Preface

The 29th edition of the Italian Convention of Computational Logic (CILC 2014), the annual meeting organized by GULP (Gruppo ricercatori e Utenti Logic Programming), was hosted by the University of Turin, from June the 16th to June the 18th 2014.

Since the first conference, which took place in Genoa in 1986, the annual conference organized by GULP is the most important occasion for meeting and exchanging ideas and experiences between Italian researchers, working in the field of computational logic. Over the years, CILC has extended its topics of interest, from the traditional logic programming to most general ones, such as the declarative programming and the application of computational logic to Artificial Intelligence and Deductive Databases.

The technical program of CILC 2014 included 29 presentations. Different topics related to computational logic have been addressed, including verification of logic programs, answer set programming, proof theory, computable set theory, machine learning, constraint logic programming, description logics, semantic web, inductive logic programming, argumentation. Associated papers (21 full papers, 8 short contributions) have been selected for publication in the proceedings of the event.

The 29th edition of CILC included also two invited talks:

- “From logic programming to argumentation and back”, given by Francesca Toni from the Department of Computing, Imperial College London;
- “Tractable approaches to consistent query answering in ontology-based-data access”, given by Riccardo Rosati from DIAG (Dipartimento di Ingegneria informatica, automatica e gestionale), Sapienza Università di Roma).

A selection of the accepted papers will appear in a special issue of a scientific journal. The complete program with links to full papers is available at http://cilc2014.di.unipmn.it/accepted.php.

We have been able to organize CILC 2014 thanks to the support of the:

- Department of Computer Science of the University of Turin;
- Department of Science and Technological Innovation (DISIT) of the University of Piemonte Orientale "Amedeo Avogadro"
- Association for Logic Programming (ALP)
- Gruppo ricercatori e Utenti Logic Programming (GULP)
- Gruppo Nazionale per il Calcolo Scientifico (GNCS) of the Istituto Nazionale di Alta Matematica (INDAM) Turismo Torino e Provincia.

We would also like to thank all the people who have contributed to the success of CILC 2014, including authors, speakers, reviewers, participants and local organizers, in particular Daniele Theseider Dupré and Matteo Spiotta. A special
thanks goes to the President of GULP, Agostino Dovier, and the secretary of GULP, Marco Gavanelli, for their helpful suggestions and support to the organization of the event.

June 2014 Laura Giordano, Valentina Gliozzi, Gian Luca Pozzato
Conference Chairs

Laura Giordano, Università del Piemonte Orientale
Valentina Gliozzi, Università degli Studi di Torino
Gian Luca Pozzato, Università degli Studi di Torino

Program Committee

Elena Bellodi, Università degli Studi di Ferrara
Stefano Bistarelli, Università degli Studi di Perugia
Davide Bresolin, Università degli Studi di Bologna
Federico Chesani, Università di Bologna
Simona Colucci, Università della Tuscia, Viterbo
Stefania Costantini, Università degli Studi di L’Aquila
Alessandro Dal Palù, Università degli Studi di Parma
Agostino Dovier, Università degli Studi di Udine
Wolfgang Faber, University of Huddersfield, UK
Stefano Ferilli, Università degli Studi di Bari “Aldo Moro”
Fabio Fioravanti, Università “G. D’Annunzio” di Chieti-Pescara
Camillo Fiorentini, Università degli Studi di Milano
Andrea Formisano, Università degli Studi di Perugia
Enrico Francioni, Free University of Bozen - Bolzano
Marco Gavanelli, Università degli Studi di Ferrara
Chiara Ghidini, Fondazione Bruno Kessler - Trento
Laura Giordano, Università del Piemonte Orientale, CHAIR
Valentina Gliozzi, Università degli Studi di Torino, CHAIR
Francesca Alessandra Lisi, Università degli Studi di Bari “Aldo Moro”
Marco Maratea, Università degli Studi di Genova
Alberto Martelli, Università degli Studi di Torino
Alessandra Mileo, National University of Ireland, Galway
Marianna Nicolosi Asmundo, Università degli Studi di Catania
Nicola Olivetti, Aix-Marseille University (AMU)
Eugenio Omodeo, Università degli Studi di Trieste
Fabio Patrizi, DIAG - Sapienza Università di Roma
Alberto Pettorossi, Università di Roma Tor Vergata
Enrico Pontelli, New Mexico State University
Gian Luca Pozzato, Università degli Studi di Torino, CHAIR
Manzirio Proietti, IASI - Consiglio Nazionale delle Ricerche
Alessandro Provetti, Università degli Studi di Messina
Luca Pulina, Università degli Studi di Sassari
Francesco Ricca, Università della Calabria
Fabrizio Riguzzi, Università degli Studi di Ferrara
Gianfranco Rossi, Università degli Studi di Parma
Pietro Sala, Università degli Studi di Verona
Luigi Sauro, Università degli Studi di Napoli “Federico II”
Local Organizing Committee

Laura Giordano, Università del Piemonte Orientale
Valentina Gliozzi, Università degli Studi di Torino
Adam Jalal, BE/CO/DA CERN, Ginevra
Gian Luca Pozzato, Università degli Studi di Torino
Matteo Spiotta, Università degli Studi di Torino
Daniele Theseider Dupré, Università del Piemonte Orientale
# Table of Contents

## Abstracts of invited talks

*Francesca Toni.*
From logic programming to argumentation and back  

*Riccardo Rosati.*
Tractable approaches to consistent query answering in ontology-based-data access  

## Full papers

*Agostino Dovier.*
Set Graphs VI: Logic Programming and Bisimulation  

*Davide Ancona, Daniela Briola, Amal El Fallah Seghrouchni, Viviana Mascardi and Patrick Taillibert.*
Exploiting Prolog for Projecting Agent Interaction Protocols  

*Mauro Ferrari, Camillo Fiorentini and Guido Fiorino.*
JTabWb: a Java framework for implementing terminating sequent and tableau calculi  

*Marco Gavanelli, Michela Milano, Stefano Bragaglia, Federico Chesani, Elisa Marengo and Paolo Cagnoli.*
Multi-Criteria Optimal Planning for Energy Policies in CLP  

*Stefania Costantini and Andrea Formisano.*
Query Answering in Resource-Based Answer Set Semantics  

*Marco Montali, Diego Calvanese and Giuseppe De Giacomo.*
Specification and Verification of Commitment-Regulated Data-Aware Multiagent Systems  

*Stefano Ferilli.*
Toward an Improved Downward Refinement Operator for Inductive Logic Programming  

*Emanuele De Angelis, Fabio Fioravanti, Alberto Pettorossi and Maurizio Proietti.*
Program Verification using Constraint Handling Rules and Array Constraint
<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generalizations</td>
<td>114</td>
</tr>
<tr>
<td><em>Loris Bozzato, Thomas Eiter and Luciano Serafini.</em></td>
<td>132</td>
</tr>
<tr>
<td>Defeasibility in contextual reasoning with CKR</td>
<td></td>
</tr>
<tr>
<td><em>Piero A. Bonatti, Iliana Petrova and Luigi Sauro.</em></td>
<td>147</td>
</tr>
<tr>
<td>A mechanism for ontology confidentiality</td>
<td></td>
</tr>
<tr>
<td><em>Domenico Cantone, Cristiano Longo and Marianna Nicolosi-Asmundo.</em></td>
<td>162</td>
</tr>
<tr>
<td>Herbrand-satisfiability of a Quantified Set-theoretical Fragment</td>
<td></td>
</tr>
<tr>
<td><em>Francesca Alessandra Lisi and Floriana Esposito.</em></td>
<td>178</td>
</tr>
<tr>
<td>Semantic Web Services for Integrated Tourism in the Apulia region</td>
<td></td>
</tr>
<tr>
<td><em>Domenico Cantone, Marianna Nicolosi-Asmundo and Ewa Orlowska.</em></td>
<td>194</td>
</tr>
<tr>
<td>A Dual Tableau-based Decision Procedure for a Relational Logic with the Universal Relation</td>
<td></td>
</tr>
<tr>
<td><em>Mathew Joseph, Gabriel Kuper and Luciano Serafini.</em></td>
<td></td>
</tr>
<tr>
<td>Query answering over Contextualized RDF knowledge with Forall-Existential Bridge Rules: Attaining Decidability using Ayclicity</td>
<td>210</td>
</tr>
<tr>
<td><em>Roberto Micalizio and Gian Luca Pozzato.</em></td>
<td></td>
</tr>
<tr>
<td>Revising Description Logic Terminologies to Handle Exceptions: a First Step</td>
<td>225</td>
</tr>
<tr>
<td><em>Stefania Costantini and Giovanni De Gasperis.</em></td>
<td></td>
</tr>
<tr>
<td>Runtime Self-Checking via Temporal (Meta-)Axioms for Assurance of Logical Agent Systems</td>
<td>241</td>
</tr>
<tr>
<td><em>Stefania Costantini and Regis Riveret.</em></td>
<td></td>
</tr>
<tr>
<td>Complex Events and Actions in Logical Agents</td>
<td>256</td>
</tr>
<tr>
<td><em>Antonis Kakas, Francesca Toni and Paolo Mancarella.</em></td>
<td>272</td>
</tr>
<tr>
<td>Argumentation for Propositional Logic and Nonmonotonic Reasoning</td>
<td></td>
</tr>
<tr>
<td><em>Rodica Ceterchi, Eugenio G. Omodeo and Alexandru I. Tomescu.</em></td>
<td>287</td>
</tr>
<tr>
<td>The representation of Boolean algebras in the spotlight of a proof checker</td>
<td></td>
</tr>
</tbody>
</table>

**Short papers**

<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Francesco Alberti, Silvio Ghilardi and Natasha Sharygina.</em></td>
<td>303</td>
</tr>
<tr>
<td>A framework for the verification of parameterized infinite-state systems</td>
<td></td>
</tr>
</tbody>
</table>
Table of Contents

Irene Benedetti, Stefano Bistarelli and Paolo Piersanti.
On Relating Voting Systems and Argumentation Frameworks 309

Davide Bresolin, Emilio Muñoz-Velasco and Guido Sciavicco.
A First Study of the Horn Fragment of the Modal Logic of Time Intervals 314

Daniela Briola, Viviana Mascardi and Davide Ancona.
Distributed Runtime Verification of JADE and Jason Multiagent Systems with Prolog 319

Simona Colucci, Silvia Giannini, Francesco M. Donini and Eugenio Di Sciascio.
Finding Commonalities in Linked Open Data 324

Carlo Combi and Pietro Sala.
Keeping interval-based functional dependencies up-to-date 330

Tommaso di Noia, Marina Mongiello and Eugenio Di Sciascio.
A computational model for MapReduce job flow 335

Eugenio Omodeo, Carla Piazza, Alberto Policriti and Alexandru I. Tomescu.
Hyper-extensionality and one-node elimination on membership graphs 341
Abstracts of invited talks
Abstract. Argumentation has gained popularity in recent years as a knowledge representation formalism to support, in particular, non-monotonic and paraconsistent reasoning. I will trace back the origins of two well-known argumentation frameworks (namely abstract argumentation and assumption-based argumentation) to work on the semantics of logic programming and abductive logic programming in the late eighties and early nineties. I will then discuss recent work with Claudia Schulz on the use of (assumption-based) argumentation to provide justifications for (non-)membership of literals in answer sets, illustrating one way in which argumentation can benefit back logic programming.
Tractable approaches to consistent query answering in ontology-based-data access

Riccardo Rosati
Dipartimento di Ingegneria informatica, automatica e gestionale (DIAG)
Sapienza Università di Roma
rosati@dis.uniroma1.it

Abstract. In this talk, we address the problem of consistent query answering in ontology-based data access (OBDA). A robust system for ontology-based data access should provide meaningful answers to queries even when the data conflicts with the ontology. This can be accomplished by adopting an inconsistency-tolerant semantics, with the consistent query answering (CQA) semantics being the most prominent example. Unfortunately, query answering under the CQA semantics has been shown to be computationally intractable, even when extremely simple ontology languages are considered. First, we present and compare the CQA semantics and other inconsistency-tolerant semantics that have been proposed to overcome the above computational problem. Then, we propose two new families of inconsistency-tolerant semantics which approximate the CQA semantics from above and from below and converge to it in the limit. We study the data complexity of conjunctive query answering under these new semantics, and show a general tractability result for all known first-order rewritable ontology languages. We also analyze the combined complexity of query answering for ontology languages of the DL-Lite family. This is joint work with Meghyn Bienvenu (CNRS and Université Paris-Sud).
Full papers
Abstract. We analyze the declarative encoding of the set-theoretic graph property known as bisimulation. This notion is of central importance in non-well-founded set theory, semantics of concurrency, model checking, and coinductive reasoning. From a modeling point of view, it is particularly interesting since it allows two alternative high-level characterizations. We analyze the encoding style of these modelings in various dialects of Logic Programming. Moreover, the notion also admits a polynomial-time maximum fix point procedure that we implemented in Prolog. Similar graph problems which are NP hard or not yet perfectly classified (e.g., graph isomorphism) can benefit from the encodings presented.

1 Introduction

Graph bisimulation is the key notion for stating equality in non-well-founded-set theory [1]. The notion is used extensively whenever cyclic properties need to be checked (e.g., in conductive reasoning [16]), in the semantics of communicating systems [12], as well as in minimizing graphs for hardware verification, and in model checking in general [9]. The problem of establishing whether two graphs are bisimilar (hence, the sets 'represented' by those graphs are equivalent) is easily shown to be equivalent to the problem of finding a maximum bisimulation of a graph into itself. This problem admits fast polynomial time algorithms that optimize a naive maximum fix point algorithm [14, 7]. As far as we know, the problem of establishing whether there exists or not a linear-time algorithm for the general case is still open.

The maximum bisimulation problem has the beauty of having two (equivalent) declarative formalizations. The first one is the definition of a particular morphism that is similar to the one used for defining other “NP” properties such as graph/subgraph simulation or isomorphism. The second one is based on the notion of coarsest stable partition which is itself similar to the property exploited for computing the minimum deterministic finite automata for a given regular language. The focus of the paper is the analysis of the programming style to be used for modeling the maximum bisimulation problem in as much declarative way as possible in some dialects of logic programming, namely, Prolog, Constraint Logic Programming on Finite Domains, Answer Set Programming, the less known, but developed for coinductive reasoning, Co-inductive Logic Programming, and the

* This research is partially supported by INdAM-GNCS.
set-based constraint logic programming language \{log\} (read setlog). The contribution of this paper is not on the direction of improving existing polynomial time algorithms; however, we also encode in Prolog a polynomial-time max fixpoint algorithm.

The paper is inserted either in the series of papers on “Set Graphs” (e.g., [13]) or in the series of papers aimed at comparing relative expressiveness of logic programming paradigms on families of problems (e.g., [4, 20]). Proposed models can be slightly modified to address the other similar properties recalled above, some of which are not believed to admit a fast implementation and, therefore, they can exploit the declarative style of logic languages and the speed of their implementations, in particular, in the case of ASP modeling.

2 Sets, Graphs, and Bisimulation

We assume the reader has some basic notions of set theory and of first-order logic with equality. We add here a set of notions needed for understanding the contribution of the paper; the reader is referred, e.g., to [1, 11], for details. Basic knowledge of Logic Programming is also assumed.

Sets are made by elements. The *extensionality principle* \((E)\) states that two sets are equal if and only if they contain the same elements:

\[ \forall z \left( z \in x \leftrightarrow z \in y \rightarrow x = y \right) \quad (E) \]

(the \(\leftarrow\), apparently missing, direction is a consequence of equality). In “classical” set theory sets are assumed to be well-founded; in particular the \(\in\) relation fulfills the so-called *foundation axiom* \((FA)\):

\[ \forall x \left( x \neq \emptyset \rightarrow (\exists y \in x)(x \cap y = \emptyset) \right) \quad (FA) \]

that ensures that a set cannot contain an infinite descending chain \(x_0 \ni x_1 \ni x_2 \ni \cdots\) of elements. In particular, let us observe that a set \(x\) such that \(x = \{x\}\) can not exist since \(x\) is not empty, its unique element \(y\) is \(x\) itself, and \(x \cap y = \{y\} \neq \emptyset\) contradicting the axiom.

On the other side, cyclic phenomena are rather common in our experience. For instance in knowledge representation, argumentation theory, operating systems design, concurrency theory, and so on. Representing and reasoning on these problems lead us in working on (cyclic) directed graphs with a distinguished entry point. Precisely, an *accessible pointed graph* \((apg)\) \(\langle G, \nu \rangle\) is a directed graph \(G = \langle N, E \rangle\) together with a distinguished node \(\nu \in N\) (the point) such that all the nodes in \(N\) are reachable from \(\nu\).

Intuitively, an edge \(a \rightarrow b\) means that the set “represented by \(b\)” is an element of the set “represented by \(a\)”. The graph edge \(\rightarrow\) stands, in a sense, for the Peano symbol \(\ni\).\(^1\) The above idea can be used to *decorate* an \(apg\), namely,

\(^1\)Let us observe the morphing \( \rightarrow \rightarrow \rightarrow \ni\), pointed out by Carla Piazza.
assigning a (possibly non-well founded) set to each of the nodes. Sinks, i.e., nodes without outgoing edges have no elements and are therefore decorated as the empty set $\emptyset$. In general, if the \textit{apg} is acyclic, it represents a well-founded set and it can be decorated uniquely starting from sinks and proceeding backward to the point (theoretically, this follows from the Mostowski’s Collapsing Lemma [11]). See Figure 1 for two examples; in particular observe that redundant nodes and edges can occur in a graph.

![Fig. 1. Two acyclic pointed graphs and their decoration with well-founded sets](image)

If the graph contains cycles, interpreting edges as membership implies that the set that decorates the graph is no longer well-founded. Non well-founded sets are often referred to as \textit{hypersets}. \textit{Anti Foundation Axiom (AFA)} [1] states that every \textit{apg} has a unique decoration. Figure 2 reports some examples. In particular, the leftmost and the central \textit{apg}s both represent the hyperset $\Omega$ which is the singleton set containing itself. Applying extensionality axiom ($E$) for verifying their equality would lead to a circular argument.

![Fig. 2. Three cyclic pointed graphs and their decoration with hypersets](image)
2.1 The notion of Bisimulation

Each apg has a unique decoration. Therefore two apgs denote the same hyperset if and only if their decoration is the same. The notion introduced to establish formally this fact is the notion of bisimulation.

Let \( G_1 = (N_1, E_1) \) and \( G_2 = (N_2, E_2) \) be two graphs, a bisimulation between \( G_1 \) and \( G_2 \) is a relation \( b \subseteq N_1 \times N_2 \) such that:

1. \( u_1 v_2 \land (u_1, v_1) \in E_1 \Rightarrow \exists v_2 \in N_2 (v_1 v_2 \land (u_2, v_2) \in E_2) \)
2. \( u_1 v_2 \land (u_2, v_2) \in E_2 \Rightarrow \exists v_1 \in N_1 (v_1 v_2 \land (u_1, v_1) \in E_1) \).

In case \( G_1 \) and \( G_2 \) are apgs pointed in \( \nu_1 \) and \( \nu_2 \), respectively, it is also required that \( \nu_1 \nu_2 \). If there is a bisimulation between \( G_1 \) and \( G_2 \) then the two graphs are bisimilar.

**Remark 1 (Bisimulation and Isomorphism).** Let us observe that if \( b \) is required to be a bijective function then it is a graph isomorphism. Establishing whether two graphs are isomorphic is an NP-problem neither proved to be NP-complete nor in P. Establishing whether \( G_1 \) is isomorphic to a subgraph of \( G_2 \) (subgraph isomorphism) is NP-complete [15]. Establishing whether \( G_1 \) is similar to a subgraph of \( G_2 \) (subgraph bisimulation) is NP-complete [6]. Instead, establishing whether \( G_1 \) is bisimilar to \( G_2 \) is in P (actually, \( O(|E_1 + E_2| \log |N_1 + N_2|) \)) [14].

In case \( G_1 \) and \( G_2 \) are the same graph \( G = (N, E) \), a bisimulation on \( G \) is a bisimulation between \( G \) and \( G \). It is immediate to see that there is a bisimulation between two apg’s \((G_1, \nu_1)\) and \((G_2, \nu_2)\) if and only if there is a bisimulation \( b \) on the graph \( G = (\{\nu\} \cup N_1 \cup N_2, \{(\nu, \nu_1), (\nu, \nu_2)\} \cup E_1 \cup E_2) \) such that \( \nu_1 \nu_2 \) (see, e.g., [7], for a proof). Therefore, we can focus on the bisimulations on a single graph; among them, we are interested in computing the maximum bisimulation (i.e., the one maximizing the number of pairs \( u \nu v \)). It can be shown that it is unique, that is an equivalence relation, and that contains all other bisimulations on \( G \). Therefore, we might restrict our search to bisimulations on \( G \) that are equivalence relations on \( N \) such that:

\[
\forall u_1, u_2, v_1, v_2 \in N (u_1 b u_2 \land (u_1, v_1) \in E \Rightarrow (\exists v_2 \in N) (v_1 b v_2 \land (u_2, v_2) \in E)) (1)
\]

The fact that we look for equivalence (hence, symmetric) relations makes the case 2 of the definition of bisimulation superfluous. We will use the following logical rewriting of (1) in some encodings:

\[
\neg \exists u_1, u_2, v_1 \in N (u_1 b u_2 \land (u_1, v_1) \in E \land \neg ((\exists v_2 \in N) (v_1 b v_2 \land (u_2, v_2) \in E))) (1')
\]

The graph obtained by collapsing nodes according to the equivalence relation is the one that allows to obtain the apg decoration, using the following procedure.

Let \( G = (\langle N, E \rangle, \nu) \) be an apg. For each node \( i \in N \) assign uniquely a variable \( X_i \), then add the equation \( X_i = \{X_j : (i, j) \in E\} \). The set of equations obtained defines the set decorating \( G \), that can be retrieved as the solution of \( X_\nu \).
Another characterization of the maximum bisimulation is based on the notion of stability. Given a set \( N \), a partition \( P \) of \( N \) is a collection of non-empty disjoint sets \( B_1, B_2, \ldots \) such that \( \bigcup B_i = N \). Let \( E \) be a relation on the set \( N \), with \( E^{-1} \) we denote its inverse relation.

A partition \( P \) of \( N \) is said to be stable with respect to \( E \) if and only if
\[
(\forall B_1 \in P)(\forall B_2 \in P)(B_1 \subseteq E^{-1}(B_2) \vee B_1 \cap E^{-1}(B_2) = \emptyset) \quad (2)
\]
which is in turn equivalent to state that there do not exist two blocks \( B_1 \in P \) and \( B_2 \in P \) such that:
\[
(\exists x \in B_1)(\exists y \in B_1)(x \in E^{-1}(B_2) \land y \notin E^{-1}(B_2)) \quad (2')
\]

We say that a partition \( P \) refines a partition \( Q \) if each block (i.e., class) of \( P \) is contained in a block of \( Q \). A class \( B_2 \) of \( P \) splits a class \( B_1 \) of \( P \) if \( B_1 \) is replaced in \( P \) by \( C_1 = B_1 \cap E^{-1}(B_2) \) and \( C_2 = B_1 \setminus E^{-1}(B_2) \); if \( C_1 \) or \( C_2 \) is empty, it is not added in \( P \). The split operation produces a refinement of a partition \( P \); if \( P \) is stable with respect to \( E \), no split operations changes \( P \).

It can be shown that given a graph \( G = \langle N, E \rangle \), starting from the partition \( P = \{N\} \), after at most \( |N| - 1 \) split operations a procedure halts determining the coarsest stable partition (CSP) w.r.t. \( E \). Namely, the partition is stable and any other stable partition is a refinement of it. Moreover, and this is relevant to our task, the CSP corresponds to the partition induced by the maximum bisimulation, hence this algorithm can be employed to compute it in polynomial time. Paige and Tarjan showed us the way for fast implementations in [14].

3 Logic Programming Encoding of Bisimulation

We first focus on the logic programming encoding or the definition of bisimulation (1) or (1') and of the part needed to look for the maximum bisimulation on an input \( \text{apg} \). We impose the relation is symmetric and reflexive. In the remaining part of the paper we assume that \( \text{apg}'s \) are represented by facts \( \text{node}(1) \). \( \text{node}(2) \). \( \text{node}(3) \). \ldots \) for enumerating the nodes, and facts \( \text{edge}(u,v) \) where \( u \) and \( v \) are nodes, for enumerating the edges. For the sake of simplicity, we also assume that node 1 is the point of the \( \text{apg} \).\(^2\)

3.1 Prolog

The programming style used in the Prolog encoding is generate & test. The core of the encoding is reported in Figure 3. A bisimulation is represented by a list of pairs of nodes \( (U, V) \). Assuming a “guessed” bisimulation is given as input, for every guessed pair the morphism property (1) is checked. As usual in Prolog, the “for all” property is implemented by a recursive predicate (although a slightly

\(^2\) Complete codes are available at http://clp.dimi.uniud.it
more compact \texttt{foreach} statement is available in most Prolog systems and will be used in successive encodings).

\texttt{bis/1} is called by a predicate that guesses a bisimulation of size at least \( k \) between nodes, itself called by a meta predicate that increases the value of \( k \) until no solution is found. The \texttt{guess} predicate forces all identities, all the pairs between nodes without outgoing edges, and imposes symmetries; this extra part of the code is rather boring and we have omitted the code. As a (weak) search strategy, the guess predicate tries first to insert as much pairs as possible: this will explain the difference of computational times on different benchmarks of the same size.

### 3.2 CLP(FD)

The programming style is constraint \& generate. In this case the bisimulation is stored in matrix, say \( B \), of Boolean variables. \( B[i, j] = 1 \) means that \( i \) \textit{bis} \( j \) \( (B[i, j] = 0 \) means that \( \neg(i \textit{bis} j) \)). We omit the definitions of the \texttt{reflexivity} predicate that sets \( B[i, i] = 1 \) for all nodes \( i \) and of the \texttt{symmetry} predicate that sets \( B[i, j] = B[j, i] \) for all pair of nodes \( i \) and \( j \). Let us focus on the morphism requirements (1). \texttt{morphism/2} collects all edges and calls \texttt{morphism/3}. This predicate scans each edge \((U, V)\) and then each node \( U \) and adds the property that if \( B[U, U1] = 1 \) then \( \sum_{(U1, V1) \in E} B[V, V1] = 1 \). Let us observe that \( O(|E||N|) \) of these constraints are generated. We omit the definitions of some auxiliary predicates, such as \texttt{access(X,Y,B,N,BXY)} that simply sets \( BXY = B[X,Y] \). The whole encoding is longer and perhaps less intuitive than the Prolog one. However, the search of the \textit{maximum} bisimulation is not delegated to a meta predicate as in Prolog but it is encoded directly into the \texttt{maximize} option of the labeling primitive. The “down” search strategy, trying to assign 1 first, is similar to the strategy used in the Prolog code.

### 3.3 ASP

ASP encodings allow to define explicitly the bisimulation relation. Two rules are added for forcing symmetry and reflexivity. Then a non-deterministic choice is added to each pair of nodes. The great declarative advantage of ASP in this case is the availability of constraint rules that allows to express universal quantification (negation of existential quantification). The morphism requirement (1') can be therefore encoded as it is, with the unique addition of the \texttt{node} predicates needed for grounding (Figure 5). Then we define the notion of representative nodes (the nodes of smaller index among the nodes equivalent to it) and minimize the number of them. This has proven to be much more efficient that maximizing the size of \texttt{bis}. A final remark on the expected size of the grounding. Both the constraint and the definition of \texttt{one\_son\_bis} ranges over all edges and another free node: this generates a grounding of size \( O(|E||N|) \).
bis(B) :- bis(B,B).
% Recursively analyze B
bis([],_).
bis([ (U1,U2) |RB],B) :- %%% if U1 bis U2
  successors(U1,SU1), %%% Collect the successors SU1 of U1
  successors(U2,SU2), %%% Collect the successors SU2 of U2
  allbis(SU1,SU2,B), %%% Then recursively consider SU1
  bis(RB,B).
allbis([],_,_).
allbis([V1 | SU1],SU2,B) :- %%% If V1 is a successor of U1
  member(V2,SU2), %%% there is a V2 successor of U2
  member((V1,V2),B), %%% such that V1 bis V2
  allbis(SU1,SU2,B).
successors(X,SX) :- findall(Y,edge(X,Y),SX).

Fig. 3. Prolog encoding of the bisimulation definition. Maximization code is omitted.

bis :- size(N), M is N*N, %%% Define the N * N Boolean
  length(B,M), domain(B,0,1), %%% Matrix B
  constraint(B,N), Max #= sum(B), %%% Max is the number of pairs
  labeling([maximize(Max),ffc,down],B). %%% in the bisimulation

constraint(B,N) :- reflexivity(N,B), symmetry(1,2,N,B), morphism(N,B).
morphism(N,B) :-
  findall( (X,Y),edge(X,Y),EDGES),
  foreach( E in EDGES, U2 in 1..N, morphismcheck(E,U2,N,B)).
morphismcheck( (U1,V1),U2,N,B) :-
  access(U1,U2,B,N,BU1U2), % Flag BU1U2 stands for (U1 B U2)
  successors(U2, SuccU2), % Collect all edges (U2,V2)
  collectlist(SuccU2,V1,N,B,BLIST),% BLIST contains all possible flags BV1V2
  BU1U2 #=< sum(BLIST). % If (U1 B U2) there is V2 s.t. (V1 B V2)

Fig. 4. Portion of the CLP(FD) encoding of the bisimulation definition

%% Reflexivity and Symmetry
bis(I,I) :- node(I).
bis(I,J) :- node(I;J), bis(J,I).
%%% Nondeterministic choice
{bis(I,J)} :- node(I;J).
%%% Morphism requirement (1’)
:- node(U1;U2;V1), bis(U1,U2), edge(U1,V1), not one_son_bis(V1,U2).
one_son_bis(V1,U2) :- node(V1;U2;V2), edge(U2,V2), bis(V1,V2).

%%% Minimization  (max bisimulation)
non_rep_node(A) :- node(A), bis(A,B), B < A.
rep_node(A) :- node(A), not non_rep_node(A).
rep_nodes(N) :- N=#sum[rep_node(A)].
#minimize [rep_nodes(N)=N].

Fig. 5. ASP encoding of the bisimulation definition
3.4 co-LP

In this section we exploit a less standard logic programming dialect. Coinductive Logic Programming (briefly co-LP) was introduced by Gupta et al. [17] and recently presented in a concise way in [2], where computability results and a working SWI interpreter are provided. The same syntax of pure Prolog should be used. The differences lay in the semantics: the maximum fix point of a coinductive predicate is looked for, as opposite to the least fix point of classical logic programming. Although this can easily lead to a non recursively enumerable semantics, the finiteness of the graphs makes this option available for this problem. As a matter of fact, the piece of code reported in Figure 6 encodes the problem and, by looking for the maximum fix point, the maximum bisimulation is computed without the need of additional minimization/maximization directives. \texttt{bis} and \texttt{allbis} are declared as coinductive. The definition of \texttt{successors} is the same as in Figure 3 and declared as inductive, as well as the \texttt{member} predicate.

\begin{verbatim}
bis(U,V) :- successors(U,SU), successors(V,SV),
                allbis(SU,SV), allbis(SV,SU).
allbis([],_).
allbis([U|R],SV) :- member(V,SV), bis(U,V), allbis(R,SV).
\end{verbatim}

\textbf{Fig. 6.} co-LP (complete) encoding of the definition of Bisimulation

4 Logic Programming Encoding of CSP

We focus first on the encoding of the definition of stable partition (2) and finally on the (less declarative) computation of the CSP.

4.1 Prolog

The programming style is generate & test. A partition is a list of non-empty lists of nodes (blocks). Sink nodes (if any) are deterministically set in the first block. Possible partitions of increasing size are non-deterministically generated until the first stable one is found. Once the partition is guessed the verify part is made by a double selection of blocks within the list of blocks. The main predicate that encodes property (2) is the following:

\begin{verbatim}
stablecond(B1,B2) :- edgeinv(B2,InvB2),
                   (subseteq(B1,InvB2) ; emptyintersection(B1,InvB2)).
\end{verbatim}

where \texttt{edgeinv} collects the nodes that enter into \texttt{B2} (definable as \texttt{findall(X, (edge(X,Y), member(Y,B)), REVB)}) while the two set-theoretic predicates are defined through list operations.
4.2 CLP(FD)

In this case the data structure used is a mapping from nodes to blocks indexes, stored as a list of finite domain variables. The set inclusion and empty intersection requirement of (2) are not naturally implemented by a constraint & generate style. As in the encoding 3.2 maximization is forced by a parameter of the labeling; some symmetry breaking is encoded (e.g., sink nodes are deterministically forced to stay in partition number one). We only report the excerpt of the encoding, where we made use of the `foreach` built-in. With a rough analysis, the number of constraints needed is \(O(|N|^3)\) but each constraint generated by `alledge` can be of size \(|N|\) itself.

4.3 ASP

Also in this case ASP allows a concise encoding (Figure 8). The assignment is implemented defining the function `inblock/2`. The possibility of reasoning “a posteriori” and the availability of the constraint rule allows to naturally encode the property (‘2′). The remaining part of the code is devoted to symmetry breaking and minimization of the number of blocks. The bottleneck for the grounding stage is the constraint rule that might generate \(O(|N|^4)\) ground instantiations.

4.4 `{log}

The CLP language `{log}, originally presented in [5], populated with several set-based constraints such as the disjoint constraint (`disj`—imposing empty intersection) in [8] and later augmented with Finite Domain constraints in [3] is a set-based extension of Prolog (and a particular case of constraint logic programming language). Encoding the set-theoretic stable property (2) is rather natural in this case. We report the definition in Figure 9. `subset`, `disj`, `in` are built-in constraints. Similarly, restricted universal quantifiers (`forall(X in S, Goal)` and intensional set formers (`{X : Goal(X)}`) are accepted.

4.5 Computing the coarsest stable partition

We have implemented the maximum fixpoint procedure for computing the coarsest stable partition in Prolog. Initially nodes are split into (at most) two classes: internal and non internal nodes. For each node \(U\), a list of pairs \(U-I\) is computed by stating that \(U\) is assigned to block \(I\). Then a possible splitter is found and, in case, a split is executed. The procedure terminates in at most \(n-1\) steps where \(n\) is the number of nodes. The Prolog code is reported in Appendix (Figure 12).

5 Experiments

Although the focus of this work is on the expressivity of the declarative encoding (being this problem solved by fast algorithms in literature, such as [14,
stability(B,N) :-
    foreach( I in 1..N, J in 1..N, stability_cond(I,J,B,N)).

stability_cond(I,J,B,N) :- % Blocks BI and BJ are considered
    inclusion(I,N,I,J,B, Cincl), % Nodes in 1..N are analyzed
    emptyintersection(I,N,I,J,B,Cempty), % Cincl and Cempty are reified
    Cincl + Cempty #> 0. % OR condition

inclusion(X,N,_,_,_, 1) :- X>N,!.
inclusion(X,N,I,J,B, Cout) :- % Node X is considered
    alledges(X,B,J,Flags), % Flags stores existence of edge (X,Y) with Y in BJ
    LocFlag #= ((B[X] #= I) #=> (Flags #> 0)), %% Inclusion check:
    X1 is X+1, % If X in BI then X in E-1(BJ)
    inclusion(X1,N,I,J,B,Ctemp), % Recursive call
    Cout #= Ctemp*LocFlag. % AND condition (forall nodes it should hold)

alledges(X,B,J,Flags) :- % Collect the successors of X
    successors(X,OutgoingX), % And use them for assigning the Flags var
    alledgesaux(OutgoingX,B,J,Flags).
alledgesaux([],B,J,0).
alledgesaux([Y|R],B,J,Flags) :- % The Flags variable is created
    alledgesaux(R,B,J,F1), % Recursive call.
    Flags #= (B[Y] #= J) + F1. % Add "1" iff there is edge (X,Y) and BY = J

Fig. 7. Excerpt of the CLP(FD) encoding of the stable partition property

blk(I) :- node(I).

% Brooklyn
inblock(A,B):blk(B)1 :- node(A).
% STABILITY (2’)
:- blk(B1;B2), node(X;Y), X != Y, inblock(X,B1), inblock(Y,B1),
   connected(X,B2), not connected(Y,B2).
connected(Y,B) :- edge(Y,Z),blk(B),inblock(Z,B).
% Basic symmetry-breaking rules (optional)
:- node(A), internal(A), inblock(A,1).
internal(X) :- edge(X,Y).
leaf(X) :- node(X), not internal(X).
non_empty_block(B) :- node(A), blk(B), inblock(A,B).
empty_block(B) :- blk(B), not non_empty_block(B).
:- blk(B1;B2), 1 < B1, B1 < B2, empty_block(B1), non_empty_block(B2).
% Minimization
number_blocks(N) :- N=#sum[non_empty_block(B)].
#minimize [number_blocks(N)=N].

Fig. 8. ASP complete encoding of the stable partition property

stable(P) :-
    forall(B1 in P, forall(B2 in P, stablecond(B1,B2) ) ).
stablecond(B1,B2) :-
    edgeinv(B2,InvB2) &
    (subset(B1,InvB2) or disj(B1,InvB2)).
edgeinv(A,B) :-
    B = {X : exists(Y,(Y in A & edge(X,Y)))}.

Fig. 9. {log} encoding of the stable partition property
7), we have reported the excerpt of the running times of the various proposed encodings on some families of graphs, parametric on their number of nodes (Figure 10). Results can give us some additional information on the possibilities and on the intrinsic limits of the logic programming dialects analyzed. All experiments are made on a laptop 2.4GHz Intel Core i7, 8GB memory 1600MHz DDR3, OSX 10.9.2. Systems used are B-Prolog Version 7.8#5 [19], clingo 3.0.5 (clasp 1.3.10) [10], and SWI Prolog Version 6.4.1 [18]. In particular, SWI Prolog has been used in the co-LP tests, thanks to its rational terms handling. On the other Prolog encodings B-Prolog proved to be from 2 to 3 times faster than SWI and it has been therefore used. Speed-up increased still further using tabling for the predicate edge but we have exploited this additional feature in the experiments in Table 5 only. We tested the codes on five families of graphs $G_1$–$G_5$ parametric on the number of nodes $n$ (see Figure 10).

- Graph $G_1$ is an acyclic graph with $n - 1$ edges, where $n$ classes are needed.
- $G_2$ is a cyclic graph with $n$ nodes and edges. If $n$ is even, just two classes are sufficient; if $n$ is odd, $\frac{n+1}{2}$ classes are needed. This is why in some experiments we have two columns with this family of graphs.
- $G_3$ is a binary tree populated following a breadth-first visit, with $n - 1$ edges.
- $G_4$ is, in a sense, symmetrical w.r.t. $G_1$: it is a complete graph with $n^2$ edges but just one class is sufficient.
- $G_5$ is a multilevel (cyclic) graph.

The results on the encoding of the bisimulation definition 1 are reported in Tables 1–3. From a quick view one might notice that the ASP encoding is a clear winner. Prolog generate & Test and the co-LP interpreter run in reasonable time on very small graphs only (Prolog is used without tabling, tabling the edge predicate allows a speed-up of roughly 4 times). The CLP approach becomes unpractical soon in the case for the complete graph $G_4$ where the $n^2$ edges generate too many constraints for the stack size when $n \geq 50$ (as reported in Section 3.2, $O(|E||N|)$ constraints are added: in this case they are $O(n^5)$; moreover each of those constraints includes a sum of $n$ elements in this case).

Let us observe that the complete graph $G_4$ produces the highest grounding
times in the ASP case. As reported in Section 3.3 grounding size is expected $O(|E||N|) = O(n^3)$ in this case. This has been verified experimentally (table not reported); in particular, for $G_4, n = 200$ the grounded file (obtained with the option $-t$) is of size 275MB. Moreover, by a simple regression analysis of Table 3, the time needed for grounding is shown to be proportional to $n^6$ for graph $G_4$.

The results on the encoding of the coarsest stable partition definition 2 are reported in Table 4. Also in this case ASP is a clear winner, although in this case smaller graphs can be handled by all approaches. We have omitted the $\{\log\}$ running times. This system proved to be definitely the slowest; just to have an idea, for $G_1, n = 5$ the computation took roughly 5 hours.

We conclude with the testing of the encoding of the polynomial time procedure of coarsest stable partition computation by maximum fixpoint and splits. In graphs $G_2'$ and $G_3$ tabling the edge predicate improved the running time of two orders of magnitude (reported times are those using tabling). As a further consideration, we started finding stack overflow for $n = 5000$. Moreover, the experimental complexity detected by a regression analysis on table 5 is $O(|N|^3)$ in all columns, which is rather good, considering the purely declarative nature of the encoding (fast solvers such as [7] run in $O(|N|)$ in acyclic graphs such as $G_1$ and $G_3$, and in the cyclic multi-level graph $G_5$, while they run in $O(|E| \log |N|) = O(|N|^2 \log |N|)$ in the other cases. By the way, complete graph $G_4$ could be solved in time $O(1)$ with a simple preprocessing).

6 Conclusions

We have encoded the two properties characterizing the bisimulation definition, and in particular, solving the maximum bisimulation problem, using some dialects of Logic Programming. As a general remark, the guess & verify style of Prolog (and of ASP) allows to define the characterizing properties to be verified ‘a posteriori’, on ground atoms. In CLP instead, those properties are added as constraints to lists of values that are currently non instantiated and this makes things much more involved, and has a negative impact on code readability. The expressive power of the constraint rule of ASP allows a natural and compact encoding of “for all” properties and this improved the conciseness of the encoding (and readability in general); recursion should be used instead for it in Prolog and CLP. co-LP (resp., $\{\log\}$) allows to write excellent code for property (1) (resp., property (2)). However, since they are implemented using meta interpreters (naive in the case of co-LP) their execution times are prohibitive for being used in practice.

The ASP encoding is also the winner from the efficiency point of view, as far as a purely declarative encoding of the NP property is concerned. This would suggest the reader that this is the best dialect to be used to encode graph properties if a polynomial time algorithm is not yet available (or it does not exist at all). This is not the case of the maximum bisimulation problem where polynomial time algorithms for computing the coarsest stable partition can be.
Table 1. Property (1). Running time (ms) for the Prolog (BP) and the co-LP encoding on very small graphs. so=stack overflow. \(G_5\) is not considered for these values of \(n\), since its structure requires at least 11 nodes. Let us observe that \(G_3\) is a tree of height 3 for \(n = 7\) and of height 4 for \(n = 8\). This explain the strange behavior of co-LP.

<table>
<thead>
<tr>
<th>(n)</th>
<th>(G_1) (\text{BP} )co-LP</th>
<th>(G_2) (\text{BP} )co-LP</th>
<th>(G_3) (\text{BP} )co-LP</th>
<th>(G_4) (\text{BP} )co-LP</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>18</td>
<td>3</td>
<td>16</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>508</td>
<td>8</td>
<td>179</td>
<td>33</td>
</tr>
<tr>
<td>7</td>
<td>33252</td>
<td>14</td>
<td>8240</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>4303576</td>
<td>28</td>
<td>203191</td>
<td>118</td>
</tr>
</tbody>
</table>

Table 2. Property (1). Running time (ms) for clp(fd) on small graphs (C=constraint, S=search). so=stack overflow. For \(G_5\) nodes are \(n + 1\)

<table>
<thead>
<tr>
<th>(n)</th>
<th>(G_1) (C)</th>
<th>(G_2) (C)</th>
<th>(G_3) (C)</th>
<th>(G_4) (C)</th>
<th>(G_5) (C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>17</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>0</td>
<td>7</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>30</td>
<td>22</td>
<td>0</td>
<td>31</td>
<td>3</td>
<td>29</td>
</tr>
<tr>
<td>40</td>
<td>59</td>
<td>0</td>
<td>64</td>
<td>6</td>
<td>68</td>
</tr>
<tr>
<td>50</td>
<td>147</td>
<td>0</td>
<td>141</td>
<td>12</td>
<td>141</td>
</tr>
<tr>
<td>60</td>
<td>240</td>
<td>0</td>
<td>277</td>
<td>38</td>
<td>259</td>
</tr>
<tr>
<td>70</td>
<td>428</td>
<td>0</td>
<td>492</td>
<td>41</td>
<td>460</td>
</tr>
<tr>
<td>80</td>
<td>705</td>
<td>0</td>
<td>810</td>
<td>61</td>
<td>762</td>
</tr>
<tr>
<td>90</td>
<td>1119</td>
<td>0</td>
<td>1463</td>
<td>99</td>
<td>1179</td>
</tr>
<tr>
<td>100</td>
<td>1703</td>
<td>0</td>
<td>1915</td>
<td>158</td>
<td>1803</td>
</tr>
</tbody>
</table>

Table 3. Property (1). Running time (ms) for clingo on medium graphs (G=grounding+preprocessing, S=search). For \(G_5\) nodes are \(n + 1\)

<table>
<thead>
<tr>
<th>(n)</th>
<th>(G_1) (G)</th>
<th>(G_2) (G)</th>
<th>(G_3) (G)</th>
<th>(G_4) (G)</th>
<th>(G_5) (G)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>670</td>
<td>50</td>
<td>650</td>
<td>100</td>
<td>620</td>
</tr>
<tr>
<td>110</td>
<td>880</td>
<td>80</td>
<td>830</td>
<td>170</td>
<td>820</td>
</tr>
<tr>
<td>120</td>
<td>1180</td>
<td>100</td>
<td>1090</td>
<td>170</td>
<td>1090</td>
</tr>
<tr>
<td>130</td>
<td>1510</td>
<td>130</td>
<td>1340</td>
<td>300</td>
<td>1440</td>
</tr>
<tr>
<td>140</td>
<td>1890</td>
<td>150</td>
<td>1670</td>
<td>290</td>
<td>1690</td>
</tr>
<tr>
<td>150</td>
<td>2270</td>
<td>180</td>
<td>2030</td>
<td>430</td>
<td>2120</td>
</tr>
<tr>
<td>160</td>
<td>2740</td>
<td>230</td>
<td>2510</td>
<td>440</td>
<td>2590</td>
</tr>
<tr>
<td>170</td>
<td>3310</td>
<td>240</td>
<td>2960</td>
<td>760</td>
<td>3120</td>
</tr>
<tr>
<td>180</td>
<td>3930</td>
<td>280</td>
<td>3520</td>
<td>690</td>
<td>3600</td>
</tr>
<tr>
<td>190</td>
<td>4600</td>
<td>320</td>
<td>4000</td>
<td>1130</td>
<td>4250</td>
</tr>
<tr>
<td>200</td>
<td>5350</td>
<td>360</td>
<td>4910</td>
<td>1140</td>
<td>4850</td>
</tr>
</tbody>
</table>
Table 4. Property (2). Running time (ms) for the Prolog (BP) and CLP(FD) (C=constraints, S=search) and ASP (G=grounding+preprocessing, S=search) encodings on very small graphs.

<table>
<thead>
<tr>
<th>n</th>
<th>G_1 BP</th>
<th>C</th>
<th>S</th>
<th>G_2 BP</th>
<th>C</th>
<th>S</th>
<th>G_3 BP</th>
<th>C</th>
<th>S</th>
<th>G_4 BP</th>
<th>C</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>11</td>
<td>4</td>
<td>2</td>
<td>13</td>
<td>2</td>
<td>0</td>
<td>13</td>
<td>1</td>
<td>0</td>
<td>129</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>26</td>
<td>31</td>
<td>58</td>
<td>0</td>
<td>38</td>
<td>1</td>
<td>0</td>
<td>32</td>
<td>5</td>
<td>0</td>
<td>230</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>351</td>
<td>78</td>
<td>146</td>
<td>15</td>
<td>79</td>
<td>12</td>
<td>0</td>
<td>79</td>
<td>5</td>
<td>0</td>
<td>372</td>
<td>19</td>
</tr>
<tr>
<td>8</td>
<td>5057</td>
<td>115</td>
<td>1257</td>
<td>1</td>
<td>232</td>
<td>8</td>
<td>3</td>
<td>119</td>
<td>11</td>
<td>0</td>
<td>803</td>
<td>47</td>
</tr>
<tr>
<td>9</td>
<td>76044</td>
<td>145</td>
<td>29507</td>
<td>196</td>
<td>384</td>
<td>197</td>
<td>5</td>
<td>151</td>
<td>33</td>
<td>0</td>
<td>1895</td>
<td>104</td>
</tr>
<tr>
<td>10</td>
<td>1338632</td>
<td>179</td>
<td>222047</td>
<td>1</td>
<td>198</td>
<td>5</td>
<td>8</td>
<td>355</td>
<td>139</td>
<td>0</td>
<td>5352</td>
<td>143</td>
</tr>
</tbody>
</table>

Table 5. Running time (ms) of the B-Prolog encoding of the fixpoint procedure for computing the Coarsest Stable Partition on large graphs. * indicates that in those columns the number of nodes is n + 1.

<table>
<thead>
<tr>
<th>n</th>
<th>G_1</th>
<th>G_2^*</th>
<th>G_2</th>
<th>G_3</th>
<th>G_4</th>
<th>G_5^*</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>24</td>
<td>38</td>
<td>22</td>
<td>15</td>
<td>40</td>
<td>34</td>
</tr>
<tr>
<td>200</td>
<td>137</td>
<td>208</td>
<td>87</td>
<td>54</td>
<td>253</td>
<td>244</td>
</tr>
<tr>
<td>300</td>
<td>885</td>
<td>968</td>
<td>324</td>
<td>204</td>
<td>1659</td>
<td>1458</td>
</tr>
<tr>
<td>600</td>
<td>2769</td>
<td>2804</td>
<td>697</td>
<td>463</td>
<td>5613</td>
<td>4553</td>
</tr>
<tr>
<td>800</td>
<td>6544</td>
<td>6169</td>
<td>1365</td>
<td>809</td>
<td>12895</td>
<td>10735</td>
</tr>
<tr>
<td>1000</td>
<td>12639</td>
<td>11650</td>
<td>2145</td>
<td>1210</td>
<td>24462</td>
<td>20069</td>
</tr>
</tbody>
</table>

Fig. 11. An overall picture on the computational results on graph G_2. Encoding (1)—left, encoding (2)—right. Logarithmic scales for axis have been used.
employed. The one implemented in Prolog and reported in Appendix proved also to be the fastest approach presented in this paper.

References

Appendix

stable_comp(Final, Nclasses) :-
  findall(X, node(X), Nodes),
  initialize(Nodes, Initial),
  maxfixpoint(Initial, 2, Final, Nclasses). % start with "2"
%%% maxfixpoint procedure. If possible, split, else stop.
maxfixpoint(AssIn, I, AssOut, C) :-
  split(I, AssIn, AssMid),!,
  I1 is I+1,
  maxfixpoint(AssMid, I1, AssOut, C).
%%% When stop, simply compute the number of classes used
maxfixpoint(Stable, C, Stable, C1) :-
  count_classes(C, Stable, C1).
%%% Split operation.
%%% First locate a block that can be split. Then find the splitter
split(MaxBlock, AssIn, AssMid) :-
  between(1, MaxBlock, I),
  findall(X, member(X-I, AssIn), BI),
  BI = [[] | _ | _], % BI might be split (not empty, not singleton)
  findall(Q, (member(V-Q, AssIn), edge(W, V), member(W, BI)), SP),
  sort(SP, SPS), member(J, SPS),
  findall(Z, (member(Y-J, AssIn), edge(Z, Y)), BJinv),
  my_delete(BI, BJinv, [D | DELTA]), % The difference is computed when not empty
  MaxBlock1 is MaxBlock + 1,
  update(AssIn, AssMid, MaxBlock1, [D | DELTA]).

%%% Initial partition: Sinks -> B1; Internal -> B2
initialize([], []).
initialize([A|R], [A-B|Ass]) :- (internal(A), !, B=2; B=1), initialize(R, Ass).
%%% AUXILIARY
count_classes(C, Stable, C1) :- (C > 3, !, C1 = C;
  C =< 2, member(_-1, Stable), member(_-2, Stable), !, C1=2; C1 = 1).
my_delete([], []).
my_delete([A|R], DEL, S) :- select(A, DEL, DEL1), !,
  my_delete(R, DEL1, S).
my_delete([A|R], DEL, [A|S]) :- my_delete(R, DEL, S).
update([], []).
update([X-[_]|R], [X-I|S], I, D) :- select(X, D, D1), !,
  update(R, S, I, D1).
update([X-J|R], [X-J|S], I, D) :- update(R, S, I, D).
internal(X) :- edge(X, _).

Fig. 12. Prolog computation of the CSP as a maxfixpoint procedure (complete code)
Exploiting Prolog for Projecting Agent Interaction Protocols

Davide Ancona, Daniela Briola, Amal El Fallah Seghrouchni, Viviana Mascardi, and Patrick Taillibert

1DIBRIS, University of Genova, Italy
{Davide.Anocona,Daniela.Briola,Viviana.Mascardi}@unige.it
2LIP6, University Pierre and Marie Curie, Paris, France
{Amal.Elfallah,Patrick.Taillibert}@lip6.fr

Abstract. Constrained global types are a powerful means to represent agent interaction protocols. In our recent research we demonstrated that they can be used to represent complex protocols in a very compact way, and we exploited them to dynamically verify correct implementation of a protocol in a real MAS framework, Jason. The main drawback of our previous approach is the full centralization of the monitoring activity which is delegated to a unique monitor agent. This approach works well for MASs with few agents, but could become unsuitable in communication-intensive and highly-distributed MASs where hundreds of agents should be monitored.

In this paper we define an algorithm for projecting a constrained global type onto a set of agents $Ags$, by restricting it to the interactions involving agents in $Ags$, so that the outcome of the algorithm is another constrained global type that can be safely used for verifying the compliance of the sub-system $Ags$ to the protocol specified by the original constrained global type. The projection mechanism is implemented in SWI Prolog and is the first step towards distributing the monitoring activity, making it safer and more efficient: the compliance of a MAS to a protocol could be dynamically verified by suitably partitioning the agents of the MAS into small sets of agents, and by assigning to each partition $Ags$ a local monitor agent which checks all interactions involving $Ags$ against the projected constrained global type. We leave for further investigation the problem of finding suitable partitions of agents in a MAS, to guarantee that verification through projected types and distributed agents is equivalent to verification performed by a single centralized monitor with a unique global type.

Keywords: Constrained Global Type, Projection, Dynamic Verification, Agent Interaction Protocol, SWI Prolog

* The long version of this paper appears in the informal proceedings of the Second International Workshop on Engineering Multi-Agent Systems (EMAS 2014) with title “Efficient Verification of MASs with Projections”. This is the shortened version presented at CILC 2014.
1 Introduction and Motivation

Distributed monitoring of agent interaction protocols is interesting for various reasons. First, the distribution of monitoring reduces the bottleneck issue due to the potentially high number of communications between the central monitor and the agents of the system. Consequently, the communications are localized according to the distribution topology (how many local monitors are available and where they are localized in the system), improving the efficiency of the monitoring. As usual, distribution increases the robustness of the whole system and prevents for a breakdown, crash or failure of the system. In particular, in the context of distributed environments, having a robust monitoring system requires to distribute the monitoring on several agents which ensure their prompt reaction to events. In addition, the distributed approach is more suitable than the centralized one for asynchronous and/or distributed contexts.

In order to distribute the monitoring activity, the first step to face is to distribute the specification of the global interaction protocol in such a way that a subset of agents can monitor a subset of the interactions, still respecting the constraints stated by the global protocol.

In this paper, we address this first step by defining and implementing an algorithm for projecting the protocol representation onto subsets of agents, and then allowing interactions taking place within these subsets to be monitored by local monitors. Automatically identifying these subsets of agents in order to guarantee that the distributed monitoring behaves like the centralized one goes beyond the aims of this paper, but is matter of our current research activity.

Another interesting issue concerns dynamic redistribution of monitoring agents; even if not explored in this work, projected types could be recomputed dynamically to balance the load among local monitors depending on the currently available resources, and according to some “meta-protocol”.

The formalism that we exploit for representing and dynamically verifying agent interaction protocols, is constrained global types [2]. Global types [4] are behavioral types for specifying and verifying multiparty interactions between distributed components. We took inspiration from global types to propose “constrained global types”, suitable for representing agent interaction protocols. They are based on interactions, namely communicative events between two agents; interaction types, modeling the message pattern expected at a certain point of the conversation; producers and consumers which allow us to express constrained shuffle of interaction traces. On top of these components, type constructors are used to model sequences, choices, concatenation and shuffle of protocols.

In our recent research we demonstrated that constrained global types can be used to represent complex protocols in a very compact way, and we exploited them to detect deviations from the protocol in a real MAS framework based on logic programming, Jason [1], and in the Java-based JADE framework1, thanks to a bidirectional Java-Prolog interface [3]. Extensions of the original formalism with attributes have been described [5] and exploited to model a complex, real

1 http://jade.tilab.com/.
protocol in the railway domain [6]. The integration of these This paper shows how a constrained global type can be projected onto a set of agents $Ags$, obtaining another constrained global type which contains only interactions involving agents in $Ags$. Although the projection is always possible, this does not mean that it is always useful: as an example, the Alternating Bit Protocol discussed in this paper can be projected onto any individual agent in the MAS, but needs to be monitored in a centralized way to verify all its constraints.

The paper is organized in the following way: Section 2 briefly overviews the state of the art in distributing the monitoring activity of complex systems; Section 3 gives the technical background on constrained global types needed for presenting the projection algorithm in Section 4, Section 5 describes the implementation of the algorithm in SWI Prolog, Section 6 describes the algorithm at work, and Section 7 concludes.

2 State of the art

Many frameworks and formalism for monitoring the runtime execution of a distributed system have been proposed in the last years.

One of the most recent and relevant works in this area is SPY (Session Python) [7], a tool chain for runtime verification of distributed Python programs against Scribble (http://www.scribble.org) protocol specifications. Scribble is a language to describe application-level protocols among communicating systems initially proposed by Kohei Honda. Given a Scribble specification of a global protocol, the SPY tool chain validates consistency properties, such as race-free branch paths, and generates Scribble (i.e. syntactic) local protocol specifications for each participant (role) defined in the protocol. At runtime, an independent monitor (internal or external) is assigned to each Python endpoint and verifies the local trace of communication actions executed during the session. This work shares the same motivations and approach with our work, and like our work concentrates on the projection of the global type to the local one rather than on the criteria for identifying in an automatic way how to distribute the monitoring activity. The main differences lie in the expressive power of the two languages, which is higher for constrained global types due to the constrained shuffle operator which is missing in Scribble, and in the availability of tools for statically verifying properties of Scribble specifications, which are not available for constrained global types.

Many other approaches for runtime monitoring of distributed systems and MASs exist, but with no emphasis on the projection from global to local monitors. This represents the main difference between those proposals and ours; the long version of this paper provides a detailed overview of many recent ones.
3 Background

This section briefly recaps on constrained global types, omitting their extension with attributes [5] because the projection algorithm discussed in Section 4 is currently defined on “plain” constrained global types only.

Constrained global types (also named “types” in the sequel, when no ambiguity arises) are defined starting from the following elements:

**Interactions**\(^2\). An interaction \(a\) is a communicative event taking place between two agents. For example, \(\text{msg(}\text{right_robot, right_monitor, tell, put_sock}\text{)}\) is an interaction involving the sender \(\text{right_robot}\) and the receiver \(\text{right_monitor}\), with performative \(\text{tell}\) and content \(\text{put_sock}\).

**Interaction types**. Interaction types model the message pattern expected at a certain point of the conversation. An interaction type \(\alpha\) is a predicate on interactions. For example, \(\text{msg(}\text{right_robot, right_monitor, tell, put_sock}\text{)} \in \text{put_right_sock}\) means that interaction \(\text{msg(}\text{right_robot, right_monitor, tell, put_sock}\text{)}\) has type \(\text{put_right_sock}\).

**Producers and consumers**. In order to model constraints across different branches of a constrained fork, we introduce two different kinds of interaction types, called *producers* and *consumers*, respectively. Each occurrence of a producer interaction type must correspond to the occurrence of a new interaction; in contrast, consumer interaction types correspond to the same interaction specified by a certain producer interaction type. The purpose of consumer interaction types is to impose constraints on interaction traces, *without introducing new events*. A consumer is an interaction type, whereas a producer is an interaction type \(\alpha\) equipped with a natural superscript \(n\) specifying the exact number of consumer interactions which are expected to coincide with it.

**Constrained global types**. A constrained global type \(\tau\) represents a set of possibly infinite traces of interactions, and is a possibly cyclic term defined on top of the following type constructors:

- \(\lambda\) (empty trace), representing the singleton set \(\{\epsilon\}\) containing the empty trace \(\epsilon\).
- \(\alpha^n : \tau\) (\(\text{seq-prod}\)), representing the set of all traces whose first element is an interaction \(a\) matching type \(\alpha (a \in \alpha)\), and the remaining part is a trace in the set represented by \(\tau\). The superscript\(^3\) \(n\) specifies the number \(n\) of corresponding consumers that coincide with the same interaction type \(\alpha\); hence, \(n\) is the least required number of times \(a \in \alpha\) has to be “consumed” to allow a transition labeled by \(a\).
- \(\alpha : \tau\) (\(\text{seq-cons}\)), representing a consumer of interaction \(a\) matching type \(\alpha (a \in \alpha)\).
- \(\tau_1 + \tau_2\) (\(\text{choice}\)), representing the union of the traces of \(\tau_1\) and \(\tau_2\).

\(^2\) “Interactions” were named “sending actions” in our previous work. We changed terminology to be consistent with the one used in the choreography community.

\(^3\) In the examples throughout the paper we use the concrete syntax of Prolog where producer interaction types are represented by pairs \((\alpha,n)\).
– \( \tau_1 | \tau_2 \) (fork), representing the set obtained by shuffling the traces in \( \tau_1 \) with the traces in \( \tau_2 \).

– \( \tau_1 \cdot \tau_2 \) (concat), representing the set of traces obtained by concatenating the traces of \( \tau_1 \) with those of \( \tau_2 \).

Constrained global types are regular terms, that is, can be cyclic (recursive), and they can be represented by a finite set of syntactic equations. We limited our investigation to types that have good computational properties, namely contractiveness and determinism.

Since constrained global types are interpreted coinductively, it is possible to specify protocols that are not allowed to terminate like for example the PingPong protocol defined by the equation

\[
\text{PingPong} = (\text{ping},0):(\text{pong},0):\text{PingPong}
\]

where \( \text{PingPong} \) is a logical variable which is unified with a recursive (or cyclic, or infinite) Prolog term consisting of the producer interaction type \( \text{ping} \), followed by the producer interaction type \( \text{pong} \) (both requiring 0 consumers), followed by the term itself. The only valid interaction trace respecting this constrained global type is the infinite sequence \( \text{ping pong ping pong ping pong} \ldots \). The valid traces for the type

\[
\text{PingPong} = ((\text{ping},0):(\text{pong},0):\text{PingPong} + \text{lambda})
\]

instead, are \( \{\epsilon, \text{ping pong, ping pong ping pong,} \ldots \} \), namely all the traces consisting of an arbitrary number (even none or infinite) of \( \text{ping pong} \).

Let us consider the following simple example where there are two robots (right and left), two monitors (right and left) associated with each robot, and a plan monitor which supervises them (Figure 1). The goal of the MAS is to help mothers in speeding up dressing their kids by putting their shoes on: robots must put a sock and a shoe on the right (resp. left) foot of the kid they help. As robots are autonomous, they could perform the two actions in the wrong order, making the life of the mothers even more crazy... Monitors are there to ensure that wrong actions are immediately rolled back. Robots communicate their actions to their corresponding monitors, which, in turn, notify the plan monitor when the robots accomplish their goal. Each robot can start by putting the sock, which is the
correct action to do, or by putting the shoe, which requires a recovery by the
(right or left, resp.) robot monitor.

As we will see, the left and right monitors play two different roles: they interact with robots to detect wrong actions and recover them, and they also verify part of the protocol, notifying the user of protocol violations. In this MAS, monitors are part of the protocol itself. In the MASs described in our previous papers, monitors performed a runtime verification of all the other agents but themselves, and their sole goal was to detect and signal violations. Extending monitors with other capabilities (or, taking another perspective, extending “normal” agents with the capability to monitor part of the protocol) does not represent an extension of the language or framework. The possibility of having agents that can monitor, can be monitored, and can perform whatever other action, was already there, but we did not exploit it before.

The interactions involved in the protocol and their types are as follows:

\[
\begin{align*}
\text{msg(right_robot, right_monitor, tell, put_sock)} & \in \text{put_right_sock} \\
\text{msg(right_robot, right_monitor, tell, put_shoe)} & \in \text{put_right_shoe} \\
\text{msg(right_robot, right_monitor, tell, removed_shoe)} & \in \text{rem_right_shoe} \\
\text{msg(right_monitor, right_robot, tell, obl_remove_shoe)} & \in \text{obl_rem_right_shoe} \\
\text{msg(left_robot, left_monitor, tell, put_sock)} & \in \text{put_left_sock} \\
\text{msg(left_robot, left_monitor, tell, put_shoe)} & \in \text{put_left_shoe} \\
\text{msg(left_robot, left_monitor, tell, removed_shoe)} & \in \text{rem_left_shoe} \\
\text{msg(left_monitor, left_robot, tell, obl_remove_shoe)} & \in \text{obl_rem_left_shoe} \\
\text{msg(left_monitor, plan_monitor, tell, ok)} & \in \text{ok_left}
\end{align*}
\]

The protocol can be specified by the following types, where SOCKS corresponds to the whole protocol.

\[
\begin{align*}
\text{RIGHT} & = ((\text{put_right_sock},0):(\text{put_right_shoe},0):(\text{ok_right},0):\lambda) + \\
& (\text{put_left_sock},0):(\text{put_left_shoe},0):(\text{ok_left},0):\lambda) + \\
\text{LEFT} & = ((\text{put_right_sock},0):(\text{put_right_shoe},0):(\text{ok_right},0):\lambda) + \\
& (\text{put_left_sock},0):(\text{put_left_shoe},0):(\text{ok_left},0):\lambda)
\end{align*}
\]

SOCKS = (RIGHT | LEFT)

The type SOCKS specifies the shuffle (symbol “|”) of two sets of traces of interactions, corresponding to RIGHT and LEFT, respectively. The shuffle expresses the fact that interactions in RIGHT are independent (no causality) from interactions in LEFT, and hence traces can be mixed in any order.

Types RIGHT and LEFT are defined recursively, that is, they correspond to cyclic terms. RIGHT consists of a choice (symbol “+”) between the finite trace (the constructor for trace is “”) of interaction types (put_right_sock,0), (put_right_shoe,0), (ok_right,0) corresponding to the correct actions of the right robot, and the trace of interaction types (put_right_shoe,0), (obl_rem_right_shoe,0), (rem_right_shoe,0) corresponding to the wrong initial action of the robot, followed by an attempt to perform the RIGHT branch again. Basically, either the right robot tells the right monitor that it put the sock on first, and then it can go on by putting the shoe, or it tells that it started its execution by
putting the shoe on. In this case, the right monitor forces the robot to remove the shoe, the robot acknowledges that it removed the shoe, and then starts again. The LEFT branch is the same as the RIGHT one, but involves the left robot and the left node monitor.

An example where sets of traces could be expressed with a fork, but are not completely independent, is given by the Alternating Bit Protocol ABP. We consider the instance of ABP where six different sending actions may occur (Figure 2): Bob sends \textit{msg1} to Alice (interaction type \textit{m1}), Alice sends \textit{ack1} to Bob (sending action type \textit{a1}), Bob sends \textit{msg2} to Carol (interaction type \textit{m2}), Carol sends \textit{ack2} to Bob (sending action type \textit{a2}), Bob sends \textit{msg3} to Dave (interaction type \textit{m3}), Dave sends \textit{ack3} to Bob (interaction type \textit{a3}) The ABP is an infinite iteration, where the following constraints have to be satisfied for all occurrences of the sending actions:

- The \textit{n}-th occurrence of an interaction of type \textit{m1} must precede the \textit{n}-th occurrence of an interaction of type \textit{m2} which in turn must precede the \textit{n}-th occurrence of an interaction of type \textit{m3}.
- For \( k \in \{1, 2, 3\} \), the \textit{n}-th occurrence of \textit{msg}k must precede the \textit{n}-th occurrence of the acknowledge \textit{ack}k, which, in turn, must precede the \((n + 1)\)-th occurrence of \textit{msg}k.

The ABP cannot be specified with forks of independent interactions, hence a possible solution requires to take all the combinations of interactions into account in an explicit way. However with this solution the size of the type grows exponentially with the number of the different interaction types involved in the protocol.

With producer and consumer interaction types it is possible to express the shuffle of non independent interactions which have to verify certain constraints. In this way the ABP can be specified in a very compact and readable way. The whole protocol is specified by the following constrained global type \textbf{ABP3}:

\[
\text{M1M2M3}=m1:m2:m3:=M1M2M3, \quad M1A1=(m1,1):(a1,0):M1A1, \\
M2A2=(m2,1):(a2,0):M2A2, \quad M3A3=(m3,1):(a3,0):M3A3, \\
\text{ABP3}=((M1M2M3|M1A1)|(M2A2|M3A3))
\]

Fork is associative and the way we put brackets in \textbf{ABP3} does not matter.
4 Projection Algorithm

In the “socks and shoes” example the monitors, besides checking that the robots accomplish their goal, verify also the compliance of the system to the specification of the protocol, given by the type \(\text{SOCKS} \). If we assume that the right robot and the right monitor reside on the same node, then it is reasonable that the right monitor verifies only the interactions which are local to its node; to do that, we must project the type \(\text{SOCKS} \) onto the agents of the node, that is, the right robot and the right monitor.

What we would like to obtain is the type

\[
\text{RIGHT}_P = ((\text{put\_right\_sock},0):(\text{put\_right\_shoe},0):(\text{ok\_right},0):\lambda) + \\
((\text{put\_right\_shoe},0):(\text{obl\_rem\_right\_shoe},0):(\text{rem\_right\_shoe},0):\text{RIGHT}_P),
\]

\(\text{SOCKS}_P = (\text{RIGHT}_P|\lambda)\)

which only contains interactions where the right robot and the right monitor are involved, either as sender or as receiver.

We can project any protocol onto any set of agents (although it is not necessarily meaningful or useful). For example, projecting the \(\text{ABP3} \) on Dave should result into

\[
\text{ABP3}_P\_\text{compact} = (m3,0):(a3,0):\text{ABP3}_P\_\text{compact}
\]

which just states that Dave must ensure to respect the order between messages and acknowledges that involve it (Dave cannot be aware of the order among messages coming from other agents). That projected type can be represented in an equivalent way, even if less compact, as

\[
\begin{align*}
\text{M1M2M3}_P &= m3:\text{M1M2M3}_P, \\
\text{M3A3}_P &= (m3,1):(a3,0):\text{M3A3}_P, \\
\text{ABP3}_P &= ((\text{M1M2M3}_P|\lambda)|\lambda)|\lambda) \\
&= (\text{M1M2M3}_P|\lambda)|\lambda)
\end{align*}
\]

Projecting the \(\text{ABP3} \) on Bob, instead, should result into the \(\text{ABP3} \) itself as Bob is involved in all communications and hence no interaction will be removed from the projection.

In order to allow agents to verify only a sub-protocol of the global interaction protocol, we designed a projection algorithm that takes a constrained global type and a set of agents \(\text{Ags} \) as input, and returns a constrained global type which contains only interactions involving agents in \(\text{Ags} \). The intuition besides the algorithm is that interactions that do not involve agents in \(\text{Ags} \) are removed from the projected constrained global type. Given the finite set \(\text{AGS} \) of all the agents that could play a role in the MAS and an interaction type \(\alpha \), \(\text{senders}(\alpha) \) is the set of all the agents in \(\text{AGS} \) that could play the role of sender in actual interactions having type \(\alpha \) and \(\text{receivers}(\alpha) \) is the set of all the agents in \(\text{AGS} \) that could play the role of receiver in interactions of type \(\alpha \). The \(\text{involves} \) predicate holds on one interaction type \(\alpha \) and one set of agents \(\text{Ags} \), \(\text{involves}(\alpha,\text{Ags}) \), iff \(\text{senders}(\alpha) \subseteq \text{Ags} \lor \text{receivers}(\alpha) \subseteq \text{Ags} \).
Projection can be described as a function \( \Pi : \mathcal{C}T \times \mathcal{P}(\text{AGS}) \rightarrow \mathcal{C}T \) where \( \mathcal{C}T \) is the set of constrained global types. \( \Pi \) is driven by the syntax of the type to project; as a first attempt, the function could be coinductively defined as follows:

(i) \( \Pi(\lambda, \text{AGS}) = \lambda \)

(ii) \( \Pi(a : \tau, \text{AGS}) = a : \Pi(\tau, \text{AGS}) \) if \( \text{involves}(a, \text{AGS}) \)

(iii) \( \Pi(\alpha : \tau, \text{AGS}) = \Pi(\tau, \text{AGS}) \) if \( \neg\text{involves}(a, \text{AGS}) \)

(iv) \( \Pi(\tau' \text{ op } \tau'', \text{AGS}) = \Pi(\tau', \text{AGS}) \text{ op } \Pi(\tau'', \text{AGS}) \), where \( \text{op} \in \{+, |, \cdot\} \).

We have to consider the greatest fixed point (coinductive interpretation) of the recursive definition above, since the least fixed point (inductive interpretation) would only include non cyclic types (that is, non recursive types).

Let us consider a simple non recursive term \( T \) defined by \( T = a : b : \lambda \).

We want to project \( T \) on \( \text{AGS} \). Suppose for that \( \text{involves}(a, \text{AGS}) \) holds, whereas \( \text{involves}(b, \text{AGS}) \) does not, meaning that interaction type \( a \) must be kept in the projection and \( b \) must be removed. From (ii) we get \( \Pi(a : b : \lambda, \text{AGS}) = a : \Pi(b : T, \text{AGS}) = a : \Pi(T) = a : \Pi(T) \) (\( T \) is discarded from the projection), and finally, from (i) we know that \( \Pi(\lambda) = \lambda \), therefore \( \Pi(T, \text{AGS}) = a : \lambda \).

Let us now consider the recursive type \( T \) s.t. \( T = a : T' \) and \( T' = b : T \).

Again, the projection is driven by the syntax of \( T \); from the definition above we have \( \Pi(a : T', \text{AGS}) = a : \Pi(T', \text{AGS}) = a : \Pi(b : T, \text{AGS}) = a : \Pi(T) = a : \Pi(a : T', \text{AGS}) \); while in the previous case we can conclude by applying the base case corresponding to the \( \lambda \) type, in this case we do not have any basis, but we can conclude by coinduction that \( \Pi(a : T', \text{AGS}) \) has to return the unique recursive type \( T'' \) s.t. \( T'' = a : T'' \) (see lhs picture in Figure 3).

The definition above however needs to be refined because it does not always specify a unique result for \( \Pi \); to see that, let us consider the recursive type \( T \) s.t. \( T = a : T' \) and \( T' = b : T' \). Now from the definitions above we get \( \Pi(a : T', \text{AGS}) = a : \Pi(T', \text{AGS}) = a : \Pi(b : T', \text{AGS}) = \Pi(T', \text{AGS}) \); since \( \Pi(T', \text{AGS}) = \Pi(T', \text{AGS}) \) is an identity, \( \Pi \) is allowed to return any type when applied to \( T' \), while the expected correct type should be \( \lambda \), so that \( \Pi(a : T', \text{AGS}) = a : \lambda \) (see rhs picture in Figure 3).

Finally, let us consider the recursive type \( T \) s.t. \( T = (a : T) + (b : T) \); by (iv) \( \Pi(T, \text{AGS}) = \Pi(a : T, \text{AGS}) + \Pi(b : T, \text{AGS}) \), by (ii) \( \Pi(a : T, \text{AGS}) = a : \Pi(T, \text{AGS}) \), and by (iii) \( \Pi(b : T, \text{AGS}) = \Pi(T, \text{AGS}) \); therefore by coinduction the returned type is \( T'' \) s.t. \( T'' = (a : T') + T' \); although in this case there exists a unique type that can returned by \( \Pi \), such a type is not \text{contractive}. A type is contractive if all possible cycles in it contain an occurrence of the sequence...
constructor “;” Figure 4 shows that type $T'$ s.t. $T' = (a : T') + T'$ is not contractive, since the rhs cycle contains only the “+” operator. The notion of contractive type is crucial for implementing efficient runtime verification.

To ensure that the projection function always returns a contractive type and that the correct coinductive definition is implemented, we need to keep track of all types visited along a path; each type is associated with its depth, and with a fresh variable which will be unified with the corresponding computed projection. During the visit the depth $DeepestSeq$ of the deepest visited sequence operator is kept. If a type $\tau$ has been already visited, then a cycle is detected: if its depth is less than $DeepestSeq$ then the cycle contains an occurrence of the sequence constructor, therefore the projected type associated with $\tau$ is contractive and, hence, is returned; otherwise, the projection would not be contractive, therefore $\lambda$ is returned.

Let us consider again the type $T = (a : T) + (b : T)$; when computing its projection, the depth of $T$ is 0, and initially $DeepestSeq$ contains the value -1. When visiting the lhs path starting from the “+” operator, the type $a : T$ is visited at depth 1, and $DeepestSeq$ is set to 1, since the root of $a : T$ is the sequence constructor. Then $T$ is revisited, and since its depth 0 is less than $DeepestSeq$, the projection of the lhs is $T' = a : T'$. When visiting the rhs path starting from the “+” operator, $DeepestSeq$ contains again the value -1, and the type $b : T$ is visited at depth 1, but because $involves(b, Ags)$ does not hold, $b$ is discarded with the corresponding sequence constructor, hence $DeepestSeq$ is not updated. Then $T$ is revisited, and since its depth 0 is not less than $DeepestSeq$, the projection of the rhs is $\lambda$.

5 Implementation

The projection algorithm has been implemented in SWI Prolog, http://www.swi-prolog.org/, which manages infinite (cyclic, recursive) terms in an efficient way. Since we need to record the association between any type and its projection in order to correctly detect and manage cycles, we exploited the SWI Prolog library assoc for association lists, http://www.swi-prolog.org/pldoc/man?section=assoc. Elements of an association list have 2 components: a (unique) key and a value. Keys should be ground, values need not be. An association list can be used to
fetch elements via their keys and to enumerate its elements in ascending order of their keys. The library(assoc) module uses AVL trees to implement association lists which makes inserting, changing and fetching a single element an \( O(\log(N)) \) operation. The three predicates of the library assoc that we use for our implementation are

- \texttt{empty_assoc(Assoc)}: Assoc is unified with an empty association list.
- \texttt{get_assoc(+Key, +Assoc, ?Value)}: Value is the value associated with Key in the association list Assoc.
- \texttt{put_assoc(+Key, +Assoc, +Value, ?NewAssoc)}: NewAssoc is an association list identical to Assoc except that Key is associated with Value. This can be used to insert and change associations.

The projection is implemented by a predicate \texttt{project(T, ProjAgs, ProjT)} where \( T \) is the constrained global type to be projected, \( ProjT \) is the result, and \( ProjAgs \) is the set of agents onto which the projection is performed. The algorithm exploits the predicate \texttt{involves(IntType, ProjAgs)} succeeding if \( IntType \) may involve one agent, as a sender or a receiver, in \( ProjAgs \).

Currently \texttt{involves} looks for actual interactions \( ActInt \) whose type is \( IntType \) and assumes that senders and receivers in \( ActInt \) are ground terms, but it could be extended to take agents’ roles into account or in other more complex ways. It uses the “or” Prolog operator ; and the \texttt{member} predicate offered by the library \texttt{lists}. It exploits the predicate \texttt{has_type(ActInt, IntType)} implementing the definition of the type \( IntType \) of an actual interaction \( ActInt \).

\begin{verbatim}
    involves(IntType, List) :-
      has_type(msg(Sender, Receiver, _, _), IntType),
      (member(Sender, List);member(Receiver, List)).
\end{verbatim}

For the implementation of \texttt{project/3} we use an auxiliary predicate \texttt{project} with six arguments, which are the same as those of the main predicate plus

- an initially empty association \( A \) to keep track of cycles;
- the current depth of the constrained global type under projection, initially set to 0;
- the depth of the deepest sequence operator belonging to the projected type, initially set to -1.

\begin{verbatim}
    project(T, ProjAgs, ProjT) :-
      empty_assoc(A), project(A, 0, -1, T, ProjAgs, ProjT).
\end{verbatim}

The predicate is defined by cases.

1. \texttt{lambda} is projected into \texttt{lambda}.

\begin{verbatim}
    project(_, _, _, _ProjAgs, lambda, lambda):- !.
\end{verbatim}

2. If \( Type \) has been already met while projecting the global type \( \texttt{get_assoc(Type, Assoc, (AssocProjType,LoopDepth)) succeeds} \), then its projection \( ProjT \) is \( AssocProjType \) if \( LoopDepth \leq \text{DeepestSeq} \) and is \texttt{lambda} otherwise.
The “if-then-else” construct is implemented in Prolog as Condition -> ThenBranch ; ElseBranch.

project(Assoc, Depth, DeepestSeq, Type, ProjAgs, ProjT) :-
    get_assoc(Type, Assoc, (AssocProjType, LoopDepth)), !,
    (LoopDepth <= DeepestSeq -> ProjT=AssocProjType; ProjT=lambda).

3. T = (IntType:T1). IntType is a consumer as it has no integer number associated with it. ProjT is recorded in the association A along with the current depth Depth (put_assoc((IntType:T1), Assoc, (ProjT, Depth), NewAssoc)). If IntType involves ProjAgs, ProjT=(IntType:ProjT1) where ProjT1 is obtained by projecting T1 onto ProjAgs, with association NewAssoc, depth of the type under projection increased by one, and depth of the deepest sequence operator equal to Depth. If IntType does not involve ProjAgs, then the projection on T is the same of T1 with association NewAssoc, depth of the type under projection equal to Depth, and depth of the deepest sequence operator equal to DeepestSeq.

project(Assoc, Depth, DeepestSeq, (IntType:T1), ProjAgs, ProjT) :- !,
    put_assoc((IntType:T1), Assoc, (ProjT, Depth), NewAssoc),
    (involves(AMsg, ProjAgs) ->
      IncDepth is Depth+1,
      project(NewAssoc, IncDepth, Depth, T1, ProjAgs, ProjT1),
      ProjT=(IntType:ProjT1);
      project(NewAssoc, IncDepth, DeepestSeq, T1, ProjAgs, ProjT)).

4. T = ((IntType,N):T1). (IntType,N) is a producer as it has an integer number N associated with it. The projection is identical to the previous case, apart from the fact that ProjT=((IntType,N):ProjT1) in the first branch of the condition in the clause’s body.

5. T = T1 op T2, where op ∈ {+, |, *}: the association between T1 op T2 and the projected type ProjT is recorded in the association Assoc along with the current depth Depth. T1 and T2 are projected into ProjT1 and ProjT2 respectively, with association equal to NewAssoc, depth of the type under projection increased by one and depth of the deepest sequence operator equal to DeepestSeq. The result of the projection is ProjT=(ProjT1 op ProjT2). For example, if op is +, the Prolog clause is:

project(Assoc, Depth, DeepestSeq, (T1+T2), ProjAgs, ProjT) :- !,
    put_assoc((T1+T2), Assoc, (ProjT, Depth), NewAssoc),
    IncDepth is Depth+1,
    project(NewAssoc, IncDepth, DeepestSeq, T1, ProjAgs, ProjT1),
    project(NewAssoc, IncDepth, DeepestSeq, T2, ProjAgs, ProjT2),
    ProjT=(ProjT1+ProjT2).

Types SOCKS_P and AP3_P shown at the beginning of Section 4 have been obtained by applying the projection algorithm to types SOCKS and ABP3 respectively. The reason why they are not as compact as possible, which is mainly evident in AP3_P, is that the projection algorithm does not implement a further
normalization step and hence some types which have been projected into lambda and might be removed, are instead kept.

The result of the projection may be a type equivalent to lambda. For example, if we project ABP to the set \{eric\}, no interaction involves it and the result is \((\text{lambda}|\text{lambda})|\text{lambda}|\text{lambda}\). On the other hand, we have already observed that the projection may be the same as the projected type. This happens for example if we project ABP to the set \{bob\}, which interacts with all the agents in the MAS.

6 Projection at Work

In SWI Prolog we have implemented a mechanism for generating all the different traces (sequences of interactions) with length \(N\), where \(N\) can be set by the user, that respect a given protocol. This mechanism is necessary during the design of the protocol and allows the protocol designer to make an empirical assessment of the conversations that will be recognized as valid ones during the runtime verification. We used this mechanism for validating both the complete protocols and the projected ones; also with projected types, the generated traces are correct w.r.t. the protocol specification.

For example, Table 1 (top left) shows one of the 16380 different traces with length 12 of the SOCKS protocol and Table 1 (top right) shows one of the 2 different traces with length 12 of the SOCKS protocol projected onto \{right_robot, right_monitor\} (for sake of presentation, we abbreviate right_robot in right_r, right_monitor in right_m, left_robot in left_r, left_monitor in left_m, msg in m, and we drop the tell performative from interactions). Both traces correspond to an execution where the protocol reached a final state and no other interactions could be accepted after the last one. In the output produced by the SWI Prolog algorithm, this information is given by means of an asterisk after the last interaction. Traces that are prefixes of longer (maybe infinite) ones have no asterisk at their end.

Table 1 (bottom left) shows an excerpt of one of the 30713 different traces with length 16 of the ABP3 protocol and Table 1 (bottom right) shows the first 12 interactions of the only trace with length 16 of the ABP3 protocol projected onto \{dave\}. Since the ABP3 is an infinite protocol, both traces are prefixes of infinite ones.

By generating traces of different length and inspecting some of them, the protocol designer can get a clear picture of whether the protocol he/she designed behaves in the expected way. Of course this manual inspection gives no guarantees of correctness, but in our experience it was enough to early detect flaws in the protocol specification.

We have implemented the “socks and shoes” MAS in Jason. The MAS is represented in Figure 1. We projected the SOCKS constrained global type shown in Section 3 onto the three sets of agents \{left_monitor\}, \{right_monitor\} and \{plan_monitor\}. The three resulting constrained global types are used by agents left_monitor, right_monitor and plan_monitor respectively.
SOCKS protocol

| m(right_r, right_m, put_sock) | m(right_r, right_m, put_shoe) |
| m(left_r, left_m, put_shoe)   | m(right_m, right_r, oblige_remove_shoe) |
| m(left_m, left_r, oblige_remove_shoe) | m(right_r, right_m, removed_shoe) |
| m(left_robot, left_m, removed_shoe) | m(right_r, right_m, put_shoe) |
| m(right_r, right_m, put_shoe) | m(right_m, right_r, oblige_remove_shoe) |
| m(right_m, plan_monitor, ok) | m(right_r, right_m, removed_shoe) |
| m(left_robot, left_m, put_shoe) | m(right_r, right_m, put_shoe) |
| m(left_m, left_r, oblige_remove_shoe) | m(right_m, right_r, oblige_remove_shoe) |
| m(left_r, left_m, removed_shoe) | m(right_r, right_m, removed_shoe) |
| m(left_r, left_m, put_sock) | m(right_r, right_m, put_sock) |
| m(left_r, left_m, put_shoe) | m(right_r, right_m, put_sock) |
| m(left_m, plan_monitor, ok) | m(right_m, plan_monitor, ok) |

ABP3 protocol

| msg(bob, alice, tell, m1) | msg(bob, dave, tell, m3) |
| msg(bob, carol, tell, m2) | msg(dave, bob, tell, a3) |
| msg(carol, bob, tell, a2) | msg(dave, bob, tell, a3) |
| msg(alice, bob, tell, a1) | msg(dave, bob, tell, a3) |
| msg(bob, dave, tell, m3) | msg(bob, dave, tell, m3) |
| msg(dave, bob, tell, a3) | msg(bob, dave, tell, m3) |
| msg(bob, alice, tell, m1) | msg(dave, bob, tell, m3) |
| msg(bob, carol, tell, m2) | msg(dave, bob, tell, m3) |
| msg(alice, bob, tell, a1) | msg(dave, bob, tell, a3) |
| msg(bob, dave, tell, m3) | msg(bob, dave, tell, m3) |
| msg(bob, alice, tell, m1) | msg(bob, dave, tell, m3) |
| msg(carol, bob, tell, a2) | msg(dave, bob, tell, a3) |

Table 1. Examples of traces compliant with complete and projected protocols.

Each of these agents monitors all the messages that it either receives or sends, using the “message sniffing” mechanism described in [1].

We run different experiments by changing the actual messages sent by the agents in the MAS, in order to obtain both correct and wrong executions. All our experiments gave the expected outcome. As an example, Figure 5 shows an interaction where left_robot sends a put_boot message instead of put_shoe, which is correctly identified by the left_monitor as a violation. The conversation between the other agents goes on.

7 Conclusions and Future Work

In this paper we have defined an algorithm for projecting a constrained global type onto a set of agents $A_G$, to allow distributed dynamic verification of the compliance of a MAS to a protocol. This is important in communication-intensive and highly-distributed large MASs, where a centralized approach with a unique monitoring agent would be unfeasible.
Besides describing the algorithm and its SWI Prolog implementation, we have shown some preliminary experiments in Jason with the running example “socks and shoes” where two local monitors with projected types are sufficient for verifying the whole system.

For what concerns future work, we are investigating on the possible ways to partition the set of agents for projecting types, to minimize the number of monitors, while ensuring safety of dynamic verification.

We are also planning to extend the projection algorithm in order to be able to properly deal with a more general form of type: attribute global types.

Finally, in the examples considered in this paper, types are projected statically (that is, before the system is started) because we have assumed that agents cannot move between nodes, but monitoring would be also possible in the presence of agent mobility. However, in this case the implementation of a self-monitoring MAS is more challenging, because monitor agents have to dynamically project the global type in reaction to any change involving the set of monitored agents. Tackling scenarios of this kind is the final goal of our research.
References


JTabWb: a Java framework for implementing terminating sequent and tableau calculi

Mauro Ferrari\textsuperscript{1}, Camillo Fiorentini\textsuperscript{2}, Guido Fiorino\textsuperscript{3}

\textsuperscript{1} DiSTA, Univ. degli Studi dell’Insubria, Via Mazzini, 5, 21100, Varese, Italy
\textsuperscript{2} DI, Univ. degli Studi di Milano, Via Comelico, 39, 20135 Milano, Italy
\textsuperscript{3} DISCO, Univ. degli Studi di Milano-Bicocca, Viale Sarca, 336, 20126, Milano, Italy

Abstract. JTabWb is a Java framework for developing provers based on terminating sequent or tableau calculi. It provides a generic engine which performs proof-search driven by a user-defined specification. The user is required to define the components of a prover by implementing suitable Java interfaces. The implemented provers can be used as standalone applications or embedded in other Java applications. The framework also supports proof-trace generation, \LaTeX rendering of proofs and counter-model generation.

1 Introduction

JTabWb is a Java framework for developing provers based on terminating sequent or tableau calculi. The project originated as a tool for experimenting and comparing on the same ground different calculi and proof-search strategies for intuitionistic propositional logic. Now it is an advanced general framework which can be used to implement different logics and calculi. It can be used either to implement provers as stand-alone Java applications or as APIs to be integrated in other Java applications. Differently from other frameworks like \cite{1, 4, 5}, in JTabWb the user can specify all the components of a prover including: formulas, proof-search nodes, rules and strategies. This allows one to easily implement provers for different logics and different calculi (sequent-style or tableau-style calculi). Its main limitation is that all the components are provided as Java classes, hence the user is expected to be experienced with the language. The object oriented nature of the target language and the compositionality of the framework supports the reuse of the components of a prover. This allows one to easily develop different variants of a prover, so to compare different implementations of data structures (formulas, sequents,...) and different strategies. The framework also provides support for generation of proof-traces (histories of proof-search), \LaTeX rendering of proofs and counter-model generation. JTabWb and some provers for intuitionistic propositional logic implemented in it are available at \url{http://www.dista.uninsubria.it/~ferram}. 

46
### 2 Basic notions and their implementation

JTabWb provides a generic engine that searches for a proof of a goal driven by a user-defined prover. In particular the engine searches for a proof of the goal visiting the proof-search space in a depth-first fashion; at any step of the search, the engine asks to the strategy component of the prover the next rule to apply. The user defines the prover by implementing the interfaces modeling the logical components of the proof-search procedure in Fig. 1. For every component we indicate the corresponding interface and its main methods.

Formulas are the basic elements of the formal system at hand; one can define formulas of any kind, e.g., propositional, first-order or modal formulas, but also “formulas with a sign” or “labelled formulas”. The data structure storing formulas during proof-search is modeled by a node-set. E.g., in the case of a sequent calculus, node-sets represent sequents $\Gamma \Rightarrow \Delta$ where $\Gamma$ and $\Delta$ are finite sets or multisets of formulas. Formulas and node-sets only require the implementation of a method `format()`, which is invoked by the engine to print detailed information about the proof-search process.

JTabWb models four kinds of rules: **regular**, **clash-detection**, **branch-exists** and **meta-backtrack**. Regular and clash-detection rules directly correspond to
the rules of a calculus; \textit{branch-exists} and \textit{meta-backtrack} rules are meta-rules to encode the proof-search strategy. All the rules have \texttt{AbstractRule} as a common supertype.

Regular rules directly correspond to the usual formalization of rules in tableau calculi. A regular rule has the form displayed in Fig. 1: \( \mathcal{R} \) is the name of the rule, \( \sigma \) is the premise of \( \mathcal{R} \) and \( \sigma_1, \ldots, \sigma_n \) (\( n \geq 1 \)) its conclusions. We use a double line to represent rules of the framework so to distinguish them from the rules of the sequent calculus we use in the examples.

A rule \( \mathcal{R}_s \) of a sequent calculus can be mapped to a regular rule \( \mathcal{R} \) writing \( \mathcal{R}_s \) bottom-up (the conclusion of \( \mathcal{R}_s \) becomes the premise of \( \mathcal{R} \)). As an example, let us consider the rule for left disjunction of G3i [6]

\[
\frac{[A, \Gamma \Rightarrow H]}{[A \lor B, \Gamma \Rightarrow H], \lor L}
\]

This rule is represented in our framework by the regular rule:

\[
\frac{[A \lor B, \Gamma \Rightarrow H]}{[A, \Gamma \Rightarrow H], \lor L | [B, \Gamma \Rightarrow H]}
\]

The main formula of a regular rule is the formula put in evidence in the premise of the rule (e.g., \( A \lor B \) in the above example). An instance of a regular rule is an object implementing the \texttt{RegularRule} interface. Conclusions of a regular rule are returned as an enumeration of objects of type \texttt{AbstractNodeSet}. To represent an enumeration of objects we use the Java \texttt{Iterator} interface: it defines the method \texttt{next()} to get the next element in the enumeration and the method \texttt{hasNext()} to check whether the enumeration contains more elements or not.

Clash-detection rules model rules without conclusions corresponding to the end-points of a derivation (closure rules of tableau calculi and axiom rules of sequent calculi). We represent such a rule by a function \( \mathcal{CD} \) that, given a node-set \( \sigma \), returns \texttt{SUCCESS} if \( \sigma \) is an end-point of the derivation, \texttt{FAILURE} otherwise. As an example, the axiom rules of G3i

\[
\frac{[A, \Gamma \Rightarrow A]}{\text{Ax}} \quad \frac{[\bot, \Gamma \Rightarrow H]}{L \bot}
\]

correspond to the following clash-detection rule:

\[
\mathcal{CD}([\Gamma \Rightarrow A]) = \begin{cases} 
\text{SUCCESS} & \text{if } \bot \in \Gamma \text{ or } A \in \Gamma \\
\text{FAILURE} & \text{otherwise}
\end{cases}
\]

An instance of a clash-detection rule is an object that implements the interface \texttt{ClashDetectionRule} providing the \texttt{checkStatus()} method encoding the corresponding \( \mathcal{CD} \) function.

The distinction between invertible and non-invertible\footnote{We adopt the formalization of invertible rule of [6].} rules has a crucial role in the definition of a proof-search procedure, since non-invertible rules introduce backtrack points in proof-search. E.g., the rules of G3i for right disjunction
public class Rule_Right_Or implements _BranchExistsRule {
    private Sequent premise;
    private Formula or_formula;
    public Rule_Right_Or (Sequent sequent, Formula or_formula) {
        this.premise = sequent; this.or_formula = or_formula;
    }
    ...
    public Iterator<_AbstractNodeSet> iterator() {
        LinkedList<Sequent> list = new LinkedList<Sequent>();
        Formula[] disjuncts = or_formula.getSubformulas();
        for (int i=0; i < disjuncts.length; i++) {
            Sequent conclusion = premise.clone();
            conclusion.removeRight(or_formula);
            conclusion.addRight(disjuncts[i]);
            list.add(conclusion);
        }
        return list.iterator();
    }
}

Fig. 2. Implementation of the rule $\lor R_i$ of $G3_i$

$$
\frac{[\Gamma \Rightarrow A_1]}{[\Gamma \Rightarrow A_1 \lor A_2]} \lor R_i \; i \in \{1, 2\}
$$

are not invertible. Indeed, it could be the case that $[\Gamma \Rightarrow A_2]$ is provable while $[\Gamma \Rightarrow A_1]$ is not. Hence, searching for a proof of $[\Gamma \Rightarrow A_1 \lor A_2]$, we have to try both the rules; if the construction of a proof for $[\Gamma \Rightarrow A_1]$ fails, we have to reconsider the premise (backtrack) and try the other way. The two rules above can be formalized in our framework by means of a branch-exists rule. A branch-exists rule $\mathcal{R}$ with premise $\sigma$ and conclusions $\sigma_1, \ldots, \sigma_n$ ($n \geq 1$) means that $\sigma$ is provable iff at least one of the $\sigma_i$ is provable. An instance of a branch-exists rule is an object implementing the _BranchExistsRule interface. The iterator method returns the conclusions of the rule as an enumeration of objects of type _AbstractNodeSet. As an example, in Fig. 2 we describe an implementation of the rules $\lor R_i$ of $G3_i$.

A calculus $\mathcal{C}$ is a finite set of regular rules, clash-detection rules and branch-exists rules. A C-tree $\pi$ is a tree of node-sets such that, if $\sigma$ is a node of $\pi$ with $\sigma_1, \ldots, \sigma_n$ as children, then either there exists a regular-rule of $\mathcal{C}$ having $\sigma$ as premise and $\sigma_1, \ldots, \sigma_n$ as conclusions, or $n = 1$ and there exists a branch-exists rule of $\mathcal{C}$ having $\sigma$ as premise and $\sigma_1$ as one of its conclusions. A C-proof is a C-tree $\pi$ such that, for every leaf $\sigma$ of $\pi$, there exists a clash-detection rule $\mathcal{CD}$ of $\mathcal{C}$ such that $\mathcal{CD}(\sigma) = \text{SUCCESS}$.

To define proof-search strategies we need to encode another kind of backtracking arising from the application of non-invertible rules. Let us consider the non-invertible rule $\to L$ of $G3_i$...
If we are searching for a proof of a sequent $\sigma$ containing more than one implication in the left-hand side, we must try all the possible instances of the rule $\to L$ to assert the provability status of $\sigma$. To express this situation we use a meta-backtrack rule, which is a function $MB$ associating with a node-set $\sigma$ an enumeration of rule instances having $\sigma$ as premise (the non-invertible rules applicable to $\sigma$). We remark that a meta-backtrack rule is not a rule of the calculus but a meta-rule to encode the proof-search strategy in presence of non-invertible rules.

The strategy is a function that, taken the current goal of the proof-search (a node-set) and the current state, returns the next rule to apply in the proof-search. The method nextRule($\text{AbstractNodeSet}$ goal, $\text{IterationInfo}$ info) of the _Strategy interface is a callback method invoked by the engine when it needs to determine the next rule to apply in the proof-search. The engine invokes this method providing as arguments the current goal of the proof-search and a bunch of data describing the last move performed by the engine. E.g., the method getAppliedRule() of IterationInfo returns the rule applied by the engine in the last move; in many cases this is the only data needed to choose the next rule to apply to goal. For instance, this is an high-level description of the strategy for a terminating sequent calculus for intuitionistic propositional logic (as, e.g., the calculus LSJ described in [2]).

```
_AbstractRule nextRule(_AbstractNodeSet sequent, _IterationInfo info) {
    if (an invertible rule r is applicable to sequent)
        return r;
    _AbstractRule lastAppliedRule = info.getAppliedRule();
    if (lastAppliedRule is not a clash-detection rule)
        return the clash-detection rule for sequent;
    if (non invertible rules r1...rN are applicable to sequent)
        return the meta-backtrack rule collecting r1...rN
    return null
}
```

A prover is an object implementing the _Prover interface which provides the getStrategy() method and some other methods that are not essential to our discussion.

3 High-level description of the engine

We give in Fig. 3 an high-level description of the algorithm implemented by the engine to perform proof-search. An instance of the engine is built by specifying the prover and the goal; searchProof() searches for a proof of the goal driven by
the strategy specified by the prover. To simplify the presentation we assume that the data passed to the strategy only consist of the rule applied in the last step. The search space is visited in a depth-first fashion using a stack to store the information related to branch points and backtrack points. More precisely, the stack contains elements (rule, iterator), where rule is the rule that caused the push action, iterator is the associated enumeration. If rule is a regular rule, the element of the stack represents a branch point, if rule is a branch-exists or a meta-backtrack rule, the element represents a backtrack point.

The method searchProof() essentially consists of a loop; we call iteration of the engine an iteration of such a loop. At every iteration the state of the engine is characterized by the current goal of the proof-search (i.e., current_goal), the rule to apply in the current iteration (i.e., current_rule) and the rule selected for the next iteration (i.e., next_rule). If current_rule is a regular rule or a branch-exists rule, the engine replaces the goal with the first conclusion of the rule and determines the next rule to apply by invoking the strategy. If the applied rule has more than one conclusion, then an element e is pushed on the stack by the call push(current_rule, iterator): if current_rule is a regular rule, e is a branch point, otherwise e is a backtrack point.

If current_rule is a meta-backtrack rule, the associated enumeration iterator collects the rules to be applied to current_goal. The first rule in the enumeration (returned by the method next()) is applied and, if iterator contains one or more rules, the backtrack point (current_rule, iterator) is pushed on the stack.

If current_rule is a clash-detection rule, then the engine invokes checkStatus() to determine if the current goal is an end-point of the proof-search. If checkStatus() returns FAILURE, then the strategy selects the next rule to apply. If it returns SUCCESS, then restoreBranchPoint() searches the stack for a branch point. If such a point exists, it provides the new goal and the strategy selects the next rule to be applied; if the stack does not contain any branch point, the proof-search successfully terminates. The method restoreBranchPoint() searches the stack for a branch point, namely an element (rule, iterator), where rule is a regular rule. If such an element does not exist, it returns null; otherwise, it returns iterator.next(), representing the new goal to be proved. If iterator is empty, the branch point is removed.

If current_rule is null, it means that the strategy failed to select a rule for current_goal in the last iteration of the engine, hence the proof-search for current_goal has failed. In this case the engine searches the stack for a backtrack point invoking restoreBacktrackPoint(). If a backtrack point exists, the engine appropriately updates its state and starts a new iteration. Otherwise, it returns FAILURE to signal that a proof for the input goal does not exist. The method restoreBacktrackPoint() searches the stack for a backtrack point, that is an element (rule, iterator), where rule is a branch-exists rule or a meta-backtrack rule. If such an element does not exist, null is returned; otherwise, the pair (rule, iterator.next()) is returned. In the latter case, if rule is a branch-exists rule, then iterator.next() is a node-set (the next goal to be proved); otherwise, rule is a meta-backtrack rule and iterator.next() is the next rule to be applied.
// goal and prover are the arguments of the engine constructor
current_goal = goal
strategy = prover.getStrategy()
next_rule = strategy.nextRule(current_goal,null)
while true do
  current_rule = next_rule // the only data about the last iteration
  if current_rule is a regular rule then
    iterator = current_rule.iterator()
    current_goal = iterator.next()
    next_rule = strategy.nextRule(current_goal,current_rule)
    if iterator.hasNext() then push(current_rule,iterator)
  end
  else if current_rule is a branch-exists rule then
    iterator = current_rule.iterator()
    current_goal = iterator.next()
    next_rule = strategy.nextRule(current_goal,current_rule)
    if iterator.hasNext() then push(current_rule,iterator)
  end
  else if current_rule is a clash-detection rule then
    if current_rule.checkStatus() == SUCCESS then
      current_goal = restoreBranchPoint() // restored is a goal or
      null
      if current_goal == null then return SUCCESS
    else
      next_rule = strategy.nextRule(current_goal,null)
    end
    else
      next_rule = strategy.nextRule(current_goal,current_rule)
  end
  else if current_rule is a meta-backtrack rule then
    iterator = current_rule.iterator()
    next_rule = iterator.next()
    if iterator.hasNext() then push(current_rule,iterator)
  end
  else if current_rule is null then
    r = restoreBacktrackPoint() // r is a pair
    if r == null then return FAILURE
    if snd(r) is a rule then //rule is a meta-backtrack rule
      next_rule = snd(r) // snd(r) is the next rule to try
      current_goal = fst(r).premise() // restore the goal from the
      rule
    else //rule is a branch-exists rule
      current_goal = snd(r) // snd(r) is the next sub-goal to try
      next_rule = strategy.nextRule(current_goal,current_rule)
    end
  end
end

Fig. 3. engine.searchProof()
4 Implemented provers and other features

The engine can be executed in verbose mode to get a detailed description of the proof-search or in trace-mode to generate a trace of the proof-search. It is possible to generate the \LaTeX{} rendering of the C-trees visited during the proof-search (this only requires to provide the data rendering for node-sets and rule names). The trace of a proof-search can be used to generate the counter-model for unprovable goals. JTabWb also provides some useful support APIs (jtabwbx.* packages): two implementations of propositional formulas, a parser for propositional formulas and a command line launcher for a prover.

We have implemented several provers for intuitionistic propositional logic in the JTabWb framework. g3ibu is a prover based on the sequent calculi Gbu and Rbu [3]. lsj is a prover based on the sequent calculi LSJ and RJ [2]. Both these provers allow one to generate counter-models for unprovable sequents. Finally, jpintp provides the implementation of several well-known tableau and sequent calculi for intuitionistic propositional logic.

As for future work, we are developing a language to specify the components of a JTabWb prover and a library of formulas implementations and node-set implementations that can be used as building blocks for provers. Finally, we remark that the JTabWb can be used also to implement calculi for first-order logic and, in general, non terminating calculi. What is missing for fruitfully use these kind of calculi is a support which allows the user to directly control the engine execution. We are developing it as an interactive version of the engine.

References

Multi-Criteria Optimal Planning for Energy Policies in CLP

Marco Gavanelli\textsuperscript{1}, Michela Milano\textsuperscript{2}, Stefano Bragaglia\textsuperscript{3}, Federico Chesani\textsuperscript{2}, Elisa Marengo\textsuperscript{4}, and Paolo Cagnoli\textsuperscript{5}

\textsuperscript{1} EnDiF - Università di Ferrara, Italy
\textsuperscript{2} DISI - Università di Bologna, Italy
\textsuperscript{3} Department of Computer Science, University of Bristol, UK
\textsuperscript{4} Faculty of Computer Science - Free University of Bozen-Bolzano
\textsuperscript{5} ARPA Emilia-Romagna, Italy

Abstract. The number of issues, stakeholders needs and regulations that a policy must consider in the current world is so high that addressing them only with common-sense is unthinkable. Policy makers have to consider disparate issues, as diverse as economic development, environmental aspects, as well as the social acceptance of the policy. A single person cannot be expert in all these subjects. Thus, to obtain a well assessed policy in the current complex world one can adopt decision support systems featuring optimization components.

Leveraging on previous work on Strategic Environmental Assessment, we developed a fully-fledged system that is able to provide optimal plans with respect to a given objective, to perform multi-objective optimization and provide sets of Pareto optimal plans, and to visually compare them. Each plan is environmentally assessed and its environmental footprint is provided in terms of emissions, global warming effect, human toxicity, and acidification. The heart of the system is an application developed in a popular Constraint Logic Programming system on the Reals sort. It has been equipped with a web service module that can be queried through standard interfaces. An intuitive graphic user interface has been added to provide easy access to the web service and the CLP application.

Keywords: CLP applications, Strategic Environmental Assessment, Regional Energy Planning

1 Introduction

Policy making, in the current connected world, has to consider such a number of issues that a single person cannot possibly consider without introducing vast approximations. For example European regions should provide Regional Energy Plans to define strategic objectives and political actions for the energy sector. These policies must take into consideration

\footnote{This paper is an extended version of [1]}
the current energy balance in the region: how much energy is produced/consumed in the region (both electrical and thermal energy can be included) per year, how much is imported/exported;

- forecasts for the future, such as the foreseen energy request or the cost of energy for the following years;

- existing and new directives, one example being the EU 20-20-20 initiative that poses three challenging targets for 2020: 20% improvement of energy efficiency, 20% of the energy should come from renewable sources, and reduction of 20% of greenhouse gas emissions.

The policy contains strategic objectives on the energy share and energy efficiency, measures and activities to cope with the increased energy needs, new regulations, etc. In the case of regional planning, the plan is typically very high-level: it includes activities such as building new power plants for some total output power, the share of each fuel type (nuclear, fossil fuels, biomasses, photovoltaic, windmills, etc.) and the type of produced energy (electric or thermal); but it lacks information about, for example, the actual placement of the plants in the region, or the size of each of the plants. More detailed plans will be done at lower scale, like the province or municipality levels.

By EU directives, regional policies on the energy sector should also include an environmental assessment of the plan. Being the plan so high-level, usually the assessment is done only in a qualitative way.

In a previous work [2], we proposed and compared two alternative logic programming formulations for the strategic environmental assessment of regional plans; one was based on probabilistic logic programming, the other on Constraint Logic Programming (CLP) [3]. We also developed four fuzzy-logic formulations of the assessment problem [4]. All these programs consider a regional plan, given in input, and provide its environmental assessment. An evaluation of the results by an environmental expert suggested that the CLP version provided the most reliable results.

In a following work [5], the CLP program was extended to a first prototype of a regional planner, that generates plans together with their assessment. Although the software was used during the definition of the Regional Energy Plan 2011-2013 of the Emilia-Romagna region [6], the first version had several limitations. First, it had only a command-line interface, and could be used only by an experienced logic programmer (not to mention configuring it). Second, it was able to provide optimal solutions only for one objective function; a serious limitation for a system to be used in the multi-faceted world of policy-making. Third, it was not able to provide any quantitative information on the environmental assessment of the plans. Fourth, it did not consider all the possible actions that a regional plan can implement, but only those actions that amount to creating new infrastructures, plants, or activities, while it was unable to assess the effect of closing power plants or decommissioning obsolete infrastructures.

In this work, we show how the first prototype of the planner was extended to a fully-fledged application. The current version of the software supports

- plans that consider decommissioning obsolete power plants;
– computation of emissions of the power plants for various types of pollutants, in a quantitative way;
– quantitative assessment of the effect of the plan on human health, global warming, and acidification potential;
– multi-criteria optimization considering a variety of objective functions based on qualitative and quantitative information;
– computation of the Pareto front, for two or more objective functions;
– a web interface, that can be accessed both as a Graphical User Interface (GUI) and as a web service.

This work is one of the components of the EU ePolicy project\(^6\). The final application will use the optimal planner as the center of a large application, that will include an opinion mining component, to assess the acceptance of the policies from the public considering information coming from blogs and social networks; a social simulator component, that will simulate how the population will react to the policies adopted by the Region; a mechanism design component, that will include information from game theory to provide the best allocation schemes of regional subsidies to the stakeholders in the Region; and an integrated visualization component.

The rest of the paper is organized as follows. We first explain the planning and environmental assessment as they are currently done by experts in the Emilia-Romagna region of Italy, and recap the basic CLP program of the first prototype (Section 2). In Section 3, extend it for the new features. We show the design and features of the web service and GUI in Section 4. Finally, we show an experimental evaluation in Section 5, and we conclude in Section 6.

\(2\) Problem considered and CLP solution

The strategic environmental assessment, in the Emilia-Romagna region of Italy, is currently performed by considering two matrices, called coaxial matrices [7]. They are a development of the network method [8], and they contain qualitative relations.

The first matrix, \(M\), considers the activities that can be undertaken in a plan, and links them with the so-called environmental pressures. Environmental pressures can be positive or negative, and they account for the impact on the environment of human activities. Each element \(m_{ij}\) of the matrix \(M\) can take values \{high, medium, low, null\}, and defines a qualitative dependency between the activity \(i\) and the negative or positive pressure \(j\).

The second matrix, \(N\), relates the pressures with the environmental receptors, that register the effect of the pressures on the environment. For example, the activity “coal-fueled power plant” generates the pressure “emission of pollutants in the atmosphere” on the environment; on its turn, this influences the receptor “air quality” (as well as other receptors, like “human wellbeing” or “wildlife

\(^6\) http://www.epolicy-project.eu
In the same way, the pressure “emission of greenhouse gases” influences receptor “global warming”; while the pressure “emission of pollutants in the water” influences the “quality of river waters”. Again, each element $n_{ij}$ of the matrix can take the same qualitative values: high, medium, low or null.

Currently, the matrices relate 115 activities with 29 negative pressures, 19 positive pressures and 23 receptors. They can be used to assess a variety of regional plans, including Agriculture plans, Forest, Fishing, Energy, Industrial, Transport, Waste, Water, Telecommunications, Tourism, Urban plans. The environmental assessment is usually done using a spreadsheet and eliminating (by hand) those activities that do not belong to the given type of plan; then pressures that are not influenced by remaining activities are removed, and again receptors that are not influenced by the remaining pressures are removed as well. Finally, these reduced matrices are evaluated by environmental experts, that state which parts are most important, mainly considering clusters of High values.

Clearly, this process is very slow, experts might overlook combinations of medium or low values that might produce a significant effect, and, most importantly, it can be done only after the plan has been provided by the policy maker. At this stage, usually only minor modifications can be backpropagated to the plan, and it is practically impossible to compare the effect of the plan with other alternative plans, because this would need to start again another planning phase. As a matter of fact, although the evaluation of alternative plans is obligatory by EU regulations, this is usually not done, because planning and environmental assessment are done in a strictly sequential way.

To overcome the limitations and improve on current practices, we devised an expert system able to automatically assess regional plans [2]; then we extended it to include, in a single software component, a Decision Support System (DSS) able to provide optimal plans together with their environmental assessment [5], in particular for regional energy plans. We now recap the CLP model of such DSS in Section 2.1, and then extend it with new features in Section 3.

## 2.1 A first CLP solution

We model the planning problem in CLP on the Reals sort (CLP($\mathbb{R}$)). In CLP, one can define a problem through a set of variables, ranging on given domains; the possible assignments are restricted through a set of constraints; a solution is an assignment of values to variables such that all the constraints are satisfied. In many cases, solutions are not all equivalent, but there is an objective function to be maximized or minimized.

Given a number $N_a$ of activities, we consider a vector $A = (a_1, \ldots, a_{N_a})$ in which we associate to each activity a variable $a_i$ that defines its magnitude. The domain of $a_i$ depends on the availability of the resource on the given Region; for example some regions are very windy, while others can exploit better biomasses or solar panels.

We distinguish primary from secondary activities: primary activities are of primary importance for the given type of plan, while secondary activities are those supporting the primary activities by providing the needed infrastructures.
E.g., in an Energy plan, the activities that produce energy (e.g., power plants) are of primary importance; such activities require other activities (e.g., power lines, waste stocking, streets, etc.) to be performed, and they also should be considered in the environmental assessment. Let \( A^P \) be the set of indexes of primary activities and \( A^S \) that of secondary activities. The dependencies between primary and secondary activities are considered by the constraint:

\[
\forall j \in A^S \quad a_j = \sum_{i \in A^P} d_{ij} a_i
\]  

(1)

Each activity \( a_i \) has a cost \( c_i \); given a budget \( B_{Plan} \) available for a given plan, we have a constraint limiting the overall plan cost:

\[
\sum_{i=1}^{N_a} a_i c_i \leq B_{Plan}
\]  

(2)

Moreover, given an expected outcome \( out_{Plan} \) of the plan, we have a constraint ensuring to reach the outcome:

\[
\sum_{i=1}^{N_a} a_i out_i \geq out_{Plan}.
\]  

(3)

For example, in an energy plan the outcome can be to have more energy available in the region, so \( out_{Plan} \) could be the increased availability of electrical power (e.g., in kilo-TOE, Tonnes of Oil Equivalent). In such a case, \( out_i \) will be the production in kTOE for each unit of activity \( a_i \).

Concerning the impacts of the regional plan, an environmental expert suggested to convert the qualitative values in the matrices into the following numeric coefficients: high = 0.75, medium = 0.5, low = 0.25 and null = 0. We sum up the contributions of all the activities and obtain the estimate of the impact on each environmental pressure:

\[
\forall j \in \{1, \ldots, N_p\} \quad p_j = \sum_{i=1}^{N_a} m^i_j a_i.
\]  

(4)

In the same way, given the vector of environmental pressures \( \mathbf{P} = (p_1, \ldots, p_{N_p}) \), one can estimate their influence on the environmental receptor \( r_i \) by means of the matrix \( \mathbf{N} \), that relates pressures with receptors:

\[
\forall j \in \{1, \ldots, N_r\} \quad r_j = \sum_{i=1}^{N_p} n^i_j p_i.
\]  

(5)

Possible objective functions include maximizing the produced energy, minimizing the cost, or maximizing one of the receptors (e.g., maximizing the “air quality”), or a linear combination of the above.
3 Extended solution

The CLP(R) program described in Section 2.1 was practically used in the development of the 2011-13 Regional Energy plan of the Emilia-Romagna region of Italy. The main objective of the plan was to increase significantly the share of renewable energy in the energy mix, to fulfill the 20-20-20 directive. Indeed, during the years from 2011 to 2013, a large number of new renewable power plants was installed in the region, thanks to subsidies, that make them appealing from the market viewpoint. For the next years, technicians in the region foresee that old power plants fueled by fossil fuels will become obsolete, and they will have to be shut down completely, or, possibly, used only when renewable energy is unavailable or in peak hours. They asked us to extend the DSS to allow for possible closing of power plants, which means that some of the activities could have a negative magnitude: the magnitude, in MW, of oil-based power plants could be reduced with respect to the previous years.

First of all, one should notice that negative activities introduce nonlinearities. For example, if building a new power plant \( i \) has a cost \( c_i \) in Euros per MW, decommissioning it will not give a profit of \( c_i \, €/MW \).

Our implementation is based on the ECLiPSe CLP language [9,10], using the \texttt{eplex} library [11]. The \texttt{eplex} library uses very fast solvers based on linear programming or mixed-integer linear programming algorithms; this means that one can impose linear constraints on variables ranging either on continuous domains, or on integer domains. It is well known that linear programming is polynomially solvable, while (mixed) integer linear programming is NP-hard; thus the efficiency of the solution depends on whether there are integer variables or not. To address the non-linearity, we introduced, for each activity \( a_i \) that has negative values in its domain, an integer variable \( IsPos_i \) and a real variable \( Pos_i \); we wish to obtain that

\[
IsPos_i = \begin{cases} 
1 & \text{if } a_i \geq 0 \\
0 & \text{if } a_i < 0 
\end{cases} \quad Pos_i = \begin{cases} 
a_i & \text{if } a_i \geq 0 \\
0 & \text{if } a_i < 0 
\end{cases}.
\]

this can be obtained by imposing the following linear constraints:

\[
Pos_i \geq a_i \\
Pos_i \geq 0 \\
Pos_i \leq a_i + (1 - IsPos_i) \cdot M \\
Pos_i \leq IsPos_i \cdot M
\]

(where \( M \) is a sufficiently large positive number), and with the further integrality constraint \( IsPos_i \in \{0, 1\} \). The cost constraint (2) is now rewritten as

\[
\sum_{i=1}^{N} Pos_i \, c_i \leq B_{Plan}. \tag{6}
\]

Similarly, we do not want that secondary activities are decommissioned automatically when decommissioning primary activities; so we impose their relationship only with the positive part of primary activities.
Concerning the environmental assessment, as a first attempt we left the original linear constraints of equations (4-5), but the results were not considered satisfactory by the environmental expert. In fact, a new activity has a number of impacts, some for the construction of the activity (e.g., land use for building a coal power plant, pollution due to the construction site, etc.), and others due to running the activity (e.g., air pollution for burning fuel, water for cooling the plant, etc.). If we assume that the same coefficients in equation (4) can be used also for negative activities, we would correctly account for the second type of impacts, but we would wrongly assume that decommissioning a power plant means restoring the construction site as it was before. Moreover, we would not consider the end-life of the power plants, which can be significant (for example, consider nuclear power plants).

To account correctly for these cases, the environmental expert added new activities on the co-axial matrices (e.g., “Reduced use of fossil fueled power plants”), together with their impacts on environmental pressures. All the pressures are now computed only on positive activities, i.e., Equation (4) is substituted with

\[ \forall j \in \{1, \ldots, N_p\} \quad p_j = \sum_{i=1}^{N_a} m_{ij}^{+} Pos_i. \]  

(7)

Then, we considered the new activities as a new type of secondary activities: the “Reduced use of fossil fueled power plants” is a secondary activity that becomes positive only when one of the activities “Coal-based power plant”, “oil-based power plant”, etc., has a negative value. More precisely, we have secondary activities that are linked to the decomposition of other activities: e.g., activity “Reduction of fossil fuel power plants” is a secondary activity that is positive if one of the fossil fueled power plants has a negative value. Associated to activities we now have two matrices of dependencies between activities. In particular we have a \( N_a \times N_a \) square matrix \( D^+ \) where each element \( d_{ij}^+ \) represents the magnitude of activity \( j \) per unit of activity \( i \), and another \( N_a \times N_a \) square matrix \( D^- \) where each element \( d_{ij}^- \) represents the magnitude of activity \( j \) per unit of reduction of activity \( i \).

The dependency primary-secondary activities in Equation (1) is now substituted with

\[ \forall j \in A^S \quad a_j = \sum_{i \in A^P} K_{ij} \]  

(8)

where

\[ K_{ij} = \begin{cases} d_{ij}^+ \cdot a_i & \text{if } a_i \geq 0 \\ d_{ij}^- \cdot (-a_i) & \text{if } a_i < 0 \end{cases} . \]

### 3.1 Computing emissions

The base CLP program in Section 2.1 was able to provide the environmental assessment only in terms of qualitative information. We extended it to consider
also quantitative information, in particular the emission of pollutants in the air for each power plant type. We rely on the data provided by two databases: INEMAR [12] and ISPRA [13]. Both databases provide the various types of pollutants emitted per energy unit (in GJ) in input to the power plant. The considered types of pollutants include Sulfur Oxides (SO\textsubscript{X}), Nitrogen Oxides (NO\textsubscript{X}), methane, CO, CO\textsubscript{2}, N\textsubscript{2}O, ammonia, Hexachlorobenzene (HCB), various metals (Arsenic, Cadmium, Chromium, Copper, Mercury, Nickel, lead, Selenium, Zinc), particulate matter (PM10), Dioxins, and some families of compounds, like Polycyclic Aromatic Hydrocarbon compounds (PAH), Polychlorinated biphenyls (PCB), and Non-Methane Volatile Organic Compounds (NMVOC).

While ISPRA provides the average emission for each type of plant (biomasses, oil, coal, etc.), INEMAR provides fine grained information, in which emissions depend also on the type of boiler and the size of the plant (in MW).

We relate the power produced by plants with that of each boiler type. Let \( N_B \) the number of boiler types, we have a vector of constrained variables \( B = (b_1, \ldots, b_{N_B}) \) where \( b_i \) is the total output power of the plants using boiler type \( i \). Let \( O \) be the matrix that relates power plants and the different kinds of boiler: each element \( o_{ij} \) of the matrix is set to 1 if the boiler \( b_j \in B \) can be used for the power plant \( a_i \in A \), and zero otherwise. We impose that the output power of each plant type is the sum of the power of its boilers:

\[
\forall i \in \{1, \ldots, N_a\} \quad a_i = \sum_{j \in N_B} o_{ij} b_j
\]  

(9)

Let \( E = (e_1, \ldots, e_{N_e}) \) be the vector of emissions and \( T \) the matrix that relates them with the boilers. An element \( t_{ij} \) of the matrix represents the grams of pollutant \( e_i \in E \) emitted when 1GJ of fuel is provided to the boiler \( b_j \in B \).

To calculate the emissions, we have to compute the input energy for each boiler type \( j \), provided the output power \( b_j \):

\[
\forall i \in \{1, \ldots, N_e\} \quad e_i = \sum_{j \in N_B} t_{ij} \left( \frac{T^U}{\eta} b_j \right).
\]  

(10)

\( T^U \) is the average running time of a power plant per year (necessary to convert energy into power, and estimated in 8000 hours by our environmental expert) and \( \eta \) is the average efficiency (output power/input power) of power plants, which is prescribed by law as 39\% [14].

3.2 Indicators

With the computation of emissions (Section 3.1), the DSS provides new quantitative information, and lets the user find plans that are optimal with respect to objective functions that include emissions; for example, the user might require the plan that minimizes the emission of NO\textsubscript{X} or that of CO\textsubscript{2}, or even a weighted sum of the two. However, although useful, these might be too fine-grained for the environmental expert, not to mention for a policy maker: indeed, a policy maker
could know that \( NOX \) are toxic for humans, but how does that compare with the emission of heavy metal compounds? Instead, the policy maker knows that \( CO_2 \) is not harmful for human health, but it is responsible for the greenhouse effect; are there other emissions that worsen global warming?

The European Commission [15] published a set of indicators quantifying the effect of various substances on human toxicity, global warming and acidification. For example, Annex 1 of [15] contains 100 chemical substances together with their human toxicity factor, defined as the toxicity of the substance compared to that of lead (Pb). The following annexes contain global warming potentials, relative to \( CO_2 \), and acidification potentials, relative to \( SO_2 \). By using the weights in the tables, one can provide, e.g., the effect of the plan in terms of human toxicity (in kg of equivalent emitted lead), global warming (in kg of equivalent \( CO_2 \)) and acidification (in kg of equivalent \( SO_2 \)). Moreover, a policy maker may want to optimize on these indicators, and find the plan that minimizes human toxicity, or the greenhouse effect, or any weighted sum of the two.

However, the tables provided by the EC do not always have the same granularity of the information available for emissions. For example, for each plant type we know the emissions of \( NOX \); unluckily, in [15] we do not have an aggregated value for the toxicity of all the nitrogen oxides, but we have the single toxicity values of \( NO \) and \( NO_2 \), and they are quite different (respectively, 95 and 300 times that of lead). Even more complicated is for PAH, which include many compounds, e.g., Benzo-a-pyrene (toxicity 0.05 times that of Pb) and Naphthalene (500 times Pb). Our environmental expert suggested that we provided as output, for each indicator, three cases: best, worst, and average, considering respectively the highest toxicity in the compound class, the lowest and an average. Instead, when one of the indicators is in the objective function (e.g., one wants to find the plan with minimum human toxicity), we should optimize the worst case to be more conservative.

### 3.3 Computing the Pareto front

In the case of regional planning it is very hard (if not impossible) to devise a unique function that includes all the objectives that are important for the user. The optimization component described in Section 2 can provide optimal solutions with respect to one objective function that can be either the total amount of energy produced (both electrical and thermal), or the total cost, or the values of receptors, emissions and indexes explained in the previous sections. We decided to extend it to support also multi-objective optimization, to let the user compute more than one solution, and compare them.

In a multi-objective optimization problem, a Pareto optimal solution is such that it is not possible to improve the result for one objective function, without worsening at least one other objective function. More precisely, in a multi-objective problem with \( n \) functions to minimize, a solution \( \mu^* \) is Pareto-optimal if there does not exist another solution \( \mu \) such that \( \mu_j \leq \mu^*_j \) for \( 1 \leq j \leq n \) and there exists at least one \( i \), \( 1 \leq i \leq n \) such that \( \mu_i < \mu^*_i \). The set of Pareto points is distributed on the so-called Pareto frontier.
We implemented the *normalized normal constraint method* [16], an algorithm that works with any type of constraints (linear and nonlinear) and variables (continuous and discrete), and that is able to find an *evenly distributed* set of Pareto solutions. This is an important feature for a DSS, since it supplies the policy maker with a set of solutions that are a good representation of the whole space of Pareto solutions.

### 4 Graphical User Interface

A software for policy making should be usable by non-IT experts, and have an intuitive GUI able to visualize properly the heterogeneous information inherently present in environmental policies. We deployed the CLP planner as a stateless Web service and access it by means of a stateful Web application. This choice is also convenient from the perspective of a possible composition with the other services of the ePolicy application.

The CLP program is embedded inside a Java wrapper (Fig. 1a) that encodes the requests in CLP terms and decodes the results. This component provides a plethora of Java classes that represent the Business Object Model (BOM) of this domain. Any query addressed to this component and all the returned results are expressed in terms of these objects. Then we use the Apache CXF framework to define a Web Service’s Service Endpoint Interface (WS SEI) – an interface containing the signature of the method to call the service – and later to implement such a service taking advantage of the wrapper.

The Web application that stands as a GUI for the Web service is a standard Java servlet (Fig. 1b) following the Model-View-Controller (MVC) pattern: any request made through a browser is intercepted by the servlet which acts as a controller. The requests are forwarded to the BOM objects inside the model; these objects interact with our Web service and persistence layer to produce results. The controller then uses the JavaServer Pages (JSPs) to generate the view that becomes the response to display in the user’s browser. Both the Web service and the Web application are finally deployed to an application server. The
Web application can be accessed at: http://globalopt.epolicy-project.eu/Pareto/.

After a welcome page that introduces the software, there are an input page, and a results page. In input, the user can select the language to use (currently, Italian or English), insert bounds (minimum and maximum bounds for each energy source), constraints, and objective functions for the Pareto optimization, as well as the number of Pareto points (s)he wishes to compare. Constraints and objectives can include linear combinations of cost, produced power, receptors, emissions, or indicators. To simplify the input, the user can load the data for the Regional Energy Plan 2011-2013 of the Emilia-Romagna region for Electricity; in the following section we will show the results for this instance. The user can then compute the Pareto optimal plans, and a set of graphs is presented, as described in the next section.

4.1 Interpreting the Results

The results page consists of two master-slave panels. The left side-panel is the master and by clicking on any of its entries in one of its three sections, the view in the main panel (the slave) changes accordingly. Each view hosted in the main panel has many tabs and by selecting one of them, an appropriate graph or table is shown.

In particular, the left panel let the user select either a Scenario comparison, to compare all the generated scenarios, or to get detailed information on one scenario. Scenarios are divided into boundary scenarios, that are those that optimize one of the objective functions, and intermediate scenarios, that try to balance the various objectives.

Scenarios comparison. By clicking on General overview on the left panel, the user can compare the scenarios. One comparison is through a spiderweb chart (Fig. 2a) that has an axis for each objective function. Along each axis, the optimal values are far from the origin. Each scenario is represented by a polygon where each vertex is on a different axis. Generally speaking, a bigger polygon implies a better scenario (note that these solutions are Pareto optimal, so one polygon cannot be completely included into another polygon). Hovering the mouse on the axes, one can obtain the values for each plan. To improve legibility, one can deactivate one (or more) plans by clicking on it on the legend.

The scenarios can also be compared through stacked bar chart, showing, for each scenario, the distribution of costs per energy source (Fig. 2b), or the amount of electric/thermal energy per source. Again, hovering with the mouse over the graphs, more detailed information is provided.

Another scenario comparison is with respect to the values of the Objective functions selected by the users.

Finally, by clicking on Emissions and pollutants, three tabs (Fig. 3) show, in basic column charts, the amount of pollutants divided into the categories Heavy metals, Greenhouse gases, and Other pollutants. With the default data, heavy
metals are not present, because in the 2011-2013 plan there are no fossil fuels, the only sources emitting metals.

**Boundary and Intermediate scenarios.** These sections show detailed information for each of the computed scenarios on the Pareto front. Scenarios are divided into boundary scenarios, that are those that optimize one of the objective functions, and intermediate scenarios. For each scenario, the following views are available:

- **Receptors.** This composite view uses 7 VU-meter charts (Fig. 4a). The top part shows the 3 receptors with the best normalised value, while the bottom one the 3 with the worst normalised value. The main chart allows the user to select any receptor and appraise its normalised value. This graph was explicitly requested by an environmental expert to highlight the best and worst receptors.
- There are then four interactive tabular views (Fig. 4b) showing respectively, for the chosen scenario, the amount of produced energy per source, the total
5 Experiments

The software computes the optimal solution for one objective in a very short time; on modern computers it is well below 1 second. The multi-objective version has to compute a number of solutions, that depend on the number of scenarios (points on the Pareto front) requested by the user, so the computing time can grow up to some seconds, to compute around 5-10 scenarios (a number of scenarios that can be visualized and compared visually).

In order to assess the scalability, we performed a series of tests by randomly generating a set of data, including the co-axial matrices, the matrix relating primary and secondary activities, the activity costs, etc. In this way, we were able to stress-test the software with instances containing a number of activities, pressures and receptors larger than those in the actual data provided by ARPA.

The experiments were performed on a laptop computer running Linux with a 8x Intel Core i7-3720QM CPU at 2.60GHz; only one core was used in the experiments. The results are plotted in Figure 5, for a single objective. The $x$-axis is the size of the instance, i.e., the size of the co-axial matrices (each matrix is $N \times N$). The $y$-axis shows the time required to find the optimal solution.

The computing time, for sizes below 100, is always less than a second. Note that if $N = 100$, the matrix that relates activities and pressures has size $100 \times 100 = 10,000$, while in the real instance it is just $93 \times 48 = 4,464.
6 Conclusions and Future Work

We presented a DSS with optimization capabilities based on CLP for the regional planning and with particular emphasis on the environmental aspects of regional policies. The program was practically used to produce the energy plan 2011-2013 of the Emilia-Romagna region in Italy [6], and it is foreseen to use it also for the forthcoming plans. The CLP program is included into a web service, with an intuitive graphical user interface (http://globalopt.epolicy-project.eu/Pareto/), and that can be easily integrated with other components. The CLP program will be the heart of the platform of the EU FP7 ePolicy project, that will also include a social simulator, an opinion mining component, and a mechanism design component (based on game theory), all governed by the described CLP program. Preliminary work has been done on the integration of the CLP program with the mechanism design component [17], and a social simulator [18].

Acknowledgements This work was partially supported by EU project ePolicy, FP7-ICT-2011-7, grant agreement 288147.

References

1. Gavanelli, M., Milano, M., Bragaglia, S., Chesani, F., Marengo, E., Cagnoli, P.: Multi-criteria optimal planning for energy policies in CLP. Theory and Practice of Logic Programming, on-line supplement (2014)
Query Answering in Resource-Based answer set semantics

Stefania Costantini$^1$ and Andrea Formisano$^2$

$^1$ DISIM, Università di L’Aquila, Italy stefania.costantini@univaq.it
$^2$ DMI, Università di Perugia, Italy formis@dmi.unipg.it

Abstract. In recent work, we defined Resource-Based answer set semantics, which is an extension to traditional answer set semantics stemming from the study of its relationship with linear logic. In this setting there are no inconsistent programs, and constraints are defined “per se” in a separate layer. In this paper, we propose a query-answering procedure reminiscent of Prolog for answer set programs under this extended semantics.

1 Introduction

Answer set programming (ASP) is nowadays a well-established and successful programming paradigm based upon answer set semantics [1, 2, 3], with applications in many areas (cf., e.g., [4, 5, 6] and the references therein). Nevertheless, as noted in [7, 8], few attempts to construct a goal-oriented proof procedure exist. Rather, an ASP-solver is used (see [9]) to find the answer sets, if any exists. This is due to the very nature of the answer set semantics, where a program may admit none or several answer sets, and where the semantics enjoys no locality, or, better, no relevance in the sense of [10]: i.e., no subset of the given program can in general be identified, from where the decision of atom $A$ (intended as a goal, or query) belonging or not to some answer set can be drawn. The work of [7] suggests an incremental construction of approximations of answer sets, so as to provide local computations and top-down query answering. A sound and complete proof procedure for the approach is provided. The work of [8] can be used with non-ground queries and with non-ground, and possibly infinite, programs. Soundness and completeness results are proved for large classes of logic programs.

However, another problem related to goal-oriented answer-set-based computation is that of repeated queries. Assume that one would be able to pose a query ?− $Q_1$ receiving an answer “yes”, to signify that $Q_1$ is entailed by some answer set of given program $Π$. Presumably, one might intend subsequent queries to be answered in the same context, i.e., a subsequent query ?− $Q_2$ might reasonably ask whether some of the answer sets entailing $Q_1$ also entail $Q_2$. This might go on until the user explicitly “resets” the context. Such an issue, though reasonable in practical applications, has been hardly addressed up to now, due to the semantic difficulties that we have mentioned.

In recent work, stemming from our research on RASP [11, 12, 13], which is a recent extension of ASP, obtained by explicitly introducing the notion of resource, we

---

$^*$ This research will be presented at the ASPOCP14 workshop. This research is partially supported by GNCS-13 and GNCS-14 projects.
proposed in [14] a comparison between RASP, ASP and linear logic [15]. For establishing this correspondence, we introduced a RASP and linear-logic modeling of default negation as understood under the answer set semantics. This led to the definition of an extension to the answer set semantics which we called Resource-Based answer set semantics (RAS) [16]. This extension finds an alternative equivalent definition in a variation of the auto-epistemic logic characterization of answer set semantics discussed in [17]. In resource-based answer set semantics we have no inconsistent programs, and constraints are defined “per se” in a separate layer.

This allows us to propose here top-down procedure for the new semantics which, via a form of tabling, also provides contextualization. Differently from [7], this procedure does not require actual incremental answer set construction when answering a query. Rather, it exploits the fact that resource-based answer set semantics enjoys the property of relevance [10] (where answer set semantics does not), which guarantees that truth value of an atom can be established on the basis of the subprogram it depends upon, and thus allows for top-down computation starting from a query. As answer set semantics and resource-based answer set semantics extend the well-founded semantics [18], we take as a starting point XSB-resolution [19, 20], an efficient, fully described and fully implemented procedure which is correct and, for the class programs considered in the answer set semantics, always complete w.r.t. the well-founded semantics of given program. In this paper we do not provide the full detail of the proposed procedure, which we call RAS-XSB-resolution. In fact, this would imply suitably extending and reworking all definitions related to XSB. We rather lay the foundation, however with a precision and formality that should be sufficient to allow such a refinement as the next step. We also provide formal properties of the proposed procedure.

Notice that RAS-XSB resolution can be used for “traditional” answer set programming under the software engineering discipline of dividing the program into a consistent “base” level and a “top” level including constraints. Therefore, even to readers not particularly interested in the new semantics, the paper proposes a full top-down query-answering procedure for ASP, though applicable with previously mentioned (reasonable) limitation. With respect to top-down procedure proposed [8], we do not aim at managing function symbols (and thus programs with infinite grounding), so under this extent our work is more limited. However, we get correctness and completeness for every program (under the new semantics).

In the rest of the paper, after a short introduction of the answer set semantics we summarize resource-based answer set semantics. We then proceed to present the original contributions of this paper, that consist in introducing some useful properties of RAS, and in the definition of RAS-XSB-resolution. Background notions which are useful for a better understanding and proof of the main theorem are provided in the extended version of this paper, available online [21].

2 Background on ASP

In the Answer Set Semantics (originally named “stable model semantics”), a (logic) program II (cf., [1]) is a collection of rules of the form $H \leftarrow L_1, \ldots, L_n$, where $H$ is an atom, $n \geq 0$ and each literal $L_i$ is either an atom $A_i$ or its default negation not $A_i$. 
An answer set program can be seen as a Datalog program with negation (cf. [22, 23] for definitions about logic programming and Datalog). In what follows, unless explicitly differently specified we refer to ground programs, i.e., programs not containing variables. Below is the specification of the answer set semantics, reported from [1].

**Definition 1.** Given ASP program $\Pi$ and set of atoms $I$, the $\Gamma$ operator performs the following steps: (a) Computes the reduct $\Pi^I$ of $\Pi$, by:

1. removing from $\Pi$ all rules which contain a negative premise not $A$ such that $A \in I$;
2. removing from the remaining rules those negative premises not $A$ such that $A \notin I$; notice that $\Pi^I$ is a positive logic program.

(b) Computes the Least Herbrand Model of $\Pi^I$, denoted as $\Gamma_{\Pi}(I)$.

**Definition 2.** A set of atoms $I$ is an answer set for a program $\Pi$ iff $\Gamma_{\Pi}(I) = I$.

Answer sets are minimal supported models of the program interpreted in the obvious way as a first-order theory ($\leftarrow$ stands for implication, comma for conjunction and not for classical negation). It will be useful in what follows to consider a simple property of $\Gamma_{\Pi}$ (see [24]): if $M$ is a minimal model of $\Pi$, then, $\Gamma_{\Pi}(M) \subseteq M$.

In the answer set semantics, a rule of the form $\leftarrow L_1, \ldots, L_n$, is called constraint, and states that the $L_i$s cannot be all true w.r.t. any answer set. It is rephrased into a standard rule $Q \leftarrow \text{not } Q, L_1, \ldots, L_n$. with $Q$ fresh atom, as a contradiction on $Q$ leads to inconsistency, i.e., non-existence of answer sets (which in fact can in general be several, one, or none) unless one of the $L_i$s is false.

In this paper we refer for lack of space to the basic version of the answer set semantics. Therefore, we do not consider the various extensions and useful programming constructs that have appeared in the wide existing literature about ASP.

### 3 Resource-Based answer set semantics

In this section we introduce a formal definition of resource-based answer set semantics, which is needed in order to be able to define the proposed proof procedure and prove its properties. However, some preliminary observations are in order so as to explain why resource-based answer set semantics is reasonable, and might possibly be adopted as a proper extension of the answer set semantics. As it is well-known, the answer set semantics extends the well-founded semantics [18], which assigns to a logic program $\Pi$ a unique, three-valued model, called well-founded model and denoted by $WFS(\Pi) = \langle W^+, W^- \rangle$, where $W^+$ and $W^-$ are disjoint. In particular, $W^+$ is the set of atoms deemed true, $W^-$ is the set of atoms deemed false, while atoms belonging to neither set are deemed undefined. Atoms with truth value ‘undefined’ under the well-founded semantics are exactly the atoms which are of interest for finding the answer sets, and are, in particular, atoms involved in negative cycles, i.e. cycles through negation (as extensively discussed, e.g., in [24, 25, 26] and in the references therein).

In particular, the answer set semantics selects some of the (two-valued) classical models of given program so as for each atom $A$ which is true w.r.t. an answer set $M$, two conditions hold: (i) $A$ is supported in $M$ by some rule of the given program; (ii) the support of $A$ does not depend (directly or indirectly) upon the negation of another true
atom, including itself. For even cycles\(^3\) such as \(\{e \leftarrow \text{not } f. f \leftarrow \text{not } e.\}\), two answer sets can be found, namely \(\{e\}\) and \(\{f\}\), respecting both conditions. This extends to wider programs including such cycles. For odd cycles (such as unary odd cycles of the form \(\{p \leftarrow \text{not } p.\}\) and ternary odd cycles of the form \(\{a \leftarrow b. b \leftarrow \text{not } c. c \leftarrow \text{not } a.\}\)) it is not possible to assign truth values to their composing atoms in classical models. Thus, under the answer set semantics a program including such cycles is \textit{inconsistent}, i.e., it has no answer sets\(^4\). In a sense, the answer set semantics is still three-valued, as sometimes it is able to assign truth value to atoms, and sometimes (when the program is inconsistent) leaves them all undefined.

Resource-based answer set semantics is able to cope with any kind of cycle, and always assigns a truth value to all atoms while fulfilling conditions (i) and (ii). This by resorting to \textit{supported subsets} of classical models. For the unary odd cycle, \(p\) is deemed to be false because were it true, it would depend upon the negation of a true atom (itself). The ternary odd cycle has the three resource-based answer sets \(\{a\},\{b\}\) and \(\{c\}\). Taking for instance \(\{a\}\), atom \(b\) must be false to fulfill condition (i), and atom \(c\) must be false to fulfill condition (i) for itself and (ii) for \(a\).

Logical foundations of resource-based answer set semantics are discussed in depth in [27]. A characterization can be obtained by elaborating the auto-epistemic-logic characterization of answer set semantics discussed in [17]. Intuitively (for precise definitions please refer to [27]), a rule \(A \leftarrow A_1,\ldots,A_n,\text{not }B_1,\ldots,\text{not }B_m\) can be seen as standing for \(A_1 \land \ldots \land A_n \land \lnot B_1 \land \ldots \land \lnot B_m \land \lnot \bar{A} \supset A\). Where \(L\varphi\) can be understood as “\(\varphi\) is believed”, or also “we assume \(\varphi\)”, and \(\bar{A}\) can be understood as “one intends to prove \(A\)”. The overall reading is that \(A\) is derived whenever the positive conditions hold, we assume that the negative conditions hold as well, and we assume that we indeed intend to prove \(A\). Clearly, we have to state that \(A \land L\lnot A \supset \bot\) (if we have \(A\) we cannot believe its negation) and \(L\bar{A} \land L\lnot A \supset \bot\) (we cannot intend to prove \(A\) if we believe its negation). This characterization can be transposed into plain ASP by interpreting modal literals as fresh atoms. The answer sets (which are among the classical models) of the transposition \(\Pi'\) of a given program \(\Pi\) coincide, when removing fresh atoms, with the resource-based answer sets of \(\Pi\). Every program admits at least one (possibly empty) resource-based answer set.

The denomination of resource-based answer set semantics stems from the linear logic formulation of ASP that we proposed in [14, 16], which constituted the original inspiration for the new semantics. This formulation interprets negation \(\text{not } A\) of atom \(A\) as a resource that is unlimitedly available unless \(A\) is proved. Therefore, \(\text{not } A\) can be freely used whenever needed but: (1) \(\text{not } A\) becomes unavailable if \(A\) is proved; (2) whenever \(\text{not } A\) has been used, \(A\) can no longer be proved. For unary odd cycles such as \(\{p \leftarrow \text{not } p.\}\) in the linear logic formulation upon the attempt of using \(\text{not } p\) for proving \(p\), by condition (2) \(p\) becomes no longer provable (and thus it is false). Similarly for ternary odd cycles. Thus, under the resource-based answer set semantics a 3-atoms odd cycle is interpreted as an exclusive disjunction, exactly like 2-atoms even cycles. In the generate-and-test perspective which is the basis of the ASP programming methodology,

\(^3\) Even (resp. odd) cycles are cycles involving an even (resp. odd) number of negative dependencies, cf., e.g., [24, 25, 26] for precise definitions.

\(^4\) Unless “handles” are provided from other parts of the program, see [24, 25, 26] for details.
even cycles are commonly used to generate the search space. Thus, our new semantics provides some additional flexibility in this sense.

Resource-based answer set semantics is significantly different from other valuable semantic approaches aimed at managing odd cycles, such as [28, 29] and [30, 31]. Such a semantics can be characterized, similarly to traditional answer set semantics, by means of the $\Gamma$ operator (cf. Definition 1 in Sect. 2). In fact, the resource-based interpretation of negation requires that the negation of an atom (seen as a resource) that has been proved, becomes unavailable: the effect of $\Gamma$ is in fact exactly that of eliminating rules that make such use of negation.

For providing the formal definition, some preliminary consideration is needed. As discussed in [21], a nonempty answer set program $\Pi$ (that below we call simply “program”) can be seen as divided into a sequence of components, or layers, $C_1, \ldots, C_n$, $n \geq 1$, where each $C_i$ is the union of a set of cyclic or acyclic subprograms (subcomponents) independent of each other (with no atoms in common); each subcomponent of $C_1$, which is called the bottom of $\Pi$, is standalone, i.e., the atoms occurring therein do not depend upon other parts of the program; each subcomponent of $C_i$, $i > 1$, is on top of some subcomponent of $C_{i-1}$, i.e., the atoms occurring therein depend upon atoms occurring in $C_{i-1}$. For the formal definition of cyclic, acyclic on-top and standalone subprograms, refer to [21]. Based upon such a decomposition, as first discussed in [32], the answer sets of a program can be computed incrementally in a bottom-up fashion.

**Proposition 1.** Consider a nonempty ASP program $\Pi$, divided into components $C_1, \ldots, C_n$, $n \geq 1$. An answer set $S$ of $\Pi$ (if any exists) can be computed incrementally by means of the following steps:

0. Set $i = 1$.
1. Compute an answer set $S_1$ of component $C_1$ (for $i = 1$, this accounts to computing an answer set of the bottom component).
2. Simplify program $C_{i+1}$ by: (i) deleting all rules in which have not $B$ in their body, $B \in S_i$; (ii) deleting (from the body of the remaining rules) every literal not $F$ where $F$ does not occur in the head of rules of $C_{i+1}$, $F \notin S_i$, and every atom $E$ with $E \in S_i$. Notice that, once simplified, $C_{i+1}$ becomes standalone.
3. If $i < n$ set $i = i + 1$ and go to step 1, else set $S = S_1 \cup \ldots \cup S_n$.

Resource-based answer sets can be computed in a similar way. We start by defining the notion of resource-based answer sets of a given standalone program. In particular, they are obtained from some of its minimal models, specifically from the $\Pi$-based minimal models:

**Definition 3.** A $\Pi$-based minimal model $I$ of an ASP program $\Pi$ is either the empty set (in case it is the unique minimal model), or a nonempty minimal model such that $\forall A \in I$, there is a rule in $\Pi$ with head $A$, where $A$ does not occur positively in the body.

The restriction to $\Pi$-based minimal models is due to the fact that resource-based answer sets are supported sets of atoms. Thus, we aim at avoiding unsupported minimal models, such as, for sample one-rule program $a \leftarrow \text{not } c$, the minimal model $\{c\}$. Being $\Pi$-based is only a prerequisite for supportedness which however will be guaranteed by other conditions. Below we provide a variation of the answer set semantics that defines resource-based answer sets.
Definition 4. Let $\Pi$ be a standalone program, and let $I$ be a $\Pi$-based minimal model. $M$ is a resource-based answer set of $\Pi$ iff $M = \Gamma_\Pi(I)$ (we remind the reader that, for any model $I$ and program $\Pi$, $\Gamma_\Pi(I) \subseteq I$).

We are now ready to define resource-based answer sets of a generic program $\Pi$.

Definition 5. Consider a nonempty ASP program $\Pi$, divided into components $C_1, \ldots, C_n$, $n \geq 1$. A resource-based answer set $S$ of $\Pi$ is defined as $M_1 \cup \ldots \cup M_n$ where $M_1$ is a resource-based answer set of $C_1$, and each $M_i$, $1 < i \leq n$, is a resource-based answer set of standalone component $C_i'$, obtained by simplifying $C_i$ w.r.t. $S = M_1 \cup \ldots \cup M_{i-1}$, where the simplification consists in: (i) deleting all rules in $C_i$ which have not $B$ in their body, $B \in S$; (ii) deleting (from the body of remaining rules) every literal not $D$ where $D$ does not occur in the head of rules of $C_i$ and $D \notin S$, and also every atom $D$ with $D \in S$ (notice in fact that, once simplified, $C_{i+1}$ becomes standalone and therefore Definition 4 can be applied).

The above definition brings clear analogies to the procedure for answer set computation specified in [21]. Therefore, it is easy to see that, for consistent ASP programs, answer sets are among resource-based answer sets. Proposed program decomposition is also reminiscent of the one adopted in [7]. However, in general, resource-based answer sets are not models in the classical sense: rather, they are sets of atoms which are subsets of some minimal model, and are supported (similarly to answer sets, which are minimal supported models): in fact, from the above definitions it can be easily seen that for every atom $A$ in a resource-based answer set $M$, there exists a rule with head $A$, and body which is true w.r.t. $M$. Non-empty resource-based answer sets still form an anti-chain w.r.t. set inclusion.

We now explain by means of an example why the incremental construction of resource-based answer set is needed. Let $\Pi^E$ be the following:

$$a \leftarrow \neg p. \quad p \leftarrow \neg p. \quad q \leftarrow e. \quad e \leftarrow \neg q.$$  

Suppose to apply Definition 4 directly to the overall program. It admits a unique $\Pi^E$-based minimal model $S = \{p, q\}$, and we have $\Gamma_{\Pi^E}(S) = \emptyset$. This is reasonable for $p$ and $q$: in fact, they depend upon their own negation, so in our perspective there is not "enough" negation to prove them, thus they must be deemed to be false. It is reasonable also for $e$, which is involved (though through negation) in a positive circularity. It is however not reasonable for $a$, which depends upon negation of a false atom. However, according to Definition 5, we divide $\Pi^E$ into a standalone bottom component $C_1$, consisting of the last three rules, with $M_1 = \emptyset$ as the unique resource-based answer set, and a top component $C_2$ consisting of the first rule $a \leftarrow \neg p$: after simplification, $C_2'$ is simply fact $a$, unique resource-based answer set $M_2 = \{a\}$, which coincides with the unique resource-based answer set of the overall program, thus meeting the intuition.

We have called the new semantics Resource-Based Answer Set semantics (RAS), w.r.t. AS (Answer Set) semantics. Differently from answer sets, a (possibly empty) resource-based answer set always exists. Complexity of RAS semantics is however higher than complexity of AS semantics: in fact, [33] proves that deciding whether a set of formulas is a minimal model of a propositional theory is co-NP-complete. Clearly, checking whether a minimal model $I$ is $\Pi$-based and computing $\Gamma_{\Pi}(I)$ has polyno-
mial complexity. Then, checking whether a set of atom $I$ is a resource-based answer set of program $\Pi$ is co-NP-complete.

In resource-based answer set semantics there are no inconsistent programs. This means that constraints cannot be modeled (as done in ASP) in terms of odd cycles. Hence, they have to be modeled explicitly. Without loss of generality we will assume in the rest of the paper the following simplification concerning constraints. Each constraint $\leftarrow L_1, \ldots, L_k$, $k > 0$, where each $L_i$ is a literal, can be rephrased as simple constraint $\leftarrow H$, where $H$ is a fresh atom, plus rule $H \leftarrow L_1, \ldots, L_k$ to be added to given program $\Pi$. We will from now on implicitly consider the version of $\Pi$ enriched by such rules.

Definition 6. Let $\Pi$ be a program and $\{C_1, \ldots, C_k\}$ be a set of constraints, each $C_i$ in the form $\leftarrow H_i$. A resource-based answer set $M$ for $\Pi$ is admissible if it fulfills all constraints, i.e., if for all $i \leq k$, $H_i \notin M$. $M$ is admissible w.r.t. a single constraint $C_j$ if $H_j \notin M$.

4 Properties of Resource-Based answer set semantics

It is relevant, also for what follows, to evaluate RAS with respect to general properties of semantics of logic programs introduced in [10], that we recall below.

Definition 7. The sets of atoms a single atom $A$ depends upon, directly or indirectly, positively or negatively, is defined as $\text{dependencies}_{of}(A) = \{B : A \text{ depends on } B\}$.

The former definition is provided with some approximation, as dependencies should be formally checked on the dependency graph of given program [22, 23].

Definition 8. Given a program $\Pi$ and an atom $A$, $\text{rel\_rul}(\Pi; A)$ is the set of relevant rules of $\Pi$ with respect to $A$, i.e. the set of rules that contain an atom $B \in \text{dependencies}_{of}(A)$ in their heads.

Note that the notions introduced by Definitions 7 and 8 for an atom $A$ are plainly generalized to any set $X$ of atoms. Notice, moreover, that given an atom (or a set of atoms) $X$, $\text{rel\_rul}(\Pi; X)$ is a subprogram of $\Pi$.

Definition 9. Given any semantics $\text{SEM}$ and a ground program $\Pi$, Relevance states that for all literals $L$ it holds that $\text{SEM}(\Pi)(L) = \text{SEM}(\text{rel\_rul}(\Pi; L))(L)$.

Relevance implies that the truth value of any literal under that semantics in a given programs is determined solely by the subprogram consisting of the relevant rules. The answer set semantics does not enjoy relevance [10]. This is one reason for the lack of goal-oriented proof procedures. Instead, it is easy to see that

Proposition 2. Resource-based answer set semantics enjoys Relevance.

Resource-based answer set semantics, like most semantics for logic programs with negation, enjoys Reduction, which simply assures that the atoms not occurring in the heads of a program are always assigned truth value ‘false’. Resource-based answer set semantics also enjoys Modularity [10] (where the reduct $\Pi^M$ of program $\Pi$ w.r.t. set of atoms $M$ is recalled in Definition 1):
Definition 10. Given any semantics \( SEM \), a ground program \( \Pi \) let \( \Pi = \Pi_1 \cup \Pi_2 \) where for every atom \( A \) occurring in \( \Pi_2 \), \( \text{rel}_\text{rul}(\Pi; A) \subseteq \Pi_2 \). \( SEM \) enjoys Modularity if \( SEM(\Pi) = SEM(\Pi_1^{SEM(\Pi_2)} \cup \Pi_2) \).

We can in fact prove the following proposition:

Proposition 3. Given a ground program \( \Pi \) let \( \Pi = \Pi_1 \cup \Pi_2 \), where for every atom \( A \) occurring in \( \Pi_2 \), \( \text{rel}_\text{rul}(\Pi; A) \subseteq \Pi_2 \). A set \( M \) of atoms is a resource-based answer set of \( \Pi \) iff there exists a resource-based answer set \( S \) of \( \Pi_2 \) such that \( M \) is a resource-based answer set of \( \Pi_1^{S} \cup \Pi_2 \).

Modularity is an important property, that also impacts on constraint checking, i.e., on the check of admissibility of resource-based answer sets. Considering, in fact, a set of constraints \( \{C_1, \ldots, C_n\} \), \( n > 0 \), each \( C_i \) in the form \( \leftarrow H_i \), and letting for each \( i \leq n \) \( \text{rel}_\text{rul}(\Pi; H_i) \subseteq \Pi_2 \), from Prop. 3 it follows that, if a resource-based answer set \( X \) of \( \Pi_2 \) is admissible (in terms of Definition 6) w.r.t. \( \{C_1, \ldots, C_n\} \), then any resource-based answer set \( M \) of \( \Pi \) such that \( X \subseteq M \) is also admissible w.r.t. this set of constraints. In particular, \( \Pi_2 \) can be identified in relation to a certain query, i.e.:

Definition 11. Given a program \( \Pi \), a constraint \( \leftarrow H \) associated to \( \Pi \) is relevant for query \( \leftarrow A \) if \( \text{rel}_\text{rul}(\Pi; A) \subseteq \text{rel}_\text{rul}(\Pi; H) \).

5 A Proof Procedure for RAS

As said before, the answer set semantics extends the well-founded semantics. Resource-based answer set semantics still extends the well-founded semantics, as it still deals with assigning a truth value to atoms which are undefined under the this semantics: however, it is able to cope with odd cycles that the answer set semantics interprets as inconsistencies. Assuming to devise a query-answering device for ASP, query \( \leftarrow A \) to ASP program \( \Pi \) may be reasonably expected to succeed or fail if \( A \) belongs to \( W^+ \) or \( W^- \) respectively, but how to find an answer if \( A \) is undefined because it is involved in a negative circularity remains to be understood.

An additional problem with answer set semantics is that query \( \leftarrow A \) might locally succeed, but still, for the lack of relevance, the overall program may not have answer sets (i.e., the program is inconsistent). In resource-based answer set semantics instead, there are no inconsistent programs and every program has at least one (possibly empty) resource-based answer set: each of them taken singularly is then admissible or not w.r.t. the integrity constraints. This allows one to defer constraint checking in case the proof of query \( A \) succeeds. In this section, we present and discuss the foundations of a proof procedure for logic programs under resource-based answer set semantics.

We take as a starting point a well-established proof procedure for the well-founded semantics, namely XSB-resolution. An ample literature exists for XSB-resolution, from the seminal work in [20] to the most recent work in [19] where many useful references can also be found. XSB resolution is fully implemented, and information and downloads can be find on the XSB web site, xsb.sourceforge.net/index.html.

For lack of space, here we do not describe XSB-resolution in detail. We provide in [21] some definitions and results useful for a general understanding. In the rest of this
section, we proceed to illustrate how we mean to extend XSB in order to cope with undefined atoms.

XSB-resolution [19] adopts tabling, that will also be useful in what follows. Tabled logic programming was first formalized in the early 1980’s, and several formalisms and systems have been based both on tabled resolution and on magic sets, which can also be seen as a form of tabled logic programming (c.f. [19] for references). In the Datalog context, tabling simply means that whenever atom \(S\) is established to be true or false, it is recorded in a table. Thus, when subsequent calls are made to \(S\), the evaluation ensures that the answer to \(S\) refers to the record rather than being re-derived using program rules. Seen abstractly, the table represents the given state of a computation: in this case, subgoals called and their answers so far derived. One powerful feature of tabling is its ability to maintain other global elements of a computation in the “table”, such as information about whether one subgoal depends on another, and whether the dependency is through negation. By maintaining this global information, tabling is useful for evaluating logic programs under the well-founded semantics. The essential idea is that global information about dependencies is used to determine the truth value of literals that do not have a derivation. If such literals are involved in a cyclic dependency through negation, they are undefined under WFS; if not, the literals belong to an unfounded set and are false in WFS. In fact, it can be shown that tabling allows Datalog programs with negation to terminate with polynomial data complexity under the well-founded semantics.

We will now define the foundations of a top-down proof procedure for resource-based answer set semantics, which we call RAS-XSB-Resolution. The procedure has to cope with the fact that there are atoms which are involved in negative circularities, and must be assigned a truth value according to some resource-based answer set. We build upon XSB-Resolution, which is by no means elementary, so we refer the reader to the references for a proper understanding. An abridged specification is provided below for the reader’s convenience, based upon preliminary definition reported in [21]. In order to give an intuitive idea, we resort, in fact, to the following “naive” formulation, relying upon general definitions reported in [22, 23].

**Definition 12 (A “naive” XSB-resolution).** Given a program \(\Pi\), let \(\text{Table}(\Pi)\) be the data structure used by the proof procedure for tabling purposes, i.e., the table associated with the program (or simply “program table”). Given a query \(\neg - A\), the list of current subgoals is initially set to \(L_1 = \{A\}\) and \(\text{Table}(\Pi)\) is initialized to be the empty set. If in the construction of a proof-tree for \(\neg - A\), a literal \(L_i\) is selected in the list of current subgoals \(L_i\), we have that: if \(L_i\) definitely succeeds (in case of a negative literal \(L_i = \text{not } B\), it definitely succeeds if \(B\) definitely fails) then we take \(L_i\) as proved and proceed to prove \(L_{i+1}\) after the related updates to the program table. Otherwise, we have to backtrack to previous list \(L_{i-1}\) of subgoals. Success and failure determine suitable modifications to \(\text{Table}(\Pi)\).

On Datalog programs XSB is correct and complete, therefore, under XSB-resolution, atom \(A\) definitely succeeds iff \(A \in W^+\) and definitely fails iff \(A \in W^-\). Note that definite failure occurs not only when at some point there is an atom not defined by any rules, but also whenever an atom depends positively in any possible (even
indirect) way upon itself. In our extension, we take the above result as a starting point for success and failure.

In order to represent the notion of negation as a resource, we initialize the program table prior to posing queries and we manage the table during a proof so as to state that: the negation of any atom which is not a fact is available unless this atom has been proved; the negation of an atom which has been proved becomes unavailable; the negation of an atom which cannot be proved is always available.

**Definition 13 (Table Initialization in RAS-XSB-Resolution).** Given a program $Π$ and an associated table $\text{Table}(Π)$, Initialization of $\text{Table}(Π)$ is performed by inserting, for each atom $A$ occurring as the conclusion of some rule in $Π$, a fact $\text{yes}A$ (where $\text{yes}A$ is a fresh atom).

The meaning of $\text{yes}A$ is that the whole amount of $A$'s negation is still available. If $\text{yes}A$ is present then $A$ can possibly succeed. Success of $A$ “absorbs” $\text{yes}A$ and prevents $\neg A$ from success. Resource-based answer set semantics in fact dictates that proving $A$ consumes the whole amount of $A$’s negation.

$\text{Table}(Π)$ will evolve during a proof into subsequent “knowledge states”. In the following, without loss of generality we can assume that a query is of the form $\sim A$, where $A$ is an atom. A proof of query $\sim A$ is performed by XSB-resolution, though with the following additive modifications.

**Definition 14 (Success and failure in RAS-XSB-Resolution).** Given program $Π$ and its associated table $\text{Table}(Π)$, notions of success and failure and of modifications to $\text{Table}(Π)$ are extended as follows with respect to XSB-Resolution.

1. Atom $A$ succeeds if one of the following is the case:
   (a) $A$ is present in $\text{Table}(Π)$.
   (b) Fact $\text{yes}A$ is present in $\text{Table}(Π)$, and there exists in $Π$ either fact $A$ or a rule of the form $A ← L_1, \ldots, L_n$, $n > 0$, such that every $L_i$, $i ≥ n$, succeeds. Define success of $A$ as a particular case.

   In consequence of success of $A$, fact $A$ is added to $\text{Table}(Π)$ (if not already present), and fact $\text{yes}A$ is removed.

2. Atom $A$ fails if one of the following is the case:
   (a) Fact $\text{yes}A$ is not present in $\text{Table}(Π)$, and therefore $A$ is unprovable.
   (b) A definitely fails.
   (c) There exists no rule of the form $A ← L_1, \ldots, L_n$, $n > 0$, such that every $L_i$ succeeds, as one of the following is the case:
      (i) Some positive literal, among $L_1, \ldots, L_n$, fails.
      (ii) Some negative literal, among $L_1, \ldots, L_n$, fails.
      (iii) Any possible derivation of some of the $L_i$s, $i ≤ n$, incurs into not $A$ directly, i.e., not through layers of negation.
      (iv) Any possible derivation of some of the $L_i$s, $i ≤ n$, incurs into $A$ through layers of negation that do not involve not $A$.

   In cases (iii) and (iv) we say that $A$ is forced to failure. In consequence of failure of $A$, fact $\text{yes}A$ is removed from $\text{Table}(Π)$ (if present). In case $A$ is forced to failure, for every positive literal $B$ encountered in the derivation from $A$ to, respectively, $A$ or not $A$, fact $\text{yes}B$ is removed from $\text{Table}(Π)$ (if present).

3. Literal $\neg A$ succeeds if one of the following is the case:
   (a) Fact $\neg A$ is present in $\text{Table}(Π)$. 

(b) \( A \) fails.
(c) \( A \) does not fail, rather any derivation of not \( A \) incurs through layers of negation again into not \( A \). In this case we say that not \( A \) is allowed to succeed.

In consequence of success of not \( A \), fact yes\( A \) is removed from Table(II) (if present), and fact not \( A \) is added to Table(II). In case however not \( A \) is allowed to succeed, in case the parent subgoal fails yes\( A \) is restored and not \( A \) is removed.

(4) Literal not \( A \) fails if \( A \) succeeds.

From this extension of the notions of success and failure we obtain RAS-XSB-Resolution as an extended XSB-Resolution. The “naive” definition is the following (a precise operational definition will require a punctual modification of all definitions related to XSB).

**Definition 15 (A “naive” RAS-XSB-resolution).** Given a program II, let assume as input the data structure Table(II) used by the proof procedure for tabling purposes, i.e., the table associated with the program (or simply “program table”). Given a query \(? - A \), the list of current subgoals is initially set to \( L_1 = \{ A \} \). If in the construction of a proof-tree for \(? - A \) a literal \( L_{ij} \) is selected in the list of current subgoals \( L_i \), we have that: if \( L_{ij} \) succeeds then we take \( L_{ij} \) as proved and proceed to prove \( L_{i+1} \) after the related updates to the program table. Otherwise, we have to backtrack to the previous list \( L_{i-1} \) of subgoals. Conditions for success and failure are those specified in Definition 14. Success and failure determine the modifications to Table(II) specified for XSB-resolution, plus those specified in Definition 14. Backtracking involves restoring previous contents of Table(II).

**Definition 16.** Given a program II, its associated table Table(II), a free query is a query \(? - A \) which is posed on II when the table has just been initialized. A contextual query is a query \(? - B \) which is posed on II leaving the associated table in the state created by former queries.

Success of query \(? - A \) means that there exist resource-based answer sets that contain \( A \). These sets are further characterized by the final content of Table(II), which encompasses a number of literals which hold in therein. Backtracking on \(? - A \) accounts to asking whether there are other different resource-based answer sets containing \( A \), and implies accordingly backtracking Table(II) to previous contents. Instead, posing a subsequent query \(? - B \) without resetting the contents of Table(II), which constitutes a context, accounts to asking whether some of the already-computed resource-based answer sets containing \( A \) also contain \( B \). Contextual queries and sequences of contextual queries are formally defined below.

**Definition 17 (Query sequence).** Given a program II and \( k > 1 \) queries \(? - A_1, \ldots, ? - A_k \) performed one after the other, Table(II) is initialized only before posing \(? - A_1 \). Thus, \(? - A_2 \) is a free query where each \(? - A_i \) is contextual w.r.t. the previous ones.

To show the application of RAS-XSB-resolution to single queries and to a query sequence, let us consider the following sample program II, which includes virtually all cases of potential success and failure. The well-founded model of this program is \( \langle \{ e \}, \{ d \} \rangle \), since \( e \) is true as it is a fact, \( d \) is false as it has no defining rules, and
all the other atoms are undefined. In fact, they are involved in negative circularities either directly or indirectly (through dependencies, like $s$ and $f$). There is an even cycle involving $a$ and $g$, and a unary odd cycle on $p$, which however depends upon its own negation indirectly, i.e., $p$ depends upon $h$ which in turn depends upon not $p$.

$T_1$: $a \leftarrow \neg not\ g.$  
$T_3$: $s \leftarrow \neg not\ p.$  
$T_5$: $h \leftarrow \neg not\ p.$  
$T_7$: $f \leftarrow \neg not\ g, e.$  
$T_2$: $g \leftarrow \neg not\ a.$  
$T_4$: $p \leftarrow h.$  
$T_6$: $f \leftarrow \neg not\ a, d.$  
$T_8$: $e.$

The resource-based answer sets of such $II$ are $M_1 = \{e, a, f, s\}$ and $M_2 = \{e, s, g\}$.

Below we illustrate some derivations. Initially, $\mathcal{Table}(II)$ includes yes$_A$ for every atom occurring in some rule head: $\mathcal{Table}(II) = \{yes, yesb, yesc, yesf, yesg, yesp, yesh, yess\}$. Let us go through the proof of query $?← f$, assuming to adopt a Prolog-like search strategy: applicable rules from first to last as they occur in the program, literals in rule bodies from left to right. Each additional layer of $?←$ indicates nested derivation of $A$ whenever literal not $A$ is encountered. In the comment, we refer to cases of RAS-XSB-resolution as specified in Definition 14.

\begin{verbatim}
?← f.
?← not a, d.  % via $T_6$
?← a.  % via $T_1$
?← not g.  % via $T_1$
?← g.  % via $T_1$
?← not a.  % via $T_2$, not a succeeds by case 3.c, $\mathcal{Table}(II) = \mathcal{Table}(II) \cup \{not\ a\} \setminus \{yes\}$
?← d.  % $d$ fails by case 2b, previous $\mathcal{Table}(II)$ restored, backtracking
?← not g, e.  % via $T_7$
?← g.  % via $T_7$
?← not a.  % via $T_2$
?← a.  % via $T_1$
?← not g.  % via $T_1$, not $g$ succeeds by case 3.c, $a$ succeeds by case 1.b,  
$\mathcal{Table}(II) = \mathcal{Table}(II) \cup \{a, not\ g\} \setminus \{yes, yesg\}$
?← e.  % $e$ succeeds by case 1.b, overall query $f$ succeeds by case 1.b
$\mathcal{Table}(II) = \mathcal{Table}(II) \cup \{e, f\} \setminus \{yes, yesf\}$
\end{verbatim}

Assuming now to go on to query the same context, i.e., without re-initializing $\mathcal{Table}(II)$, queries $?← e$ and $?← g$ quickly fail by cases 3.c and 1.a, since $a ∈ \mathcal{Table}(II)$. Query $?← e$ succeeds immediately by case 1.a as $e ∈ \mathcal{Table}(II)$. We can see that the context we are within corresponds to resource-based answer set $M_1$, where only $s$ remains to be proved. This can be done as follows:

\begin{verbatim}
?← s.
?← not p.  % via $T_3$
?← p.  % via $T_3$
?← h.  % via $T_4$
?← not p.  % via $T_5$, $p$ fails by case 2.c.iii, $h$ fails by case 2.c.ii, $\mathcal{Table}(II) = \mathcal{Table}(II) \setminus \{yes, yesh\}$, not $p$ succeeds by case 3.b, $s$ succeeds by case 1.b,  
$\mathcal{Table}(II) = \mathcal{Table}(II) \cup \{not\ p, s\} \setminus \{yes\}$
\end{verbatim}

It remains to show how the derivation of $h$ proceeds, as it involves the tricky case of a positive dependency through negation, where $h$ is still undefined under the well-founded semantics.

?← h.
?← not p.  % via $T_5$
\[ ?- - h. \% h \text{ fails by case 2.c.iv. not } p \text{ succeeds by case 3.b} \]

\[ \text{Table}(I) = \text{Table}(I) \setminus \{\text{yesp, yesh}\} \cup \{\text{not } p\} \]

6 Properties of RAS-XSB-resolution

Properties of resource-based answer set semantics are strictly related to properties of RAS-XSB-resolution. In fact, thanks to Relevance we can have soundness and correctness, and Modularity allows for contextual query and locality in constraint-checking.

**Theorem 1.** RAS-XSB-resolution is correct and complete w.r.t. resource Answer Set semantics, in the sense that, given program \( \Pi \), query \( ?- A \) succeeds under RAS-XSB-resolution with an initialized \( \text{Table}(\Pi) \) iff there exists resource-based answer set \( M \) for \( \Pi \) where \( A \in M \).

**Theorem 2.** RAS-XSB-resolution is contextually correct and complete w.r.t. resource Answer Set semantics, in the sense that, given program \( \Pi \) and query sequence \( ?- A_1, \ldots, ?- A_k, k > 1 \), we have that, for \( \{B_1, \ldots, B_r\} \subseteq \{A_1, \ldots, A_k\} \) and \( \{D_1, \ldots, D_s\} \subseteq \{A_1, \ldots, A_k\} \), the queries \( ?- B_1, \ldots, ?- B_r \) succeed while \( ?- D_1, \ldots, ?- D_s \) fail under RAS-XSB-resolution, iff there exists resource-based answer set \( M \) for \( \Pi \) where \( \{B_1, \ldots, B_r\} \subseteq M \) and \( \{D_1, \ldots, D_s\} \cap M = \emptyset \).

In fact, keeping in \( \text{Table}(\Pi) \) atoms and literals proved so far accounts to performing the simplification of given program \( \Pi \) w.r.t. a resource-based answer set computed for the subprogram including relevant rules of previous queries. Therefore, the result descends from Modularity of resource-based answer set semantics. It remains to consider the issue of constraint checking. Notice that, due to modularity of RAS, if \( \Pi \) is admissible, then only constraints relevant to given query need to be checked.

**Proposition 4.** Let \( \Pi \) be an admissible program w.r.t. the constraints \( \leftarrow H_1, \ldots, \leftarrow H_h \) (it has admissible answer sets). Let \( \leftarrow H_1, \ldots, \leftarrow H_k, k \leq h \) be the relevant constraints for a query \( ?- A \). Then, if \( ?- A \) succeeds and each \( H_i, i \leq k \), considered as a query, contextually succeeds as well, then there exists some admissible resource-based answer set \( M \) for \( \Pi \) with \( A \in M \).

If admissibility of \( \Pi \) is unknown, all constraints must instead be checked. Checking constraints on the state of \( \text{Table}(\Pi) \) left by a query alleviates the efficiency problem. “Smart” heuristics, such as those presently adopted by answer set solvers, for checking constraints during the proof process might also be in order.

7 Discussion and Concluding Remarks

A relevant question about RAS-XSB-resolution concerns whether it is applicable to non-ground queries and programs. By resorting to standard unification, non-ground queries on ground programs are managed without substantial modifications. We claim that the procedure can be extended to non-ground programs without requiring preliminary program grounding. Transforming this claim into evidence requires however an actual reworking of XSB-resolution definitions and proofs. This is a topic for future work.
Another relevant question is whether RAS-XSB-resolution might be extended to plain ASP. Unfortunately, an ASP program may have a quite complicated structure: the effort of in [7] has been in fact that of performing a layer-based computation upon some conditions. Thus, the adoption of RAS-XSB-resolution is possible at the condition of structuring an ASP program so that constraints are at the top layer. Many applications are already expressed in this form, which means that the proposed procedure may have an impact beyond resource-based answer set semantics.

In summary, we have proposed the theoretical foundations of a proof procedure related to a reasonable extension of answer set programming. The procedure has been obtained by exploiting properties of both answer set semantics and resource-based answer set semantics, which enable us to resort as a starting point to XSB-resolution. The new procedure has drawn inspiration from the tabling feature of XSB-resolution. Future work includes a precise definition of RAS-XSB-resolution, and an implementation, that should then be checked and experimented on (suitable versions of) well-established benchmarks (see, e.g., [34, 35]). We also intend to investigate an integration of RAS-XSB-resolution with principle and techniques proposed in [8], so as to further enlarge its applicability.

References

Abstract. In this paper we investigate multiagent systems whose agent interaction is based on social commitments that evolve over time, in presence of (possibly incomplete) data. In particular, we are interested in modeling and verifying how data maintained by the agents impact on the dynamics of such systems, and on the evolution of their commitments. This requires to lift the commitment-related conditions studied in the literature, which are typically based on propositional logics, to a first-order setting. To this purpose, we propose a rich framework for modeling data-aware commitment-based multiagent systems. In this framework, we study verification of rich temporal properties, establishing its decidability under the condition of “state-boundedness”, i.e., data items come from an infinite domain but, at every time point, each agent can store only a bounded number of them.

1 Introduction

In this paper we investigate multiagent systems (MASs) whose agent interaction is based on social commitments that evolve over time, in presence of possibly incomplete data. MASs based on social commitments have been extensively studied in the literature [8]. Intuitively, a social commitment $\text{CC}(d, c, q_p, q_d)$ models a relationship between a debtor agent $d$ and a creditor agent $c$, in which $d$ commits towards $c$ that, whenever condition $q_p$ holds in the system, it will bring about condition $q_d$ in the following course of interaction. Commitments provide a semantics for the agent interaction that abstracts away from the internal agent implementation, and can be thus employed to specify business protocols and contracts. The establishment of commitments is regulated by contracts, which depend on domain-specific events and conditions. Established commitments, in turn, have a lifecycle that is regulated by a so-called commitment machine [12] on the basis of such contracts. While in the literature, virtually all the work is based on propositional contents for such commitments [8], here we explicitly manage data described through first-order formalisms, in line with [7]. In other words, we study how data maintained by the agents impact on the dynamics of such systems, and on the evolution of their commitments. Technically, this requires to lift to first-order the notions related to contracts, commitments, and commitment machines.

As a result, we obtain a powerful framework of data-aware commitment-based MASs (DACMASs), which incorporates the typical notions of commitment-based MASs

* This paper is a short version of [10], to be presented at AAMAS 2014.
but in a rich, data-aware context. In our framework, the commitment machine itself becomes a special agent, called institutional, which is a agent, in charge of supporting the evolution of the system according to the commitments. In addition, this agent manages core information about the MAS itself, such as the list of participating agents, which changes over time as the system unfolds. In this light, it maintains and manipulates a common knowledge base, of interest for all the interacting agents. The data manipulated by the agents are described in terms of a domain ontology, expressed in a lightweight description logic (DL), tailored towards ontology-based data access. This ontology provides a common ground for the agent interaction and commitments, establishing the vocabulary that is shared by all of them. In particular, we rely on DLR-Lite [5], which is the n-ary version of the DL at the base of the OWL 2 QL profile of the OWL 2 semantic web standard. Each agent has its own data about the domain and the contracts it is involved in, expressed in terms of such ontology. Such data are manipulated through actions, in response to events and according to the commitments in place. At each point in time, only a finite number of data is present in the system. However, such data change over time: old data are removed by the agents, and new data (coming from a countably infinite domain $\Delta$) are inserted.

Our main result is that, when a DACMAS is state-bounded, i.e., the number of data that are simultaneously present at each moment in time is bounded, verification of rich temporal properties becomes decidable. More specifically, we are able to check DACMASs against properties expressed in a sophisticated first-order variant of $\mu$-calculus with a controlled form of quantification across states. We do this by exploiting recent results in [2], and reducing verification of state-bounded DACMASs to finite-state model checking through a faithful form of abstraction, essentially obtained by replacing real data items with a finite number of symbolic values, while correctly preserving the relationships among the real data items themselves.

2 Preliminaries

Description Logics (DLs) [1] are logics that represent the domain of interest in terms of objects, concepts, denoting sets of objects, and relations between objects. We consider here the DL DLR-Lite [5], which is a DL that belongs to the DL-Lite family of lightweight DLs and that is equipped with relations of arbitrary arity. In DLR-Lite, concepts $C$ and relations $R$ are built from atomic concepts $N$ and atomic relations $P$ (of arity $\geq 2$):

$C \rightarrow N \mid P[i_1,\ldots,i_h] \mid C \land C \mid R \rightarrow P \mid P[i_1,\ldots,i_m]$ where $h \geq 2$ and for $i_1,\ldots,i_h$, which denote pairwise distinct components of relation $P$, we have that $\{i_1,\ldots,i_h\} \subseteq \{1,\ldots,n\}$, where $n$ is the arity of $P$. Similarly, $i \in \{1,\ldots,n\}$. Intuitively, $\land$ denotes concept conjunction, while $P[i_1,\ldots,i_m]$ denotes the projection of relation $P$ on its components $i_1,\ldots,i_m$. This results in a concept if $m = 1$ and in a relation otherwise.

Formally, the semantics of DLs is given in terms of first-order interpretations $I = (\Delta^I, \cdot^I)$, where $\Delta^I$ is a nonempty interpretation domain, and $\cdot^I$ is an interpretation function, assigning to each concept $C$ a subset $C^I$ of $\Delta^I$, and to each $n$-ary relation $R$ an $n$-ary relation $R^I$ over $\Delta^I$ such that

$$(C_1 \land C_2)^I = C_1^I \cap C_2^I$$

$$(P[i_1,\ldots,i_m])^I = \{ (o'_1,\ldots,o'_m) \mid \text{there is } o \in P^I \text{ s.t. } o[i_j] = o'_j, \text{ for } j \in \{1,\ldots,m\} \}$$

85
(Here, $o_i^j$ denotes the $i$-th component of tuple $o$.) Also, $\mathcal{I}$ assigns to each constant $a$ an object $a^\mathcal{I}$ of $\Delta^\mathcal{I}$. We adopt the unique name assumption, i.e., $a_1 \neq a_2$ implies $a_1^\mathcal{I} \neq a_2^\mathcal{I}$.

In DLs, knowledge about the domain of interest is encoded in an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A}\rangle$, which is formed by a TBox $\mathcal{T}$, encoding intensional knowledge, and an ABox $\mathcal{A}$, encoding extensional knowledge about individuals objects.

A DLR-Lite TBox is a finite set of assertions of the form:
\[
E_1 \sqsubseteq E_2 \quad \text{(concept/relation inclusion assertion),}
\]
\[
E_1 \sqsubseteq \neg E_2 \quad \text{(concept/relation disjointness assertion),}
\]
\[
\{\text{key } i_1, \ldots, i_k; R\} \quad \text{(key assertion),}
\]
where $R$ has arity $n$, and $1 \leq i_1 < i_2 < \cdots < i_k \leq n$. To ensure decidability of inference, and good computational properties, we require that no relation $P$ can appear both in a key assertion and in the right hand side of a relation inclusion assertion [11,5].

A DLR-Lite ABox is a finite set of assertions of the form:
\[
N(a_1) \quad \text{(concept membership assertion),}
\]
\[
P(a_1, \ldots, a_n) \quad \text{(relation membership assertion),}
\]
where $P$ has arity $n$, and $a_1, \ldots, a_n$ denote constants.

The semantics of an ontology is given by stating when an interpretation $\mathcal{I}$ satisfies an assertion, where $\mathcal{I}$ satisfies: $E_1 \sqsubseteq E_2$, if $E_1^\mathcal{I} \subseteq E_2^\mathcal{I}$; $E_1 \sqsubseteq \neg E_2$, if $E_1^\mathcal{I} \cap E_2^\mathcal{I} = \emptyset$; (key $i_1, \ldots, i_k; R$), if there are no two distinct tuples in $R^\mathcal{I}$ that agree on all their components $i_1, \ldots, i_k$; $N(a_1)$, if $a_1^\mathcal{I} \in N^\mathcal{I}$; and $P(a_1, \ldots, a_n)$, if $(a_1^\mathcal{I}, \ldots, a_n^\mathcal{I}) \in P^\mathcal{I}$.

A model of an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A}\rangle$ is an interpretation that satisfies all assertions in $\mathcal{T}$ and $\mathcal{A}$. An ontology $\mathcal{O}$ is satisfiable if it has at least one model, and it logically implies an assertion $\alpha$, written $\mathcal{O} \models \alpha$, if all models of $\mathcal{O}$ satisfy $\alpha$.

Next we introduce queries, whose answers, as usual in ontologies, are formed by constants denoting individuals explicitly mentioned in the ABox. A union of conjunctive queries (UCQ) $q$ over an ontology $\langle \mathcal{T}, \mathcal{A}\rangle$ is a FOL formula of the form $\bigwedge_{i=1}^n \exists y_i, \text{conj}_i(x, y_i)$ with free variables $x$ and existentially quantified variables $y_1, \ldots, y_n$. Each $\text{conj}_i(x, y_i)$ in $q$ is a conjunction of atoms of the form $N(z)$, $P(z)$, where $N$ and $P$ respectively denote a concept and a role name occurring in $\mathcal{T}$, and $z$, $y_i$ are constants of $\mathcal{A}$ or variables in $x$, $y_1, \ldots, y_n$. The (certain) answers to $q$ over $\langle \mathcal{T}, \mathcal{A}\rangle$ is the set $\text{ANS}(q, \mathcal{T}, \mathcal{A})$ of substitutions $\theta$ of the free variables of $q$ with constants in $\mathcal{A}$ such that $q\theta$ evaluates to true in every model of $\langle \mathcal{T}, \mathcal{A}\rangle$, denoted $\langle \mathcal{T}, \mathcal{A}\rangle \models q\theta$. DLR-Lite enjoys nice computational properties, in particular w.r.t. query evaluation: computing the certain answers to a UCQ can be done in polynomial time in the size of $\langle \mathcal{T}, \mathcal{A}\rangle$, and in $\text{AC}^0$ in the size of $\mathcal{A}$ alone (i.e., in data complexity) [5]. Such result is based on the FOL rewritability property of DLR-Lite [5], which states that for every UCQ $q$ and TBox $\mathcal{T}$, we can rewrite $q$ into a new UCQ $\text{rew}_\mathcal{T}(q)$ such that $\text{ANS}(q, \mathcal{T}, \mathcal{A}) = \text{ANS}(\text{rew}_\mathcal{T}(q), \emptyset, \mathcal{A})$, for every ABox $\mathcal{A}$. In other words, the TBox can be “compiled away”.

We also consider ECQs, which are FOL queries whose atoms are UCQs evaluated according to the certain answer semantics above [4]. An ECQ over $\mathcal{T}$ and $\mathcal{A}$ is a possibly open formula of the form (where $q$ is a UCQ):
\[
Q \rightarrow [q] \mid \neg Q \mid Q_1 \land Q_2 \mid \exists x.Q
\]
The (certain) answers to $Q$ over $\langle \mathcal{T}, \mathcal{A}\rangle$, is the set of substitutions $\theta$ of the free variables of $Q$ with constants in $\mathcal{A}$ defined by composing the certain answers of the UCQs $q$ in $Q$.
through first-order constructs, and interpreting existential variables as ranging over the constants in $\mathcal{A}$. Hence, the first-order constructs in ECQs are interpreted under a (weaker) epistemic semantics. ECQs over DLR-Lite ontologies enjoy the same computational properties as UCQs, in particular FOL rewritability of query answering [4].

3 Framework

We introduce now our framework for modeling DACMASs. Formally, a DACMAS is a tuple $\langle T, \mathcal{E}, \mathcal{X}, \mathcal{I}, \mathcal{C}, B \rangle$, where: (i) $T$ is a global DLR-Lite TBox; (ii) $\mathcal{E}$ is a set of predicates denoting events (where the predicate name is the event type, and the arity determines the content/payload of the event); (iii) $\mathcal{X}$ is a finite set of agent specifications; (iv) $\mathcal{I}$ is a (partial) specification for the institutional agent; (v) $\mathcal{C}$ is a contractual specification; (vi) and $B$ is a Commitment Box (CBox).

3.1 The Global TBox

The global TBox represents the key concepts, relations and constraints characterizing the domain in which the agents operate, so as to provide a common ground for the agent interaction. Part of this TBox is fixed for every agent system, and is used to model core notions related to the system itself. The extension of such core notions is maintained by a single, special institutional agent, which is also responsible for the manipulation of commitments (cf. Section 3.6). The data maintained by such an agent are publicly available and can be queried by the other agents, but only modified by the institutional agent. Specifically, the institutional agent maintains data about the following relations:

- Agent denotes the set of (names of) agents that currently participate to the system. Since the institutional agent is always part of the system, we fix its name as inst, and enforce that inst always belongs to the extension of Agent.
- Spec, whose extension is immutable, denotes the set of agent specification names mentioned in $\mathcal{X}$ (cf. Section 3.2).
- hasSpec connects agents with their current specification(s): $\text{hasSpec}[1] \sqsubseteq \text{Agent}$, $\text{hasSpec}[2] \sqsubseteq \text{Spec}$.

Each agent, including the institutional agent, maintains a proprietary DLR-Lite ABox, in which it stores its own data. Such data can be queried only by the agent itself and by the institutional agent, which exploits the results of such queries to keep track of the evolution of commitments. Furthermore, each agent progresses its own ABox during the execution in such a way that it is always consistent with the global TBox $T$. Notice that the overall collection of ABoxes is not assumed to be consistent with the TBox, i.e., the TBox assertions are only required to be satisfied by each agent individually.

Since, in general, queries may involve the ABoxes of several agents, to disambiguate to which ABox a query atom refers, we augment the vocabulary of the TBox with a location argument that points to an agent. We use $R^{a}(x)$ to denote an atomic query returning the extension of $R$ in the ABox of $a$. If $a$ does not point to an agent currently in the system, then $R^{a}(x)$ evaluates to empty. Beside the special constant inst, we also use self to implicitly refer to the agent that is posing the query (similarly to “this” in object-oriented programming). When clear from the context, we omit $^{a}$ and just use relations without the location argument. We denote with $UCQ_{\ell}$ (resp., $ECQ_{\ell}$) the language obtained from UCQ (resp., ECQ) by extending atoms with a location argument.
3.2 Agent Specifications

In a DACMAS, agents interact by exchanging messages. A message is sent by a sender agent to a receiver agent, and is about the occurrence of an event with a payload, containing data to be communicated. All agents but the institutional one are only aware of the events they send and receive. As for data, the institutional agent has instead full visibility of all exchanged messages, so as to properly handle the evolution of commitments. Agents determine the events they may send, and also how they react to events, through proactive and reactive rules. Such rules are grouped into behavioural profiles called agent specifications, and model: (i) the possible, proactive emission of an event, directed to another agent (communicative rule); (ii) conditional internal (re)actions, which lead to update the agent ABox when sending/receiving an event to/from another agent (update rule). The update could result in the insertion of new data items (from the countably infinite domain $\Delta$), not already present in the system.

The exchange of a message represents a synchronization point among the sender, receiver and institutional agent. Hence, the reaction of the three agents is interpreted as a sort of transaction, such that each of them effectively enforces the update on its own ABox only if each of the three resulting ABoxes is consistent with $\mathcal{T}$. An inconsistency could potentially arise when reacting to an event either because the same data item is asserted to be member of two disjoint classes, or because a key assertion is violated.

Formally, an agent specification is a tuple $\langle sn, \Pi \rangle$, where $sn$ is the specification name, and $\Pi$ is a set of communicative and update rules. Such rules are defined over the vocabulary of $\mathcal{T}$ and $B$, and are applied over the ABoxes of the agent and of inst. This allows the agent to query the status of commitments and obtain the names of the other participants. A communicative rule has the form

$$Q(r, x) \text{ enables } EV(x) \text{ to } r$$

where $Q$ is an $ECQ_r$, and $EV(x)$ is an event supported by the system, i.e., predicate $EV/|x|$ belongs to $\mathcal{E}$. The semantics of a communicative rule is as follows. Whenever $Q(r, x)$ evaluates positively, the agent autonomously selects one of the answers $\theta$ returned by the query, using it to determine the event receiver and its payload. This states that the ground event $EV(x)\theta$ can be sent by the agent to $r\theta$, provided that $r\theta$ points to an actual agent name in the system (including the two special names inst and self).

Example 1. Consider a DACMAS where customers and sellers interact to exchange goods. We model the behavioural rules for customers and sellers using two agent specifications. To buy from a seller, a customer must register to that seller. A registration request is modeled in the customer specification as:

$$\text{Spec}@\text{inst}(sel, seller) \text{ enables } \text{REQ_REG to } sel$$

Assuming that each seller maintains its customers and items respectively in relations MyCust and Item, the proposal of an item to a customer is modeled in the seller specification as: $\text{MyCust}(m) \land \text{Item}(i) \text{ enables } \text{PROPOSE}(i) \text{ to } m$.

Update rules are ECA-like rules of the form:

- on $EV(x)$ to $r$ if $Q(r, x)$ then $\alpha(r, x)$ (on-send)
- on $EV(x)$ from $s$ if $Q(s, x)$ then $\alpha(s, x)$ (on-receive)
where $\text{EV}/|x|$ is an event type from $\mathcal{E}$, $Q$ is an $\text{ECQ}_\ell$, and $\alpha$ is an update action with parameters (described below). Each such rule triggers when an event is sent to/received from another agent, and $Q$ holds. This results in the application of $\alpha$ using the actual event payload and receiver/sender. Action $\alpha$ queries the ABox of the agent and of $\text{inst}$, using the answers to add and remove facts to the ABox.

Formally, an update action is an expression $\alpha(p) : \{e_1, \ldots, e_n\}$, where $\alpha(p)$ is the action signature (constituted by the name $\alpha$ and by a list $p$ of parameters), and $\{e_1, \ldots, e_n\}$ are update effects, each of which has the form

$$[q^+(p, x)] \land Q^-(p, x) \rightarrow \text{add } A, \text{del } D$$

- $q^+$ is an $\text{UCQ}_\ell$, and $Q^-$ is an $\text{ECQ}_\ell$ whose free variables occur all among those of $q^+$; intuitively, $q^+$ selects a set of tuples from the agent ABox and that of $\text{inst}$, while $Q^-$ filters away some of them.\(^3\) During the execution, the effect is applied with a ground substitution $\theta$ for the action parameters, and for every answer $\theta$ to the query $[q^+(d, x)] \land Q^-(d, x)$.

- $A$ is a set of facts (over the alphabet of $T$ and $B$) which include as terms: free variables $x$ of $q^+$, action parameters $p$ and/or Skolem terms $f(x, p)$. We use $\text{SKOLEM}(A)$ to denote all Skolem terms mentioned in $A$. At runtime, whenever a ground Skolem term is produced by applying $\theta$ to $A$, the agent autonomously substitutes it with a possibly new data item taken from $\Delta$. This mechanism is exploited by the agent to inject new data into the system. The ground set of facts so obtained is added by the agent to its ABox.

- $D$ is a set of facts which include as terms free variables $x$ of $q^+$ and action parameters $p$. At runtime, whenever a ground fact in $D$ is obtained by applying $\theta$, it is removed from the agent ABox.

As in STRIPS, we assume that additions have priority over deletions (i.e., if the same fact is asserted to be added and deleted during the same execution step, then the fact is added). The “add $A$” (resp. “del $D$”) part can be omitted if $A = \emptyset$ (resp., if $D = \emptyset$).

**Example 2.** Consider three possible reaction rules for the seller. The fact that the seller makes every agent that sends a request become one of its customers is modeled as:

```plaintext
on ASK_REG from c if true then makeCust(c)
```

where $\text{makeCust}(x) : \{\{\text{true}\} \rightarrow \text{add}\{\text{MyCust}(x)\}\}$. Assume now that the seller maintains the item cart for a customer, using relation $\text{InCart}(i, c)$ to model that item $i$ is in the cart of $c$. The seller reaction to an “empty cart” request is modeled as:

```plaintext
on EMPTY_CART_REQ from c if MyCust(c) then doEmpty(c)
```

where $\text{doEmpty}(c) : \{\{\text{InCart}(i, c)\} \rightarrow \text{del}\{\text{InCart}(i, c)\}\}$. Note that the effect is applied to each $i$ in the cart of $c$. Consider now the case where the seller receives a new item $i$ to be sold. It reacts by adding $i$ and deciding its price (denoted with Skolem $p(i)$):

```plaintext
on NEW_ITEM(i) from a if true then addItem(i)
```

where $\text{addItem}(i) : \{\{\text{true}\} \rightarrow \text{add}\{\text{Item}(i), \text{Price}(i, p(i))\}\}$.

\(^3\)The distinction between $q^+$ and $Q^-$ is needed for technical reasons, borrowed from [2].
3.3 Institutional Agent Specification

The institutional agent inst manages the core information of the DACMAS. Its behaviour is (partially) captured by the institutional agent specification \( \mathcal{I} \), which differs from the other agent specifications in two respects. First, since inst is aware of all messages exchanged by the other agents and can query their ABoxes, its specification is not only constituted by communicative rules and on-send/on-receive reactive rules, but also by on-exchange rules of the form:

\[
\text{on } \text{EV}(x) \text{ from } s \text{ to } r \text{ if } Q(s, r, x) \text{ then } \alpha(s, r, x)
\]

where \( Q \) and \( \alpha \) can query the internal ABox of the institutional agent, and the ABoxes of \( s \) and \( r \). To conveniently specify reactions of inst that do not depend on a specific event, but trigger whenever an event is exchanged, we use:

\[
\text{on any event from } s \text{ to } r \text{ if } Q(s, r) \text{ then } \alpha(s, r)
\]

Second, \( \mathcal{I} \) is only a partial specification for inst. In fact, inst is also responsible for the manipulation of commitments, which results in a set of additional on-exchange rules that, starting from the contractual specification (cf. Section 3.4), encode the commitment machines for the commitments involved in the contract. These rules are automatically extracted from the contractual specification (cf. Section 3.6).

Example 3. Consider a portion of institutional agent specification, modeling the creation of a new agent whenever inst receives a request (whose payload denotes the specification to be initially followed by that agent). To handle this request, inst uses relation NewA to store a newly created agent together with its initial specification. The axiom \( \text{NewA}[1] \models \neg \text{Agent} \) is part of \( \mathcal{T} \), and enforces that a new agent has indeed a new name. The behaviour is defined in two steps. In the first step, inst reacts to a creation request by choosing an agent name (using Skolem term \( n() \)). The reaction is applied only if there is no pending new agent to be processed.

\[
\text{on AG.REQ}(s) \text{ from } a \text{ if } \neg (\exists x \exists y. \text{NewA}(x, y)) \text{ then } create(s)
\]

\[
\text{create}(s) : \{ \text{true} \leadsto \text{add}\{\text{NewA}(n(), s)\} \}
\]

Note that axiom \( \text{NewA}[1] \models \neg \text{Agent} \) ensures that the update is blocked if the chosen name is already used in the system.

In the second step, inst informs itself that a new agent has to be processed; the corresponding reaction finalizes the insertion of the new agent, moving it to the set of participating agents:

\[
\text{NewA}(a, s) \text{ enables } \text{INSERT.AG}(a, s) \text{ to self}
\]

\[
\text{on INSERT.AG}(a, s) \text{ from self if } \text{true then } do_{\text{ins}}(a, s)
\]

\[
\text{do}_{\text{ins}}(a, s) : \{ \text{true} \leadsto \text{add}\{\text{Agent}(a), \text{Spec}(a, s)\}, \text{del}\{\text{NewA}(a, s)\} \}
\]

3.4 Contractual Specification

The contractual specification \( C \) consists of a set of commitment rules, which are reactive rules similar to on-exchange rules. The main difference is that, instead of actions, they
describe (first-order) conditional commitments and their creation:

\[
\text{on } \text{EV}(x) \text{ from } s \text{ to } r \text{ if } Q_c(s, r, x) \quad (\ast)
\]

then \( CC_n(s, r, [q^+(s, r, x, y)] \land Q^-_p(s, r, x, y), Q_d(s, r, x, y)) \)

where \( n \) is the commitment name, the ECQ\(_t\) \( Q_c \) is the condition for the creation of the conditional commitment, \( [q^+_p] \land Q^-_p \) (where, as in update effects, \( q^+_p \) is a UCQ\(_t\), and \( Q^-_p \) is an ECQ\(_t\) whose free variables all occur among those of \( q^+_p \) ) is the precondition determining the generation of a corresponding base-level commitment, and the ECQ\(_t\) \( Q_d \) is the discharge condition for such base-level commitment. All the aforementioned queries can be posed over the ABoxes of \( s, r, \) and \( \text{inst} \). We use GET-CC(\( C \)) to extract the set of conditional commitments contained in \( C \).

According to the literature, commitments are manipulated either explicitly via specific events (such as a commitment cancellation or delegation), or implicitly when the commitment precondition or discharge condition becomes true. The allowed commitment manipulations, together with the resulting commitment states, are captured by means of a commitment machine [12]. In this work, we consider a simple commitment machine, inspired by [12,13], and show how to lift it to a first-order setting, taking into account that in our framework the precondition and the discharge condition are specified through queries over the data of the involved agents. More elaborated commitment machines, in terms of events and states, can be seamlessly incorporated.

Specifically, every commitment in GET-CC(\( C \)) is associated to a specific first-order commitment machine, which is activated using the corresponding commitment rule in \( C \) of the form above, instantiated possibly multiple times, depending on the agent data. The machine evolves as follows:

1. When an event of type \( \text{EV} \) is sent by agent \( a \) to agent \( b \) with payload \( d \), if \( Q_c(a, b, d) \) is satisfied, an instance of the conditional commitment \( n \) is created. The debtor, creditor, and payload of this instance are respectively \( a, b, \) and \( d \).
2. Such instance is explicitly or implicitly manipulated by the involved agents. Explicit manipulation is done via specific message exchanges; we consider in particular the case of delegation from the debtor \( a \) to a new debtor, and the case of cancellation. Implicitly, instead, the instance can generate one or more corresponding base-level commitment instances; whenever \( [q^+_p(a, b, d, y)] \land Q^-_p(a, b, d, y) \) is satisfied with actual values \( y \) for variables \( y \), the conditional commitment instance creates a base-level commitment instance with payload \( d \) and \( y \). Such base-level instance is put into the active state. The discharge condition for this instance is the instantiation of \( Q_d \) with the involved agents and specific payload, i.e., \( a \) is committed to bring about \( Q_d(a, b, d, y) \).
3. Also a base-level commitment instance is explicitly and implicitly manipulated by the involved agents. Explicit manipulation of an active base-level instance resembles that of conditional commitment instances, with the difference that, when canceled, a base-level commitment instance enters into the violated state. Implicit manipulation determines instead the discharge of the instance as soon as \( Q_d(a, b, d, y) \) holds, moving the instance from active to satisfied.

Example 4. Consider a commitment rule establishing a conditional commitment that the seller takes whenever it accepts the registration of a customer \( c \). The conditional commitment is about the delivery of items paid by \( c \). Specifically, for each item sold by the seller, if \( c \) has paid that item, then the seller commits to ensure that \( c \) will hold that
item. Note that the two conditions are correlated by the same item, and that a base-level commitment is created for each paid item. This cannot be expressed in propositional logic. Assuming that the seller stores a fact Paid\((i, c)\) if \(c\) has paid for \(i\), and that the customer stores a fact Owned\((i)\) whenever it owns \(i\), the commitment rule can be specified as:

\[
\text{on ACCEPT_REG from } s \text{ to } c \text{ if } \text{MyCust}@s(c) \\
\text{then } \text{CCDelivery}(s, c, [\text{Item}@s(i) \land \text{Paid}@s(i, c), \text{Owns}@s(i) ])
\]

Note the use of location arguments, reflecting that payments are maintained by the seller, whereas the items owned by the customer are maintained by the customer itself.

### 3.5 Commitment Box

The commitment box \(B\) is a set of relations used by \(\text{inst}\) to maintain the concrete instances of conditional commitments, and the instances of their corresponding base-level commitments (with their states). In fact, due to the presence of data, commitments do not only require to keep track of the involved agents, but also of the payload associated to each of their instances. Such relations are extracted from the contractual specification as follows. Each commitment \(CC_n(s, r, [q_n(s, r, x, y)] \land Q^P_n(s, r, x, y), Q^Q_n(s, r, x, y))\) in GET-CC\((C)\) induces two relations in \(B\), on the basis of the commitment name \(n\) and the payloads \(x\) and \(y\): (i) \(nCC/ar\), where \(ar = 2 + |x|\) for debtor, creditor, and conditional commitment payload; (ii) \(nC/ar\), where \(ar = 3 + |x| + |y|\) for debtor, creditor, state, and base-level commitment payload.

**Example 5.** The commitment in Example 4 induces the following relations in \(B\): \(\text{DeliveryCC}(\text{debtor, creditor})\) and \(\text{DeliveryC}(\text{debtor, creditor, state, item})\).

### 3.6 Commitment Machine Formalization

As anticipated in Section 3.3, the specification of \(\text{inst}\) must be complemented with a set of additional on-exchange rules, used to properly manipulate the evolution of commitments as the interaction unfolds. Commitment instances are stored by \(\text{inst}\) using the vocabulary of the CBox \(B\), and evolved through the application of these rules. Specifically, these rules ground the (first-order) commitment machine described in Section 3.4 to each specific commitment of GET-CC\((C)\), according to the “templates” described in the remainder of this section. We denote with \(\text{cc-rules}(C)\) all the commitment manipulation rules produced from \(C\).

When discussing the templates, we refer to a commitment rule \(\rho \in C\) of the form \((*)\) in Section 3.4. Notice that, when \(n\), \(x\) and \(y\) are mentioned in the rule templates, they are meant to be replaced with the actual commitment name and payload variables.

**CC creation.** For each \(\rho \in C\), a corresponding creation rule is obtained, depending on \(n\) and \(x\). When the rule triggers, a new instance of the conditional commitment \(nCC\) is created, with the actual agents and payload:

\[
\text{on EV}(x) \text{ from } s \text{ to } r \text{ if } Q_n(s, r, x) \text{ then } \text{create}_nCC(s, r, x) \\
\text{create}_nCC(s, r, x) : \{ \text{true} \rightarrow \text{add}(nCC(s, r, x)) \}
\]

**CC delegation.** The delegation of a conditional commitment instance for commitment \(n\) is triggered when the old debtor \(d_o\) sends to the new debtor \(d_n\) a DELEGATE\(_nCC\)}
event, specifying in the event payload the creditor and the payload of the instance to be delegated. If such an instance exists, the debtor is updated by inst:

```plaintext
on DELEGATE \_nCC(c, x) from d_o to d_n if \_nCC(d_o, c, x) then changedeb \_nCC(d_o, d_n, c, x)
```

`changedeb \_nCC(d_o, d_n, c, x) : \{ [true] \rightarrow add\{nCC(d_n, c, x)\}, del\{nCC(d_o, c, x)\}\}`

**CC cancelation.** The cancelation of a conditional commitment instance for commitment \_n is triggered when the debtor sends to the creditor a `CANCEL \_nCC` event, providing the instance payload. If the instance exists, it is removed:

```plaintext
on CANCEL \_nCC(x) from d to c if \_nCC(d, c, x) then delete \_nCC(d, c, x)
```

`delete \_nCC(d, c, x) : \{ [true] \rightarrow del\{nCC(d,c, x)\}\}`

**C creation.** Every conditional commitment instance for relation \_nCC creates a base-level commitment instance whenever the precondition (whose variables x are grounded with the instance payload) holds with an answer substitution \(\theta\) for variables y. This results in the creation of a new tuple for relation \_nC with the actual, full payload. This does not depend on the specific exchanged event, but only on the actual configuration of the data. Hence, a single “any-event” rule can be used to manage the creation of all base-level instances at once:

```plaintext
on any event from d to c if true then createC(d, c)
```

where, for each commitment \_CC_n(s, r, [q_p^+(s, r, x, y)] \wedge Q_p^-(s, r, x, y), Q_n(s, r, x, y))

in GET-CC(C), action `createC(d, c)` contains the following detachment effect:

\[ [nCC(d, c, x) \wedge q_p^+(d, c, x, y)] \wedge Q_p^-(d, c, x, y) \rightarrow add\{nC(d, c, active, x, y)\} \]

Differently from the propositional formalization of a commitment machine, in which the conditional commitment detaches to a base-level one, in our setting the conditional commitment instance is maintained, and keeps waiting for other situations matching the precondition with different data.

**C delegation.** It resembles the CC delegation:

```plaintext
on DELEGATE \_nC(c, x, y) from d_o to d_n
if \_nC(d_o, c, active, x, y) then changedeb \_nC(d_o, d_n, c, x)
```

`changedeb \_nC(d_o, d_n, c, x, y) : \{ [true] \rightarrow add\{nC(d_n, c, active, x, y)\}, del\{nC(d_o, c, active, x, y)\}\}`

**C cancelation.** It determines a transition for the base-level commitment instance from the active to the violated state:

```plaintext
on CANCEL_C(x, y) from d to c if \_nC(d, c, x, y) then viol \_nC(d, c, x, y)
```

`viol \_nC(d, c, x, y) : \{ [true] \rightarrow add\{nC(d, c, viol, x, y)\}, del\{nC(d, c, active, x, y)\}\}`

**C discharge.** Similarly to the case of C creation, the discharge of base-level commitment instances is handled by a single “any-event” rule, which checks the discharge condition
for each active commitment instance with the actual payload, evolving the instance to the satisfied state if it holds:

\[ \text{on any event from } d \text{ to } c \text{ if } true \text{ then } dischargeC(d, c) \]

where, for each \( cc_n(s, r, [q_p^p(s, r, x, y)] \land Q_p^p(s, r, x, y), Q_d(s, r, x, y)) \) in \( cc(C) \), action \( dischargeC(d, c) \) contains:

\[ [nC(d, c, \text{active}, x, y)] \land Q_d(d, c, x, y) \]
\[ \leadsto \text{add}\{\text{nC}(d, c, \text{sat}, x, y)\}, \text{del}\{\text{nC}(d, c, \text{active}, x, y)\} \]

C removal. A last “any-event” reactive rule is used by \( inst \) to remove those instances of base-level commitments that already achieved a final state (sat or viol):

\[ \text{on any event from } a \text{ to } b \text{ if } true \text{ then } removeFinal() \]

where, for each base-level commitment relation \( nC \) in \( B \), action \( removeFinal() \) contains:

\[ [nC(d, c, s, x, y)] \land (s = \text{sat} \lor s = \text{viol}) \leadsto \text{del}\{nC(d, c, s, x, y)\} \]

**Example 6.** Assume that the only rule in \( C \) is that of Example 4. The following CC creation rule is produced

\[ \text{on } \text{ACCEPT}_\text{REG} \text{ from } s \text{ to } c \text{ if MyCust}@s(c) \text{ then } create_{DeliveryCC}(s, c) \]
\[ create_{DeliveryCC}(s, c) : \{[true] \leadsto \text{add}\{\text{DeliveryCC}(s, c)\}\} \]

Furthermore, the following C creation and C discharge update actions are produced:

\[ createC(d, c) : \{[\text{DeliveryCC}(d, c) \land \text{Item}@d(i) \land \text{Paid}@d(i, c)] \]
\[ \leadsto \text{add}\{\text{DeliveryC}(d, c, \text{active}, i)\}\} \]
\[ dischargeC(d, c) : \{[\text{DeliveryC}(d, c, \text{active}, i)] \land \text{Owns}@c(i) \]
\[ \leadsto \text{add}\{\text{DeliveryC}(d, c, \text{sat}, i)\}, \text{del}\{\text{DeliveryC}(d, c, \text{active}, i)\}\} \]

### 4 Execution Semantics

The execution semantics of a DACMAS is defined in terms of a transition system that, starting from a given initial state, accounts for all the possible system dynamics, considering all the (possibly infinite) sequences of message exchanges, and all the possible substitutions that the agents choose during the application of update actions to provide concrete values for the Skolem terms. Given a DACMAS \( S = (T, \mathcal{E}, \mathcal{X}, \mathcal{I}, C, B) \) and an initial state \( \sigma_0 \), the execution semantics of \( S \) over \( \sigma_0 \) is defined by a transition system \( T_S^{\sigma_0} = (\Delta, T \cup \mathcal{B}, \Sigma, \sigma_0, \Rightarrow) \), where:

- \( \Sigma \) is a (possibly infinite) set of states. Each state \( \sigma \in \Sigma \) is equipped with a function \( \text{abox} \) that, given the name \( a \) of an agent, returns the ABox \( \sigma\text{.abox}(a) \) of \( a \) in \( \sigma \), if and only if \( a \) participates to the system in state \( \sigma \), i.e., a belongs to the extension Agent in \( \sigma\text{.abox}(\text{inst}) \). Hence, \( \sigma\text{.abox}(\text{inst}) \) is always defined.
- \( \sigma_0 \in \Sigma \) is the initial state. We assume that every ABox \( \mathcal{A} \) in \( \sigma_0 \) is satisfiable, and that \( \text{Spec}(sn) \in \sigma_0\text{.abox}(\text{inst}) \) if and only if \( (sn, \_ ) \in \mathcal{X} \).
- \( \Rightarrow \subseteq \Sigma \times \Sigma \) is a transition relation.
Instrumental to the definition of the transition system is the extension of answering \( \text{ECQ}_\ell \) queries so as to take into account location arguments. Formally, given a TBox \( T \), we define that \( R@b(x) \) holds in state \( \sigma \) from the perspective of agent \( a \) under substitution \( \theta \) for \( x \), written \( T, \sigma, a, \theta \models R@b(x) \), if:\(^4\)

\[
\begin{align*}
\{ a \cdot \text{abox}(a) \} & \text{ is defined and } (T, \sigma, a \cdot \text{abox}(a)) \models R(x)\theta, \text{ if } b\theta = \text{self} \\
\{ a \cdot \text{abox}(b) \} & \text{ is defined and } (T, \sigma, a \cdot \text{abox}(b)) \models R(x)\theta, \text{ if } b\theta \neq \text{self}
\end{align*}
\]

Note that the semantics supports a sort of dynamic binding of location arguments, using \( \theta \) to substitute a variable location argument with an agent name. This relation extends in the natural way to \( \text{UCQ}_\ell \) and \( \text{ECQ}_\ell \), considering that quantification ranges over the active domain \( \text{ADOM}(\sigma) \) of \( \sigma \), which is defined as the union of the active domains of the ABoxes maintained by the agents present in \( \sigma \). This, in turn, allows us to define the certain answers to \( Q \) obtained by agent \( a \) in state \( \sigma \), denoted \( \text{ANS}_\ell(Q, T, \sigma, a) \), as the set of substitutions \( \theta \) for the free variables in \( Q \) such that \( Q \) holds in state \( \sigma \) from the perspective of \( a \), i.e., \( \text{ANS}_\ell(Q, T, \sigma, a) = \{ \theta \mid T, \sigma, a, \theta \models Q \} \).

The construction of the transition system \( \Upsilon_0 \) is given in Figure 1.

## 5 Verification of DACMAS

To specify dynamic properties over DACMASs, we use a first-order variant of \( \mu \)-calculus [14,6]. \( \mu \)-calculus is virtually the most powerful temporal logic used for model checking of finite-state transition systems, and is able to express both linear time logics such as LTL and PSL, and branching time logics such as CTL and CTL* [9]. In our variant of \( \mu \)-calculus, local properties are expressed as \( \text{ECQ}_\ell \) queries over the current state of the DACMAS. We allow for a controlled form of first-order quantification across states, inspired by [2], where the quantification ranges over data items across time only as long as such items persist in the active domain. Formally, we define the logic \( \mu \mathcal{L}_{ECQ}^p \) as:

\[
\Phi ::= Q_\ell \mid \neg \Phi \mid \Phi_1 \land \Phi_2 \mid \exists x. \text{LIVE}(x) \land \Phi \mid \text{LIVE}(x) \land (\neg)\Phi \mid \text{LIVE}(x) \land (\neg\neg)\Phi \mid Z \mid \mu Z.\Phi
\]

where \( Q \) is a (possibly open) \( \text{ECQ}_\ell \) query, in which the only constants that may appear are those in the initial state of the system, \( Z \) is a second order predicate variable (of arity 0), and \( \text{LIVE}(x_1, \ldots, x_n) \) abbreviates \( \bigwedge_{i \in \{1, \ldots, n\}} \text{LIVE}(x_i) \). For \( \mu \mathcal{L}_{ECQ}^p \), the following assumption holds: if \( \text{LIVE}(x) \land (\neg\neg)\Phi \) and \( \text{LIVE}(x) \land (\neg)\Phi \), the variables \( x \) are exactly the free variables of \( \Phi \), once we substitute to each bounded predicate variable \( Z \) in \( \Phi \) its bounding formula \( \mu Z.\Phi \). We adopt the usual abbreviations, including \( \nu Z.\Phi \) for greatest fixpoints. Intuitively, the use of \( \text{LIVE}(\cdot) \) in \( \mu \mathcal{L}_{ECQ}^p \) ensures that data items are only considered if they persist along the system evolution, while the evaluation of a formula with data that are not present in the current state trivially leads to false or true. This is in line with DACMASs, where the evolution of a commitment instance persists until the commitment is discharged or canceled, and where an agent name is meaningful only while it persists in the system: when an agent leaves the system and its name \( a \) is canceled by \( \text{inst} \), \( \text{inst} \) could reuse \( a \) in the future to identify another agent.

The formula \( \mu Z.\Phi \) denotes the least fixpoint of the formula \( \Phi \). As usual in \( \mu \)-calculus, formulae of the form \( \mu Z.\Phi \) must obey to the syntactic monotonicity of \( \Phi \) w.r.t. \( Z \), which

\(^4\)We assume that \( \theta \) is the identity on data items (including the special constants \( \text{self} \) and \( \text{inst} \)).
1: procedure BUILD-TS
2: input: DACMAS $S = \langle T, E, \mathcal{X}, \mathcal{I}, \mathcal{C}, \mathcal{B} \rangle$ and initial state $\sigma_0$
3: output: Transition system $\langle \Delta, T \cup B, \Sigma, \sigma_0, \Rightarrow \rangle$
4: $\Sigma := \{ \sigma_0 \}$, $\Rightarrow := \emptyset$
5: while true do
6:      pick $\sigma \in \Sigma$ and $a \in \{ \text{ag} | \text{Agent}(\text{ag}) \in \sigma.abox(\text{inst}) \}$
7:    Fetch all current behavioural rules for $a$; Calculate enabled events with receivers for $a$
8:    if There exists at least an enabled event then
9:       pick an enabled event $\text{EV}(e)$ for $a$ with receiver $b$
10:       $A_i := \text{APPLY}(S, \sigma, \text{inst}, a, b, \text{EV}(e))$ \hspace{1cm} \triangleright \text{New inst ABox}
11:       $\Sigma := \Sigma \cup \{ \sigma' \}$ \hspace{1cm} \triangleright \text{Tentatively add a new state } \sigma'$
12:       for all $x \in \{ \text{ag} | \text{Agent}(\text{ag}) \in A_i \}$ do
13:           $\sigma'.abox(x) := \text{APPLY}(S, \sigma, x, a, b, \text{EV}(e))$
14:       if for every $x \in \{ \text{ag} | \text{Agent}(\text{ag}) \in A_i \}$, $\langle T, \sigma'.abox(x) \rangle$ is satisfiable then
15:       $\Rightarrow := \Rightarrow \cup \langle \sigma, \sigma' \rangle$
16:       else $\Sigma := \Sigma \setminus \{ \sigma' \}$ \hspace{1cm} \triangleright \text{Inconsistent execution step}
17: function $\text{APPLY}(S, \sigma, x, a, b, \text{EV}(e))$
18: output: new ABox for $x$ after reacting to $\text{EV}(e)$ from $a$ to $b$
19: if $x \not\in \{ \text{inst}, a, b \}$ then return $\sigma.abox(x)$
20: Fetch all current behavioural rules for $x$
21: if $x = a$ then \hspace{1cm} \triangleright $x$ is the sender agent
22: Fetch on-send and “self” on-receive rules and compute actions with actual params
23: if $x = b$ then \hspace{1cm} \triangleright $x$ is the receiver agent
24: Fetch on-receive and “self” on-send rules and compute actions with actual params
25: if $x = \text{inst}$ then \hspace{1cm} \triangleright $x$ is the institutional agent
26: Fetch matching “any-event” on-exchange rules and compute actions with actual param
27: $\text{TOADD} := \emptyset$, $\text{TODEL} := \emptyset$
28: for all $\alpha(v) \in \text{ACT}$ do \hspace{1cm} $\triangleright$ ACT = set of fetched actions
29: $\text{TOADDSK} := \emptyset$
30: for all effect “$q^\rightarrow(p, x) \land Q^\leftarrow(p, x) \leftarrow \text{add } A$, del $D$” in the definition of $\alpha$ do
31: for all $\theta \in \text{ANS}_x([q^\rightarrow(v, x)] \land Q^\leftarrow(v, x); T, \sigma, x)$ do
32: $\text{TOADDSK} := \text{TOADDSK} \cup \theta[p/v]$
33: $\text{TODEL} := \text{TODEL} \cup \theta[p/v]$
34: pick a substitution $\theta_{sk}$ of the Skolem terms with data
35: $\text{TOADD} := \text{TOADD} \cup \text{TOADDSK}\theta_{sk}$
36: if $x = \text{inst}$ then $\text{TOADD} := \text{TOADD} \cup \{ \text{Agent}(\text{inst}) \}$
37: return $(\sigma.abox(x) \setminus \text{TODEL}) \cup \text{TOADD}$

Fig. 1. Transition system construction

states that every occurrence of the variable $Z$ in $\Phi$ must be within the scope of an even number of negation symbols. This ensures that the least fixpoint $\mu Z.\Phi$ always exists.

The semantics of $\mu\mathcal{L}_p^{\text{ECQ}_I}$ formulae is defined over a possibly infinite transition system $\mathcal{T} = \langle \Delta, T \cup B, \Sigma, \sigma_0, \Rightarrow \rangle$ (cf. Section 4), assuming that $\text{ECQ}_I$ queries are posed from the point of view of inst. This does not prevent the possibility to query the ABoxes of the other agents, thanks to the dynamic binding for location arguments. Since $\mu\mathcal{L}_p^{\text{ECQ}_I}$ contains formulae with both individual and predicate free variables, we introduce an individual variable valuation $v$, i.e., a mapping from individual variables $x$ to $\Delta$, and a predicate variable valuation $V$, i.e., a mapping from the predicate variables $Z$ to subsets of $\Sigma$. With these three notions in place, we assign meaning to formulae by
associating to \( T, v, \) and \( V \) an extension function \( (\cdot)^{T}_{v,V} \), which maps formulae to subsets of \( \Sigma \). Formally, the extension function \( (\cdot)^{T}_{v,V} \) is defined inductively as shown in Figure 2. When \( \Phi \) is a closed formula, \( (\Phi)^{T}_{v,V} \) does not depend on \( v \) or \( V \), and we denote the extension of \( \Phi \) simply by \( (\Phi)^{T} \). A closed formula \( \Phi \) holds in a state \( s \in \Sigma \) if \( s \in (\Phi)^{T} \).

In this case, we write \( T, s \models \Phi \). Given DACMAS \( S \), an initial state \( \sigma_0 \) and a \( \mu L^{ECQ}_{p} \) formula \( \Phi \), we are interested in the following verification problem: \( T_{S}^{\sigma_0}, \sigma_0 \models \Phi \).

**Example 7.** Consider the contract of Example 4. Assume that \( T \) contains that gold customers are seller customers: MyGoldCust \( \sqsubseteq \) MyCust. The \( \mu L^{ECQ}_{p} \) property

\[
\nu Z. (\forall s, c, i. \text{DeliveryC}(s, c, \text{active}, i) \land \text{MyGoldCust}@s(c) \\
\to \mu Y. (\text{DeliveryC}(s, c, \text{sat}, i)) \\
\lor (\text{LIVE}(s, c, i) \land (\langle - \rangle Y) \lor \neg Z)
\]

models that, for every delivery commitment instance a seller has towards a gold customer, there must exist a run where the instance persists in the system until it is satisfied. ■

The number of states of \( T_{S}^{\sigma_0} \) is in general infinite, and verification of (even propositional) temporal properties of simple forms (e.g., reachability) turns out to be undecidable [2, 6]. This calls for identifying interesting classes of DACMASs for which verification is decidable. Recently, the notion of state-bounded system has been proposed in the context of both data-aware business processes [2] and MASs [3], as an interesting condition that ensures decidability of verification for rich first-order temporal properties, while reflecting naturally occurring working assumptions in real-world systems. Intuitively, state-boundedness allows for encountering infinitely many different data during the evolution of the system, provided that such data do not accumulate in a single state.

We take this general notion and adapt it to DACMASs. In particular, a DACMAS is state-bounded if, for every agent active in the system, there exists a bound on the number of data items simultaneously stored in its ABox. Since the ABox of inst stores the names of the active agents, this implicitly bounds also the number of simultaneously active agents. Observe, however, that the overall number of data items (and hence also agents) encountered across and along the runs of the system can still be infinite. With this notion in place, we obtain the following fundamental result:

**Theorem 1 ([10]).** Verifying state-bounded DACMASs against \( \mu L^{ECQ}_{p} \) properties is decidable and reducible to finite-state model checking.

This means that state-bounded DACMASs can be verified, in principle, using standard model checkers for propositional \( \mu \)-calculus.
6 Conclusion

DACMASs are readily implementable in standard technologies such as JADE (which supports dynamic agent creation) and lightweight ontologies. Observe that a system execution requires polynomial time at each step (actually logspace w.r.t. the data, as any system based on relational databases). Only offline verification of the system is (as usual) exponential in the representation. Our framework complements that of [7], which employs data-aware commitments to monitor a system execution and track the state of commitment instances, but cannot be exploited for static analysis.

We consider extending our framework with the possibility of checking epistemic properties, in the line of [3]. Notice that, if instead of relying on the $\mu$-calculus, we rely on CTL, we can relax the persistence requirement in the logic, as in [3]. We also intend to study how to derive skeletons for the local agent specifications from a global, choreographic commitment-based protocol.

References

Abstract. In real-world supervised Machine Learning tasks, the learned theory can be deemed as valid only until there is evidence to the contrary (i.e., new observations that are wrongly classified by the theory). In such a case, incremental approaches allow to revise the existing theory to account for the new evidence, instead of learning a new theory from scratch. In many cases, positive and negative examples are provided in a mixed and unpredictable order, which requires generalization and specialization refinement operators to be available for revising the hypotheses in the existing theory when it is inconsistent with the new examples. The space of Datalog Horn clauses under the OI assumption allows the existence of refinement operators that fulfill desirable properties. However, the versions of these operators currently available in the literature are not able to handle some refinement tasks. The objective of this work is paving the way for an improved version of the specialization operator, aimed at extending its applicability.

1 Introduction

Supervised Machine Learning approaches based on First-Order Logic representations are particularly indicated in real-world tasks in which the relationships among objects play a relevant role in the definition of the concepts of interest. Given an initial set of examples, a theory can be learned from them by providing a learning system with the whole ‘batch’ of examples. However, being inductive inference only falsity preserving, the learned theory can be deemed as valid only until there is evidence to the contrary (i.e., new observations that are wrongly classified by the theory). In such a case, either a new theory is to be learned from scratch using the new batch made up of both the old and the new examples, or the existing theory must be incrementally revised to account for the new evidence as well. To distinguish these two stages, we may call them ‘training’ and ‘tuning’, respectively. In extreme cases, the initial batch is not available at all, and learning must start and proceed incrementally from scratch. In many real cases, positive and negative examples are provided in a mixed and unpredictable order to the tuning phase, which requires two different refinement operator to be available for revising the hypotheses in the existing theory when it is inconsistent
with the new examples. A generalization operator is needed to refine a hypothesis that does not account for a positive example, while a specialization operator must be applied to refine a hypothesis that erroneously accounts for a negative example. So, the kind of modifications that are applied to the theory change its behavior non-monotonically. The research on incremental approaches is not very wide, due to the intrinsic complexity of learning in environments where the available information about the concepts to be learned is not completely known in advance, especially in a First-Order Logic (FOL) setting. Thus, the literature published some years ago still represents the state-of-the-art for several aspects of interest.

The focus of this paper is on supervised incremental inductive learning of logic theories from examples, and specifically on the extension of existing specialization operators. Indeed, while these operators have a satisfactory behavior when trying to add positive literals to a concept definition, the way they handle the addition of negative information has some shortcomings that, if solved, would allow a broader range of concepts to be learned. Here we point out these shortcomings, and propose both improvements of the existing operator definitions, and extensions to them. Theoretical results on the new version of the operator are sketched, and an algorithm for it is provided and commented. The solution has been implemented and embedded in the multistrategy incremental learning system InTheLEx [3]. The next section lays the logic framework in which we cast our proposal; then, Sections 3 and 4 introduce the learning framework in general and the state-of-the-art specialization operator for it in particular. Section 5 describes our new proposal, and finally Section 6 concludes the paper. Due to lack of space, proofs of theoretical results will not be provided.

2 Preliminaries

The logic framework in which we build our solution exploits Datalog [1, 4] as a representation language. Syntactically, it can be considered as a sublanguage of Prolog in which no function symbols are allowed. I.e., a Datalog term can only be a variable or a constant, which avoids potentially infinite nesting in terms and hence simplifies clause handling by the operators we will define in the following. The missing expressiveness of function symbols can be recovered by Flattening [9], a representational change that transforms a set of clauses containing function symbols into another, semantically equivalent to it, made up of function-free clauses1. In a nutshell, each n-ary function symbol is associated to a new (n+1)-ary predicate, where the added argument represents the function result. Functions are replaced, in the literals in which they appear, by variables or constants representing their result.

In the following, we will denote by body(C) and head(C) the set of literals in the body and the atom in the head of a Horn clause C, respectively. Pure

---

1 Flattening potentially generates an infinite function free program. This is not our case, where we aim at learning from examples, and thus our universe is limited by what we see in the examples.
Datalog does not allow the use of negation in the body of clauses. A first way to overcome this limitation is the Closed World Assumption (CWA): if a fact does not logically follow from a set of Datalog clauses, then we assume its negation to be true. This allows the deduction of negative facts, but not their use to infer other information. Conversely, real world description often needs rules containing negative information. Datalog allows to use negated literals in clauses body, at the cost of a further safety condition: each variable occurring in a negated literal must occur in another positive literal of the body too.

Since there can be several minimal Herbrand models instead of a single a least one, CWA cannot be used. In Stratified Datalog this problem is solved by partitioning a program \( P \) in \( n \) sets \( P^i \) (called layers) s.t.:

1. all rules that define the same predicate in \( P \) are in the same layer;
2. \( P^1 \) contains only clauses without negated literals, or whose negated literals correspond to predicates defined by facts in the knowledge base;
3. each layer \( P^i, i > 1 \), contains only clauses whose negated literals are completely defined in lower level layers (i.e., layers \( P^j \) with \( j < i \)).

Such a partition is called stratification, and \( P \) is called stratified.

A stratified program \( P \) with stratification \( P^1, \ldots, P^n \) is evaluated by growing layers, applying to each one CWA locally to the knowledge base made up by the original knowledge base and by all literals obtained by the evaluation of the previous layers.

There may be different stratifications for a given program, but all are equivalent as regards the evaluation result. Moreover, not all programs are stratified. The following notion allows to know if they are.

**Definition 1 (Extended dependence graph)** Let \( P \) be a Datalog program. The extended dependence graph of \( P \), \( EDG(P) \), is a directed graph whose nodes represent predicates defined by rules in \( P \), and there is an edge \( \langle p, q \rangle \) if \( q \) occurs in the body of a rule defining \( p \). An edge \( \langle p, q \rangle \) is labeled with \( \neg \) if there exists at least one rule having \( p \) as its head and \( \neg q \) in its body.

A program \( P \) is stratified if \( EDG(P) \) contains no cycles containing edges marked with \( \neg \). The evaluation of a stratified program produces a minimal Herbrand model, called perfect model.

A specific kind of negation is expressed by the inequality built-in predicate \( \neq \). Using only this negation in Datalog yields an extension denoted by Datalog\( ^\neq \).

### 2.1 Object Identity

We deal with Datalog under the Object identity (OI) assumption, defined as follows:

**Definition 2 (Object Identity)**

*Within a clause, terms denoted with different symbols must be distinct.*
This notion is the basis for the definition of an equational theory for Datalog clauses that adds one rewrite rule to the set of the axioms of Clark’s Equality Theory (CET) [6]:

\[ t \neq s \in \text{body}(C) \text{ for each clause } C \text{ in } L \text{ and for all pairs } t, s \text{ of distinct terms that occur in } C \]  

where \( L \) denotes the language that consists of all the possible Datalog clauses built from a finite number of predicates. The \( \text{OI} \) rewrite rule can be viewed as an extension of both Reiter’s \textit{unique-names} assumption [8] and axioms (7), (8) and (9) of CET to the variables of the language.

\( \text{Datalog}^{\text{OI}} \) is a sublanguage of \( \text{Datalog}^{\neq} \) resulting from the application of \( \text{OI} \) to Datalog. Under \( \text{OI} \), any Datalog clause \( C \) generates a new Datalog
\( \neq \) clause \( C_{\text{OI}} \) consisting of two components, called \textit{core} and \textit{constraints}:

\[ \begin{align*}
\text{core}(C_{\text{OI}}) & = C \\
\text{constraints}(C_{\text{OI}}) & = \{ t \neq s \mid t, s \in \text{terms}(C) \wedge t, s \text{ distinct} \}
\end{align*} \]

are the inequalities generated by the \( \text{OI} \) rewrite rule.

Formally, a Datalog\( ^{\text{OI}} \) program is made up of a set of Datalog\( ^{\neq} \) clauses of the form

\[ l_0 : l_1, \ldots, l_n, c_1, \ldots, c_m \]

where the \( l_i \)'s are as in Datalog, and the \( c_j \)'s are the inequalities generated by the \( \text{OI} \) rule and \( n \geq 0 \). Nevertheless, Datalog\( ^{\text{OI}} \) has the same expressive power as Datalog, that is, for any Datalog program we can find a Datalog\( ^{\text{OI}} \) program equivalent to it [11].

### 2.2 \( \theta_{\text{OI}} \)-subsumption

Applying the \( \text{OI} \) assumption to the representation language causes the classical ordering relations among clauses to be modified, thus yielding a new structure of the corresponding search spaces for the refinement operators.

The ordering relation defined by the notion of \( \theta \)-subsumption under \( \text{OI} \) upon Datalog clauses [2, 10] is \( \theta_{\text{OI}} \)-subsumption.

**Definition 3 (\( \theta_{\text{OI}} \)-subsumption ordering)** Let \( C, D \) be Datalog clauses. \( D \) \( \theta_{\text{OI}} \)-subsumes \( C \) under \( \text{OI} \) (\( D \theta_{\text{OI}} \)-subsumes \( C \)), written \( C \leq_{\text{OI}} D \), iff \( \exists \sigma \) substitution s.t. \( D_{\text{OI}}.\sigma \subseteq C_{\text{OI}} \). This means that \( D \) is more general than or equivalent to \( C \) (in a theory revision setting, \( D \) is an upward refinement of \( C \) and \( C \) is a downward refinement of \( D \)) under \( \text{OI} \). \( C \prec_{\text{OI}} D \) stands for \( C \leq_{\text{OI}} D \wedge D \nsubseteq_{\text{OI}} C \). \( C \) and \( D \) are equivalent under \( \text{OI} \) (\( C \sim_{\text{OI}} D \)) when \( C \leq_{\text{OI}} D \) and \( D \leq_{\text{OI}} C \).

A substitution, as usual, is a mapping from variables to terms [13]. Its domain can be extended to terms: applying a substitution \( \sigma \) to a term \( t \) means that \( \sigma \) is applied to all variables in \( t \). In our case, applying a substitution to a constant leaves it unchanged.

Like \( \theta \)-subsumption, \( \theta_{\text{OI}} \)-subsumption induces a quasi-ordering upon the space of Datalog clauses, as stated by the following result.
Proposition 1  Let \( C, D, E \) be Datalog clauses. Then:

1. \( C \leq_{OI} C \) \hspace{1cm} (reflexivity)
2. \( C \leq_{OI} D \) and \( D \leq_{OI} E \) \( \Rightarrow \) \( C \leq_{OI} E \) \hspace{1cm} (transitivity)

Other interesting properties of \( \theta_{OI} \)-subsumption are the following:

Proposition 2  Let \( C, D \) be Datalog clauses.

- \( C \leq_{OI} D \) \( \Rightarrow \) \( C \leq_{\theta} D \), where \( \leq_{\theta} \) denotes \( \theta \)-subsumption
  (i.e., \( \theta_{OI} \)-subsumption is a weaker relation than \( \theta \)-subsumption).
- \( C \leq_{OI} D \) \( \Rightarrow \) \( |C| \geq |D| \).
- \( C \sim_{OI} D \) iff \( C \) and \( D \) are renamings.

Non-injective substitutions would yield contradictions when applied to constraints (e.g., \( [x \neq y], \{x/a, y/a\} = [a \neq a] \)). So, under \( OI \), substitutions are required to be injective.

Requiring that terms are distinct ‘freezes’ the number of literals of the clause (since they cannot unify among each other), hence \( \theta_{OI} \)-subsumption maps each literal of the subsuming clause onto a single, different literal in the subsumed one. In particular, equivalent clauses under \( \leq_{OI} \) must have the same number of literals, hence the only way to have equivalence is through variable renaming. Thus, a search space ordered by \( \theta_{OI} \)-subsumption is made up of non-redundant clauses, i.e. no subset of a clause can be equivalent to the clause itself under \( OI \). This yields smaller equivalence classes than those in a space ordered by \( \theta \)-subsumption.

Proposition 3 (Decidability of \( \theta_{OI} \)-subsumption)  Given two clauses \( C \) and \( D \), \( C \leq_{OI} D \) is a decidable relationship.

In the worst case, i.e. when \( |D| \leq |C| \), all literals in \( D \) match with all literals in \( |C| \), and all such matchings are pairwise compatible, an upper bound to the complexity of the \( \theta_{OI} \)-subsumption test is \( (\binom{|C|}{|D|}) \).

3  Incremental Inductive Synthesis

ILP aims at learning logic programs from examples. In our setting, examples are represented as clauses, whose body describes an observation, and whose head specifies a relationship to be learned, referred to terms in the body. Negative examples for a relationship have a negated head. A learned program is called a theory, and is made up of hypotheses, i.e. sets of program clauses all defining the same predicate. A hypothesis covers an example if the body of at least one of its clauses is satisfied by the body of the example. The search space is the set of all clauses that can be learned, ordered by a generalization relationship.

In ILP, a standard practice to restrict the search space is imposing biases on it [7]. In the following, we are concerned with logic theories expressed as hierarchical (i.e., non-recursive) programs, for which it is possible to find a level
mapping [6] s.t., in every program clause, the level of every predicate symbol occurring in the body is less than the level of the predicate in the head. This has strict connections with stratified programs, that are needed when the language is extended to deal with negation. Another bias on the representation language is that, whenever we write about clauses, we mean Datalog linked clauses. A clause is linked if, for any term appearing in its body, it also appears in the head or it is possible to find a chain of terms such that adjacent terms appear in the same literal and at least a term in the chain appears in the head.

The canonical inductive paradigm requires the learned theory to be complete and consistent. For hierarchical theories, the following definitions are given (where \( E^- \) and \( E^+ \) are the sets of all the negative and positive examples, resp.):

Definition 4 (Inconsistency)
- A clause \( C \) is inconsistent wrt \( N \in E^- \) iff \( \exists \sigma \) s.t.\( \text{body}(C).\sigma \subseteq \text{body}(N) \land \neg \text{head}(C).\sigma = \text{head}(N) \land \text{constraints}(C_{OI}).\sigma \subseteq \text{constraints}(N_{OI}) \).
- A hypothesis \( H \) is inconsistent wrt \( N \) iff \( \exists C \in H: C \) is inconsistent wrt \( N \).
- A theory \( T \) is inconsistent iff \( \exists H \subseteq T, \exists N \in E^- : H \) is inconsistent wrt \( N \).

Definition 5 (Incompleteness)
- A hypothesis \( H \) is incomplete wrt \( P \) iff \( \forall C \in H: \text{not}(P \preceq_{OI} C) \).
- A theory \( T \) is incomplete iff \( \exists H \subseteq T, \exists P \in E^+: H \) is incomplete wrt \( P \).

When the theory is to be learned incrementally, it becomes relevant to define operators that allow a stepwise (incremental) refinement of too weak or too strong programs [5]. A refinement operator, applied to a clause, returns one of its upward or downward refinements. Refinement operators are the means by which wrong hypotheses in a logic theory are changed in order to account for new examples with which they are incomplete or inconsistent. In the following, we will assume that logic theories are made up of clauses that have only variables as terms, built starting from observations described as conjunctions of ground facts (i.e., variable-free atoms). This assumption causes no loss in expressive power, since a reification process allows to express through predicates all the information that may be carried out by constants. Put another way, we take to the extreme the flattening procedure, considering even constants as 0-ary functions that are replaced by 1-ary predicates having the same name and meaning. This restriction simplifies the refinement operators for a space ordered by \( \theta_{OI} \)-subsumption defined in [2, 10], and the associated definitions and properties.

Definition 6 (Refinement operators under OI) Let \( C \) be a Datalog clause.
- \( D \in \rho_{OI}(C) \) (downward refinement operator) when \( \text{body}(D) = \text{body}(C) \cup \{l\} \), where \( l \) is an atom s.t. \( l \notin \text{body}(C) \).
- \( D \in \delta_{OI}(C) \) (upward refinement operator) when \( \text{body}(D) = \text{body}(C) \setminus \{l\} \), where \( l \) is an atom s.t. \( l \in \text{body}(C) \).

\[^2\] \( \neg \text{head}(C).\sigma = \text{head}(N) \) because the relationship must be the same as for \( N \).
Particularly important are locally finite, proper and complete (ideal) refinement operators [10]. Such a kind of operators are not feasible when full Horn clause logic is chosen as representation language and either $\theta$-subsumption or implication is adopted as generalization model, because of the existence of infinite unbound strictly ascending/descending chains. On the contrary, in the space of Datalog clauses ordered by $\theta_{OI}$-subsumption such a kind of chains do not exist, since equivalence among clauses coincides with alphabetic variance. This makes possible the existence of ideal refinement operators under the ordering induced by $\theta_{OI}$-subsumption [2, 10].

By the definition of $\delta_{OI}$, any possible generalization of a clause must have as body a subset of its body, and hence there are $2^{\mid body(C)}$ such generalizations.

**Proposition 4** [10] The refinement operators in Definition 6 are ideal for Datalog clauses ordered by $\theta_{OI}$-subsumption.

Inspired to a concept given by Shapiro [12], we have a measure for the complexity of a clause:

**Definition 7** ($\text{size}_{OI}$) The size of a clause $C$ under OI ($\text{size}_{OI}(C)$) is the number of literals in the body of $C$:

$$\text{size}_{OI}(C) = \mid body(C)\mid$$

Under $\theta_{OI}$-subsumption it allows to predict the exact number of steps required to perform a refinement, based only on the syntactic structure of the clauses involved (that could be known or bounded a priori): given two clauses $C$ and $D$, if $D <_{OI} C$, then $C \in \delta_{OI}^k(D), D \in \rho_{OI}^k(C)$, $k = \text{size}_{OI}(D) - \text{size}_{OI}(C) = \mid body(D)\mid - \mid body(C)\mid$.

### 4 Downward Refinement

When a negative example is covered, a specialization of the theory must be performed. Starting from the current theory, the misclassified example and the set of processed examples, the specialization algorithm outputs a revised theory. In our framework, specializing means adding proper literals to a clause that is inconsistent with respect to a negative example, in order to avoid its covering that example. The possible options for choosing such a literal might be so large that an exhaustive search is not feasible. Thus, we want the operator to focus the search into the portion of the space of literals that contains the solution of the diagnosed commission error, as a result of an analysis of its algebraic structure.

According to the theoretical operator in Definition 6, only positive literals can be added. To this aim, we try to add to the clause one (or more) atom(s), which characterize all the past positive examples and can discriminate them from the current negative one. The search for such atoms is performed in the space of positive literals, that contains information coming from the positive examples used to learn the current theory, but not yet exploited by it. First of all, the process of abstract diagnosis detects all the clauses that caused the inconsistency. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be the positive examples $\theta_{OI}$-subsumed by
a clause \( C \) that is inconsistent wrt a negative example \( N \). The set of all possible most general downward refinements under \( \text{OI} \) of \( C \) against \( N \) is:

\[
\text{mgdr}_{\text{OI}}(C, N) = \{ M \mid M \leq_{\text{OI}} C, M \text{ consistent wrt } N, \forall D \text{ s.t. } D \leq_{\text{OI}} C, D \text{ consistent wrt } N : \text{not}(M \prec_{\text{OI}} D) \}
\]

Among these, the search process aims at finding one that is compliant with the previous positive examples in \( P \), i.e. one of the most general downward refinements under \( \text{OI} \) (\( \text{mgdr}_{\text{OI}} \)) of \( C \) against \( N \) given \( P_1, \ldots, P_n \):

\[
\text{mgdr}_{\text{OI}}(C, N \mid P_1, \ldots, P_n) = \{ M \in \text{mgdr}_{\text{OI}}(C, N) \mid P_j \leq_{\text{OI}} M, j = 1, \ldots, n \}
\]

Since the downward refinements we are looking for must satisfy the property of maximal generality, the operator tries to add as few atoms as possible. Thus, it may happen that, even after some refinement steps, the added atoms are still not sufficient to rule out the negative example, i.e. the specialization of \( C \) is still overly general. This suggests to further exploit the positive examples in order to specialize \( C \). Specifically, if there exists a literal that, when added to the body of \( C \), is able to discriminate from the negative example \( N \) that caused the inconsistency of \( C \), then the downward refinement operator should be able to find it. The resulting specialization should restore the consistency of \( C \), by refining it into a clause \( C' \) which still \( \theta_{\text{OI}} \)-subsumes the positive examples \( P_i \), \( i = 1, 2, \ldots, n \).

The process of refining a clause by means of positive literals can be described as follows. For each \( P_i \), \( i = 1, 2, \ldots, n \), suppose that there exist \( n_i \) distinct substitutions s.t. \( C \theta_{\text{OI}} \)-subsumes \( P_i \), and consider all the possible \( n \)-tuples of substitutions obtained by picking one of such substitutions for every positive example. Each of these substitutions yields a distinct residual, consisting of all the literals in the example that are not involved in the \( \theta_{\text{OI}} \)-subsumption test, after having properly turned their constants into variables. Formally:

**Definition 8 (Residual)** Let \( C \) be a clause, \( E \) an example, and \( \sigma_j \) a substitution s.t. \( \text{body}(C).\sigma_j \subseteq \text{body}(E) \) and \( \text{constraints}(C_{\text{OI}}).\sigma_j \subseteq \text{constraints}(E_{\text{OI}}) \). A residual of \( E \) wrt \( C \) under the mapping \( \sigma_j \), denoted by \( \Delta_j(E, C) \), is:

\[
\Delta_j(E, C) = \text{body}(E) \cdot \sigma_j^{-1} \cdot \backslash \text{body}(C)
\]

where \( \sigma_j^{-1} \) is the extended antisubstitution of \( \sigma_j \). An antisubstitution is a mapping from terms onto variables. When a clause \( C \theta_{\text{OI}} \)-subsumes an example \( E \) through a substitution \( \sigma \), then it is possible to define a corresponding antisubstitution, \( \sigma^{-1} \), which is the inverse function of \( \sigma \), mapping some constants in \( E \) to variables in \( C \). Since not all constants in \( E \) have a corresponding variable according to \( \sigma^{-1} \), we introduce the extension of \( \sigma^{-1} \), denoted with \( \underline{\sigma}^{-1} \), that is defined on the whole set \( \text{consts}(E) \), and takes values in the set of the variables of the language\(^3\):

\[
\underline{\sigma}^{-1}(c_n) = \begin{cases} 
\sigma^{-1}(c_n) & \text{if } c_n \in \text{vars}(C).\sigma \\ - & \text{otherwise}
\end{cases}
\]

\(^3\) Variables denoted by - are new variables, managed as in Prolog.
The residuals obtained from the positive examples $P_i, i = 1, \ldots, n$, can be exploited to build a space of complete positive downward refinements, denoted with $P$, and formally defined as follows.

$$P = \bigcup_{i=1, \ldots, n} \bigcap_{k=1, \ldots, n} \Delta_{j_k}(P_k, C)$$

where the symbol $\Delta_{j_k}(P_k, C)$ denotes one of the $n_k$ residuals of $P_k$ wrt $C$, and $\cap_{k=1, \ldots, n} \Delta_{j_k}(P_k, C)$, when $j_k \in \{1, \ldots, n_k\}$, is the set of the literals common to an $n$-tuple of residuals (one residual for each positive example $P_k, k = 1, \ldots, n$). Moreover, denoted with $\theta_j, j = 1, \ldots, m$, all the substitutions which make $C$ inconsistent wrt $N$, let us define a new space:

$$S = \bigcup_{j=1, \ldots, m} \Delta_j(N, C)$$

which includes all the literals that cannot be used for refining $C$, because they would still be present in $N$.

**Proposition 5** Given a clause $C$ that $\theta_{OI}$-subsumes the positive examples $P_1, \ldots, P_n$ and is inconsistent wrt the negative example $N$, then:

$$\{C' \mid \text{head}(C') = \text{head}(C) \land \text{body}(C') = \text{body}(C) \cup \{l\}, l \in P - S \} \subseteq \subseteq \text{mgd}_{OI}(C, N \mid P_1, \ldots, P_n)$$

Hence, every downward refinement built by adding a literal in $P - S$ to the inconsistent clause $C$ restores the properties of consistency and completeness of the original hypothesis. Moreover, it is one of the most general downward refinements of $C$ against $N$.

It may happen that no (set of) positive literal(s) is able to characterize the past positive examples and discriminate the negative example that causes inconsistency. In such a case, the above version of the operator would fail. However, we don’t want to give up yet, since the addition of a negative literal to the clause body might restore consistency\(^4\). To take this opportunity, we extend the search space to Datalog\(^-\). These literals are interpreted according to the CWA. Of course, suitable adaptations of the notions presented in Section 2 are used to handle these literals\(^5\). So, in case of failure on the search for positive literals, the algorithm autonomously performs a representation change, that allows it to extend the search to the space of negative literals, built by taking into account the negative example that caused the commission error. The new version of the operator tries to add the negation of a literal, that is able to discriminate the negative example from all the past positive ones. Revisions performed by this

\(^4\) Note that a negative literal in the body corresponds to a positive literal in the clause. However, here we are expressing the fact that a condition must not hold in an observation in order to infer the relationship in the head.

\(^5\) E.g., given two clauses under OI, $C = C^+ \cup C^-$ and $D = D^+ \cup D^-$, where $C^+$ and $D^+$ include the positive literals and the OI-constraints, and $C^-$ and $D^-$ are sets of negative literals, $C \leq_{OI} D$ iff $\exists \sigma$ substitution s.t. $D^+, \sigma \subseteq C^+$ and $\forall d \in D^- : \beta c \in C^- \land \text{d} \cdot \sigma = c$.  

S. Ferilli. Toward an Improved Downward Refinement Operator for Inductive Logic Programming
operator are always minimal [14], since all clauses in the theory contain only variables as arguments. Moreover, this operator is ideal in the space of constant-free clauses. The definitions and results in the rest of this section are taken from [2].

When the space $P - S$ does not contain any solution to the problem of specializing an inconsistent clause, a change of representation must be performed in order to search for literals in another space, corresponding to the quotient set of the Datalog linked clauses. In the following, a slightly different but equivalent specification of the operator in this space will be provided with respect to [2].

First of all, we define the new target space, called the space of negative downward refinements:

$$S_n = \neg S = \neg (\cup_{j=1,...,m} \Delta_j(N, C))$$

where, given a set of literals $\varphi = \{l_1, \ldots, l_n\}, n \geq 1$: $\neg \varphi = \{-l_1, \ldots, -l_n\}$. Again, we are interested in a specific subset of $S_n$, because of the properties satisfied by its elements. Let us introduce the following notation:

$$S = \cap_{j=1,...,m} \Delta_j(N, C)$$

Note that $S \subseteq S$. Based on $S$, the space of consistent negative downward refinements can be defined as:

$$S_c = \neg S = \neg (\cap_{j=1,...,m} \Delta_j(N, C))$$

Indeed, $S_c$, compared to $S_n$, fulfills the following property:

**Proposition 6** Given a clause $C$ and an example $N$, then:

$$\{C' \mid \text{head}(C') = \text{head}(C) \land \text{body}(C') = \text{body}(C) \cup \{l \in S_c\} \subseteq \text{mgdr}_{OI}(C, N)$$

Overall, the search for a complete and consistent hypothesis can be viewed as a two-stage process: the former stage searches into the space $P - S$, the latter into $S_c$. It is now possible to formally define the downward refinement operator $\rho_{OI}^{cons}$ on the space $L$ of constant-free Datalog linked program clauses.

**Definition 9** ($\rho_{OI}^{cons}$) $\rho_{OI}^{cons} : L \rightarrow 2^L \forall C \in L : \rho_{OI}^{cons}(C) = \{C' \mid \text{head}(C') = \text{head}(C) \land \text{body}(C') = \text{body}(C) \cup \{l \in (P - S) \cup S_c\} \}

**Proposition 7** The downward refinement operator $\rho_{OI}^{cons}$ is ideal.

The ideality of $\rho_{OI}^{cons}$ is owed to the peculiar structure of the search space when ordered by the relation $\leq_{OI}$.

5 Discussion and Extension of the Specialization Operator

The existing definition of $\rho_{OI}^{cons}$ aims at identifying a set of literals each of which, when added to a clause $C$, yields a new clause that is both consistent with the given negative example and complete with respect to all the previous positive examples. Now, this is true if the added literal belongs to the space of complete
positive downward refinements \( P - S \). Conversely, the space of consistent negative downward refinements does not ensure completeness wrt the previous positive examples, since it is computed considering only all possible residuals of the negative example. This can be easily shown in the following example\(^6\).

**Example 1.** Consider the following situation.

Two positive examples: \( P_1 = h : -p, q, t, u, v \) and \( P_2 = h : -p, q, v \).

produce as a least general generalization the clause \( C_1 = h : -p, q \).

Then, the negative example \( N_1 = h : -p, q, t, u, v \). arrives.

The residuals of \( P_1 \) and \( P_2 \) wrt \( C_1 \) are \( \{t, u\} \) and \( \{v\} \), respectively.

The residual of \( N_1 \) is \( \{t, u, v\} \). So, \( P - S = (\{t, u\} \cap \{v\}) - \{t, u, v\} = \emptyset - \{t, u, v\} = \emptyset \), hence no specialization by means of positive literals can be obtained (as expected, since \( C \) was a least general generalization). Switching to the space of negative literals, we have that \( S_c = \neg(\{t, u, v\}) = \{\neg t, \neg u, \neg v\} \). However, none of these literals generates a clause that is complete with all previous positive examples:

\[
\begin{align*}
C'_2 &= h : -p, q, \neg t, \text{ where } P_1 \not\leq_{OI} C'_2 \\
C''_2 &= h : -p, q, \neg u, \text{ where } P_1 \not\leq_{OI} C''_2 \\
C'''_2 &= h : -p, q, \neg v, \text{ where } P_2 \not\leq_{OI} C'''_2 \\
\end{align*}
\]

So, what we need to consider is not \( S_c \). Intuitively, we want to select a literal that is present in all residuals of the negative example and that is not present in any residual of any positive example. Let us define:

\[
P = \bigcup_{i=1,\ldots,n} \Delta_{j_i}(P_i, C)
\]

Now, what we need to consider is \( S_c' = \neg(S - P) \).

**Example 2.** In the previous example, we would have \( S_c' = (\{t, u, v\}) - (\{t, u\} \cup \{v\}) = \{t, u, v\} - \{t, u, v\} = \emptyset \) which shows, as expected, that no complete refinement can be obtained for the given case.

Consider now another set of positive examples: \( P_1 = h : -p, q, t, u, \) and \( P_2 = h : -p, q, s, t \).

whose least general generalization is, again, the clause \( C_1 = h : -p, q \).

Then, the negative example \( N_1 = h : -p, q, t, u, v, w \). arrives.

The residuals of the positive examples are: \( \Delta(C_1, P_1) = \{t, u\} \), \( \Delta(C_1, P_2) = \{r\} \) and \( \Delta(C_1, P_3) = \{s, t\} \).

The residual of \( N_1 \) is \( \{t, u, v, w\} = S \).

So, \( P - S = (\{t, u\} \cap \{r\} \cap \{s, t\}) - \{t, u, v, w\} = \emptyset - \{t, u, v, w\} = \emptyset \), hence again no specialization by means of positive literals can be obtained. Switching to the space of negative literals, we have that \( S_c' = \neg(S - P) = \neg(\{t, u, v, w\} - (\{t, u\} \cup \{r\} \cup \{s, t\})) = \neg(\{t, u, v, w\} -

\footnote{For the sake of readability, in the following we will often switch to a propositional representation. This means that the residual is unique for each example, so the subscript in \( \Delta_i(\cdot, \cdot) \) is no more necessary.}
\{r, s, t, u\} = \neg\{(v, w)\} = \{\neg v, \neg w\} and indeed, adding any of these literals to C generates a clause that is complete with all previous positive examples:

\begin{align*}
C'_2 &= h : -p, q, \neg v \text{ where } P_1 \leq_{OI} C'_2, P_2 \leq_{OI} C'_2 \text{ and } P_3 \leq_{OI} C'_2; \\
C''_2 &= h : -p, q, \neg w \text{ where } P_1 \leq_{OI} C''_2, P_2 \leq_{OI} C''_2, P_3 \leq_{OI} C''_2.
\end{align*}

This is captured by the following result:

**Proposition 8** Given a clause \( C = h : \text{body}(C) \) that \( \theta_{OI} \)-subsumes the positive examples \( P_1, \ldots, P_n \) and is inconsistent wrt the negative example \( N \), then:

\[
\{ C' \mid \text{head}(C') = \text{head}(C) \land \text{body}(C') = \text{body}(C) \cup \{ l \mid l \in S'_c \} \subseteq mgd_{OI}(C, N \mid P_1, \ldots, P_n) \}
\]

Now, an additional problem arises. Indeed, in some cases a single negative literal is not enough to ensure that the correctness of the theory is restored. The situation may be clarified by the following example.

**Example 3.** Consider again the situation described in Example 1.

While no single (positive or negative) literal can restore completeness and consistency of the theory, either \( C'_2 = h : -p, q, \neg (t, v) \) or \( C''_2 = h : -p, q, \neg (u, v) \) would be correct refinements of \( C_1 \) wrt \( \{ P_1, P_2, N_1 \} \). These solutions are not permitted in the representation language, since only literals may appear in the body of clauses. However, any of the two above clauses corresponds to the conjunction of two clauses, e.g. \( C'_2 \) is equivalent to \( \{ h : -p, q, \neg t, h : -p, q, \neg v \} \). This solution would introduce some redundancy in the theory, since the body of the original clause \( C_1 \) would appear in both specialized clauses. This might be undesirable, in which case we may leverage Datalog implication and solve the problem by inventing a new predicate \( s \) as follows: \( \{ h : -p, q, \neg(s) , s : -t, v \} \). The intuition behind this choice is that the need to place together the literals in the negation might be a hint of a more general relationship among them. This relationship might be captured by a so far unknown concept, that is explicitly added. A useful side effect of this setting is that when the same combination will occur in future observations, it will be recognized and explicitly added by saturation, this way obtaining higher level descriptions.

So, the extension comes into play when no single literal is sufficient to restore correctness of the theory. Indeed, when a single literal is to be negated there is no need for inventing any predicate. More formally, we are not looking anymore for a single \( l \in S'_c \) to be added to \( C \), but we need a \( S \subseteq S'_c \) s.t. \( \forall i : \exists l \in S \) s.t. \( l \notin P_i \). In particular, we would like to find a minimal such set. Minimality may be in terms of set inclusion or of number of elements: \( \overline{S} = \arg \min_S(|S|) \).

To formally express our operator, let us define:

\[
\begin{align*}
&\forall r \in \Delta(N, C) : \\
&\quad \overline{P}_r = \{ P_i \in P \mid r \in \Delta(P_i, C) \} \\
&\quad \overline{P}_r = \{ P_i \in P \mid r \notin \Delta(P_i, C) \} \\
&\forall S \subseteq \Delta(N, C) : \\
&\quad \overline{P}_S = \cap_{r \in S} \overline{P}_r \\
&\quad \overline{P}_S = \cup_{r \in S} \overline{P}_r
\end{align*}
\]
is the set of positive examples that are no more covered when adding \(-r\) to \(C\);
\(\overline{P}_r\) is the set of positive examples that are still covered when adding \(-r\) to \(C\);
\(P_S\) is the set of positive examples that are no more covered when adding \(neg(S)\) to \(C\);
\(\overline{P}_S\) is the set of positive examples that are still covered when adding \(neg(S)\) to \(C\). This helps us to define what we are looking for. Specifically, we need a \(S \subseteq \Delta(N, C)\) s.t. \(P_S = \emptyset \land \overline{P}_S = \mathcal{P}\), as shown by the following example:

**Example 4.** Consider the set of positive examples \(P = \{P_1, P_2, P_3\}\) where:

\(P_1 = h : -p, q, s, t\) \hspace{1cm} \(P_2 = h : -p, q, t, u\) \hspace{1cm} \(P_3 = h : -p, q, u, r\)

Given their least general generalization \(C = h : -p, q, r\), the corresponding residuals are:

\(\Delta(P_1, C) = \{s, t\}\) \hspace{1cm} \(\Delta(P_2, C) = \{t, u\}\) \hspace{1cm} \(\Delta(P_3, C) = \{u, r\}\)

Now, given the negative example \(N = h : -p, q, s, t, u, r\) covered by \(C\), with residual \(\Delta(N, C) = \{s, t, u, r\}\), we have:

\(P_{\{s,u\}} = P_s \cap P_u = \{P_1\} \cap \{P_2, P_3\} = \emptyset;\) \hspace{1cm} \(P_{\{s,u\}} = \overline{P}_s \cap \overline{P}_u = \{P_2, P_3\} \cup \{P_1\} = \mathcal{P}\): NOT A SOLUTION!

\(P_{\{t,u\}} = P_t \cap P_u = \{P_1, P_2\} \cap \{P_2, P_3\} = \emptyset;\) \hspace{1cm} \(P_{\{t,u\}} = \overline{P}_t \cap \overline{P}_u = \{P_3\} \cup \{P_3, P_1\} \neq \mathcal{P}\): NOT A SOLUTION!

\(\ldots\) and so on.

Of course, a trial-and-error approach would solve the problem, but there is an exponential number of subsets to be tried. In order to devise a more efficient algorithm, let us analyze the sets \(P_r, \overline{P}_r, P_S\) and \(\overline{P}_S\) to better understand them and their behavior. First of all, the \(P_s\)'s and \(\overline{P}_s\)'s are complementary:

**Proposition 9** Given a clause \(C\) and a negative example \(N\) covered by \(C\):

1. \(\forall r \in \Delta(N, C) : \{P_r, \overline{P}_r\}\) is a partition of \(P\);
2. \(\forall S \subseteq \Delta(N, C) : \{P_S, \overline{P}_S\}\) is a partition of \(P\).

This ensures, in particular, that \(P_S = \emptyset \land \overline{P}_S = \mathcal{P} \iff P_S = \emptyset \iff \overline{P}_S = \mathcal{P}\). We also note that positive example (un-)coverage is monotonic:

**Proposition 10** \(\forall S' \subset S'' \subseteq \Delta(N, C) : P_{S''} \subseteq P_{S'} \land \overline{P}_{S''} \subseteq \overline{P}_{S'}\)

Finally, let us note that any element of the residual of the negative example, added to \(C\), causes some positive example to become uncovered (which will be used in the first iteration of our algorithm):

**Proposition 11** If \(\rho_{cons}^r\) fails, then \(\forall r \in \Delta(N, C) : P_r \neq \emptyset\).

We propose a sequential covering-like strategy to find such an \(S\), according to Algorithm 1. Note that, at the beginning of the algorithm, \(S = \emptyset \Rightarrow P_S = \emptyset \Rightarrow |P_S| = 0\) and \(S = \emptyset \Rightarrow \overline{P}_S = \mathcal{P} \Rightarrow |\overline{P}_S| = n\). However, as soon as the loop is entered, the selection and addition of the first \(r\) makes \(P \neq \emptyset\) by Proposition 11; so, the condition of the IF statement is true, hence \(S\) is updated and a second round of the loop is guaranteed to take place. At each round, a new
Algorithm 1 Backtracking specialization strategy

\[ S = \emptyset; \ R = \Delta(N, C) \]

repeat
\[ r \leftarrow \text{select from } \mathcal{R} \text{ OR } \text{backtrack} \]
\[ \mathcal{R} \leftarrow \mathcal{R} \setminus \{ r \}; \ S' \leftarrow S \cup \{ r \} \]
if \( \mathcal{P}_S \neq \overline{\mathcal{P}_S} \) (\( \Leftrightarrow \mathcal{P}_{S'} \neq \overline{\mathcal{P}_S} \)) (\( \Leftrightarrow \mathcal{P}_{S'} \subset \mathcal{P}_S \Leftrightarrow \overline{\mathcal{P}_{S'}} \supset \overline{\mathcal{P}_S} \)) then
\[ S \leftarrow S' \]
end if
until \( \mathcal{P}_S = \emptyset \) (\( \Leftrightarrow \overline{\mathcal{P}_S} = \mathcal{P} \)) OR no more backtracking available
if \( \mathcal{P}_S = \emptyset \) (\( \Leftrightarrow \overline{\mathcal{P}_S} = \mathcal{P} \)) then
return \( S \)
else
return \( \text{failure} \)
end if

\( r \) is removed from \( \mathcal{R} \) and added to \( S \) only if the coverage improves, otherwise it is discarded. If the last \( r \) does not satisfy the loop condition, the overall solution is not complete and backtracking is applied. Note that, in the worst case, adding the whole residual to \( C \) would be a solution, which ensures termination of the algorithm (unless there is a positive example that includes the whole residual, which can be checked before starting the algorithm). If different solutions are requested, backtracking can be applied to non-discarded items.

6 Conclusions and Future Work

Incremental supervised Machine Learning approaches using First-Order Logic representations are mandatory when tackling complex real-world tasks, in which relationships among objects play a fundamental role. A noteworthy framework for these approaches is based on the space of Datalog Horn clauses under the Object Identity assumption, which ensures the existence of (upward and downward) refinement operators fulfilling desirable requirements. The refinement operators for this framework proposed in the current literature have some limitations that this paper aims at overcoming. So, after recalling the most important elements of the framework and of the current operators, this paper points out these deficiencies and proposes solutions that result in improved operators. Specifically, the downward refinement operator is considered. A preliminary prototype of the operator has been implemented, and is currently being integrated in the InTheLEx learning system.

Future work includes a study of the possible connections of the extended operator with related fields of the logic-based learning, such as deduction, abstraction and predicate invention. Experiments aimed at assessing the efficiency and effectiveness of the operator in real-world domains are also planned.
Acknowledgments

This work was partially funded by the Italian PON 2007-2013 project PON02_00563_3489339 ‘Puglia@Service’.

References

Abstract. The transformation of constraint logic programs (CLP programs) has been shown to be an effective methodology for verifying properties of imperative programs. By following this methodology, we encode the negation of a partial correctness property of an imperative program \( \text{prog} \) as a predicate \( \text{incorrect} \) defined by a CLP program \( P \), and we show that \( \text{prog} \) is correct by transforming \( P \) into the empty program through the application of semantics preserving transformation rules. Some of these rules perform replacements of constraints that encode properties of the data structures manipulated by the program \( \text{prog} \). In this paper we show that Constraint Handling Rules (CHR) are a suitable formalism for representing and applying constraint replacements during the transformation of CLP programs. In particular, we consider programs that manipulate integer arrays and we present a CHR encoding of a constraint replacement strategy based on the theory of arrays. We also propose a novel generalization strategy for constraints on integer arrays that combines the CHR constraint replacement strategy with various generalization operators for linear constraints, such as widening and convex hull. Generalization is controlled by additional constraints that relate the variable identifiers in the imperative program \( \text{prog} \) and the CLP representation of their values. The method presented in this paper has been implemented and we have demonstrated its effectiveness on a set of benchmark programs taken from the literature.

1 Introduction

It has long been recognized that Constraint Logic Programming (CLP) is a formalism that provides very suitable inference mechanisms for the verification of properties of imperative programs. The landmark paper [41] has shown that: (i) the operational semantics of imperative programs can easily be formalized as an interpreter written in CLP, and (ii) by specializing that interpreter with
respect to a given imperative program, say \textit{prog}, one can derive a new CLP program, say \textit{VC}, representing the verification conditions for \textit{prog} in purely logical form. In particular, in the specialized CLP program \textit{VC} there are no references to the imperative constructs of \textit{prog}. Relevant properties of the execution of \textit{prog} (such as its loop invariants) can then be inferred by analyzing the program \textit{VC}.

Many verification methods within the CLP paradigm have been developed. Some methods, directly following the approach presented in [41], are based on \textit{abstract interpretation} [8] and compute an overapproximation of the least model of the CLP program under consideration by a bottom-up evaluation of an abstraction of the program [2, 28, 39]. Other methods use goal directed evaluation of CLP programs combined with other symbolic techniques such as \textit{interpolation} [17, 20, 31, 30]. Some other methods, like the ones presented in [5, 25, 43, 45], combine CLP (also called \textit{constrained Horn clauses} in those papers) with different reasoning techniques developed in the areas of Software Model Checking and \textit{Automated Theorem Proving}, such as CounterExample-Guided Abstraction Refinement (CEGAR) and \textit{Satisfiability Modulo Theory} (SMT).

In this paper we follow the approach based on transformations of CLP programs presented in [12, 13]. We encode the negation of a partial correctness property of an imperative program \textit{prog} as a predicate \textit{incorrect} defined by a CLP program \textit{P}. Similarly to [41], we generate a CLP program \textit{VC} representing the verification conditions for \textit{prog}, by specializing \textit{P} with respect to the CLP representation of \textit{prog}. However, at this point the transformation-based method departs from the ones considered above. Indeed, it continues by applying further equivalence preserving transformations to \textit{VC} with the objective of deriving either (i) the empty CLP program, hence proving that \textit{incorrect} does not hold and \textit{prog} is correct, or (ii) a CLP program containing the fact \textit{incorrect}, hence proving that \textit{prog} is incorrect. Due to the undecidability of partial correctness, it may be the case that we derive a CLP program containing one or more clauses of the form \textit{incorrect} :- \textit{G}, where \textit{G} is a non-empty conjunction, and we are able to conclude neither that \textit{prog} is correct nor that \textit{prog} is incorrect.

Thus, CLP program transformation provides a uniform framework for reasoning about the correctness of imperative programs in which, as we have explained, one can generate the verification conditions and also check their validity. Moreover, that framework is \textit{parametric} with respect to the syntax and the semantics of the programs to be verified, and optimizing transformations considered in the literature [42] can be applied to improve the efficiency of the verification method. Finally, transformations can easily be composed together into a sequence of transformations, so as to derive very sophisticated verification methods. For instance, in [15] it is shown that the \textit{iteration} of program specialization can significantly improve the precision of our program verification method and indeed, by implementing Iterated Specialization the VeriMAP system [14] is competitive with state-of-the-art CLP-based verifiers such as ARMC [43], HSF [25], and TRACER [30].

The main contributions of this paper are the following.

115
We consider imperative programs that manipulate integers and integer arrays, and we generate verification conditions where read and write operations on arrays are represented as constraints. Then we show that Constraint Handling Rules (CHR) are a suitable formalism for manipulating constraints during the transformation of the CLP verification conditions. In particular, we present CHR rules based on the theory of arrays \([7, 23, 37]\) and we show how they can be combined with unfold/fold transformation rules for CLP programs \([18]\) with the objective of proving properties of the given imperative programs.

(2) We propose a powerful transformation strategy that guides the application of both the CHR and the unfold/fold transformation rules. In particular, we design a novel array constraint generalization strategy that automatically introduces, during CLP transformation, the new predicate definitions (corresponding to program invariants) required for the verification of the properties of interest. Our generalization strategy combines CHR manipulation of array constraints with the widening and convex hull operators for linear constraints considered in the field of abstract interpretation \([10]\). Generalization is controlled by means of additional constraints that relate the variable identifiers in the given imperative programs and the CLP representations of their values.

(3) Finally, we present an implementation of the method in the VeriMAP system \([14]\), and we demonstrate its effectiveness on a set of benchmark programs taken from the literature.

2 The Transformation-Based Verification Method

In this section we introduce a class of Constraint Logic Programs with constraints on integers and integer arrays, and we show how partial correctness properties of imperative programs can be encoded as programs of that class.

First we need the following definitions. An atomic integer constraint is either \(p_1 = p_2\), or \(p_1 \geq p_2\), or \(p_1 > p_2\), where \(p_1\) and \(p_2\) are linear polynomials with integer variables and coefficients (sum and multiplication are denoted by \(+\) and \(\ast\), respectively). An atomic array constraint is either \(\text{dim}(a, n)\) denoting that the dimension of the array \(a\) is \(n\), or \(\text{read}(a, i, v)\) denoting that the \(i\)-th element of the array \(a\) is the integer \(v\), or \(\text{write}(a, i, v, b)\) denoting that the array \(b\) is equal to the array \(a\), except that its \(i\)-th element is \(v\). The \text{read} and \text{write} constraints satisfy the following axioms \([7, 23]\), where variables are universally quantified at the front:

\[
\begin{align*}
(A1) & \quad I = J, \text{read}(A, I, U), \quad \text{read}(A, J, V) \rightarrow U = V \quad \text{(array congruence)} \\
(A2) & \quad I = J, \text{write}(A, I, U, B), \quad \text{read}(B, J, V) \rightarrow U = V \quad \text{(read-over-write: case =)} \\
(A3) & \quad I \neq J, \text{write}(A, I, U, B), \quad \text{read}(B, J, V) \rightarrow \text{read}(A, J, V) \quad \text{(read-over-write: case \neq)}
\end{align*}
\]

A constraint is either \text{true}, or an atomic (integer or array) constraint, or a conjunction of constraints. An atom is a formula of the form \(p(t_1, \ldots, t_n)\), where \(p\) is a predicate symbol not in \(\{=, \geq, >, \text{dim, read, write}\}\) and \(t_1, \ldots, t_n\) are terms constructed out of variables, constants, and function symbols different from \(+\) and \(\ast\). A CLP program is a finite set of clauses of the form \(A : - c, B\), where \(A\) is an atom, \(c\) is a constraint, and \(B\) is a (possibly empty) conjunction of
atoms. Given a clause \( A : - c, B \), the atom \( A \) is called the head, and \( c, B \) is called the body. We assume that in every clause all integer arguments in its head are distinct variables. A clause \( A : - c \) is called a constrained fact. If \( c \) is true, then it is omitted and the constrained fact is called a fact. A CLP program is said to be linear if all its clauses are of the form \( A : - c, B \), where \( B \) consists of at most one atom.

An \( \mathcal{A} \)-interpretation \( I \) is a set \( D \), together with a function \( f \) in \( D^n \to D \) for each function symbol \( f \) of arity \( n \), and a relation \( p \) on \( D^n \) for each predicate symbol \( p \) of arity \( n \), such that: (i) the set \( D \) is the Herbrand universe [36] constructed out of the set \( \mathbb{Z} \) of the integers, the constants, and the function symbols different from \(+\) and \((ii) I \) assigns to the symbols \(+, *, =, \geq, >\) the usual meaning in \( \mathbb{Z} \), (iii) for all sequences \( a_0 \ldots a_{n-1} \), for all integers \( d \), \( \dim(a_0 \ldots a_{n-1}, d) \) is true in \( I \) iff \( d = n \), (iv) for all sequences \( a_0 \ldots a_{n-1} \) and \( b_0 \ldots b_{n-1} \) of integers, for all integers \( i \) and \( v \), \( \text{read}(a_0 \ldots a_{n-1}, i, v) \) is true in \( I \) iff \( 0 \leq i \leq n-1 \) and \( v = a_i \), and \( \text{write}(a_0 \ldots a_{n-1}, i, v, b_0 \ldots b_{n-1}) \) is true in \( I \) iff \( 0 \leq i \leq n-1 \), \( n = m \), \( b_i = v \), and for \( j = 0, \ldots, n-1 \), if \( j \neq i \) then \( a_j = b_j \), \( (v) I \) is an Herbrand interpretation [36] for function and predicate symbols different from \(+, *, =, \geq, >\), \( \dim, \text{read}, \text{and write.} \)

We can identify an \( \mathcal{A} \)-interpretation \( I \) with the set of all ground atoms that are true in \( I \), and hence \( \mathcal{A} \)-interpretations are partially ordered by set inclusion. A constraint \( c \) is said to be satisfiable if \( \mathcal{A} \models \exists(c) \), where in general, for every formula \( \varphi \), \( \exists(\varphi) \) denotes the existential closure of \( \varphi \). We say that \( I \) is an \( \mathcal{A} \)-model of \( \varphi \) if \( \varphi \) is true in \( I \). We write \( \mathcal{A} \models \varphi \) if every \( \mathcal{A} \)-interpretation is an \( \mathcal{A} \)-model of \( \varphi \). In particular, every \( \mathcal{A} \)-interpretation is an \( \mathcal{A} \)-model of Axioms (A1)–(A3). A constraint \( c \) entails a constraint \( d \), denoted \( c \sqsubseteq d \), if \( \mathcal{A} \models \forall(c \to d) \). By \( \text{vars}(\varphi) \) we denote the free variables of \( \varphi \). The semantics of a CLP program \( P \) is the least \( \mathcal{A} \)-model of \( P \), denoted \( M(P) \) and constructed as usual for CLP programs [29].

We consider imperative programs with integer and array variables. Every program has a single \texttt{halt} command whose execution causes the program to terminate. The semantics of programs is defined in terms of a transition relation, denoted \( \Rightarrow \), between configurations. A configuration is a pair \( \langle c, \delta \rangle \) of a labeled command \( c \) and an environment \( \delta \) that maps: (i) every integer variable identifier \( x \) to its value \( v \), and (ii) every integer array identifier \( a \) to a finite sequence \( a_0 \ldots a_{n-1} \) of integers, where \( n \) is the dimension of the array \( a \). The transition relation specifies the ‘small step’ operational semantics and its definition is similar to that in [44] and is omitted. An environment \( \delta \) is said to satisfy a formula \( \varphi(z_1, \ldots, z_r) \) iff \( \varphi(\delta(z_1), \ldots, \delta(z_r)) \) holds.

Given two formulas \( \varphi_{\text{init}} \) and \( \varphi_{\text{error}} \) that are disjunctions of constraints with free variables \( z_1, \ldots, z_r \), we say that program \( \text{prog} \) is incorrect with respect to these formulas iff there exist two environments \( \delta_{\text{init}} \) and \( \delta_{\text{h}} \) such that: (i) \( \delta_{\text{init}} \) satisfies \( \varphi_{\text{init}} \), (ii) \( \langle \ell_0 : c_0, \delta_{\text{init}} \rangle \Rightarrow \langle \ell_h : \texttt{halt}, \delta_{\text{h}} \rangle \), and (iii) \( \delta_{\text{h}} \) satisfies \( \varphi_{\text{error}} \), where \( \ell_0 : c_0 \) is the first labeled command of \( \text{prog} \) and \( \ell_h : \texttt{halt} \) is the unique \texttt{halt} command of \( \text{prog} \). A program is said to be correct if it is not incorrect. (In [11] the reader may find an extension of these definitions where \( \varphi_{\text{init}} \) and \( \varphi_{\text{error}} \) are predicates defined by any CLP program.) Our notion of correctness
is equivalent to the Hoare notion of partial correctness specified by the triple \( \{ \varphi_{\text{init}} \} \ \text{prog} \ \{ \neg \varphi_{\text{error}} \} \).

We translate the problem of checking whether or not the program \( \text{prog} \) is \textit{incorrect} into the problem of checking whether or not the atom \text{incorrect} is a consequence of the following CLP program \( T \):

\[
\text{incorrect} :- \ \text{errorConf}(X), \ \text{reach}(X).
\]

\[
\text{reach}(Y) :- \ \text{tr}(X, Y), \ \text{reach}(X).
\]

\[
\text{reach}(Y) :- \ \text{initConf}(Y).
\]

where \( \text{initConf}(X) \), \( \text{errorConf}(X) \), and \( \text{tr}(X, Y) \) are defined by CLP clauses so that the following conditions hold. For all configurations \( X \) and \( Y \), (i) \( \text{initConf}(X) \) holds iff \( X \) is an initial configuration, that is, a configuration of the form \( \langle \ell_0 : \alpha_0, \delta_{\text{init}} \rangle \) and \( \delta_{\text{init}} \) satisfies \( \varphi_{\text{init}} \), (ii) \( \text{errorConf}(X) \) holds iff \( X \) is an error configuration, that is, a configuration of the form \( \langle \ell_h : \text{halt}, \delta_h \rangle \) and \( \delta_h \) satisfies \( \varphi_{\text{error}} \), and (iii) \( \text{tr}(X, Y) \) holds iff \( X \longrightarrow Y \) holds.

\( \text{reach}(Y) \) holds iff the configuration \( Y \) can be reached from a configuration \( X \) whose environment satisfies \( \varphi_{\text{init}} \). Program \( \text{prog} \) is correct with respect to \( \varphi_{\text{init}} \) and \( \varphi_{\text{error}} \) iff \( \text{incorrect} \notin M(T) \).

Our verification method applies unfold/fold rules to the initial program \( T \) and consists of following two steps [13]. (i) \textit{VCGen}: the Generation of the Verification Conditions, and (ii) \textit{VCTransf}: the Satisfiability Checking of the Verification Conditions. The soundness of our method follows from the fact that for each program \( U \) obtained from \( T \) by applying the unfold/fold rules, \( \text{incorrect} \in M(T) \) iff \( \text{incorrect} \in M(U) \).

\textit{VCGen} performs a specialization of program \( T \) with respect to the given \( \text{tr} \) (which depends on \( \text{prog} \) ), \( \text{initConf} \), and \( \text{errorConf} \) predicates, thereby deriving a new program \( T_1 \), whose clauses are said to be the verification conditions for \( \text{prog} \) such that \( \text{tr} \) does not occur in \( T_1 \) (for this reason this step is also called the removal of the interpreter). During this specialization step all occurrences of the \( \text{dim} \) predicate are replaced by suitable constraints on the indexes of the arrays. We say that verification conditions are satisfiable iff \( \text{incorrect} \notin M(T_1) \), and thus their satisfiability guarantees that \( \text{prog} \) is correct with respect to \( \varphi_{\text{init}} \) and \( \varphi_{\text{error}} \). \textit{VCTransf}, which will be described in detail in Section 3, checks the satisfiability of the verification conditions generated at the end of \textit{VCGen}.

Before starting the specialization, \textit{VCGen} adds to the initial program \( T \) some additional constraints that are needed for controlling the generalization strategy described in Section 3.3. These constraints use the predicate \( \text{val} \) that relates some of the variable identifiers occurring in the imperative program \( \text{prog} \) and the CLP representation of their values. The meaning of the \( \text{val} \) constraints is as follows: for every variable identifier \( i \) of the program \( \text{prog} \), for every value \( I \), the constraint \( \text{val}(i, I) \) (where \( i \) is a constant uniquely associated with \( i \)) holds iff there exists a configuration whose environment \( \delta \) maps \( i \) to \( I \). These \( \text{val} \) constraints will be used by our generalization strategy to distinguish among different \textit{read} constraints, thereby making the strategy more effective as confirmed by the experimental results reported in Section 4. For instance, the constraint ‘\( \text{val}(i, I), \text{val}(j, J), \text{read}(A, I, U), \text{read}(A, J, V) \)’ expresses the property that the
first \texttt{read} gets the array element at index \(i\) and the second \texttt{read} gets the array element at index \(j\), while without the \texttt{val} constraints, \texttt{\{read(A, I, U), read(A, J, V)\}} does not express this property.

Now, let us see our verification method in action on a simple example. Let us consider the following program that, given the array \(a[0..(n-1)]\) and any \(i \in \{0, \ldots, n-1\}\) places in \(a[n_i-1]\) the maximum value of the leftmost portion \(a[0..(n_i-1)]\) by iteratively swapping adjacent elements.

\[
\text{bubblesort-inner: for} \ (j = 0; \ j < n_i-1; \ j++) \ { \ \ \ \ \ \ \}
\text{if} \ (a[j] > a[j+1]) \ { \ \ \ \ \ \ \}
\text{tmp} = a[j]; \ a[j] = a[j+1]; \ a[j+1] = \text{tmp}; \} \}
\]

Let us also consider the two properties \(\varphi_{\text{init}}(i, n, a) \equiv 0 \leq i < n \land \dim(a, n)\) and \(\varphi_{\text{error}}(i, j, n, a) \equiv \exists k \exists x \exists y 0 \leq i < n \land 0 \leq k < j \land j < n_i-1 \land \text{read}(a, k, x) \land \text{read}(a, j, y) \land x > y\).

These two properties are expressed in CLP as follows:

\[
\begin{align*}
\phi\text{Init}(I, N, A) : & - 0 \leq I, I < N, \dim(A, N). \\
\phi\text{Error}(I, J, N, A) : & - 0 \leq I, I < N, 0 \leq K, K < J, J = N - I - 1, X > Y, \text{read}(A, X, X), \text{read}(A, J, Y), \text{val}(K, K), \text{val}(J, J). \\
\end{align*}
\]

Note the two \texttt{val} constraints that relate the index variables \(k\) and \(j\) to their respective values \(K\) and \(J\), respectively. At the end of \texttt{VCGen} we get the following CLP program \(T_1\) that expresses the verification conditions for the program \texttt{bubblesort-inner}:

1. \texttt{incorrect} : - 0 \leq I, 0 \leq K, K \leq J, J = N - I - 1, X > Y, \text{read}(A, X), \text{read}(A, J, Y), \text{val}(K, K), \text{val}(J, J), \text{new1}(I, J, N, A, Tmp, K).

2. \texttt{new1}(I, J, N, A2, W, K) : - \text{J1} = I + 1, J < N - I - 1, J \geq 0, J < N - 1, X > Y, \text{read}(A, J, X), \text{read}(A, J1, Y), \text{read}(A, J, W), \text{read}(A, J1, Z), \text{write}(A, J, Z, A1), \text{write}(A1, J1, W, A2), \text{val}(J1), \text{val}(J, J), \text{val}(K, K), \text{new1}(I, J, N, A, Tmp, K).

3. \texttt{new1}(I, J, N, A, Tmp, K) : - \text{J1} = J + 1, J < N - I - 1, J \geq 0, J < N - 1, X \leq Y, \text{read}(A, J, X), \text{read}(A, J1, Y), \text{val}(J1), \text{val}(J, J), \text{val}(K, K), \text{new1}(I, J, N, A, Tmp, K).

4. \texttt{new1}(I, J, N, A, Tmp, K) : - 0 \leq I, I < N, J = 0, \text{val}(J, J), \text{val}(K, K).

where \texttt{new1} is a new predicate symbol introduced during program specialization by \texttt{VCGen}. The definition of the predicate \texttt{new1} is associated with the \texttt{for}-loop of the \texttt{bubblesort-inner} program and consists of clauses 2–4 that represent the execution of the \texttt{for} statement. In particular, we have that (see the underlined constraints): (i) clauses 1 and 4 represent the exit and the entry of the \texttt{for}-loop, respectively; and (ii) clauses 2 and 3 represent the execution of the conditional in the \(a[j] > a[j+1]\) case and in the \(a[j] \leq a[j+1]\) case, respectively.

3 A Transformation Strategy for Verification

The \texttt{VCTransf} step of our verification method transforms the CLP program \(T_1\) derived at the end of \texttt{VCGen} to a program \(T_2\) such that \texttt{incorrect} \(\in \mathcal{M}(T_1)\) iff \texttt{incorrect} \(\in \mathcal{M}(T_2)\). This transformation makes use of transformation rules that preserve the least \(\mathcal{A}\)-model semantics of CLP programs. In particular, we apply the following rules, that are collectively called unfold/fold rules: \texttt{unfolding}, \texttt{constraint replacement}, \texttt{clause removal}, \texttt{definition}, and \texttt{folding}. These rules are an adaptation to CLP programs of the unfold/fold rules for general CLP programs (see, for instance, [18]).
VCTransf applies the unfold/fold rules according to a strategy that performs the propagation of the constraints of the error property $\phi_{\text{Error}}$ in a backward way from the error configuration towards the initial configuration, so as to derive a program $T_2$ where the predicate $\text{incorrect}$ is defined by either (i) the fact $\text{incorrect}$ (in which case the imperative program $\text{prop}$ is incorrect), or (ii) the empty set of clauses (in which case $\text{prop}$ is correct). In the case where neither (i) nor (ii) holds, that is, in program $T_2$ the predicate $\text{incorrect}$ is defined by a non-empty set of clauses not containing the fact $\text{incorrect}$, we cannot conclude anything about the correctness of $\text{prop}$. However, similarly to what has been proposed in [12], in this case we can perform again VCTransf by propagating the initial property $\phi_{\text{Init}}$, and continue alternating the propagation of the error and initial properties in the hope of deriving a program where either (i) or (ii) holds. Obviously, due to the undecidability of program correctness, it may be the case that this process does not terminate.

3.1 The Transformation Strategy

VCTransf is performed by applying the unfold/fold transformation rules according to the Transform strategy shown in Figure 1. Let us briefly describe the various rules used by the Transform strategy.

- The UNFOLDING rule performs one step of backward propagation of the error property $\phi_{\text{Error}}$.
- The CONSTRAINT REPLACEMENT rule infers new constraints on the variables of the single atom that occurs in the body of each clause obtained by UNFOLDING. CONSTRAINT REPLACEMENT makes use of a function $\text{Repl}$ that, given a clause $C$ of the form $H : \neg c_0, B$, returns a set $\{H : \neg c_1, B, \ldots, H : \neg c_n, B\}$ of clauses (with $n \geq 0$), where $c_1, \ldots, c_n$ are constraints such that $A \models \forall (\exists x_0 c_0) \leftrightarrow (\exists x_1 c_1 \lor \ldots \lor \exists x_n c_n)$ holds, and for $i=0, \ldots, n$, we have that $x_i = \text{vars}(c_i) - \text{vars}(H, B)$. In particular, if $c_0$ is unsatisfiable, then $n = 0$ and clause $C$ is removed. The function $\text{Repl}$ is implemented by a CHR program as described in Section 3.2.
- The rules of REMOVAL OF USELESS CLAUSES and REMOVAL OF SUBSUMED CLAUSES remove clauses that do not contribute to the least model of the CLP program at hand.
- The DEFINITION rule introduces new predicate definitions by suitable generalizations of the constraints. Generalization is performed by using a function $\text{Gen}$ such that, for any given clause $E$ of the form $H : \neg e(V, X), p(X)$ and set $\text{Defs}$ of predicate definitions, $\text{Gen}(E, \text{Defs})$ is a clause of the form $\text{newq}(X) : \neg \text{gen}(X), p(X)$, where: (i) $\text{newq}$ is a new predicate symbol, and (ii) $\text{gen}(X)$ is a constraint such that $e(V, X) \subseteq \text{gen}(X)$.
- The FOLDING rule replaces the clause $H : \neg e(V, X), p(X)$ by the clause $H : \neg \text{e}(V, X), \text{newq}(X)$.

Note that the input program $T_1$ of the Transform strategy is a linear CLP program. Indeed, during VCGen the atoms different from $\text{reach}$ are unfolded and hence a linear program is generated.

The new predicates introduced by the DEFINITION rule can be understood as over-approximations of the sets of configurations that are backward-reachable.
from the error configuration. Note, however, that the folding rule preserves equivalence, as \( e(V,X), p(X) \) is equivalent to \( e(V,X), \text{new}q(X) \). In Section 3.3 we present a generalization function that guarantees the termination of Transform and, at the same time, allows us to prove the correctness of non-trivial programs.

**Input:** A linear CLP program \( T_1 \).

**Output:** Program \( T_2 \) such that \( \text{incorrect} \in M(T_1) \) iff \( \text{incorrect} \in M(T_2) \).

**Initialization:**

Let \( \text{InDefs} \) be the set of all clauses of \( T_1 \) whose head is the atom \( \text{incorrect} \);

\[
T_2 := \emptyset; \quad \text{Defs} := \text{InDefs};
\]

while in \( \text{InDefs} \) there is a clause \( C \) of the form \( H :- c, A \) do

**Unfolding:** Let \( \{ K_i :- c_i, B_i | i = 1, \ldots, m \} \) be the set of the (renamed apart) clauses of \( T_1 \) such that, for \( i = 1, \ldots, m \), \( A \) is unifiable with \( K_i \) via the most general unifier \( \vartheta_i \).

Then \( \text{TransfC} := \{ (H :- c, K_i, B_i | i = 1, \ldots, m) \} \);

**Constraint Replacement:** \( \text{TransfC} := \cup_{D \in \text{TransfC}} \text{Repl}(D) \);

**Removal of Subsumed Clauses:** Remove from \( \text{TransfC} \) every clause \( H :- d, B \) such that there exists a distinct clause \( H :- e \) in \( \text{TransfC} \) with \( d \sqsubseteq e \);

**Definition & Folding:**

while in \( \text{TransfC} \) there is a clause \( E \) of the form \( H :- e(V,X), \text{p}(X) \), where \( e(V,X) \) is a constraint and \( p \) is a predicate defined in \( T_1 \) do

if in \( \text{Defs} \) there is a clause \( D \) of the form \( \text{new}p(X) :- c(X), \text{p}(X) \), where \( c \) is a constraint such that \( e(V,X) \sqsubseteq c(X) \) then \( \text{TransfC} := (\text{TransfC} \setminus \{ E \}) \cup \{ H :- e(V,X), \text{new}p(X) \} \);

else let \( \text{Gen}(E, \text{Defs}) \) be \( \text{new}q(X) :- \text{gen}(X), \text{p}(X) \);

\[
\text{Defs} := \text{Defs} \cup \{ \text{Gen}(E, \text{Defs}) \};
\]

\[
\text{InDefs} := (\text{InDefs} \setminus \{ C \}) \cup \{ \text{Gen}(E, \text{Defs}) \};
\]

\[
\text{TransfC} := (\text{TransfC} \setminus \{ E \}) \cup \{ H :- e(V,X), \text{new}q(X) \}
\]

end-while;

\[
T_2 := T_2 \cup \text{TransfC}
\]

**Removal of Useless Clauses:**

Remove from \( T_2 \) all clauses with head predicate \( p \), if in \( T_2 \) there is no constrained fact \( q(\ldots) :- c \) where \( q \) is either \( p \) or a predicate on which \( p \) depends.

**Fig. 1.** The Transform strategy.

We assume that the set \( \text{Defs} \) is structured as a tree of clauses where, with reference to Figure 1, clause \( C \) is said to be the **parent** of clause \( \text{Gen}(E, \text{Defs}) \), and the **ancestor** relation is defined as the reflexive, transitive closure of the parent relation.

### 3.2 Constraint Replacement via CHR

In this section we show how Constraint Handling Rules with disjunction can be used to realize in a very natural way the constraint rewritings based on Ax-
ions (A1)–(A3) for array operations, which allow us to apply the Constraint Replacement rule during the Transform strategy.

CHR is a committed-choice language based on rewriting rules. It was specifically designed for building custom constraint solvers [22]. A CHR program consists of a set of guarded rules that rewrite multisets of constraints. Constraint predicates are of two different kinds: (i) built-in constraints, whose entailment is checked by using a domain-specific constraint solver, and (ii) user-defined constraints, which are rewritten as specified by the CHR program. We assume that the set of built-in constraints contains the constraints true, false, and syntactic equalities. Built-in constraints and user-defined constraints are closed under conjunction. A constraint goal is either a (built-in or user-defined) constraint, or a conjunction of constraint goals, or a disjunction of constraint goals.

CHR rules are of the form: \( r \circ B \_ H_1 \_ H_2 \Leftrightarrow H \_ G \_ B \), where the \( \circ \) symbol separates the optional rule identifier \( r \) from the rest of the rule, the user-defined constraints \( H_1 \) and \( H_2 \) are the kept head and the removed head, respectively, the built-in constraint \( G \) is the guard, and \( B \) is a constraint goal. Either \( H_1 \) or \( H_2 \) is a non-empty conjunction. If \( H_2 \) is empty then the rule is called a propagation rule and can be written as follows: \( H_1 \Rightarrow G \_ B \). The logical meaning of the CHR rule \( H_1 \_ H_2 \Leftrightarrow G \_ B \) is the guarded equivalence \( \forall (G \rightarrow (\{H_1 \land H_2\} \Leftrightarrow (H_1 \land \exists Y B))) \), where \( Y \) is the set of variables occurring in \( B \) and not in the rest of the rule.

The operational semantics of CHR is formally defined in terms of a transition relation between CHR states as described in [1]. A CHR state is a triple \( \langle g, u, b \rangle \), where \( g \) is a constraint goal, \( u \) is a user-defined constraint and \( b \) is a built-in constraint. An initial state is a state of the form \( \langle g, \text{true}, \text{true} \rangle \). Starting from an initial state, constraints are rewritten as long as possible by applying CHR rules. A final state is a state from which no transition is applicable. A final state is failed if it is of the form \( \langle g, u, \text{false} \rangle \). Note that, since constraint goals may contain disjunctions, the transition relation is nondeterministic, and thus it generates a tree of computations whose leaves correspond to the final states. A terminating CHR program is one for which there is no infinite sequence of transitions, that is, the tree of computations is finite.

The CHR program \( \text{Arr} \) used for constraint replacement in the Transform strategy consists of the following rules:

\begin{verbatim}
ac \textcircled{\textbf{ac}} \textcircled{\textbf{read}}(A1, I, X)\_\textcircled{\textbf{read}}(A2, J, Y) \Leftrightarrow A1 \Leftrightarrow A2, I = J \mid X = Y.

cac \textcircled{\textbf{cac}} \textcircled{\textbf{read}}(A1, I, X)\_\textcircled{\textbf{read}}(A2, J, Y) \Rightarrow A1 \Leftrightarrow A2, X \leftrightarrow Y \mid I \leftrightarrow J.

row \textcircled{\textbf{row}} \textcircled{\textbf{write}}(A1, I, X, A2)\_\textcircled{\textbf{read}}(A3, J, Y) \Leftrightarrow A2 \Leftrightarrow A3 \mid (I = J, X = Y); (I \leftrightarrow J, \text{\textbf{\textbf{read}}}(A1, J, Y)).
\end{verbatim}

These rules encode the axioms (A1)–(A3) presented in Section 2. Rules \textbf{ac} and \textbf{cac} encode the array congruence axiom (A1) and its contrapositive version, respectively, and rule \textbf{row} encodes the two so-called read-over-write axioms (A2) and (A3). The symbol ‘\Leftrightarrow’ denotes syntactic equality, while ‘\Leftrightarrow’ and ‘\leftrightarrow’ denote integer equality and inequality, respectively. Note that we use the semicolon ‘;’ for denoting disjunction in the right-hand side of the rule \textbf{row}.

If we adopt an operational semantics that prevents trivial non-termination cases by applying a propagation rule at most once to the same constraints [1], then it can be shown that the CHR program \textbf{Arr} terminates for all constraint
goals generated during the application of our transformation strategy. Indeed, the only rule that may lead to a non-terminating behavior is \texttt{row}. By using this rule, a constraint containing

\[(g1) \texttt{write}(U,I,X,V), \texttt{write}(V,I,H,U), \texttt{read}(V,J,Y)\]

could be rewritten as a constraint containing

\[(g2) \texttt{write}(U,I,X,V), \texttt{write}(V,I,H,U), \texttt{read}(U,J,Y)\]

and then, by interchanging the roles of the two \texttt{write} constraints in the application of the \texttt{row} rule, a constraint containing \((g2)\) could be rewritten to a constraint containing \((g1)\), thereby giving rise to an infinite branch in the tree of computation. However, it can be shown that a constraint goal of the form \((g1)\) cannot be generated by the Unfolding rule during the application of the Transform strategy. Informally, in every clause, the constraints can be ordered from left to right following the order of execution of the corresponding \texttt{read} and \texttt{write} operations, and hence a variable \(V\) occurring in a constraint of the form \texttt{write}(U,I,X,V), does not occur to the left of that constraint. This argument is formalized by considering the transitive closure \(\prec^+\) of the following relation between the variables of a clause: \(U \prec V\) iff the constraint \texttt{write}(U,I,X,V) occurs in the clause. It can be shown that in every clause derived by the Unfolding rule during the application of the Transform strategy, \(\prec^+\) is irreflexive. Thus, the termination of \texttt{Arr} follows from the fact that an application of the \texttt{row} rule will replace a constraint of the form \texttt{read}(V,J,Y) by a constraint of the form \texttt{read}(U,J,Y) with \(U \prec V\).

Given a clause \(D\) of the form \(H : = d, B\), derived by the Unfolding rule, let \(\{\langle g_1, u_1, b_1 \rangle, \ldots, \langle g_n, u_n, b_n \rangle\}\) be the set of all non-failed final states computed from the initial state \(\langle d, \text{true}, \text{true} \rangle\). Let \(d_i\) be the conjunction \((g_i, u_i, b_i)\). We assume that, for \(i = 1, \ldots, n\), the variables occurring in \(d_i\) and not in \(d\) are fresh, and thus they occur neither in \(H\) nor in \(B\). By the soundness of CHR we have that \(A \models \forall (d \leftrightarrow (\exists X_1 d_1 \lor \ldots \lor \exists X_n d_n))\) where, for \(i = 1, \ldots, n\), \(X_i = \text{vars}(d_i) \setminus \text{vars}(d)\). Thus, the applicability conditions of the Constraint Replacement rule are satisfied, and in the Transform strategy we define \texttt{Repl}(D) to be \(\{H : = d_1, B, \ldots, H : = d_n, B\}\).

To see how the CHR program \texttt{Arr} works, let us consider again the \texttt{bubblesort-inner} example of Section 3. By applying the Unfolding rule to clause 1 the Transform strategy derives a set of clauses including the following one:

\[
\text{new2}(I,J,N,A2,W,K) : = J_1 = 1 + J, J < N - I - 1, K \leq J, Z < W, I \geq 0, K \geq 0, J \geq N - I - 3, X > Y, \text{write}(A,J,Z,A1), \text{write}(A1,J1,W,A2), \text{read}(A,J,W), \text{read}(A,J1,Z), \text{read}(A2,K,X), \text{read}(A2,J1,Y), \text{val}(j,J1), \text{val}(k,K), \text{val}(j,J), \text{new1}(I,J,N,A,Tmp,K). \]

The CHR program \texttt{Arr} rewrites the constraint occurring in the above clauses and the Constraint Replacement rule derives the following clause:

\[
\text{new2}(I,J,N,A2,W,K) : = J_1 = 1 + J, J < N - I - 1, K \leq J, Z < W, I \geq 0, K \geq 0, J \geq N - I - 3, X > Y, \text{write}(A,J,Z,A1), \text{write}(A1,J1,W,A2), \text{read}(A,J,Y), \text{read}(A,J1,Z), \text{read}(A,K,X), Y = W, J > K, J_1 > K, \text{val}(j,J1), \text{val}(k,K), \text{val}(j,J), \text{new1}(I,J,N,A,Tmp,K). \]

123
where (i) by row, the constraint \( \text{read}(A_2, J_1, Y) \) has been replaced by the equality constraint \( Y = W \) (ii) by row, in the constraint \( \text{read}(A_2, K, X) \), the variable \( A_2 \), denoting the array \( a \) after the write operation, has been replaced by the variable \( A \), denoting the array \( a \) before the write operation, and (iii) the constraint \( 'J > K, J_1 > K' \) has been added by the built-in solver on linear constraints.

### 3.3 Generalization Strategy

The most critical step of the Transform strategy is the introduction of new predicates during DEFINITION & FOLDING. Indeed, it should guaranteed that a finite number of new predicates is introduced, to avoid the non-termination of Transform. For this reason, as usual in many program transformation techniques [19], we collect in the set \( \text{Defs} \) all predicate definitions introduced by the strategy, and before introducing a new predicate definition \( D \), we match it against the ones already in \( \text{Defs} \). If \( D \) is ‘similar’ to a definition \( A \) in \( \text{Defs} \) (formalized via the embedding relation defined below), then the function \( \text{Gen} \) introduces a new definition which is a generalization of \( A \) and \( D \), instead of \( D \). The function \( \text{Gen} \) defined in this section, makes use of operators for generalizing array constraints that ensure that no infinite number of distinct generalizations can be obtained, and hence a finite number of new predicates is introduced during the Transform strategy. The embedding relation and the generalization strategy take into consideration the \( \text{val} \) constraints between the integer CLP variables occurring in \( \text{read} \) constraints and the identifiers of the imperative program with which they are associated. By doing so we will be able to identify similarities between definitions that go beyond syntactic variance, hence improving the level of precision of the verification technique.

In the following we will denote constraints as conjunctions of the form \( i, r, w, v \), where \( i \) is an integer constraint, and \( r, w, \) and \( v \) are conjunctions of \( \text{read}, \text{write}, \) and \( \text{val} \) constraints, respectively. We assume that all integer variables in \( \text{read} \) constraints are distinct and do not occur in any (non constraint) atom of the clause at hand (this condition can always be satisfied by adding some integer equalities).

Given a clause \( D \) of the form \( H : -i, r, w, v, B \), for every integer variable \( I \) occurring in a \( \text{read} \) atom in \( r \) we compute the set \( \text{ids}(I) \) of identifiers \( \text{id} \) such that an atom \( \text{val}(\text{id}, J) \) occurs in \( v \) and the constraint \( I = J \) is entailed by \( i \). We define the clause identifier set of \( D \), denoted \( \text{ids}(D) \), as the set of pairs \( (\text{ids}(I), \text{ids}(V)) \) such that a constraint of the form \( \text{read}(A, I, U) \) occurs in \( r \). For example, if the constraint occurring in the body of clause \( D \) is

\[
M = 0, \ N > M, \ V = 0, \ \text{read}(A, M, U), \ \text{read}(A, N, V), \ \text{val}(m, H), \ \text{val}(n, N), \ \text{val}(v, V)
\]

then we have that \( \text{ids}(D) = \{\{(m, v), \{\}\}, \{(n), \{m, v\}\}\} \).

Given two clause identifier sets \( R_1 \) and \( R_2 \), we say that \( R_1 \) is embedded into \( R_2 \) via the set relation \( \text{rel} \) iff for each pair \( (I_1, U_1) \) in \( R_1 \) there exists a pair \( (I_2, U_2) \) in \( R_2 \) such that (i) \( \text{rel}(I_1, I_2) \) and \( \text{rel}(U_1, U_2) \) hold and (ii) \( R_1 - \{(I_1, U_1)\} \) is embedded into \( R_2 - \{(I_2, U_2)\} \) via \( \text{rel} \). In our experiments we have considered two embedding relations based on the following definitions of \( \text{rel}(s_1, s_2) \): (1) \( s_1 \subseteq s_2 \) (subset relation), and (2) \( s_1 \cap s_2 \) defined as \( s_1 = s_2 = 0 \) \( \lor (s_1 \cap s_2 \neq 0) \).
We say that a clause \( D_1 \) is embedded into a clause \( D_2 \) via the relation \( rel \) iff \( ids(D_1) \) is embedded in \( ids(D_2) \) via \( rel \).

Given a clause \( E \) of the form \( H : \ e(V,X), p(X) \) and a set \( Defs \) of definitions, the generalization function \( Gen \) computes a definition \( newq(X) : \ gen(X), p(X) \), where \( newq \) is a new predicate symbol and \( gen(X) \) is a constraint such that \( e(V,X) \subseteq gen(X) \), which is constructed as follows. Let \( e(V,X) \) be of the form \( i,r,w,v \) and let \( newq(X) : \ i_X,r_X,v_X,p(X) \) be the candidate definition clause for \( E \), where: (i) \( r_X \) is the conjunction of the \( read(A,I,V) \) constraints in \( r \) such that \( A \) occurs in \( X \) and, for some \( val(j,J) \) in \( v \) we have that \( J \) occurs in \( X \) and either \( I = J \) or \( V = J \) is entailed by \( i \), (ii) \( i_X \) is the constraint obtained from \( i \) by projecting away the variables not occurring in \( X \) or \( r_X \), and (iii) \( v_X \) is the conjunction of the \( val(j,J) \) constraints in \( v \) such that \( J \) occurs in \( X \).

Suppose that clause \( E \) has been derived from clause \( C \) at the end of the removal of subsumed clauses step. \( Gen(E,Ddefs) \) is defined as follows.

If in \( Ddefs \) there is an ancestor \( A \) of \( C \) of the form \( H_0 : \ i_0,r_0,v_0,p(X) \), such that \( r_0 \) is a subconjunction of \( r_X \), and \( A \) is a subdefinition of \( A \), then let \( i_X \) be the constraint obtained from \( i_0 \) by projecting away the variables not occurring in \( X \) or \( r_0 \); compute a generalization \( g \) of the constraints \( i_X \) and \( i_0 \) such that \( i_X \subseteq g \), by using a generalization operator for linear constraints. Define the constraint \( gen(X) \) as \( g,r_X,v_X \); Else define the constraint \( gen(X) \) as \( i_X,r_X,v_X \).

For the projection and generalization operations we apply the usual operators for linear constraints on the reals (and in particular the widening and convex hull generalization operators defined in \([10, 19, 40]\)). These operators are correct because they guarantee that \( i \subseteq g \).

To see an example of application of the generalization strategy let us consider the clause that was derived in Section 3.2 by applying the constraint replacement rule. The candidate definition for that clause is:

\[
new4(I,J,N,A,Tmp,K) : \ J < N - I - 1, \ I \geq 0, \ K \geq 0, \ J \geq N - I - 3, \ X > W, \ J > K, \\
read(A,J,W), \ read(A,K,X), \ val(k,K), \ val(j,J), \ new1(I,J,N,A,Tmp,K).
\]

and \( Ddefs \) contains the following ancestor definition:

\[
new2(I,J,N,A,Tmp,K) : \ J < N - I - 1, \ I \geq 0, \ K \geq 0, \ J \geq N - I - 2, \ X > W, \ J > K, \\
read(A,J,W), \ read(A,K,X), \ val(k,K), \ val(j,J), \ new1(I,J,N,A,Tmp,K).
\]

Since the ancestor definition is embedded into the candidate definition via \( \subseteq \) or \( @ \) (indeed, the two clauses have the same clause identifier set \( \{\{j\}, \{i\}\} \)), we obtain a generalization of the candidate definition by applying the widening operator between the linear constraints, hence dropping the constraint \( J \geq N - I - 2 \) of the ancestor definition, and we introduce the following generalized definition:

\[
new4(I,J,N,A,Tmp,K) : \ J < N - I - 1, \ I \geq 0, \ K \geq 0, \ X > W, \ J > K, \\
read(A,J,W), \ read(A,K,X), \ val(k,K), \ val(j,J), \ new1(I,J,N,A,Tmp,K).
\]

The correctness of the Transform strategy with respect to the least \( A \)-model semantics follows from the correctness results for the unfold/fold rules proved in \([18]\).
The termination of the \textit{Transform} strategy is based on the following facts:
(i) Constraint satisfiability and entailment are checked by a terminating solver (note that completeness is not necessary for the termination of \textit{Transform}).
(ii) The CHR program \texttt{Arr} implementing \textbf{CONSTRANT REPLACEMENT} terminates. (iii) The set of new clauses that, during the execution of the \textit{Transform} strategy, can be introduced by \textbf{DEFINITION} & \textbf{FOLDING} steps is finite. Indeed, by construction, they are all of the form \( H : - \ i, r, v, p(X) \), where: (1) \( X \) is a tuple of variables, (2) \( i \) is an integer constraint, (3) \( r \) is a conjunction of array constraints of the form \texttt{read}(A, I, V), where \( A \) is a variable in \( X \) and the variables \( I \) and \( V \) occur in \( i \) only, (4) the set of identifiers of the imperative program is finite, and hence the embedding relation is a \textit{thin well-quasi ordering} \cite{19} (this property guarantees that generalization is eventually triggered, and that a definition can be generalized a finite number of times only), (5) the cardinality of \( r \) is bounded, because if in \texttt{Defs} there exists a clause \( A \) of the form \( H_0 : - \ i_0, r_0, v_0, p(X) \), then generalization does not introduce a descendant definition clause \( D \) of the form \texttt{newp}(X) : - \( i_1, r_1, v_1, p(X) \) such that \( A \) is embedded into \( D \). (6) we assume that the generalization operator on linear constraints has the following \textit{finiteness} property: only finite chains of generalizations of any given constraint can be generated by applying the operator. The already mentioned generalization operators presented in \cite{10, 19, 40} satisfy this finiteness property. Thus, we have the following result.

**Theorem 1.** (i) The \textit{Transform} strategy terminates. (ii) Let program \( T2 \) be the output of \textit{Transform} applied to the input program \( T1 \). Then, \texttt{incorrect} \( \in M(T1) \) iff \texttt{incorrect} \( \in M(T2) \).

Let us now conclude our \textit{bubblesort-inner} example. After a few iterations, the outermost while-loop of the \textit{Transform} strategy terminates and produces the following set \( T2 \) of clauses (which we list as they have been automatically generated):

\begin{verbatim}
incorrect : A = -1+B-C, D = -1+B-C, E=F <= -1, G >= 0, C >= 0, B-G-C >= 2, read(H,D,E), read(H,G,F), val(j,A), val(k,G), new1(C,A,B,H,I,G).


\end{verbatim}
new4(A, B, C, D, E, F) : -G ≥ F+1, B = 1+G, H = 1+G, I = 1+G, ... of the one presented in [13].
The novel contributions of this paper are the following. (1) We have formalized the technique presented in this paper is an extension of the one presented in [13].

5 Related Work and Conclusions
The novel contributions of this paper are the following. (1) We have formalized

4 Experimental Evaluation
We have implemented our verification method as a module of the VeriMAP software model checker [14] (available at http://map.uniroma2.it/VeriMAP) and we have performed an experimental evaluation of our method on a benchmark set of programs taken from the literature [6, 9, 16, 27, 35] (the source code is available at http://map.uniroma2.it/smc/array-chr).

We have applied the Transform strategy presented in Section 3 using different generalization strategies that combine the widening and convex hull operators together with various embedding relations. Different embedding relations are obtained: (i) by selecting different sets of variable identifiers for the introduction of the val constraints, and (ii) by using different relations to compare sets of identifiers (see Section 3.3). In particular, we have considered the following generalization strategies: GenW,I, GenH,V, GenH,Y, GenH,I, and GenH,Y, where the subscripts should be interpreted as follows. The first subscript denotes the generalization operator: W stands for the widening operator, and H stands for the widening-and-convex-hull operator. The second subscript denotes the selected set of identifiers: I stands for the set of variable identifiers associated with the second argument (that is, the index) of the read constraints, and V stands for the set of identifiers associated with the third argument (that is, the value) of the read constraints. The third subscript denotes the relation rel ∈ {⊆, ⋓} that is used for comparing the sets of identifiers.

The results of our experiments are summarized in Table 1. The experiments have been performed on an Intel Core Duo E7300 2.66Ghz processor with 4GB of memory under GNU/Linux OS. We have that the strategies based on GenH,I,rel are more precise than those based on GenH,Y,rel, for any rel ∈ {⊆, ⋓}. Similarly, the strategies based on GenH,S,rel are more precise than those based on GenH,S,rel, for any S ∈ {I, V}. Note that by generalizing the constraints, the Transform strategy may get an empty set of identifiers associated with a given variable, thereby making the generalizations based on the operator ⋓ less useful that those based on the operator ⊆. The best trade-off between precision and performance is obtained by GenH,I,rel that allowed us to prove all programs we have considered. Note also that the bubblesort-inner program can be proved only by generalizations based on GenW,I or GenH,I.

5 Related Work and Conclusions
The technique presented in this paper is an extension of the one presented in [13]. The novel contributions of this paper are the following. (1) We have formalized
Table 1. Verification results using VeriMAP. Time is in seconds. By ‘unknown’ we indicate that VeriMAP terminates without being able to prove correctness or incorrectness. By ‘time-out’ we indicate that VeriMAP is unable to provide an answer within 5 minutes.

<table>
<thead>
<tr>
<th>Program</th>
<th>$Gen_{W,I}$</th>
<th>$Gen_{H,V} \subseteq$</th>
<th>$Gen_{H,V}$</th>
<th>$Gen_{H,I}$</th>
<th>$Gen_{H,I}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>bubblesort-inner</td>
<td>0.9</td>
<td>unknown</td>
<td>unknown</td>
<td>unknown</td>
<td>1.52</td>
</tr>
<tr>
<td>copy-partial</td>
<td>unknown</td>
<td>unknown</td>
<td>3.52</td>
<td>3.51</td>
<td>3.54</td>
</tr>
<tr>
<td>copy-reverse</td>
<td>unknown</td>
<td>unknown</td>
<td>5.25</td>
<td>unknown</td>
<td>5.23</td>
</tr>
<tr>
<td>copy</td>
<td>unknown</td>
<td>unknown</td>
<td>5.00</td>
<td>4.88</td>
<td>4.90</td>
</tr>
<tr>
<td>find-first-non-null</td>
<td>0.14</td>
<td>0.66</td>
<td>0.64</td>
<td>0.28</td>
<td>0.27</td>
</tr>
<tr>
<td>find</td>
<td>1.04</td>
<td>6.53</td>
<td>2.35</td>
<td>2.33</td>
<td>2.29</td>
</tr>
<tr>
<td>first-not-null</td>
<td>0.11</td>
<td>0.22</td>
<td>0.22</td>
<td>0.22</td>
<td>0.22</td>
</tr>
<tr>
<td>init-backward</td>
<td>unknown</td>
<td>1.04</td>
<td>1.04</td>
<td>1.03</td>
<td>1.04</td>
</tr>
<tr>
<td>init-non-constant</td>
<td>unknown</td>
<td>2.51</td>
<td>2.51</td>
<td>2.47</td>
<td>2.47</td>
</tr>
<tr>
<td>init-partial</td>
<td>unknown</td>
<td>0.9</td>
<td>0.89</td>
<td>0.9</td>
<td>0.89</td>
</tr>
<tr>
<td>init-sequence</td>
<td>unknown</td>
<td>4.38</td>
<td>4.33</td>
<td>4.11</td>
<td>4.29</td>
</tr>
<tr>
<td>init</td>
<td>unknown</td>
<td>1.00</td>
<td>0.97</td>
<td>0.98</td>
<td>0.98</td>
</tr>
<tr>
<td>insertionsort-inner</td>
<td>0.58</td>
<td>2.41</td>
<td>2.4</td>
<td>2.38</td>
<td>2.37</td>
</tr>
<tr>
<td>max</td>
<td>unknown</td>
<td>unknown</td>
<td>0.8</td>
<td>0.81</td>
<td>0.82</td>
</tr>
<tr>
<td>partition</td>
<td>0.84</td>
<td>1.77</td>
<td>1.78</td>
<td>1.76</td>
<td>1.76</td>
</tr>
<tr>
<td>rearrange-in-situ</td>
<td>unknown</td>
<td>unknown</td>
<td>3.06</td>
<td>3.01</td>
<td>3.03</td>
</tr>
<tr>
<td>selectionsort-inner</td>
<td>unknown</td>
<td>time-out</td>
<td>unknown</td>
<td>2.84</td>
<td>2.83</td>
</tr>
<tr>
<td>precision</td>
<td>6</td>
<td>10</td>
<td>15</td>
<td>15</td>
<td>17</td>
</tr>
<tr>
<td>total time</td>
<td>3.61</td>
<td>21.42</td>
<td>34.76</td>
<td>31.81</td>
<td>38.45</td>
</tr>
<tr>
<td>average time</td>
<td>0.60</td>
<td>2.14</td>
<td>2.31</td>
<td>2.12</td>
<td>2.26</td>
</tr>
</tbody>
</table>

constraint replacement as a CHR program representing the Theory of Arrays, whereas in [13] constraint replacement was implemented directly in CLP. We have shown that the approach based on CHR allows a very elegant combination of constraint manipulation with transformations based on unfold/fold rules.

(2) We have presented a novel strategy that controls the generalization of array constraints during CLP transformation by taking into account the information relating the variable identifiers in the imperative program and the CLP representation of their values. We have shown that our generalization strategy is effective on several examples taken from the literature.

In the Introduction we mentioned some CLP-based program verification methods. Here we briefly recall other methods, not based on CLP, for the verification of array programs.

Some of these methods use abstract interpretation. In [27], which builds upon [24], invariants are discovered by partitioning the arrays into symbolic slices and associating an abstract variable with each slice. A similar approach is followed in [9] where a scalable framework for the automatic analysis of array programs is introduced. In [21, 34] a predicate abstraction for inferring universally quantified properties of array elements is presented, and in [26] the authors present a similar technique which uses template-based quantified abstract domains. In [46] a backward reachability analysis based on predicate abstraction and abstraction refinement is used for verifying assertions which are universally quantified over array indexes.
The methods based on abstract interpretation construct over-approximations, that is, invariants implied by the program executions. These methods have the advantage of being quite efficient because they fix in advance a finite set of basic assertions from which the invariants can be constructed. However, for the same reason, these methods may lack flexibility as the abstraction should be re-designed when verification fails.

Also theorem provers have been applied for discovering invariants and proving verification conditions generated from the programs. In particular, in [7] a satisfiability decision procedure for a decidable fragment of a theory of arrays is presented. That fragment is expressive enough to prove properties such as sortedness of arrays. In [32, 33, 38] the authors present some techniques that use theorem proving for generating array invariants. Some theorem proving techniques for program verification are based on Satisfiability Modulo Theory (SMT) (see, for instance, [3, 4, 35]). The approaches based on theorem proving and SMT are more flexible with respect to those based on abstract interpretation because no finite set of assertions is fixed in advance and, instead, the suitable assertions needed for the proofs can be generated on demand.

Although the approach based on CLP program transformation shares many ideas and techniques with abstract interpretation and automated theorem proving, we believe that it offers a higher degree of flexibility and parametricity. Indeed, the transformation-based method for the generation of the verification conditions and their proof, is to a large extent independent of the imperative program and the property to be verified.

The use of CHR further enhances the flexibility of our transformation-based approach because CHR manipulate the constraints that represent operations on the data structures (such as the read and write operations in the case of arrays), while the unfold/fold rules manipulate the non-constraint atoms of the CLP programs. The experimental results we have reported in this paper demonstrate that the combination of the two kind of rules, those for constraints and those for non-constraint atoms, is a promising, powerful technique for proving program properties.

As future work we plan to extend our transformation-based method to the verification of programs which manipulate dynamic data structures such as lists or heaps. To this aim we may combine the CHR axiomatization of heaps proposed by [17] with the generalization strategies based on widening and convex-hull considered in this paper.

References


Defeasibility in contextual reasoning with CKR

Loris Bozzato¹, Thomas Eiter², and Luciano Serafini¹

¹ Fondazione Bruno Kessler, Via Sommarive 18, 38123 Trento, Italy
² Institut für Informationssysteme, Technische Universität Wien, Favoritenstraße 9-11, A-1040 Vienna, Austria

{bozzato,serafini}@fbk.eu, eiter@kr.tuwien.ac.at

Abstract. Recently, representation of context dependent knowledge in the Semantic Web has been recognized as a relevant issue and a number of logic based solutions have been proposed in this regard: among them, in our previous works we presented the Contextualized Knowledge Repository (CKR) framework. A CKR knowledge base has a two layered structure, modelled by a global context and a set of local contexts: the global context not only contains the metaknowledge defining the properties of local contexts, but also holds the global (context independent) object knowledge that is shared by all of the local contexts. In many practical cases, however, it is desirable to leave the possibility to “override” the global object knowledge at the local level, by recognizing the axioms that can allow exceptional instances in the local contexts. This clearly requires to add a notion of non monotonicity across the global and the local parts of a CKR.

In this paper we present an extension to the semantics of CKR to introduce such notion of defeasible axioms. By extending a previously proposed datalog translation, we obtain a representation for CKR as a datalog program with negation under answer set semantics. This representation can be exploited as the basis for implementation of query answering for the proposed extension of CKR.

1 Introduction

Representation of context dependent knowledge in the Semantic Web has been recently recognized as a relevant issue that lead to a number of logic based proposals, e.g. [7–9, 13–15, 12]; in particular, the Contextualized Knowledge Repository (CKR) framework [12, 3, 2], with its latest formulation in [4], has been developed at the FBK in Trento.

A CKR knowledge base has a two layered structure: basically, it consists of a global context and a set of local contexts. The global context contains metaknowledge defining the properties of local contexts, as well as the global (context independent) object knowledge that is shared by all of the local contexts. Local contexts, on the other hand, contain knowledge that holds under specific circumstances (e.g. an event) and thus represent different independent views of the domain. The global object knowledge is propagated from the global to the local contexts and used as a common part of the system knowledge. In many practical cases, however, it is desirable to leave the possibility to “override” the global object knowledge at the local level, by recognizing those axioms that can allow exceptional instances in their local instantiations.

* This paper has been previously presented at the 5th International Workshop on Acquisition, Representation and Reasoning with Contextualized Knowledge (ARCOE-LogIC 2013).
For example, in the scenario of an event recommendation system, we might want to assert at the global level that “by default, all of the cheap events are interesting”, but then override this inclusion for particular kind of events in the context of a participant. We might also want to express defeasibility on the propagation of information: for instance, in a CKR representing an organization, we might want to express that “by default, all the employees working the previous year also work in the current year” and override the axiom in the context of a specific year for employees that finished their working contract at that time. In other words, we want to specify that certain axioms at the global level are defeasible, thus they can allow exceptional instances in local contexts, while holding in the general case: this clearly requires to add a notion of non-monotonicity across the global and the local parts of a CKR.

In this work, we present an extension to the CKR semantics of [4] to support such defeasibility for global object knowledge. We desire to enrich previous work and to have a datalog representation of the extended CKR semantics that extends the one for the CKR semantics in [4]: we introduced defeasible axioms guided by the approach of inheritance logic programs in [5]. There the idea is that special rules recognize exceptional facts at the local level and others propagate global facts only if they are not proved to be overridden at the local level, which happens if the opposite is derived; in the same vein, we consider instances of axioms that might be overridden at the local level. The semantics for CKR we define is (as desired) representable by a datalog program with negation under answer sets semantics; furthermore, a respective translation can be used as the basis to implement query answering over defeasible CKR knowledge bases.

We can thus summarize the contributions of our work as follows. After a brief introduction of preliminary definitions in Section 2, we present in Section 3 syntax and semantics of an extension of CKR with defeasible axioms in the global context. Notably, this allow us to introduce for the first time a notion of non-monotonicity across contexts in our contextual framework. We then extend in Section 4 the datalog translation for OWL RL based CKR from [4] with rules for the translation of defeasible axioms and for considering local exceptions in the propagation of such knowledge. We express non-monotonicity using answer set semantics, such that instance checking over an CKR with defeasible axioms reduces to cautious inference from all answer set of the translation. The work reported here is in progress, and an implementation of a prototype reasoner, based on the results of [4] and this paper, is underway.

2 Preliminaries: \textit{SROIQ}-RL

This work basically builds on the materialization calculus for CKR on OWL RL recently proposed in [4]. The extension of the calculus that we present here is again formulated over the language \textit{SROIQ}-RL, which represents the restriction of \textit{SROIQ} to the OWL RL constructs [11]. The language is obtained by restricting the form of General Concept Inclusion axioms (GCIs) and concept equivalence of \textit{SROIQ} to \( C \sqsubseteq D \) where \( C \) and \( D \) are concept expressions, called left-side concept and right-side concept respectively, and defined by the following grammar:

\[
C ::= A | \{a\} | C_1 \sqcap C_2 | C_1 \sqcup C_2 | \exists R.C_1 | \exists R.\{a\} | \exists R.\top \\
D ::= A | \neg C_1 | D_1 \sqcap D_2 | \exists R.\{a\} | \forall R.D_1 | \leq nR.C_1 | \leq nR.\top
\]
where $A$ is a concept name, $R$ is role name and $n \in \{0, 1\}$. A both-side concept $E,F$ is a concept expression which is both a left- and right-side concept. TBox axioms can only take the form $C \sqsubseteq D$ or $E \equiv F$. The RBox for $SROIQ$-RL can contain every role axiom of $SROIQ$ except for $\text{Ref}(R)$. ABox concept assertions can be only stated in the form $D(a)$, where $D$ is a right-side concept.

3 CRK with defeasible axioms

We now introduce CRKs and extend them with primitives to express defeasible axioms. We first present the syntax and then define a model-based semantics for the interpretation of defeasible inheritance from the upper contexts.

A Contextualized Knowledge Repository (CRK) is a two layered structure. The upper layer consists of a knowledge base $\mathcal{G}$, which describes two types of knowledge: (i) the structure and the properties of contexts of the CRK (called global knowledge), and (ii) the knowledge that is context independent, i.e., that holds in every context (called global knowledge). The lower layer is constituted by a set of (local) contexts; each contains (locally valid) facts and can also refer to what holds in other contexts.

Meta-Language. The meta-knowledge of a CRK is expressed by a DL language defined on a meta-vocabulary, containing the elements that define the contextual structure.

Definition 1 (Meta-vocabulary). A meta-vocabulary is a DL vocabulary $\Gamma$ composed of a set $\text{NC}_\Gamma$ of atomic concepts, a set $\text{NR}_\Gamma$ of atomic roles, and a set $\text{NI}_\Gamma$ of individual constants that are mutually disjoint and contain the following sets of symbols

1. $\mathbb{N} \subseteq \text{NI}_\Gamma$ of context names.
2. $\mathbb{M} \subseteq \text{NI}_\Gamma$ of module names.
3. $\mathbb{C} \subseteq \text{NC}_\Gamma$ of context classes, including the class Ctx.
4. $\mathbb{R} \subseteq \text{NR}_\Gamma$ of contextual relations.
5. $\mathbb{A} \subseteq \text{NR}_\Gamma$ of contextual attributes.
6. For every attribute $A \in \mathbb{A}$, a set $\mathbb{D}_A \subseteq \text{NI}_\Gamma$ of attribute values of $A$.

We use the role $\text{mod} \in \text{NR}_\Gamma$ defined on $\mathbb{N} \times \mathbb{M}$ to express associations between contexts and modules. The meta-language $\mathcal{L}_\Gamma$ of a CRK is a DL language over $\Gamma$ with the following syntactic conditions on the application of role restrictions: for every $\bullet \in \{\forall, \exists, \leq n, \geq n\}$, (i) for a concept $\bullet A.B$, then $B$ is in the form $B = \{a\}$ with $a \in \mathbb{D}_A$; (ii) for a concept $\bullet \text{mod}.B$, then $B$ is in the form $B = \{m\}$ with $m \in \mathbb{M}$.

Object Language. The context (in)dependent knowledge of a CRK is expressed via a DL language called object-language $\mathcal{L}_\Sigma$ over an object-vocabulary $\Sigma = \text{NC}_\Sigma \uplus \text{NR}_\Sigma \uplus \text{NI}_\Sigma$. Expression in $\mathcal{L}_\Sigma$ will be evaluated locally to each context, i.e., each context can interpret each symbol independently. However, sometimes one wants to constrain the meaning of a symbol in a context with the meaning of a symbol in some other context. For such external references, we extend the object language $\mathcal{L}_\Sigma$ to $\mathcal{L}_\Sigma'$ with eval expressions of the form $\text{eval}(X, C)$, where $X$ is a concept or role expression of $\mathcal{L}_\Sigma$ and $C$ is a concept expression of $\mathcal{L}_\Gamma$ (with $C \sqsubseteq \text{Ctx}$).

Defeasible Axioms. Compared to [4], we extend the types of axioms that can appear in the global object knowledge $\mathcal{G}$ with defeasible axioms. Given an axiom $\alpha \in \mathcal{L}_\Sigma$, the assertion $D(\alpha)$ in $\mathcal{G}$ states that $\alpha$ is a defeasible axiom of $\mathcal{G}$. Intuitively, this statement means that $\alpha$, at the level of instantiations for individuals, propagates to a local context unless it is contradicted there, and thus an exception to $\alpha$ for individuals is tolerated.
Example 1. A defeasible fact \( D(\text{Expensive(concert)}) \) might express that a concert is expensive and propagate this to local contexts. At such a context, this might be contradicted by an assertion \( \neg\text{Expensive(concert)} \), which then overrides the global assertion. Likewise, such overriding should take place if we have a global axiom \( \text{Cheap} \sqsubseteq \neg\text{Expensive} \) and a local assertion \( \text{Cheap(concert)} \), such that \( \neg\text{Expensive(concert)} \) can be derived at the local context.

Example 2. Beyond facts, from a defeasible GCI axiom \( D(\text{Cheap} \sqsubseteq \text{Interesting}) \) (“cheap events are interesting”) and the global assertion \( \text{Cheap(fbmatch)} \) (“football matches are cheap”), we may conclude \( \text{Interesting(fbmatch)} \) at a local context; however, an assertion \( \neg\text{Interesting(fbmatch)} \) there would contradict this and should override the instance of the defeasible axiom for \( fbmatch \): that is, the fact \( \neg\text{Cheap} \sqsubseteq \text{Interesting} \) may be violated.

The DL language \( L^D_\Sigma \) extends \( L_\Sigma \) with the set of defeasible axioms in \( L^D_\Sigma \).

**CKR Syntax.** We are now ready to give our formal definition of Contextualized Knowledge Repository.

**Definition 2 (Contextualized Knowledge Repository, CKR).** A Contextualized Knowledge Repository (CKR) over a meta-vocabulary \( \Gamma \) and an object vocabulary \( \Sigma \) is a structure \( \mathcal{R} = (\emptyset, \{K_m\}_{m \in M}) \) where:

- \( \emptyset \) is a DL knowledge base over \( L_\Gamma \cup L^D_\Sigma \), and
- every \( K_m \) is a DL knowledge base over \( L^D_\Sigma \), for each module name \( m \in M \).

In particular, \( \mathcal{R} \) is a SROIQ-RL CKR, if \( \emptyset \) and all \( K_m \) are knowledge bases over the extended language of SROIQ-RL where eval-expressions can only occur in left-concepts and contain left-concepts or roles. In the following, we tacitly focus on such CKR. We show how the examples in the introduction can be formalized as CKRs.

**Example 3.** In the first example (inspired by a real application of CKR\(^3\)) we want to define an event recommendation system: we thus represent touristic events and preferences of tourists in order to be able to derive appropriate suggestions. In particular, we want to assert that, in general, all of the \( \text{Cheap} \) events are \( \text{Interesting} \); we do so by expressing this as a defeasible axiom in the global context. Furthermore, we propose local markets (\( \text{market} \)) and football matches (\( \text{fbmatch} \)) as examples of cheap events. On the other hand, we want to reflect that tourists interested in cultural events are not interested in a sportive event like a football match\(^4\); we express this by negating the interest in \( \text{fbmatch} \). Thus, our example CKR \( \mathcal{R}_{\text{tour}} = (\emptyset, \{K_{\text{ctourist,m}}\}) \) has:

\[
\begin{align*}
\emptyset : & \{ D(\text{Cheap} \sqsubseteq \text{Interesting}), \text{Cheap(fbmatch)}, \text{Cheap(market)}, \\
& \quad \text{mod(\text{cultural_tourist}, \text{ctourist_m})} \}, \\
K_{\text{ctourist,m}} : & \{ \neg\text{Interesting(fbmatch)} \}.
\end{align*}
\]

Note that the negative assertion in the local context represents, as discussed in Example 2, an exception to the defeasible axiom: we want to recognize this “overriding” for the \( fbmatch \) instance, but still apply the defeasible inclusion for \( \text{market} \).

\(^3\)http://www.investintrentino.it/News/Trentour-Trentino-platform-for-smart-tourism

\(^4\)To keep things simple, we omit modeling sportive events by a separate concept.
Example 4. Our next example shows how we can represent a form of defeasible propagation of information across local contexts using eval expressions. We want to represent the information about an organization in a CKR, using contexts to represent its situation in different years. We express the rule that every employee working the years before (WorkingBefore) also works in the current year (WorkingNow) by a defeasible inclusion. In the module associated to 2012, we say that alice, bob and charlie were working last year. In the module for 2013, we say (using an eval expression) that all of the employees working in 2012 have to be considered in the set of employees working in the past years; moreover, we say that charlie no longer works for the organization. This can be encoded in the CKR $K_{org} = \langle \emptyset, \{K_{em2012,m}, K_{em2013,m}\} \rangle$, where

$$\emptyset : \{ \text{D}(\text{WorkingBefore} \sqsubseteq \text{WorkingNow}), \text{mod}(\text{employees2012, em2012,m}), \text{mod}(\text{employees2013, em2013,m}) \}$$

$$K_{em2012,m} : \{ \text{WorkingNow}(\text{alice}), \text{WorkingNow}(\text{bob}), \text{WorkingNow}(\text{charlie}) \}$$

$$K_{em2013,m} : \{ \text{eval}(\text{WorkingNow}, \{\text{employees2012}\}) \sqsubseteq \text{WorkingBefore}, \}$$

Intuitively, at the local context employees2013, where WorkingBefore(charlie) can be derived, the negative assertion $\lnot$WorkingNow(charlie) should override the instance of the inclusion axiom in the global context for charlie, as it would lead to the opposite, i.e., WorkingNow(charlie); on the other hand, for alice and bob no overriding should happen and we can derive that they still work for the organization.

Semantics. We now define a model-based semantics for CKRs. The idea is to model exceptions of axiom instances by so called clashing assumptions, which are pairs $\langle \alpha, e \rangle$ of an axiom and a set of individuals, to the effect that in the evaluation at the local context, the instance of $\alpha$ for $e$ is disregarded. However, such a clashing assumption must be justified, in the sense that the instance of $\alpha$ for $e$ must be unsatisfiable at the local context. This is ensured if there are assertions that can be derived which prove this unsatisfiability: we call such assertions clashing sets. Models of a CKR will be then CKR interpretations in [4] extended with clashing assumptions that are all justified.

Definition 3 (CKR interpretation). A CKR interpretation for $\langle \Gamma, \Sigma \rangle$ is a structure $I = \langle \mathcal{M}, \mathcal{I} \rangle$ s.t.

- $\mathcal{M}$ is a DL interpretation of $\Gamma \cup \Sigma$ s.t., for every $c \in \mathbb{N}$, $c^\mathcal{M} \in \text{Ctx}^\mathcal{M}$ and, for every $C \in \mathcal{C}$, $C^\mathcal{M} \subseteq \text{Ctx}^\mathcal{M}$;

- for every $x \in \text{Ctx}^\mathcal{M}$, $\mathcal{I}(x)$ is a DL interpretation over $\Sigma$ s.t. $\Delta^\mathcal{I}(x) = \Delta^\mathcal{M}$ and, for $a \in \text{NI}_\Sigma$, $a^\mathcal{I}(x) = a^\mathcal{M}$.

The interpretation of ordinary DL expressions on $\mathcal{M}$ and $\mathcal{I}(x)$ in $I = \langle \mathcal{M}, \mathcal{I} \rangle$ is as usual; eval expressions are interpreted as follows: for every $x \in \text{Ctx}^\mathcal{M}$,

$$\text{eval}(X, C)^{\mathcal{I}(x)} = \bigcup_{x \in \text{Ctx}^\mathcal{M}} X^{\mathcal{I}(x)}$$

According to the previous definition, a CKR interpretation consists of an interpretation for the “upper-layer” (which includes the global knowledge and the meta-knowledge) and an interpretation of the object language for each instance of type context (i.e., for all $x \in \text{Ctx}^\mathcal{M}$), providing a semantics of the object-vocabulary in $x$. 

136
An instantiation of an axiom \( \alpha \in L_\Sigma \) with a tuple \( e \) of individuals in \( N_\Sigma \), written \( \alpha(e) \), is the specialization of \( \alpha \), viewed as its first order translation in an universal sentence \( \forall x. \phi_\alpha(x) \), to \( e \) (i.e., \( \phi_\alpha(e) \)); accordingly, e.g., \( e \) is void for assertions, a single element \( e \) for GCI, and a pair \( e_1, e_2 \) of elements for role axioms.

A clashing assumption is pair \( (\alpha, e) \) such that \( \alpha(e) \) is an axiom instantiation; intuitively, it represents the assumption that \( \alpha(e) \) is not (DL-)satisfiable. A clashing set for a \( (\alpha, e) \) is a satisfiable set \( S \) of ABox assertions such that \( S \cup \{ \alpha(e) \} \) is unsatisfiable. That is, \( S \) provides an assertional “justification” for the assumption of local overriding of \( \alpha \) on \( e \).

Given a CKR interpretation \( \mathcal{I} = \langle M, I, T \rangle \), we call the structure \( \mathcal{I}_{CAS} = \langle M, I, T, CAS \rangle \) a CAS-interpretation, where \( CAS \) is a map such that, for every \( x \in \Delta^M \), \( CAS(x) \) is a set of clashing assumptions for \( x \).

**Definition 4 (CAS-model).** Given a CKR \( \mathcal{R} = \langle \Theta, \{K_m\}_{m \in \mathcal{M}} \rangle \) and a CAS-interpretation \( \mathcal{I}_{CAS} = \langle M, I, T, CAS \rangle \), we say that \( \mathcal{I}_{CAS} \) is a CAS-model for \( \mathcal{R} \) (\( \mathcal{I}_{CAS} \models \mathcal{R} \)) if:

- for every \( \alpha \in L_\Sigma \cup L_T \) in \( \Theta \), \( M \models \alpha \);
- for every \( D(\alpha) \in \Theta \) with \( \alpha \in L_\Sigma \), \( M \models \alpha \);
- if \( \langle x, y \rangle \in \text{mod}^M \) and \( y = m^M \), then \( I(x) \models K_m \);
- for every \( \alpha \in \Theta \cap L_\Sigma \) and \( x \in \text{ctx}^M \), \( I(x) \models \alpha \), and

Thus, for every \( \alpha \in L_\Sigma \) and \( c \in N \), we write \( \mathcal{R} \models_{CAS} c : \alpha \) if for every CAS-interpretation \( \mathcal{I}_{CAS} \) it holds that \( \mathcal{I}_{CAS} \models \mathcal{R} \) implies \( \mathcal{I}(c^M) \models \alpha \); for \( \alpha \in L_T \), we write \( \mathcal{R} \models_{CAS} \alpha \) if for every CAS-interpretation \( \mathcal{I}_{CAS} \) it holds that \( \mathcal{I}_{CAS} \models \mathcal{R} \) implies \( M \models \alpha \).

We say that a CAS-model \( \mathcal{I}_{CAS} = \langle M, I, T, CAS \rangle \) of \( \mathcal{R} \) is justified if, for every \( x \in \text{ctx}^M \) and \( \langle x, e \rangle \in \text{CAS}(x) \), some clashing set \( S_{x,\langle x, e \rangle} \) exists such that for every CAS-model \( \mathcal{R}'_{CAS} = \langle M', I', T, CAS' \rangle \) of \( \mathcal{R} \) with \( \Delta^M = \Delta^{M'} \), it holds \( I'(x) \models S_{x,\langle x, e \rangle} \).

Informally, justification requires that we have factual evidence that an instantiation of an axiom can not be satisfied, and this evidence is provable. Based on this, we now give the definition of CKR model:

**Definition 5 (CKR model).** A CKR interpretation \( \mathcal{I} = \langle M, I, T \rangle \) is a CKR model of \( \mathcal{R} \) (in symbols, \( \mathcal{I} \models \mathcal{R} \)) if some \( \mathcal{I}_{CAS} = \langle M, I, T, CAS \rangle \) is a justified CAS-model for \( \mathcal{R} \).

For \( \alpha \in L_\Sigma \) and \( c \in N \), we write \( \mathcal{R} \models c : \alpha \) if for every CKR model \( \mathcal{I} \) of \( \mathcal{R} \); similarly for \( \alpha \in L_T \), we write \( \mathcal{R} \models \alpha \) if \( M \models \alpha \) for every CKR model \( \mathcal{I} \) of \( \mathcal{R} \).

We can show the following properties:

**Proposition 1.** Suppose that \( \mathcal{I} = \langle M, I, T \rangle \) and \( \mathcal{I}' = \langle M', I' \rangle \) are CKR models of \( \mathcal{R} \) such that \( \mathcal{I}_{CAS} = \langle M, I, T, CAS \rangle \) and \( \mathcal{I}'_{CAS'} = \langle M', I', CAS' \rangle \) are justified CAS-models of \( \mathcal{R} \). Then \( \text{ctx}^M = \text{ctx}^{M'} \) and \( \text{CAS}(x) \subseteq \text{CAS}(x) \) for every \( x \in \text{ctx}^M \) implies \( CAS = CAS' \).

The previous result shows that, given the justification of the CKR model, there is a notion of minimality on the sets of clashing assumptions related to each context.

Given a CAS-interpretation \( \mathcal{I}_{CAS} = \langle M, I, T, CAS \rangle \), we denote with \( \mathcal{I}_M = \langle M', I', CAS' \rangle \) the interpretation in which:

1. \( \Delta^M = \{ a^M \mid a \in N_I \cup N_\Sigma \} \);
2. \( M' \), \( I' \) and \( CAS' \) are the restrictions of \( M \), \( I \) and \( CAS \) to \( \Delta^M \), respectively.

137
Proposition 2. Let $\mathcal{J}_{\text{CAS}}$ be a justified CAS-model of $\mathfrak{R}$. Then, also the CAS-interpretation $\mathcal{J}^N_{\text{CAS}}$ is a justified CAS-model of $\mathfrak{R}$.

Basically, the result shows that justification for a CKR only depends on the “named contexts” part of the considered CAS-model: intuitively, this allow us to consider such restricted models in the correctness result of the datalog translation presented in the following section.

Example 5. We can now show an example of CKR models satisfying the example CKRs presented in Example 3. In the case of $\mathfrak{R}_{\text{tour}}$, we can consider a model $\mathcal{J}_{\text{CAS}_{\text{tour}}} = \langle M, \mathcal{I}, \text{CAS}_{\text{tour}} \rangle$ such that $\text{CAS}_{\text{tour}}(\text{CulturalTourist}^M) = \{\{\text{Cheap} \sqsubseteq \text{Interesting}, \{\text{fbmatch}\}\}\}$. Note that the interpretation is justified as it is easy to check that $\mathfrak{R}_{\text{tour}} \models \text{CulturalTourist} : \{\text{Cheap(fbmatch)}, \neg \text{Interesting(fbmatch)}\}$, that represents a clashing set for the defeasible axiom. Moreover, for the definition of satisfiability under the assumptions in $\text{CAS}_{\text{tour}}$, we obtain that $\mathcal{I}(\text{CulturalTourist}^M) \models \text{Interesting(\text{market})}.$

Similarly, for $\mathfrak{R}_{\text{org}}$, we have that the model $\mathcal{J}_{\text{CAS}_{\text{org}}} = \langle M, \mathcal{I}, \text{CAS}_{\text{org}} \rangle$ with $\text{CAS}_{\text{org}}(\text{Employees2013}^M) = \{\{\text{WorkingBefore} \sqsubseteq \text{WorkingNow}, \{\text{charlie}\}\}\}$ is a CKR model for the example CKR. For the interpretation of eval expressions, in every interpretation of $\mathcal{I}_{\text{org}}$ we have that $\{\text{alice}^x, \text{bob}^x, \text{charlie}^x\} \subseteq \text{WorkingBefore}^x$, where $x = \text{employees2013}$. Thus the justification of the model can be easily seen as $\mathfrak{R}_{\text{org}} \models \text{employees2013} : S$ for $S = \{\text{WorkingBefore(charlie),} \neg \text{WorkingNow(charlie)}\}$, which represents a clashing set for the defeasible axiom on charlie. On the other hand, for the satisfiability under the assumptions in $\text{CAS}_{\text{org}}$, we obtain that $\mathcal{I}(\text{employees}^M) \models \text{WorkingNow(alice)}$ and $\mathcal{I}(\text{employees}^M) \models \text{WorkingNow(bob)}.$

As clashing assumptions in CAS maps are ground instances of axioms, they refer merely to named individuals. We remark that using standard names for the domain elements, one could permit clashing assumptions for all elements; the results in Propositions 1 and 2 carry over to this setting.

4 CKR translation to general programs

We revise the datalog translation for $\text{SROIQ-RL}$ CKR from [4] with rules for the detection of axiom overriding and defeasible propagation of global knowledge. To simplify the presentation of rules, we introduce a normal form for the considered axioms. We say that a CKR $\mathfrak{R} = \langle \emptyset, \{K_m\}_{m \in M} \rangle$ is in normal form if:

- $\emptyset$ contains axioms in $L^\emptyset$ of the form of Table 1 or in the form $C \sqsubseteq \exists \text{mod.}\{m\}, C \sqsubseteq \exists A, \{d_a\}$ for $A, B, C \in C, R, S, T \in R, a, b \in N, m \in M, A \in A$ and $d_a \in D_A$.
- $\emptyset$ and every $K_m$ contain axioms in $L^\emptyset$ of the form of Table 1 and every $K_m$ contain axioms in $L^\Sigma$ of the form $\text{eval}(A, C) \sqsubseteq B$, $\text{eval}(R, C) \sqsubseteq T$ for $A, B, C \in NC^\Sigma, a, b \in NI^\Sigma, R, S, T \in NR^\Sigma$ and $C \in C$.
- $\emptyset$ contains defeasible axioms $D(\alpha) \in L^\Sigma$ with $\alpha$ of the form of Table 1.

It can be seen that for named interpretations, i.e., of the form $\mathcal{J}^N_{\text{CAS}}$, every CKR can be rewritten into an equivalent one in normal form (using new symbols).
In this version of the translation, the definition of program has now to be adapted to the new form of the rules (admitting negative literals) and to the answer set semantics of the resulting logic program.

**Syntax.** A signature is a tuple \((\mathcal{C}, \mathcal{P})\) of a finite set \(\mathcal{C}\) of constants and a finite set \(\mathcal{P}\) of predicates. We assume a set \(\mathcal{V}\) of variables; the elements of \(\mathcal{C} \cup \mathcal{V}\) are terms. An atom is of the form \(p(t_1, \ldots, t_n)\) where \(p \in \mathcal{P}\) and \(t_1, \ldots, t_n\), are terms. A literal \(l\) is either a positive literal \(p\) or a negative literal \(\neg p\) with \(p\) an atom and \(\neg\) the symbol of strong negation. Literals of the form \(p, \neg p\) are complementary.

A rule \(r\) is an expression of the form

\[ a \leftarrow b_1, \ldots, b_k, \text{not } b_{k+1}, \ldots, \text{not } b_m. \]

where \(a, b_1, \ldots, b_m\) are literals and \(\text{not}\) is the negation as failure symbol (NAF). We denote with \(\text{Head}(r)\) the head \(a\) of rule \(r\) and with \(\text{Body}^+(r)\) and \(\text{Body}^-(r)\) the positive \((b_1, \ldots, b_k)\) and NAF \((b_{k+1}, \ldots, b_m)\) part of the body of the rule\(^5\). A fact \(H\) is a ground rule with empty body (we then omit \(\leftarrow\)). A program \(P\) is a finite set of rules. A ground substitution \(\sigma\) for \((\mathcal{C}, \mathcal{P})\) is a function \(\sigma : \mathcal{V} \rightarrow \mathcal{C}\); (ground) substitutions on atoms and ground instances of atoms are as usual.

**Semantics.** Given a program \(P\), we define the universe \(U_P\) of \(P\) as the set of all constants occurring in \(P\) and the base \(B_P\) of \(P\) as the set of all the ground literals constructible from the predicates in \(P\) and the constants in \(U_P\). An interpretation \(I \subseteq B_P\) is a consistent subset of \(B_P\) (i.e., not containing complementary literals). We say that a literal \(l\) is true in \(I\) iff \(l \in I\) and false otherwise.

Given a rule \(r \in \text{ground}(P)\), the body of \(r\) is true in \(I\) if: (1) every literal in \(\text{Body}^+(r)\) is true w.r.t. \(I\), and (2) every literal in \(\text{Body}^-(r)\) is false w.r.t. \(I\). A rule \(r\) is satisfied in \(I\) if either the head of \(r\) is true in \(I\) or the body of \(r\) is not true in \(I\).

An interpretation \(I\) for \(P\) is a model for \(P\) \((I \models P)\), if every rule in \(\text{ground}(P)\) is satisfied in \(I\); it is a minimal model for \(P\), if no proper subset \(I' \subset I\) is a model for \(P\).

Given an interpretation \(I\) for \(P\), the (Gelfond-Lifschitz) reduct of \(P\) w.r.t. \(I\), denoted by \(G_I(P)\), is the set of rules obtained from \(\text{ground}(P)\) by (i) removing every rule \(r\) such that \(\text{Body}^-(r) \cap I \neq \emptyset\), (ii) removing the NAF part from the bodies of the remaining rules. Then \(I\) is an answer set of \(P\), if \(I\) is a minimal model of the positive version \(\text{pos}(G_I(P))\) of \(G_I(P)\) (i.e. the positive program obtained by considering each negative literal \(\neg p(t_1, \ldots, t_n)\) as a positive one with predicate symbol \(\neg p\)).

The following property is well-known.

**Lemma 1.** If \(M\) is an answer set for \(P\), then \(M\) is a minimal model of \(P\).

---

\(^5\) In the rules, we might write \((\neg)p\) to denote that the rule holds both for the positive and negative literal associated to \(p\). This will be used only as a shortcut to simplify the presentation of rules.
Using this interpretation for our programs, we say that a literal \( H \in B_P \) is a consequence of \( P \) and we write \( P \models H \) iff for every answer set \( M \) of \( P \) we have that \( M \models H \).

**Translation.** We now are ready to present our translation. As in [4], we basically instantiate and adapt the materialization calculus in [10] to meet the structure of CKR.

The translation is composed by the following sets: the input translations \( I_{glob}, I_{loc}, I_D, I_{rl} \), the deduction rules \( P_{loc}, P_D, P_{rl} \), and output translation \( O \), such that:

- every input translation \( I \) and output translation \( O \) are partial functions (defined over axioms in normal form) while deduction rules \( P \) are sets of datalog rules;
- given an axiom or signature symbol \( \alpha \) (and \( c \in \mathbb{N} \)), each \( I(\alpha) \) (or \( I(\alpha, c) \)) is either undefined or a set of datalog facts or rules;
- given an axiom \( \alpha \) and \( c \in \mathbb{N} \), \( O(\alpha, c) \) is either undefined or a single datalog fact;

We extend the definition of input translations to knowledge bases (set of axioms) \( S \) with their signature \( \Sigma \), with \( I(S) = \bigcup_{\alpha \in S} I(\alpha) \cup \bigcup_{s \in \Sigma, I(s) \text{ defined}} I(s) \) (similarly \( I(S, c) = \bigcup_{\alpha \in S} I(\alpha, c) \cup \bigcup_{s \in \Sigma, I(s) \text{ defined}} I(s, c) \)).

We briefly present the form of the different sets of translation and deduction rules involved in the translation process: the tables containing the complete set of rules can be found in the Appendix.

The set of rules in \( I_{rl}(S, c) \) define the rules to translate \( SROIQ\)-RL axioms and signature: for example, we have the following rule for concept inclusions: \( A \sqsubseteq B \rightarrow \{\text{subClass}(A, B, c)\} \). The set of rules \( P_{rl} \) are the corresponding deduction rules for axioms in \( SROIQ\)-RL: for example, for atomic concept inclusions we have the rule:

\[
\text{instd}(x, z, c) \leftarrow \text{subClass}(y, z, c), \text{instd}(x, y, c).
\]

Global input rules of \( I_{glob}(\emptyset) \), basically encode the interpretation of \( \text{Ctx} \) in the global context. Similarly, local input rules \( I_{loc}(\text{Km}, c) \) and local deduction rules \( P_{loc} \) provide the translation and rules for elements of the local object language, in particular for \( \text{eval} \) expression: e.g., for inclusion of concepts with a left \( \text{eval} \) expression we have the input rule \( \text{eval}(A, C) \sqsubseteq B \rightarrow \{\text{subEval}(A, C, B, c)\} \) and the corresponding deduction rule

\[
\text{instd}(x, b, c) \leftarrow \text{subEval}(a, c_1, b, c), \text{instd}(c', c_1, gm), \text{instd}(x, a, c').
\]

The input rules in \( I_D \) provide the translation of defeasible axioms in the global context: given a defeasible axiom \( D(\alpha) \), \( I_{rl}(\alpha, gk) \) is applied and a rule defining when the axiom is locally overridden is added to the program. For example, if \( D(A \sqsubseteq B) \in \emptyset \), the fact \( \text{subClass}(A, B, gk) \) is added to the program for the global context together with the corresponding overriding rule:

\[
\text{ovr}(\text{subClass}, x, A, B, c) \leftarrow \neg \text{instd}(x, B, c), \text{instd}(x, A, c), \text{prec}(c, g).
\]

The deduction rules in \( P_D \), on the other hand, provide the definition of the defeasible propagation of axioms from the global context to the local contexts. For example, the following rule propagates an atomic concept inclusion axiom: if the inclusion axiom is in the program of the global context and can be applied to a local instance, it is applied only if the instance is not recognized as an exception:
Finally, the set of output rules $O(\alpha, c)$ provides the translation of ABox assertions that can be verified by applying the rules of the final program. For example, the case for atomic concept assertions in a given context $c$ is given as $A(a) \mapsto \{\text{instd}(a, A, c)\}$.

Given a CKR $\mathcal{R} = \langle \mathcal{G}, \{K_m\}_{m \in M} \rangle$, the translation to its dialog program follows these steps:

1. the global program for $\mathcal{G}$ is translated to (where $g_m, g_k$ are new context names):

   $$PG(\mathcal{G}) = I_{\text{glob}}(\mathcal{G}_r) \cup I_{\text{D}}(\mathcal{G}_r) \cup I_{\text{rl}}(\mathcal{G}_r, g_m) \cup I_{\text{rl}}(\mathcal{G}_\Sigma, g_k) \cup P_{rl}$$

   where $\mathcal{G}_r = \{\alpha \in \mathcal{G} | \alpha \in L_r\}$ and $\mathcal{G}_\Sigma = \{\alpha \in \mathcal{G} | \alpha \in L_\Sigma\}$.

2. We define the set of contexts:

   $$\mathbf{N}_{\mathcal{G}} = \{c \in \mathbf{N} | PG(\mathcal{G}) \models \text{instd}(c, Ctx, g_m)\}$$

   For every $c \in \mathbf{N}_{\mathcal{G}}$, we define its associated knowledge base as:

   $$K_c = \bigcup\{K_m \in \mathcal{R} | PG(\mathcal{G}) \models \text{triplied}(c, mod, m, g_m)\}$$

3. We define each local program for $c \in \mathbf{N}_{\mathcal{G}}$ as:

   $$PC(c) := P_{\text{loc}} \cup P_{D} \cup I_{\text{loc}}(K_c, c) \cup I_{\text{rl}}(K_c, c) \cup \{\text{prec}(c, g_k)\}.$$  

4. The CKR program is then defined as:

   $$PK(\mathcal{R}) = PG(\mathcal{G}) \cup \bigcup_{c \in \mathbf{N}_{\mathcal{G}}} PC(c)$$

We say that $\mathcal{G}$ entails an axiom $\alpha \in L_r$ (denoted $\mathcal{G} \models_r \alpha$) if $PG(\mathcal{G})$ and $O(\alpha, g_m)$ are defined and $PG(\mathcal{G}) \models O(\alpha, g_m)$. Similarly, $\mathcal{G}$ entails an axiom $\alpha \in L_\Sigma$ (denoted $\mathcal{G} \models_\Sigma \alpha$) if $PG(\mathcal{G})$ and $O(\alpha, g_k)$ are defined and $PG(\mathcal{G}) \models O(\alpha, g_k)$. We say that $\mathcal{R}$ entails an axiom $\alpha \in L_\Sigma$ in a context $c \in \mathbf{N}$ (denoted $\mathcal{R} \models_c \alpha$), if the elements of $PK(\mathcal{R})$ and $O(\alpha, c)$ are defined and $PK(\mathcal{R}) \models O(\alpha, c)$.

**Correctness of the translation.** In the following we show the correctness of the translation with respect to the problem of instance checking in the presented semantics for CKR. Let $\text{CAS}_\mathbf{N}$ be a map associating every $c \in \mathbf{N}$ to a set $\text{CAS}_\mathbf{N}(c)$ of clashing assumptions. We define the set of the corresponding overriding assumptions:

$$\text{OVR}(\text{CAS}_\mathbf{N}) = \{\text{ovr}(p(e)) | (\alpha, e) \in \text{CAS}_\mathbf{N}(c), I_{rl}(\alpha, c) = p\}$$

Given a CKR $\mathcal{R}$ and its associated CKR program $PK(\mathcal{R})$, the OVR-reduct of $PK(\mathcal{R})$ (denoted by $G_{\text{OVR}}(PK(\mathcal{R}))$) is the set of rules obtained from $\text{ground}(PK(\mathcal{R}))$ by: (i) removing every rule $r$ such that $B^-(r) \cap \text{OVR}(\text{CAS}_\mathbf{N}) \neq \emptyset$; (ii) removing the NAF part (which involves only instances of ovr) from the bodies of the remaining rules.

Consider a CKR interpretation $\mathcal{I} = \langle \mathcal{M}, \mathcal{I} \rangle$ and $\text{CAS}_\mathbf{N}^M$ such that $\text{CAS}_\mathbf{N}^M(c^M) = \text{CAS}_\mathbf{N}(c)$, for all $c \in \mathbf{N}$. We can show that the following property holds:
Lemma 2. \( G_{\text{OVR}}(PK(\bar{R})) \vdash O(\alpha, c) \) iff \( \bar{R} \models_{\text{CAS}_N} c : \alpha \) (if \( O(\alpha, c) \) is defined).

We can prove the lemma by establishing the following propositions

Proposition 3 (cf. [4]). For every \( \alpha \), \( \text{ground}(PG(\emptyset)) \models O(\alpha, g) \) iff \( \emptyset \models \alpha \).

Proposition 4. For every \( \alpha \) (s.t. \( O(\alpha, c) \) is defined),

1. (CAS-soundness) If \( G_{\text{OVR}}(PK(\bar{R})) \models O(\alpha, c) \), then \( \bar{R} \models_{\text{CAS}_N} c : \alpha \).
2. (CAS-completeness) If \( \bar{R} \models_{\text{CAS}_N} c : \alpha \), then \( G_{\text{OVR}}(PK(\bar{R})) \models O(\alpha, c) \).

The completeness of the translation procedure with respect to CKR semantics can be proved using the following results. Let \( S|_p \) be the restriction of an answer set \( S \) to the set of facts for predicate \( p \).

Lemma 3. For every justified CAS-model \( I_{\text{CAS}_N} = \langle M, \mathcal{I}, \text{CAS}_N \rangle \) of \( \bar{R} \), some answer set \( S \) of \( PK(\bar{R}) \) exists such that \( S|_{\text{OVR}} = OVR(\text{CAS}_N) \).

Lemma 4. For every answer set \( S \) of \( PK(\bar{R}) \), some justified CAS-model \( I_{\text{CAS}_N} = \langle M, \mathcal{I}, \text{CAS}_N \rangle \) of \( \bar{R} \) exists such that \( \text{CAS}_S(c) = \{ (\alpha, e) \mid I_{\mathcal{I}}(\alpha, e) \models p, \text{ovr}(p(e)) \in S \} \) for every \( c \in N \).

The correctness result directly follows from previous results.

Theorem 1. For a normal form CKR \( \bar{R} \), \( \bar{R} \models c : \alpha \) iff \( PK(\bar{R}) \models O(\alpha, c) \) (provided \( O(\alpha, c) \) is not void).

5 Conclusion

We presented an extension to the Contextualized Knowledge Repository (CKR) framework introducing a notion of defeasibility of axioms across contexts. We then presented a datalog translation for the extended semantics, based on the materialization calculus for instance checking in [4]. In the translation, non-monotonicity is expressed using answer set semantics, such that instance checking over OWL RL based CKR reduces to cautious inference from all answer sets of the translation.

An implementation of the presented translation in a prototype is ongoing. It basically builds on the DReW DL datalog rewriter [16], and extends the basic translation of OWL RL ontologies to the two layered structure of CKR. Given as input an ontology representing the global context and ontologies representing the knowledge modules, the translation produces a datalog program (compliant to DLV syntax) representing the input CKR. More specifically, it encodes the rules and the two layered translation process presented in Section 4: after the translation of the global context, the set of contexts and their associations to modules are derived from the global program through interaction with the DLV solver; with the local knowledge bases defined, the programs for the local contexts are then computed. We aim at extending the current prototype to access external data sources; moreover, we want to compare the query performance with the available implementation of CKR based on SPARQL forward rules [4].
An interesting direction for future investigation is the comparison of the proposed approach to the known approaches to integrate notions of defeasibility and defaults in description logics. Notable examples of such approaches include representations of typicality in DLs concepts [6] and works employing circumscription in description logics [1]. It is interesting to see whether such notions of defeasibility can be reduced to our representation, thus leading to an implementation of such approaches.

Another natural continuation to the presented work is to allow defeasible axioms across local contexts, possibly along an explicit order relation between contexts (as the coverage relation [12]), or across knowledge modules, allowing overriding in specific instances of context classes associated to such modules.

Acknowledgments. The research leading to these results has received funding from the European Union’s Seventh Framework Programme (FP7/2007-2013) under grant agreement no.257641 (PlanetData NoE).

References

A Appendix: rules tables

Global input rules $I_{glob}(\emptyset)$

\begin{itemize}
  \item A1) $C \in C \mapsto \{\text{subClass}(C, Ctx, gm)\}$
  \item A2) $c \in N \mapsto \{\text{insta}(c, Ctx, gm)\}$
\end{itemize}

Local input rules $I_{loc}(K_m, c)$

\begin{itemize}
  \item L1) $eval(A, C) \subseteq B \mapsto \{\text{subEval}(A, C, B, c)\}$
  \item L2) $eval(R, C) \subseteq T \mapsto \{\text{subEvalR}(R, C, T, c)\}$
\end{itemize}

Local deduction rules $P_{loc}$

\begin{itemize}
  \item P1) $\text{instd}(x, b, c) \leftrightarrow \text{subEval}(a, c, b, c), \text{instd}(x, a, c')$.
  \item P2) $\text{tripled}(x, t, y, c) \leftrightarrow \text{subEvalR}(r, c, t, c), \text{instd}(x', c', gm), \text{tripled}(x, r, y, c')$.
  \item P3) $eq(x, y, c) \leftrightarrow \text{nom}(x, c), eq(x, y, c')$.
\end{itemize}

Output translation $O(\alpha, c)$

\begin{itemize}
  \item O1) $A(a) \mapsto \{\text{instd}(a, A, c)\}$
  \item O2) $R(a, b) \mapsto \{\text{tripled}(a, R, b, c)\}$
  \item O3) $\neg A(a) \mapsto \{\neg \text{instd}(a, A, c)\}$
  \item O4) $\neg R(a, b) \mapsto \{\neg \text{tripled}(a, R, b, c)\}$
\end{itemize}

Table 2. Global, local and output rules

For $D(\alpha) \in S$, apply $I_e(\alpha)$ and the corresponding rule in the following:

\begin{itemize}
  \item (ovr-inst1) $D(A(a)) \mapsto \{\text{oovr}(\text{insta}, a, A, c) \mapsto \neg \text{instd}(a, A, c), \text{prec}(c, g)\}$
  \item (ovr-inst2) $D(\neg A(a)) \mapsto \{\text{oovr}(\neg \text{insta}, a, A, c) \mapsto \text{instd}(a, A, c), \text{prec}(c, g)\}$
  \item (ovr-triple1) $D(R(a, b)) \mapsto \{\text{oovr}(\text{triples}, a, R, b, c) \mapsto \text{tripled}(a, R, b, c), \text{prec}(c, g)\}$
  \item (ovr-triple2) $D(\neg R(a, b)) \mapsto \{\text{oovr}(\neg \text{triples}, a, R, b, c) \mapsto \neg \text{tripled}(a, R, b, c), \text{prec}(c, g)\}$
  \item (ovr-inst3) $D([x] \subseteq B) \mapsto \{\text{oovr}(\text{instd}, x, A, B, c) \mapsto \neg \text{instd}(x, A, B, c), \text{prec}(c, g)\}$
  \item (ovr-subc) $D(A \subseteq B) \mapsto \{\text{oovr}(\text{subClass}, x, A, B, c) \mapsto \text{instd}(x, B, c), \text{instd}(x, A, c), \text{prec}(c, g)\}$
  \item (ovr-not) $D(A \subseteq \neg B) \mapsto \{\text{oovr}(\text{supNot}, x, A, B, c) \mapsto \neg \text{instd}(x, B, c), \text{instd}(x, A, c), \text{prec}(c, g)\}$
  \item (ovr-cnji) $D(A_1 \cap A_2 \subseteq B) \mapsto \{\text{oovr}(\text{subConj}, x, A_1, A_2, B, c) \mapsto \neg \text{instd}(x, B, c), \text{instd}(x, A_1, c), \text{instd}(x, A_2, c), \text{prec}(c, g)\}$
  \item (ovr-sube) $D(\exists R. A \subseteq B) \mapsto \{\text{oovr}(\text{subEx}, x, R, A, B, c) \mapsto \neg \text{instd}(x, B, c), \text{tripled}(x, R, y, c), \text{instd}(x, A, c), \text{prec}(c, g)\}$
  \item (ovr-sapex) $D(A \subseteq \exists R. \{a\}) \mapsto \{\text{oovr}(\text{supEx}, x, A, R, a, c) \mapsto \neg \text{tripled}(x, R, a, c), \text{instd}(x, A, c), \text{prec}(c, g)\}$
  \item (ovr-foral) $D(A \subseteq \forall R. B) \mapsto \{\text{oovr}(\text{supForall}, x, y, A, R, B, c) \mapsto \neg \text{instd}(y, B, c), \text{instd}(x, A, c), \text{prec}(c, g)\}$
  \item (ovr-leqone) $D(A \subseteq \leq 1 R. B) \mapsto \{\text{oovr}(\text{supLeqOne}, x, y, z, A, B, c) \mapsto \neg \text{eq}(x, y, c), \text{instd}(x, A, c), \text{tripled}(x, R, y, c), \text{tripled}(x, R, z, c), \text{instd}(y, B, c), \text{instd}(z, B, c), \text{prec}(c, g)\}$
  \item (ovr-subr) $D(R \subseteq S) \mapsto \{\text{oovr}(\text{subMole}, x, y, R, S, c) \mapsto \neg \text{tripled}(x, S, y, c), \text{tripled}(x, R, y, c), \text{prec}(c, g)\}$
  \item (ovr-subdc) $D(R \circ S \subseteq T) \mapsto \{\text{oovr}(\text{subChain}, x, y, z, R, S, T, c) \mapsto \neg \text{tripled}(x, T, z, c), \text{tripled}(x, R, y, c), \text{tripled}(x, S, z, c), \text{prec}(c, g)\}$
  \item (ovr-dis) $D(\text{Dis}(R, S)) \mapsto \{\text{oovr}(\text{dis}, x, y, R, S, c) \mapsto \text{tripled}(x, S, y, c), \text{tripled}(x, R, y, c), \text{prec}(c, g)\}$
  \item (ovr-inv) $D(\text{Inv}(R, S)) \mapsto \{\text{oovr}(\text{inv}, x, y, R, S, c) \mapsto \neg \text{tripled}(x, R, y, c), \text{tripled}(x, S, y, c), \text{prec}(c, g)\}$
  \item (ovr-irr) $D(\text{Irr}(R)) \mapsto \{\text{oovr}(\text{irr}, x, R, c) \mapsto \neg \text{tripled}(x, R, y, c), \text{prec}(c, g)\}$
\end{itemize}

Table 3. Input rules $I_D(S)$ for defeasible axioms
<table>
<thead>
<tr>
<th>Rule Type</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>(prop-instd)</td>
<td>( \neg )instd ( x, z, c ) ← ( \neg )instd ( x, z, g ), prec ( c, g ), not ovr ( \neg )instd ( x, z, c ).</td>
</tr>
<tr>
<td>(prop-triple)</td>
<td>( \neg )tripled ( x, r, y, c ) ← ( \neg )tripled ( x, r, y, g ), prec ( c, g ), not ovr ( \neg )tripled ( x, r, y, c ).</td>
</tr>
<tr>
<td>(prop-subc)</td>
<td>instd ( x, z, c ) ← subClass ( y, z, g ), instd ( x, y, c ), prec ( c, g ), not ovr subClass ( x, y, z, c ).</td>
</tr>
<tr>
<td>(prop-not)</td>
<td>( \neg )instd ( x, z, c ) ← supNot ( y, z, g ), instd ( x, y, c ), prec ( c, g ), not ovr supNot ( x, y, z, c ).</td>
</tr>
<tr>
<td>(prop-cnj)</td>
<td>instd ( x, z, c ) ← subConj ( y_1, y_2, g ), instd ( x, y_1, c ), instd ( x, y_2, c ), prec ( c, g ), not ovr subConj ( x, y_1, y_2, z, c ).</td>
</tr>
<tr>
<td>(prop-subex)</td>
<td>instd ( x, z, c ) ← supEx ( v, y, z, g ), tripled ( x, v, x', c ), instd ( x', y, c ), prec ( c, g ), not ovr supEx ( x, v, y, z, c ).</td>
</tr>
<tr>
<td>(prop-super)</td>
<td>tripled ( x, r, x', c ) ← supEx ( y, r, x', g ), instd ( x, y, c ), prec ( c, g ), not ovr supEx ( x, y, z, c ).</td>
</tr>
<tr>
<td>(prop-forall)</td>
<td>instd ( y, z', c ) ← supForAll ( z, r, z', g ), instd ( x, z, c ), tripled ( x, r, y, c ), prec ( c, g ), not ovr supForAll ( x, y, z, r, z', c ).</td>
</tr>
<tr>
<td>(prop-locale)</td>
<td>eq ( x_1, x_2, c ) ← supLoc ( z, r, z', g ), instd ( x, z, c ), tripled ( x, r, x_1, c ), instd ( x_1, z', c ), tripled ( x, x_2, c ), instd ( x_2, z', c ), prec ( c, g ), not ovr supLoc ( x, x_1, x_2, z, r, z', c ).</td>
</tr>
<tr>
<td>(prop-subr)</td>
<td>tripled ( x, w, x', c ) ← subRole ( v, w, g ), tripled ( x, v, x', c ), prec ( c, g ), not ovr subRole ( x, v, w, c ).</td>
</tr>
<tr>
<td>(prop-subc)</td>
<td>tripled ( x, w, z, c ) ← subChain ( u, v, w, g ), tripled ( x, u, y, c ), tripled ( y, v, z, c ), prec ( c, g ), not ovr subChain ( u, y, z, w, c ).</td>
</tr>
<tr>
<td>(prop-dis1)</td>
<td>( \neg )tripled ( x, v, y, c ) ← dis ( u, v, g ), tripled ( x, u, y, c ), prec ( c, g ), not ovr dis ( x, y, u, v, c ).</td>
</tr>
<tr>
<td>(prop-dis2)</td>
<td>( \neg )tripled ( x, u, y, c ) ← dis ( u, v, g ), tripled ( x, v, y, c ), prec ( c, g ), not ovr dis ( x, y, u, v, c ).</td>
</tr>
<tr>
<td>(prop-irr1)</td>
<td>tripled ( y, v, c ) ← inv ( u, v, g ), tripled ( x, u, y, c ), prec ( c, g ), not ovr inv ( x, y, u, v, c ).</td>
</tr>
<tr>
<td>(prop-irr2)</td>
<td>tripled ( y, u, x, c ) ← inv ( u, v, g ), tripled ( x, v, y, c ), prec ( c, g ), not ovr inv ( x, y, u, v, c ).</td>
</tr>
<tr>
<td>(prop-inv)</td>
<td>( \neg )tripled ( x, u, x, c ) ← irr ( u, g ), nom ( x, c ), prec ( c, g ), not ovr irr ( x, u, c ).</td>
</tr>
</tbody>
</table>

**Table 4:** Deduction rules \( P_3 \) for defeasible axioms

---

145
**RL input translation** $I_{rl}(S, c)$

- **(irl-nom)** $a \in NI \mapsto \{\text{nom}(a, c)\}$
- **(irl-cls)** $A \in NC \mapsto \{\text{cls}(A, c)\}$
- **(irl-rol)** $R \in NR \mapsto \{\text{rol}(R, c)\}$

- **(irl-inst1)** $\neg \text{triple}(a, u, x, c) \leftarrow \text{irr}(u, c), \text{nom}(x, c)$
- **(irl-inst2)** $\neg \text{triple}(a, R, b, c) \leftarrow \text{irr}(a, c), \text{cls}(R, c)$
- **(irl-triple)** $\text{triple}(a, R, x, c) \leftarrow \{\text{insta}(a, top, c)\}$
- **(irl-subc)** $A \subseteq B \mapsto \{\text{subClass}(A, B, c)\}$
- **(irl-bot)** $\bot(a) \mapsto \{\text{insta}(a, bot, c)\}$

**RL deduction rules** $P_{rl}$

- **(prl-instd)** $\neg \text{instd}(x, z, c) \leftarrow \neg \text{insta}(x, z, c)$
- **(prl-tripled)** $\neg \text{tripled}(x, r, y, c) \leftarrow \neg \text{triple}(x, r, y, c)$

- **(prl-eq1)** $\text{eq}(x, x, c) \leftarrow \text{nom}(x, c)$
- **(prl-eq2)** $\neg \text{eq}(y, x, c) \leftarrow \neg \text{eq}(x, y, c)$
- **(prl-eq3)** $\neg \text{eq}(y, c, z) \leftarrow \text{eq}(x, y, c), \neg \text{instd}(x, z, c)$
- **(prl-eq4)** $\neg \text{tripled}(u, y, z, c) \leftarrow \text{eq}(x, y, c), \neg \text{tripled}(u, x, z, c)$
- **(prl-eq5)** $\neg \text{tripled}(z, y, u, c) \leftarrow \text{eq}(x, y, c), \neg \text{tripled}(z, u, x, c)$
- **(prl-eq6)** $\neg \text{eq}(x, z, c) \leftarrow \neg \text{eq}(x, y, c), \neg \text{eq}(y, z, c)$

- **(prl-top)** $\text{instd}(x, top, c) \leftarrow \text{instd}(x, z, c)$
- **(prl-subc)** $\text{instd}(x, z, c) \leftarrow \text{subClass}(y, z, c), \text{instd}(x, y, c)$
- **(prl-subc)** $\neg \text{instd}(x, z, c) \leftarrow \neg \text{subClass}(y, z, c), \text{instd}(x, y, c)$
- **(prl-subc)** $\text{instd}(x, z, c) \leftarrow \text{subConj}(y_1, y_2, z, c), \text{instd}(x, y_1, c), \text{instd}(x, y_2, c)$
- **(prl-supex)** $\text{tripled}(x, r, x', c) \leftarrow \text{subEx}(y, z, c), \text{tripled}(x, v, x', c), \text{instd}(x, y, c)$
- **(prl-supex)** $\text{tripled}(x, r, x', c) \leftarrow \text{subEx}(y, r, x', c), \text{instd}(x, y, c)$
- **(prl-supforeall)** $\text{tripled}(y, z', c) \leftarrow \text{subForall}(y, z', c), \text{instd}(x, z, c), \text{tripled}(y, r, x, c)$
- **(prl-leqone)** $\text{eq}(x_1, x_2, c) \leftarrow \text{subLeqOne}(z, r, z', c), \text{instd}(x, z, c), \text{tripled}(x, r, x_1, c), \text{instd}(x_1, z', c), \text{tripled}(x, r, x_2, c), \text{instd}(x_2, z', c)$

**Table 5. RL input and deduction rules**
A mechanism for ontology confidentiality

P. A. Bonatti, I. M. Petrova and L. Sauro

Dept. of Electrical Engineering and Information Technologies
Università di Napoli “Federico II”

Abstract. We illustrate several novel attacks to the confidentiality of knowledge bases (KB). Then we introduce a new confidentiality model, sensitive enough to detect those attacks, and a method for constructing secure KB views. We identify safe approximations of the background knowledge exploited in the attacks; they can be used to reduce the complexity of constructing secure KB views. Finally, we describe a prototype implementation of the new approach that suggests its applicability in practice.

1 Introduction

Ontology languages and Linked Open Data are increasingly being used to encode the private knowledge of companies and public organizations. Semantic Web techniques make possible to merge different sources of knowledge and extract implicit information, putting on risk security and privacy of individuals. Even the authors of public ontologies may want to hide some axioms to capitalize on their formalization efforts. Several approaches have been proposed in order to tackle the confidentiality requirements that arise form these scenarios. The most popular security criterion is that the published view of the knowledge base should not entail a secret sentence. However, there exist attacks that cannot be prevented this way. The user may exploit various sources of background knowledge and metaknowledge to reconstruct the hidden part of the knowledge base. This paper contributes to the area of knowledge base confidentiality in several ways:

(i) It highlights some vulnerabilities of the approaches that can be found in the literature, (Sec. 3).
(ii) It introduces a stronger confidentiality model that takes both object-level and meta-level background knowledge into account (Sec. 4), and it defines a method for computing secure knowledge views (Sec. 5) that generalizes some previous approaches.
(iii) It proposes a safe approximation of background metaknowledge (Sec. 6 and 7).
(iv) It investigates the computational complexity of constructing secure knowledge base views with our methodology (Sec. 7).
(v) It describes a prototypical implementation of the new framework (Sec. 9)

The paper is closed by a discussion of related work (Sec. 10), and conclusions. Proofs are omitted due to space limitations.

2 Preliminaries on Description Logics

We assume the reader to be familiar with description logics, and refer to [1] for all definitions and results. We assume a fixed, denumerable signature \( \Sigma \) specifying the names of concepts, roles, and individuals. Our framework is compatible with any description.
logic DL that enjoys compactness (needed by Theorem 6) and has decidable reasoning problems (e.g., ALC, EL, SHIQ, etc.). We simply assume that our reference logical language $\mathcal{L}$ is generated from $\Sigma$ by the grammar of the selected logic DL. By axioms, we mean members of $\mathcal{L}$, unless stated otherwise. A knowledge base is any subset of $\mathcal{L}$.\(^1\)

Recall that axioms are expressions of the form $C \subseteq D$, $R \subseteq S$, $C(a)$, and $R(a, b)$ where $C, D$ are concept expressions, $R, S$ are role expressions, and $a, b$ are individual constants. In some DL, an individual constant $a$ may occur also in a nominal, that is, a concept expression $\{a\}$ denoting the singleton containing $a$. The axioms involving $\subseteq$ are called inclusions (or subsumptions), while $C(a)$ and $R(a, b)$ are called assertions. In the simplest case, $C$ and $R$ are first order predicates and assertions are actually standard first-order atomic formulae. Inclusions are syntactic variants of logical implications.

The notion of logical consequence is the classical one; for all $K \subseteq \mathcal{L}$, the logical consequences of $K$ will be denoted by $\text{Cn}(K)$ ($K \subseteq \text{Cn}(K) \subseteq \mathcal{L}$).

### 3 A simple confidentiality model

The most natural way of preserving confidentiality in a knowledge base $KB$ is checking that its answers to user queries do not entail any secret. Conceptually, the queries of a user $u$ are answered using $u$’s view $KB_u$ of the knowledge base, where $KB_u$ is a maximal subset of $KB$ that entails no secret. In order to illustrate some possible attacks to this mechanism, let us formalize the above simple confidentiality model (SCM).\(^2\) It consists of: the knowledge base $KB$ ($KB \subseteq \mathcal{L}$); a set of users $U$; a view $KB_u \subseteq KB$ for each $u \in U$; a set of secrets $S_u \subseteq \mathcal{L}$ for each $u \in U$. Secrets are axioms that may or may not be entailed by $KB$; if they do, then they are called secrets and must not be disclosed to $u$. Revealing that a secrecy is not entailed by $KB$ is harmless, cf. [4].

A view $KB_u$ is secure iff $\text{Cn}(KB_u) \cap S_u = \emptyset$. A view $KB_u$ is maximal secure if it is secure and there exists no $K$ such that $KB_u \subset K \subseteq KB$ and $\text{Cn}(K) \cap S_u = \emptyset$.

**Attacks using object-level background knowledge.** Frequently, part of the domain knowledge is not axiomatized in $KB$, therefore checking that $\text{Cn}(KB_u) \cap S_u = \emptyset$ does not suffice in practice to protect confidentiality. For example, suppose that there is one secret $S_u = \{\text{OncologyPatient}(\text{John})\}$ and $KB_u = \{\text{SSN}(\text{John}, 12345), \text{SSN}(%user123, 12345), \text{OncologyPatient}(\text{user123})\}$. $KB_u$ does not entail $\text{OncologyPatient}(\text{John})$, so according to the SCM model $KB_u$ is secure. However, it is common knowledge that a SSN uniquely identifies a person, then the user can infer that $\text{John} = \text{user123}$, and hence the secret.

In other examples, the additional knowledge used to infer secrets may be stored in a public ontology or RDF repository, and confidentiality violations may be automated.

**Attacks to complete knowledge.** Suppose the attacker knows that $KB$ has complete knowledge about a certain set of axioms. Then the attacker may be able to reconstruct some secrets from the “I don’t know” answers of a maximal secure view $KB_u$.

**Example 1.** Consider a company’s knowledge base that defines a concept Employee and a role works_for that describes which employees belong to which of the $n$ departments

---

\(^1\) Real knowledge bases are finite, but this restriction is not technically needed until Sec. 7.

\(^2\) This usage of term “model” is common in Security & Privacy.
of the company, $d_1, \ldots, d_n$. The KB consists of assertions like:

\[
\text{Employee}(e) \quad (1) \quad \text{works for}(e, d_i) \quad (2)
\]

where we assume that each employee $e$ belongs to exactly one department $d_i$. A user $u$ is authorized to see all assertions but the instances of (2) with $i = n$, because $d_n$ is a special department, devoted to controlling the other ones. So $S_u$ (the set of secrets for $u$) is the set of all assertions $\text{works for}(e, d_n)$.

Note that there is one maximal secure view $KB_u$. It consists of all instances of (1), plus all instances of (2) such that $i \neq n$. Clearly, $KB_u$ is secure according to SCM (because $Cn(KB_u) \cap S_u = \emptyset$). However, observe that $\text{works for}(e, d_n) \in Cn(KB)$ iff $\text{Employee}(e) \in Cn(KB_u)$ and for all $i = 1, \ldots, n$, $\text{works for}(e, d_i) \not\in Cn(KB_u)$ (that is, the members of $d_n$ are all the employees that apparently work for no department). Using this property (based on the knowledge that for each employee $e$, KB contains exactly one assertion $\text{works for}(e, d_i)$ and the knowledge of the protection mechanism (i.e. maximal secure views), that we assume to be known by attackers by Kerchoff's principle, a smart user can easily identify all the members of $d_n$.

In practice, it is not hard to identify complete knowledge. A hospital's KB is expected to have complete knowledge about which patients are in which ward; a company's KB is likely to encode complete information about its employees, etc.

Some approaches filter query answers rather than publishing a subset of KB [8, 13, 15]. We call our abstraction of this method simple answer confidentiality model (SACM). It is obtained from the SCM by replacing the views $KB_u \subseteq KB$ with answer views $KB^s_u \subseteq Cn(KB)$. The difference is that $KB^s_u$ is not required to be a subset of KB and—conceptually—$KB^s_u$ may be infinite. $KB^s_u$ is secure iff $Cn(KB^s_u) \cap S_u = \emptyset$.

The reader may easily verify that the SACM is vulnerable to the two kinds of attacks illustrated for the SCM. It is also vulnerable to a third kind of attacks, illustrated below.

**Attacks to the signature.** Suppose the user knows the signature of KB well enough to identify a symbol $\sigma$ that does not occur in KB. First assume that $\sigma$ is a concept name. It can be proved that:

**Proposition 1.** If $KB^s_u$ is a maximal secure answer view and $\sigma$ is a concept name not occurring in KB, then for all secrets $C \subseteq D \in S_u$, $KB^s_u \models C \cap \sigma \not\subseteq D \iff KB \models C \not\subseteq D$.

The problem is that although $C \cap \sigma \not\subseteq D$ does not entail the secret inclusion $C \not\subseteq D$, still a smart user knows that the former inclusion cannot be proved unless KB entails also the latter (then maximal secure answer views generally fail to protect secrets). This attack can be easily adapted to the case where $\sigma$ is a role name. In practice, it is not necessary to be sure that $\sigma$ does not occur in KB. The attacker may make a sequence of educated guesses (say, by trying meaningless long strings, or any word that is clearly unrelated to the domain of the KB); after a sufficient number of trials, the majority of answers should agree with the "real" answer with high probability. Rejecting queries whose signature is not contained in KB’s signature mitigates this kind of attacks but it leaks KB’s signature and it does not provide a complete solution. The attacker may still guess a $\sigma$ which is logically unrelated to $C$ and $D$ and carry out a similar attack.
4 A meta-safe confidentiality model

In this section we introduce a confidentiality model that makes the vulnerabilities illustrated above visible, by taking into account object- and meta-level background knowledge. A bk-model \( M = (KB, U, f, (S_u, PKB_u, BK_u)_{u \in U}) \) consists of a knowledge base \( KB \subseteq \mathcal{L} \), a set of users \( U \), plus:

- a filtering function \( f : \varphi(\mathcal{L}) \times U \rightarrow \varphi(\mathcal{L}) \), mapping each knowledge base \( K \) and each user \( u \) on a view \( f(K, u) \subseteq \text{Cn}(K) \);
- for all \( u \in U \):
  - a finite set of secrecies \( S_u \subseteq \mathcal{L} \);
  - a set of axioms \( BK_u \subseteq \mathcal{L} \), encoding the users’ object-level knowledge;
  - a set of possible knowledge bases \( PKB_u \subseteq \varphi(\mathcal{L}) \) (users’ metaknowledge).\(^3\)

The view of \( KB \) released to a user \( u \) is \( f(KB, u) \). We adopt \( PKB \) because at this stage we do not want to tie our framework to any specific metalanguage. \( PKB \) represents the knowledge bases that are compatible with the user’s metaknowledge.

**Definition 1.** A filtering function \( f \) is secure (w.r.t. \( M \)) iff for all \( u \in U \) and all \( s \in S_u \), there exists \( K \in PKB_u \) such that:

1. \( f(K, u) = f(KB, u) \);
2. \( s \notin \text{Cn}(K \cup BK_u) \).

Intuitively, if \( f \) is safe according to Def. 1, then no user \( u \) can conclude that any secret \( s \) is entailed by the \( KB \) she is interacting with—enhanced with the object-level background knowledge \( BK_u \)—for the following reasons: By point 1, \( KB \) and \( K \) have the same observable behavior, and \( K \) is a possible knowledge base for \( u \) since \( K \in PKB_u \); therefore, as far as \( u \) knows, the knowledge base might be \( K \). Moreover, by point 2, \( K \) and the object-level background knowledge \( BK_u \) do not suffice to entail the secret \( s \).

In the rest of the paper we tacitly assume that no secret is violated a priori, that is, for all secrets \( s \in S_u \) there exists \( K \in PKB_u \) such that \( s \notin \text{Cn}(K \cup BK_u) \).\(^4\) Moreover, in order to improve readability, we shall omit the user \( u \) from subscripts and argument lists whenever \( u \) is irrelevant to the context.

The attacks discussed in Section 3 can be easily formalized in this setting; so, in general, the maximal secure views of SCM are not secure according to Def. 1.

**Example 2.** Example 1 can be formalized in our model as follows: The set of secrecies \( S \) is the set of all assertions \( \text{works}_e(d_e) \); \( BK = \emptyset \) and \( PKB \) is the set of all the knowledge bases \( K \) that consist of assertions like (1) and (2), and such that for each axiom \( \text{Employee}(e) \), \( K \) contains exactly one corresponding axiom \( \text{works}_e(d_e) \) and vice versa. The filtering function \( f \) maps each \( K \in PKB \) on the maximal subset of \( K \) that entails none of \( S \)’s members, that is, \( f(K) = K \setminus S \) (by definition of \( PKB \)).

Note that \( f \) is injective over \( PKB \), so condition 1 of Def. 1 is satisfied only if \( K = KB \). So, if \( KB \) contains at least one secret, then the conditions of Def. 1 cannot be satisfied, that is, maximal secure SCM views are not secure in our model. Indeed, \( KB \) can be

---

\(^3\) In practice, bk-models are finite, and filterings computable, but no such assumption will be technically needed until Sec. 7.

\(^4\) Conversely, no filtering function can conceal a secret that is already known by the user.
reconstructed from the secure view by observing that $KB = f(KB) \cup \{\text{works for}(e,d_n) \mid Employee(e) \in f(KB) \land \forall i = 1, \ldots, n, \text{works for}(e,d_i) \notin f(KB)\}$.  

Similarly, the formalizations of the other attacks yield injective filtering functions (the details are left to the reader).

5 A meta-secure query answering mechanism

In this section we introduce a secure filtering function. It is formulated as an iterative process based on a censor, that is a boolean function that decides for each axiom whether it should be obfuscated to protect confidentiality. The iterative construction manipulates pairs $(X^+, X^-) \in \phi(L) \times \phi(L)$ that represent a meta constraint on possible knowledge bases: we say that a knowledge base $K$ satisfies $(X^+, X^-)$ iff $K$ entails all the sentences in $X^+$ and none of those in $X^-$ (formally, $Cn(K) \supseteq X^+$ and $Cn(K) \cap X^- = \emptyset$).

Let $PAX$ (the set of possible axioms) be the set of axioms that may occur in the knowledge base according to the user’s knowledge, i.e. $PAX = \bigcup_{K' \in PKB} K'$. Let $\nu = |PAX| + 1$ if $PAX$ is finite and $\nu = \omega$ otherwise; let $\alpha_1, \alpha_2, \ldots,\alpha_i, \ldots$ be any enumeration of $PAX$ ($i < \nu$).\(^5\) The secure view construction for a knowledge base $K$ in a bk-model $M$ consists of the following, inductively defined sequence of pairs $(K^+_i, K^-_i)_{i\geq 0}$:

- $(K^+_0, K^-_0) = (\emptyset, \emptyset)$, and for all $1 \leq i < \nu$, $(K^+_{i+1}, K^-_{i+1})$ is defined as follows:
  - if $\text{censor}_M(K^+_i, K^-_i, \alpha_{i+1}) = \text{true}$ then $(K^+_{i+1}, K^-_{i+1}) = (K^+_i, K^-_i)$;
  - if $\text{censor}_M(K^+_i, K^-_i, \alpha_{i+1}) = \text{false}$ and $K \models \alpha_{i+1}$ then $(K^+_{i+1}, K^-_{i+1}) = (K^+_i \cup \{\alpha_{i+1}\}, K^-_i)$;
  - otherwise let $(K^+_{i+1}, K^-_{i+1}) = (K^+_i \cup \{\alpha_{i+1}\}, K^-_i)$.

Finally, let $K^+ = \bigcup_{i\leq \nu} K^+_i$, $K^- = \bigcup_{i\leq \nu} K^-_i$, and $f_M(K,u) = K^+$.

Note that the inductive construction aims at finding maximal sets $K^+$ and $K^-$ that (i) partly describe what does (does not) follow from $K$ (as $K$ satisfies $(K^+, K^-)$ by construction), and (ii) do not trigger the censor (the sentences $\alpha_{i+1}$ that trigger the censor are included neither in $K^+$ nor in $K^-$, cf. the induction step).

In order to define the censor we need an auxiliary definition that captures all the sentences that can be entailed with the background knowledge $BK$ and the meta-knowledge $PKB$ enriched by a given constraint $(X^+, X^-)$ analogous to those adopted in the iterative construction: Let $CN_M(X^+, X^-)$ be the set of all axioms $\alpha \in L$ such that

for all $K' \in PKB$ such that $K'$ satisfies $(X^+, X^-)$, $\alpha \in Cn(K' \cup BK)$.

Now the censor is defined as follows: For all $X^+, X^- \subseteq L$ and $\alpha \in L$,

$$
censor_M(X^+, X^-, \alpha) = \begin{cases} 
\text{true} & \text{if there exists } s \in S \text{ s.t. either } s \in Cn_M(X^+ \cup \{\alpha\}, X^-) \\
\text{false} & \text{or } s \in Cn_M(X^+, X^- \cup \{\alpha\}); 
\end{cases}
$$

In other words, the censor checks whether telling either that $\alpha$ is derivable or that $\alpha$ is not derivable to a user aware that the knowledge base satisfies $(X^+, X^-)$, restricts the

\(^5\) We will show later how to restrict the construction to finite sequences, by approximating $PAX$. 

151
set of possible knowledge bases enough to conclude that a secret \( s \) is entailed by the knowledge base and the background knowledge \( BK \).

Note that the censor obfuscates \( \alpha_{i+1} \) if any of its possible answers entail a secret, independently of the actual contents of \( K \) (the two possible answers “yes” and “no” correspond to conditions \( s \in CnM(X^+ \cup \{\alpha\}, X^-) \) and \( s \in CnM(X^+, X^- \cup \{\alpha\}) \), respectively). In this way, roughly speaking, the knowledge bases that entail \( s \) are given the same observable behavior as those that don’t. Under a suitable continuity assumption on \( CnM \), this enforces confidentiality:

**Theorem 1.** If \( CnM(KB^+, KB^-) \subseteq \bigcup_{i<\nu} CnM(KB^+_i, KB^-_i) \), then \( f_M \) is secure w.r.t. \( M \).

Examples of the behavior of \( f_M \) are deferred until Sec.7.

6 Approximating background knowledge

Of course, the actual confidentiality of a filtering \( f(KB, u) \) depends on a careful definition of the user’s background knowledge, that is, \( PKB_u \) and \( BK_u \). If background knowledge is not exactly known, as it typically happens, then it can be safely approximated by overestimating it. More background knowledge means larger \( BK_u \) and smaller \( PKB_u \), which leads to the following comparison relation \( \leq_k \) over bK-models:

**Definition 2.** Given two bK-models \( M = \langle KB, U, f, \langle S_u, PKB_u, BK_u \rangle_{u \in U} \rangle \) and \( M' = \langle KB', U', f', \langle S'_u, PKB'_u, BK'_u \rangle_{u \in U'} \rangle \), we write \( M \leq_k M' \) if

1. \( KB = KB' \), \( U = U' \), \( f = f' \), and \( S_u = S'_u \) (for all \( u \in U \));
2. for all \( u \in U \), \( PKB_u \supseteq PKB'_u \) and \( BK_u \subseteq BK'_u \).

The next proposition proves that a bK-model \( M \) can be safely approximated by any \( M' \) such that \( M \leq_k M' \):

**Proposition 2.** If \( f \) is secure w.r.t. \( M' \) and \( M \leq_k M' \), then \( f \) is secure w.r.t. \( M \).

Consequently, a generic advice for estimating \( BK \) consists in including as many pieces of relevant knowledge as possible, for example:

(i) modelling as completely as possible the integrity constraints satisfied by the data, as well as role domain and range restrictions and disjointness constraints;

(ii) including in \( BK \) all the relevant public sources of formalized relevant knowledge (such as ontologies and triple stores).

While object-level background knowledge is dealt with in the literature, the general metaknowledge encoded by \( PKB \) is novel. Therefore, the next section is focussed on some concrete approximations of \( PKB \) and their properties.

7 Approximating and reasoning about possible knowledge bases

In this section, we investigate the real world situations where the knowledge base \( KB \) is finite and so are all the components of bK-models \( (U, S_u, BK_u, PKB_u) \); then we focus on \( PKB_u \) that contain only finite knowledge bases. Consequently, \( f_M \) will turn out to be decidable and we will study its complexity under different assumptions.
A language for defining PKB is a necessary prerequisite for the practical implementation of our framework and a detailed complexity analysis of \( f_M \). Here we express PKB as the set of all theories that are contained in a given set of possible axioms \( \text{PAX} \)\(^6\) and satisfy a given, finite set \( \text{MR} \) of metarules like:

\[
a_1, \ldots, a_n \Rightarrow \beta_1 | \ldots | \beta_m \quad (n \geq 0, m \geq 0),
\]

where all \( a_i \) and \( \beta_j \) are in \( \mathcal{L} \) (1 \( \leq i \leq n, 1 \leq j \leq m \)). Informally, (5) means that if \( KB \) entails \( a_1, \ldots, a_n \) then \( KB \) entails also some of \( \beta_1, \ldots, \beta_m \). Sets of similar metarules can be succinctly specified using metavariables; they can be placed wherever individual constants may occur, that is, as arguments of assertions, and in nominals. A metarule with such variables abbreviates the set of its ground instantiations: Given a \( K \subseteq \mathcal{L} \), let \( \text{ground}_k(\text{MR}) \) be the ground (variable-free) instantiation of \( \text{MR} \) where metavariables are uniformly replaced by the individual constants occurring in \( K \) in all possible ways.

**Example 3.** Let \( \text{MR} = \{ \exists R, [X] \Rightarrow A(X) \} \), where \( X \) is a metavariable, and let \( K = \{ R(a, b) \} \). Then \( \text{ground}_K(\text{MR}) = \{ (\exists R, [a] \Rightarrow A(a)), (\exists R, [b] \Rightarrow A(b)) \} \). \( \square \)

If \( r \) denotes rule (5), then let \( \text{body}(r) = \{ a_1, \ldots, a_n \} \) and \( \text{head}(r) = \{ \beta_1, \ldots, \beta_m \} \). We say \( r \) is Horn if \( |\text{head}(r)| \leq 1 \). A set of axioms \( K \subseteq \mathcal{L} \) satisfies a ground metarule \( r \) if either \( \text{body}(r) \not\subseteq \text{Cn}(K) \) or \( \text{head}(r) \cap \text{Cn}(K) \neq \emptyset \). In this case we write \( K \models_m r \).

**Example 4.** Let \( A, B, C \) be concept names and \( R \) be a role name. The axiom set \( K = \{ A \subseteq \exists R, B, A \subseteq C \} \) satisfies \( A \subseteq \exists R \Rightarrow A \subseteq B \mid A \subseteq C \) but not \( A \subseteq \exists R \Rightarrow A \subseteq B \). \( \square \)

Moreover, if \( K \) satisfies all the metarules in \( \text{ground}_k(\text{MR}) \) then we write \( K \models_m \text{MR} \). Therefore the formal definition of \( PKB \) now becomes:

\[
PKB = \{ K \mid K \subseteq \text{PAX} \land K \models_m \text{MR} \}.
\]

In accordance with Prop. 2, we approximate \( \text{PAX} \) in a conservative way. We will analyze two possible definitions:

1. \( \text{PAX}_0 = KB \) (i.e., as a minimalistic choice we only assume that the axioms of \( KB \) are possible axioms; of course, by Prop. 2, this choice is safe also w.r.t. any larger \( \text{PAX} \) where at least the axioms of \( KB \) are regarded as possible axioms);
2. \( \text{PAX}_1 = KB \cup \bigcup_{r \in \text{ground}_k(\text{MR})} \text{head}(r) \).

**Remark 1.** The latter definition is most natural when metarules are automatically extracted from \( KB \) with rule mining techniques, that typically construct rules using material from the given \( KB \) (then rule heads occur in \( KB \)).

**Example 5.** Consider again Example 1. The user’s metaknowledge about \( KB \)'s completeness can be encoded with:

\[
\text{Employee}(X) \Rightarrow \text{works_for}(X, d_1) | \ldots | \text{works_for}(X, d_n),
\]

\[\text{Differently from Sec. 5, here PKB is defined in terms of PAX.}\]
where $X$ is a metavariable. First let $PAX = PAX_1$. The secure view $f_M(KB)$ depends on the enumeration order of $PAX$. If the role assertions $\text{works}_\text{for}(e, d_i)$ precede the concept assertions $\text{Employee}(e)$, then, in a first stage, the sets $KB^+_i$ are progressively filled with the role assertions with $d_i \neq d_k$ that belong to $KB$, while the sets $KB^-_i$ accumulate all the role assertions that do not belong to $KB$. In a second stage, the sets $KB^+_i$ are further extended with the concept assertions $\text{Employee}(e)$ such that $e$ does not work for $d_n$. The role assertions $\text{works}_\text{for}(e, d_n)$ of $KB$ and the corresponding concept assertions $\text{Employee}(e)$ are neither in $KB^+$ nor in $KB^-$. Note that the final effect is equivalent to removing from $KB$ all the axioms referring to the individuals that work for $d_n$. Analogously, in [8] the individuals belonging to a specified set are removed from all answers.

Next suppose that the role assertions $\text{works}_\text{for}(e, d_i)$ follow the concept assertions $\text{Employee}(e)$, and that each $\text{works}_\text{for}(e, d_i)$ follows all $\text{works}_\text{for}(e, d_k)$ such that $k < i$. Now all the assertions $\text{Employee}(e)$ of $KB$ enter $KB^+$, and all axioms $\text{works}_\text{for}(e, d_i)$ with $i < n - 1$ enter either $KB^+$ or $KB^-$, depending on whether they are members of $KB$ or not. Finally, the assertions $\text{works}_\text{for}(e, d_i) \in \\text{Cn}(KB)$ with $i \in \{n - 1, n\}$ are inserted neither in $KB^+$ nor in $KB^-$, because the corresponding instance of (7) with $X = e$ has the body in $KB^+$ and the first $n - 2$ alternatives in the head in $KB^-$, therefore a negative answer to $\text{works}_\text{for}(e, d_{n-1})$ would entail the secret $\text{works}_\text{for}(e, d_n)$ by (7). This triggers the censor for all assertions $\text{works}_\text{for}(e, d_{n-1})$. Summarizing, with this enumeration ordering it is possible to return the complete list of employees; the members of $d_n$ are protected by hiding also which employees belong to $d_{n-1}$.

Finally, let $PAX = PAX_0$. Note that in this case all possible knowledge bases are subsets of $KB$, that contains exactly one assertion $\text{works}_\text{for}(e, d_{i(e)})$ for each employee $e$. To satisfy (7), every $K \in PKB$ containing $\text{Employee}(e)$ must contain also $\text{works}_\text{for}(e, d_{i(e)}())$. It follows that $f_M$ must remove all references to the individuals that work for $d_n$, as it happens with the first enumeration of $PAX_1$.

**Definition 3.** A $\text{bk-model} M$ is canonical if for all users $u \in U$, $PAX_u$ is either $PAX_0$ or $PAX_1$ and $PKB_u$ is defined by (6) for a given $MR_u$. Moreover, $M$ is in a description logic $\mathcal{DL}$ if for all $u \in U$, all the axioms in $KB, PKB_u, BK_u, \text{and } S_u$ belong to $\mathcal{DL}$.

The size of $PAX_0$ and $PAX_1$ is polynomial in the size of $KB \cup MR$, therefore $PKB$ is finite and exponential in the size of $KB \cup MR$. Finiteness implies the continuity hypothesis on $\text{Cn}_M$ of Theorem 1, and hence (using Theorem 1 and Prop. 2):

**Theorem 2.** If $M$ is canonical, then $f_M$ is secure with respect to all $M' \leq_k M$.

First we analyze the complexity of constructing the secure view $f_M(KB)$ when the underlying description logic is tractable, like $\mathcal{EL}$ and $\mathcal{DL}$-lite for example.

**Lemma 1.** If the axioms occurring in $MR$ and $K$ are in a $\mathcal{DL}$ with tractable subsumption and instance checking, then checking $K \models_m MR$ is:

1. in $P$ if either $MR$ is ground or there exists a fixed bound on the number of distinct variables in $MR$;
2. coNP-complete otherwise.

With Lemma 1, one can prove the following two lemmas.
Lemma 2. Let $M$ range over canonical bk-models. If $M, s, X^+, and X^-$ are in a DL with tractable subsumption/instance checking, and the number of distinct variables in $MR$ is bounded by a constant, then checking whether $s \in Cn_M(X^+, X^-)$ is:

1. in \( P \) if $MR$ is Horn and \( PAX = PAX_1 \);
2. coNP-complete if either $MR$ is not Horn or \( PAX = PAX_0 \).

Lemma 3. Let $M$ be a canonical bk-model. If $M, s, X^+, and X^-$ are in a DL with tractable entailment problems, and there is no bound on the number of variables in the metarules of $MR$, then checking $s \in Cn_M(X^+, X^-)$ is:

1. in \( P^{NP} \) if $MR$ is Horn and \( PAX = PAX_1 \);
2. in \( \Pi_2^P \) if either $MR$ is not Horn or \( PAX = PAX_0 \).

The value of $\text{censor}(X^+, X^-, \alpha)$ can be computed straightforwardly by iterating the tests $s \in Cn_M(X^+ \cup \{\alpha\}, X^-)$ and $s \in Cn_M(X^+, X^- \cup \{\alpha\})$ for all secrets $s \in S$. Since the set of secrets is part of the parameter $M$ of the filtering function, the number of iterations is polynomial in the input and the complexity of the censor is dominated by the complexity of $Cn_M()$. The latter is determined by Lemma 2 and Lemma 3, so we immediately get:

Corollary 1. Let $M$ be a canonical bk-model and suppose that $M, X^+, X^-$, and $\alpha$ are in a DL with tractable entailment problems. If the number of distinct variables in $MR$ is bounded by a constant, then computing $\text{censor}(X^+, X^-, \alpha)$ is:

- in \( P \) if $MR$ is Horn and \( PAX = PAX_1 \);
- coNP-complete if either $MR$ is not Horn or \( PAX = PAX_0 \).

If there is no bound on the number of variables in the metarules of $MR$, then computing $\text{censor}(X^+, X^-, \alpha)$ is:

- in \( P^{NP} \) if $MR$ is Horn and \( PAX = PAX_1 \);
- in \( \Pi_2^P \) if either $MR$ is not Horn or \( PAX = PAX_0 \).

We are now ready to analyze the complexity of filtering functions:

Theorem 3. If $M$ is a canonical bk-models in a DL with tractable entailment problems, then computing $f_M(KB)$ is:

1. P-complete if the number of distinct variables in the rules of $MR$ is bounded, $MR$ is Horn, and \( PAX = PAX_1 \);
2. P^{NP} complete if the number of distinct variables in $MR$ is bounded, and either $MR$ is not Horn or \( PAX = PAX_0 \);
3. in \( \Delta_3^p \) if the variables in $MR$ are unbounded, $MR$ is Horn, and \( PAX = PAX_1 \);
4. in \( \Delta_3^p \) if $MR$ is not restricted and \( PAX \in \{PAX_0, PAX_1\} \).

Theorem 4. Computing $f_M(KB)$ over canonical $M$ in a DL with ExpTime entailment (e.g. ALCQO, ALCIO, ALCQI, SHOQ, SHIO, SHIQ), is still in ExpTime.

Theorem 5. Computing $f_M(KB)$ over canonical $M$ in SROIQ(D) is in coNP^{NP\text{2ExpTime}}.
8 Relationships with the SCM

Here we show that the meta-secure framework is a natural generalization of the SCM. The main result—roughly speaking—demonstrates that the SCM model can be essentially regarded as a special case of our framework where $PKB \supseteq \mathcal{P}(KB)$ and $BK = \emptyset$. In this case $f_M$ is secure even if $M$ is not assumed to be canonical.

**Theorem 6.** Let $M = \langle KB, U, f_M, \langle S_u, PKB_u, BK_u \rangle_{u \in U} \rangle$. If $PKB = \mathcal{P}(KB)$, $BK = \emptyset$, and $KB$ is finite, then

1. $Cn_M(KB^+, KB^-) = \bigcup_{i < \nu} Cn_M(KB_i^+, KB_i^-)$.
2. For all enumerations of PAX, the corresponding $f_M(KB, u)$ is logically equivalent to a maximal secure view $KB_u$ of $KB$ according to the SCM; conversely, for all maximal secure view $KB_u$ of $KB$ (according to the SCM) there exists an enumeration of PAX such that the resulting $f_M(KB, u)$ is logically equivalent to $KB_u$.
3. $f_M$ is secure w.r.t. $M$ and w.r.t. any $M' = \langle KB, U, f_M, \langle S_u, PKB'_u, BK'_u \rangle_{u \in U} \rangle$ such that $PKB' \supseteq \mathcal{P}(KB)$ and $BK' = \emptyset$.

Theorem 6 applies to every canonical $M$ such that $MR = BK = \emptyset$, because $MR = \emptyset$ implies that $PAX_0 = PAX_1 = KB$ and hence $PKB = \mathcal{P}(KB)$. This shows that the SCM can be regarded as a special case of our framework where the user has no background knowledge. Moreover, by this correspondence, one immediately obtains complexity bounds for the SCM from those for $PAX_1$ and Horn, bounded-variable $MR$.

9 Framework Implementation

In this section we introduce a prototypical implementation of the framework based on $PAX_1$ and Horn metarules. Nowadays ontologies are managed with the help of the OWL API and DL reasoners which allow us to take full advantage of their rich underlying semantics. Unfortunately, the OWL reasoners publicly available do not offer native support for conjunctive query answering required to process users’ metaknowledge. A partial exception of this rule is the Pellet reasoner discussed later. Straightforward evaluation of metarules in the presence of metavariables with an OWL reasoner would need to consider all possible ways of uniformly replacing metavariables by individual constants occurring in the ontology. On the other hand, the evaluation of a ground rule $r$ with an OWL reasoner in the worst case would require checking that all the axioms $\alpha_1, \ldots, \alpha_6 \in \text{body}(r)$ and $\beta_1, \ldots, \beta_m \in \text{head}(r)$ are entailed by $KB$. Summing up, with this method, as the ontology ABox grow, metarule evaluation can easily become unmanageable in terms of execution time. Consequently, the presence of technologies that permit native conjunctive query evaluation reveals fundamental to achieve efficient implementation of the framework. SPARQL, the W3C standard that provide languages and protocols to query and manipulate RDF content (and so ontologies encoded in the XML/RDF Syntax), constitute a de facto standard when it comes to conjunctive query answering. It has recently been extended with the so-called entailment regimes, which

\footnote{http://owlapi.sourceforge.net/}
\footnote{http://www.w3.org/TR/sparql11-overview/}
define how queries are evaluated under more expressive semantics, such as OWL semantics, than the SPARQL standard simple entailment, based essentially on pattern matching on graphs. Unfortunately, most of the available engines do not provide support for OWL reasoning. To the best of our knowledge only Apache Jena Semantic Web Toolkit and Pellet support OWL inference over the queried ontologies. Moreover, Pellet query engine seems not to have been reengineered for the last few years. It was therefore an obvious option to prefer the Jena query engine for our system. The Jena inference subsystem is designed to allow usage of a number of predefined Jena OWL reasoners, as well as external reasoners. However, the usage of the internal reasoners is recommended for efficiency reasons. Note, that OWL, OWL Mini, OWL Micro Jena Reasoners are a set of useful but not a full-fledged rule-based implementations of the OWL/Lite subset of the OWL/Full language. Critical not supported constructs which go beyond OWL/Lite are complementOf and oneOf, while the support for unionOf is partial. We consider our choice to use a Jena OWL reasoner a good compromise between expressivity and efficiency. According to the theoretical framework the system consists of two modules. The first one actuates the parsing of the user’s metaknowledge represented by means of a set of metarules. The second module is in charge of the secure ontology view construction.

Algorithm 1 provides an abstract view on the implementation of our framework. It takes as input an ontology $KB$, a set of secrets $S$, a set of metarules $MR$ and the user’s background knowledge $BK$. The output is the set of axioms that constitute a secure ontology view for the user. The set $M_M$ and $M_G$ form a partition of $MR$ according to rule types (ground or containing metavariables).

**Remark 2.** By standard logic programming techniques, a minimal $PKB \subseteq PAX_1$ satisfying the set of metarules and the constraints $K^+$ can be obtained with the following PTIME construction:

$$ PKB_0 = K^+ \quad PKB_{i+1} = PKB_i \cup \{ \text{head}(r) \mid r \in \text{ground}_{PKB_i}(MR) \land \text{body}(r) \subseteq \text{Cn}(PKB_i) \} $$

The sequence’s limit $PKB_{PAX,1}$ satisfies $\langle K^+, K^- \rangle$ as well if $\text{Cn}(PKB_{PAX,1}) \cap K^- \neq \emptyset$. Then, for all $s \in S$, $s \in \text{Cn}_{M}(K^+, K^-)$ holds iff $s \in \text{Cn}(PKB_{PAX,1}) \cup BK$. For more details refer to [7].

By iterating over the axioms $\alpha \in PAX_1$ (line 6-25), $PKB$ collects at each step all parts of $PAX_1$ that can be revealed to the user. The repeat-until loop (lines 9-17) computes the deductive closure $PKB'$ of $PKB$ under $MR$. In particular, for every ground metarule (lines 10-13) we execute a SPARQL ASK query (hidden in line 11) to verify if its body is entailed by the current $PKB$. For every metarule containing metavariables (lines 14-16) we execute a SPARQL SELECT query (encoded in line 15) in order to obtain all possible bindings of the metavariables that satisfy the metarule’s body. The pair of steps described above is iterated until a fixpoint is reached (no elements are added to $PKB'$ (line 17)). At this point the condition $\text{Cn}(PKB') \cap K^- \neq \emptyset$ is checked (line 18). We are now ready to determine the value of the censor function for $\alpha$. We verify that no secret is entailed from the minimal $PKB$ (line 19) taking in consideration

---

9 To be plugged into Jena a reasoner must expose Jena API.

10 The result of Proposition 2 guarantees that considering only the minimal $PKB$ is sound.
Algorithm 1:

Data: KB, S, MR, BK.

1. \( K^+_i, K^-_i \leftarrow \emptyset \);
2. \( M_M \leftarrow \{ r_i | r_i \in MR \text{ and } r_i \text{ metarule containing metavariables} \} \);
3. \( M_G \leftarrow \{ r_i | r_i \in MR \text{ and } r_i \text{ ground metarule} \} \);
4. \( \text{PAX}_1 \leftarrow \{ \alpha \in KB \cup \bigcup_{i \in \text{ground}_{KB,MR}} \text{head}(r) \} \);
5. \( \text{PKB} \leftarrow \emptyset \);
6. \text{forall} \ \alpha \in \text{PAX}_1 \ \text{do}
7. \( \text{PKB}' \leftarrow \text{PKB} \cup \{ \alpha \} \);
8. \( M'_G \leftarrow M_G \);
9. \text{repeat}
10. \text{forall} \ m \in M'_G \text{ do}
11. \text{if} \ \text{PKB}' \models \text{body}(m) \ \text{then}
12. \( \text{PKB} \leftarrow \text{PKB}' \cup \{ \text{head}(m) \} \);
13. \( M'_G \leftarrow M'_G \setminus \{ m \} \);
14. \text{forall} \ m \in M_M \text{ do}
15. \text{forall} \ (a_0, \ldots, a_n) | \text{PKB}' \models \text{body}(m, [X_0/a_0, \ldots, X_n/a_n]) \text{ do}
16. \( \text{PKB} \leftarrow \text{PKB}' \cup \{ \text{head}(m, [X_0/a_0, \ldots, X_n/a_n]) \} \);
17. \text{until} \ \text{No element is added to PKB};
18. \text{if} \ \{ \beta \in K^-_i | \text{PKB}' \models \beta \} = \emptyset \ \text{then}
19. \text{if} \ \{ s \in S | \text{PKB}' \cup \text{BK} \models s \} = \emptyset \ \text{then}
20. \text{if} \ \text{KB} \models \alpha \ \text{then}
21. \( K^+_i \leftarrow K^+_i \cup \{ \alpha \} \);
22. \( \text{PKB} \leftarrow \text{PKB}' \);
23. \( M'_G \leftarrow M'_G \);
24. \text{else}
25. \( K^-_i \leftarrow K^-_i \cup \{ \alpha \} \);
26. \text{return} \ K^+_i

the background knowledge. In case \( \alpha \) is entailed by \( KB \), it is safe to include it in the view (line 21). Otherwise, the set \( K^-_i \) is updated (line 25). Note that we need an OWL reasoner in order to perform the entailment checks in lines 18-20. We make use of the incremental reasoner Pellet, that for each \( \alpha \) in the enumeration of \( \text{PAX}_1 \), is expected to restrict reasoning to the new inferences triggered by \( \alpha \) without repeating the inferences that involve only \( K^+_{i-1} \).

A first optimization regards the evaluation of the set of ground rules \( M_G \). During the construction of \( \text{PKB} \), due to the monotonicity of reasoning, at each iteration we can safely remove from \( M_G \) all the ground rules already satisfied at the previous iterations (line 13,23). Another optimization concerns the evaluation order of \( \text{PAX}_1 \). Checking \( \text{Cn}(\text{PKB}_{\text{PAX}_1}) \cap K^- \neq \emptyset \) (line 18) is time consuming, so we adopt an approach that main-

---

11 This corresponds to check whether \( s \in \text{Cn}_{M_G}(K^+ \cup \{ \alpha \}, K^-) \) only. The condition \( s \in \text{Cn}_{M_G}(K^+, K^- \cup \{ \alpha \}) \) is in fact embedded in line 18.
tain the set $K_i^-$ as small as possible. This is achieved processing all $\{\alpha \in PAX_1 \mid \alpha \in KB\}$ in line 6 first. Provided that the condition in line 19 is vacuously satisfied we are sure that the $K_i^-$ remains empty.

Experimental analysis show that the generation of secure views for medium sized ontologies may take several minutes. We plan to investigate module extraction techniques that are expected to improve drastically the execution time by restricting the part of the knowledge base on which metarules apply.

10 Related work

Baader et al. [2], Eldora et al. [11], and Knechtel and Stuckenschmidt [13] attach security labels to axioms and users to determine which subset of the KB can be used by each subject. These works are instances of the SCM so they are potentially vulnerable to the attacks based on background knowledge; this holds in particular for [13] that pursues the construction of maximal secure views. Moreover, in [2, 11] axiom labels are not derived from the set of secrets; knowledge engineers are responsible for checking ex post that no confidential knowledge is entailed; in case of leakage, the view can be modified with a revision tool based on pinpointing. On the contrary, our mechanism automatically selects which axioms shall be hidden in order to produce a secure view.

Chen and Stuckenschmidt [8] adopt an instance of the SACM based on removing some individuals entirely. In general, this may be secure against metaknowledge attacks (cf. Ex. 5). However, no methodology is provided for selecting the individuals to be removed given a target set of secrets. In [3], $KB$ is partitioned into a visible part $KB_v$ and a hidden part $KB_h$. Conceptually, this is analogous to axiom labelling, cf. the above approaches. Their confidentiality methodology seems to work only under the assumption that the signatures of $KB_v$ and $KB_h$ are disjoint, because in strong safety they do not consider the formulae that are implied by a combination of $KB_v$ and $KB_h$. Surely the axioms of $KB_h$ whose signature is included in the signature of $KB_v$ cannot be protected, in general. A partition-based approach is taken in [10], too. It is not discussed how to select the hidden part $KB_h$ given a set of target secrets (which includes the issue of deciding secondary protection).

Similarly, in [14] only ex-post confidentiality verification methods are provided. In their model the equivalent of $PKB$ is the set of all knowledge bases that include a given set of publicly known axioms $S \subseteq KB$; consequently, in some cases their verification method is vulnerable to the attacks to complete knowledge, that are based on more complex (conditional) metaknowledge (cf. Example 2 and Example 5) that cannot be encoded in their framework.

Cuenca Grau and Horrocks [9] investigate knowledge confidentiality from a probabilistic perspective: enlarging the public view should not change the probability distribution over the possible answers to a “sensitive query” $Q$ that represents the set of secrets. In [9] users can query the knowledge base only through a pre-defined set of views (we place no such restriction, instead). A probability distribution $P$ over the set of knowledge bases plays a role similar to metaknowledge. However, their confidentiality condition allows $P$ to be replaced with a different $P'$ after enlarging the public
view, so at a closer look $P$ does not really model the user’s a priori knowledge about the knowledge base (that should remain constant), differently from our $PKB$.

Our method is inspired by the literature on controlled query evaluation (CQE) based on lies and/or refusals ([4, 5, 6] etc). Technically we use lies, because rejected queries are not explicitly marked (the cited papers use the special answer “mum”). However, our censor resembles the classical refusal censor, so the properties of $f_M$ are not subsumed by any of the classical CQE methods. For example (unlike the CQE approaches that use lies), $f_M(KB, u)$ encodes only correct knowledge (i.e. entailed by $KB$), and it is secure whenever users do not initially know any secret (while lies-based CQE further require that no disjunction of secrets should be known a priori). Unlike the refusal method, $f_M$ can handle cover stories because users are not told that some queries are obfuscated; as an additional advantage, our method needs not to adapt existing engines to handle nonstandard answers like mum. Finally, the CQE approaches do not deal specifically with DL knowledge bases, metaknowledge, and related complexity analysis.

11 Conclusions

The confidentiality preservation methods that do not consider background knowledge are vulnerable to several attacks. We identified two vulnerabilities (attacks to complete knowledge and to the signature) and introduced a knowledge base confidentiality model that can detect these vulnerabilities, based on a fully generic formalization of object- and meta-level background knowledge. Confidentiality is enforced through a generic mechanism for constructing secure views (the filtering $f_M$) that is provably secure w.r.t. the meta-confidentiality model under a continuity assumption, and generalizes a few previous approaches (cf. Thm. 6 and Ex. 5). In order to compute secure views in practice we introduced a safe, generic method for approximating background knowledge, together with a specific rule-based language for expressing metaknowledge. Based on this instantiation of the general framework, where $f_M$ is always secure, we analyzed the computational complexity of computing secure views. If the underlying DL is tractable, then in the simplest case $f_M$ can be computed in polynomial time. The number of variables in metarules and the adoption of a more secure approximation ($PAX_0$) may increase complexity up to $P^{NP} = \Delta_2^P$ and perhaps $\Delta_3^P$. The complexity of non-Horn metarules, however, can be avoided by replacing each non-Horn $r$ with one of its Horn strengthenings: $\text{body}(r) \Rightarrow \alpha$ such that $\alpha \in \text{head}(r)$). This approximation is safe (because it restricts $PKB$), and opens the way to a systematic use of the low-complexity bk-models based on $PAX_1$ and Horn metarules. For the many ExpTime-complete DL, secure view computation does not increase asymptotic complexity. So far, the best upper complexity bound for computing secure views in the description logic underlying OWL DL (i.e. $\text{SROIQ}(\mathcal{D})$) is $\text{coNP}^{\text{ExpTime}}$.

Finally, we have provided a prototype implementation of the low-complexity frameworks based on $PAX_1$ and Horn metarules using incremental engine versions available for Pellet and ELK to avoid repeated classifications in the iterative construction of $f_M$. Metarule bodies are evaluated with SPARQL. Secure views are constructed off-line, so no overhead is placed on user queries, and this approach is expected to be applicable in practice.
Bibliography


Herbrand-satisfiability of a Quantified Set-theoretical Fragment

Domenico Cantone, Cristiano Longo, and Marianna Nicolosi Asmundo

Dipartimento di Matematica e Informatica, Università di Catania, Italy
{cantone, longo, nicolosi}@dmi.unict.it

Abstract. In the last decades, several fragments of set theory have been studied in the context of the so-called Computable Set Theory. In general, the semantics of set-theoretical languages differs from the canonical first-order semantics in that the interpretation domain of set-theoretical terms is fixed to a given universe of sets, as for instance the von Neumann standard cumulative hierarchy of sets, i.e., the class consisting of all the pure sets. Because of this, theoretical results and various machinery developed in the context of first-order logic cannot be easily adapted to work in the set-theoretical realm. Recently, quantified fragments of set-theory which allow one to explicitly handle ordered pairs have been studied for decidability purposes, in view of applications in the field of knowledge representation. Among other results, a NExpTime decision procedure for satisfiability of formulae in one of these fragments, $\forall \pi_0$, has been provided. In this paper we exploit the main features of such a decision procedure to reduce the satisfiability problem for the fragment $\forall \pi_0$ to the problem of Herbrand satisfiability for a first-order language extending it. In addition, it turns out that such reduction maps formulae of the Disjunctive Datalog subset of $\forall \pi_0$ into Disjunctive Datalog programs.

1 Introduction

The quantified fragment of set-theory $\forall \pi_0$ (see [3]) allows the explicit manipulation of ordered pairs. It is expressive enough to include a relevant amount of set-theoretic constructs, in particular map-related ones: in fact, it is characterized by the presence of terms of the form $\cdot, \cdot$ (ordered pair) and $\pi(\cdot)$ (collection of the non-pair members of its argument). This language has applications in the field of knowledge representation. In fact, a large amount of description logic constructs are expressible in it. In particular, the very expressive description logic $\mathcal{DL}(\forall \pi_0)$ can be expressed in a sublanguage of $\forall \pi_0$ which has an NP-complete decision problem, in contrast with the description logic $\mathcal{SROIQ}$ (cf. [9]), which underpins the current standard language for Semantic Web OWL2\(^1\) and whose decision problem is N2ExpTime-complete (see [10]). Despite of these desirable

---

\* This work has been supported by the project PON04a2,A “PRISMA - PiattafoRme cloud Interoperabili per SMArt-government.”

\(^1\) http://www.w3.org/TR/owl2-primer/
properties of the language $\forall \pi \_0$, no decision procedure for it has been implemented yet.

In general, the semantics of set-theoretical languages (see [11] for an introduction to set theory, and [5, 12] for an overview on decidable fragments of set-theory) differs from the canonical first-order semantics (see [7]) in that the interpretation domain of set-theoretical terms is based on the von Neumann standard cumulative hierarchy of sets $\mathcal{V}$. This is the class consisting of all the pure sets, and is recursively defined by

\[ \mathcal{V}_0 = \emptyset \]
\[ \mathcal{V}_{\gamma+1} = \mathcal{P}(\mathcal{V}_\gamma), \quad \text{for each ordinal } \gamma \]
\[ \mathcal{V}_\lambda = \bigcup_{\mu < \lambda} \mathcal{V}_\mu, \quad \text{for each limit ordinal } \lambda \]
\[ \mathcal{V} = \bigcup_{\gamma \in \text{On}} \mathcal{V}_\gamma, \]

where $\mathcal{P}(\cdot)$ is the powerset operator and $\text{On}$ denotes the class of all ordinals. Because of such peculiarity of set-theoretical languages, reusing of theoretical results and machinery developed in the context of first-order logic for set-theoretical matters is not straightforward.

In this paper we show that these difficulties can be circumvented for the fragment $\forall \_0$ by encoding some axioms of set theory (namely regularity and a weak form of extensionalilty) as $\forall \_0$-formulae. More specifically, we will prove that, for every $\forall \_0$-formula $\varphi$, one can construct in polynomial time a corresponding $\forall \_0$-formula $\chi_\varphi$ such that $\varphi$ admits a set-theoretical model if and only if $\chi_\varphi \land \varphi$ admits a Herbrand model (cf. [8]), when set-theoretic predicates in $\chi_\varphi \land \varphi$ are regarded as uninterpreted predicates.

It turns out that these formulae $\chi_\varphi$ can be considered as $\text{DATALOG}^{\lor, \neg}$ programs. As a consequence, the reduction we are going to discuss can be seen as a reduction from $\forall \_0$ to $\text{DATALOG}^{\lor, \neg}$, where $\forall \_0 \text{D} \lor$ consists of the formulae in $\forall \_0$ which satisfy the syntactic constraints required to be regarded also as $\text{DATALOG}^{\lor, \neg}$ programs.

We recall that $\text{DATALOG}^{\lor, \neg}$ (read Disjunctive Datalog, see [6]) extends Datalog by allowing disjunctions in the head of program rules. Decidability and complexity of Datalog extensions with various combinations of disjunction and negation has been studied. Also, optimization strategies have been provided for algorithms devised in this context, and a considerable amount of academic and commercial software implementing these algorithms is available (see for example [1]). Thus, the reduction we present in this paper allows one to reuse the machinery available for $\text{DATALOG}^{\lor, \neg}$ in the implementation of an optimized reasoning engine for the language $\forall \_0 \text{D} \lor$, and, consequently, for the description logic $\mathcal{DL}(\forall \_0)$ mentioned before, as all the constructs in $\mathcal{DL}(\forall \_0)$ are expressible in $\forall \_0 \text{D} \lor$.

The link between set-theoretical languages and logic programming we start to investigate here is of a certain interest to projects related with the production and usage of open data such as PRISMA*: on one hand, it provides a way to implement with small efforts a solid and efficient reasoning and query engine for the very expressive representation language $\mathcal{DL}(\forall \_0)$; on the other hand, it allows...
one to implement typical logic-programming tasks for $\mathcal{DL}(\forall_0)$ knowledge bases such as, for example, answer-set programming and non-monotonic reasoning, which may be of some interest for public utility applications.

The rest of the paper is organized as follows. In Section 2 we review some notions and definitions from first-order logic, including the definition of the Disjunctive Datalog fragment of first-order logic. Then, in Section 3, after recalling the syntax and semantics of the fragment of set-theory $\forall_0$, we briefly review a decision procedure for $\forall_0$-formulae, together with some lemmas useful to our needs. A polynomial-time reduction of the satisfiability problem for $\forall_0$-formulae to the Herbrand satisfiability problem for first-order formulae is described in Section 4. Finally, we draw our conclusions and provide some hints for future research in Section 5.

2 Preliminaries

We briefly review some notations and definitions from first-order logic which will be used throughout the paper.

We shall denote with $\text{Vars} = \{x, y, z, \ldots\}$ and $\text{Consts} = \{a, b, c, \ldots\}$ two denumerably infinite collections of variables and constants, respectively. In addition, we denote with $\text{Preds} = \{P, Q, R, \ldots\}$ a denumerably infinite set of predicate symbols, and with $\text{ar}: \text{Preds} \rightarrow \mathbb{N}$ the related arity function.

Atomic formulae are expressions of the following form

$$P(\nu_1, \ldots, \nu_n),$$

where $P \in \text{Preds}$, $\text{ar}(P) = n$, and $\{\nu_1, \ldots, \nu_n\} \in \text{Vars} \cup \text{Consts}$. Literals are atomic formulae or their negations. Quantifier-free formulae are propositional combinations of atomic formulae; thus, in particular, literals are quantifier-free formulae. Universally quantified prenex-formulae are expressions of the following form

$$(\forall x_1) \ldots (\forall x_n) \psi,$$

where $n \in \mathbb{N}$, $\{x_1, \ldots, x_n\} \subseteq \text{Vars}$ and $\psi$ is a quantifier-free formula. In the rest of the paper we will sometime abbreviate quantifier prefixes as $(\forall x_1) \ldots (\forall x_n)$ by $(\forall x_1, \ldots, x_n)$. Here we do not mention existential quantifiers because they do not serve our purposes. However, notice that constants and free variables may be regarded as existentially quantified variables when dealing with satisfiability. In addition, quantifier-free formulae can be considered as universally quantified prenex formulae with an empty quantifier prefix.

We consider also restricted universal quantifiers, i.e., quantifiers of the form

$$(\forall x_1, \ldots, x_n | P(x_1, \ldots, x_m)), \quad \text{ where } n \leq m,$$

$\forall_0$-formulae.

$^{2}$ We do not mention function symbols here as we are going to consider function-free languages only.

$^{3}$ An occurrence of a variable $x$ is free in a formula if it does not fall within the scope of the quantifier $(\forall x)$. 

164
where \(1 \leq m \leq n\), \(x_1, \ldots, x_n\) are variables, and \(P\) is a predicate of arity \(m\), whose intended meaning is that (universal) quantification is restricted to all \(x_1, \ldots, x_n\) such that \(P(x_1, \ldots, x_m)\) holds. Notice, however, that restricted universal quantifiers can easily be expressed by universally quantified prenex formulae, by way of the following equivalence:

\[
(\forall x_1, \ldots, x_n | P(x_1, \ldots, x_m)) \psi \equiv (\forall x_1, \ldots, x_n)(P(x_1, \ldots, x_m) \rightarrow \psi).
\]

Given any formula \(\varphi\), we denote with \(\text{Preds}(\varphi)\), \(\text{Consts}(\varphi)\), and \(\text{Vars}(\varphi)\) the sets of the predicate symbols, constant symbols, and variables occurring in \(\varphi\), respectively. Analogously, for a set \(\Sigma\) of formulae, we denote with \(\text{Preds}(\Sigma)\), \(\text{Consts}(\Sigma)\), and \(\text{Vars}(\Sigma)\) the sets of all the predicate symbols, constant symbols, and variables occurring in any formula \(\varphi \in \Sigma\), respectively.

We denote with \(\varphi[x \rightarrow y]\) the formula obtained from \(\varphi\) by replacing every free occurrence of \(x\) with \(y\).

A formula is said to be ground if no variable occurs in it. It is said to be closed (or to be a sentence) if no free variable occurs in it, i.e., if every variable \(x\) in it occurs only within the scope of the quantifier \((\forall x)\). Plainly, ground formulae are closed.

Finally, given any set \(\Sigma\) of atomic formulae and any atomic formula \(\gamma\), with a slight abuse of notation we write \(\Sigma \models \gamma\) to express that \(\gamma\) is a member of \(\Sigma\), i.e., \(\gamma \in \Sigma\). Likewise, we write \(\Sigma \not\models \gamma\) when \(\gamma \notin \Sigma\).

First-order interpretations, whose universe is \(\text{Consts}\) and such that each constant is interpreted by itself,\(^4\) are called Herbrand interpretations. Plainly, a Herbrand interpretation is characterized by the set of ground atomic formulae which are evaluated to \text{true} by it. A Herbrand interpretation \(\mathcal{H}\) is said to be a Herbrand model for a formula \(\varphi\) if it evaluates \(\varphi\) to \text{true}, in which case we write \(\mathcal{H} \models \varphi\).

A formula is said to be Herbrand-satisfiable if and only if it admits a Herbrand model. It is a fundamental result of first-order logic that, for satisfiability purposes, it is not restrictive to limit oneself to Herbrand satisfiability, since a formula is satisfiable if and only if it is Herbrand-satisfiable. Additionally, when considering a (function-free) universally quantified prenex sentence \(\varphi\), the search space for Herbrand models of \(\varphi\) can be limited to Herbrand interpretations over the constants occurring in it. This result can easily be generalized to finite conjunctions of universally quantified prenex sentences, as stated in the following theorem.

**Theorem 1.** Let \(\varphi\) be a finite conjunction of universally quantified prenex sentences. Then \(\varphi\) is satisfiable if and only if it admits a Herbrand model \(\mathcal{H}\) whose universe coincides with the set \(\text{Consts}(\mathcal{H})\), if \(\text{Consts}(\mathcal{H}) \neq \emptyset\), otherwise is any singleton. \(\square\)

We conclude this section by recalling the \(\text{DATALOG}^{\land \lor} \) first-order fragment. Notice that \(\text{DATALOG}^{\land \lor}\)-formulae are often referred to also as programs, so that

\(^4\) We recall that we are assuming that there are no function symbols.
the expressions “DATALOG\^{∨,¬}-formulae” and “DATALOG\^{∨,¬}-programs” must be regarded as synonyms. A DATALOG\^{∨,¬}-formula is a finite conjunction of rules, i.e., closed formulae of the following form

\[(\forall x_1)\ldots(\forall x_n)(\varphi \rightarrow \psi),\]

where \(\varphi\) (the rule body) is a conjunction of literals, \(\psi\) (the rule head) is a disjunction of literals, and \(\text{Vars}(\psi) \subseteq \text{Vars}(\varphi)\) (safety condition). Facts are special ground rules whose body is always true. For this kind of rules, one may omit to indicate the rule body. Thus facts can just be regarded as disjunctions of ground literals. They are used to express facts about real world items, such as for example \(\text{childOf}(\text{Alice, Bob}), \text{isMale}(\text{Bob}),\) etc.

Finally, we observe that restricted universal quantifiers can easily be embedded in DATALOG\^{∨,¬} rules and formulae. Indeed, let us consider a closed formula of the form

\[(\forall x_1,\ldots,x_n|P(x_1,\ldots,x_m))(\varphi \rightarrow \psi),\]

where \(\varphi\) is a conjunction of literals, \(\psi\) is a disjunction of literals, \(\text{Vars}(\psi) \subseteq \text{Vars}(\varphi) \cup \{x_1,\ldots,x_m\}\), and \(P\) is a predicate symbol of arity \(m\). Then, as remarked above, (1) is equivalent to the closed formula

\[(\forall x_1,\ldots,x_n)(P(x_1,\ldots,x_m) \rightarrow (\varphi \rightarrow \psi)),\]

which, in its turn, is equivalent to the closed formula

\[(\forall x_1,\ldots,x_n)((P(x_1,\ldots,x_m) \land \varphi) \rightarrow \psi).\]

Plainly, (2) is a (standard) DATALOG\^{∨,¬}-rule, with body \((P(x_1,\ldots,x_m) \land \varphi)\).

3 The language \(\forall_0^\pi\)

We recall the syntax and semantics of the set-theoretic language \(\forall_0^\pi\), whose decision problem has been studied in [3]. Atomic \(\forall_0^\pi\)-formulae are of the following types

\[\nu \in \bar{\pi}(\mu),\quad [\nu,\nu'] \in \mu,\quad \nu = \mu\]

with \(\nu,\nu',\mu \in \text{Vars} \cup \text{Consts}\). Intuitively, a formula of type \(\nu \in \bar{\pi}(\mu)\) expresses that \(\nu\) is a non-pair member of \(\mu\), whereas a formula of type \([\nu,\nu'] \in \mu\) expresses that the pair \([\nu,\nu']\) belongs to \(\mu\). Atomic \(\forall_0^\pi\)-formulae and their negations are called \(\forall_0^\pi\)-literals. Quantifier-free \(\forall_0^\pi\)-formulae are Boolean combinations of atomic \(\forall_0^\pi\)-formulae. Simple-prenex \(\forall_0^\pi\)-formulae have the following form:

\[(\forall x_1 \in \bar{\pi}(a_1))\ldots(\forall x_n \in \bar{\pi}(a_n))((\forall y_1,z_1 \in b_1)\ldots(\forall y_m,z_m \in b_m)\psi),\]

where \(n, m \geq 0\), \(xi \in \text{Vars}\) and \(ai \in \text{Consts}\), for \(1 \leq i \leq n\); \(y_j\) and \(z_j \in \text{Vars}\) and \(bj \in \text{Consts}\), for \(1 \leq j \leq m\), and \(\psi\) is a quantifier-free \(\forall_0^\pi\)-formula. The constants \(a_1,\ldots,a_n, b_1,\ldots,b_m\) are the domain constants of the simple-prenex \(\forall_0^\pi\)-formula under consideration. Finally, \(\forall_0^\pi\)-formulae are finite conjunctions of
closed simple-prenex $\forall\pi^0$-formulae. The collection of the domain constants of a $\forall\pi^0$-formula $\varphi$, which we denote with $\text{Consts}_D(\varphi)$, consists of the domain constants of all of its conjuncts.

Semantics of the language $\forall\pi^0$ is based on the von Neumann standard cumulative hierarchy of sets, briefly reviewed in Section 1. Other set-theoretic notions particularly useful to our purposes are those of Cartesian product and of ordered pairs. Let $\pi$ be a binary operation which associates to each pair of sets $u, v \in \mathcal{V}$ another set in $\mathcal{V}$ (i.e., $\pi(u,v) \in \mathcal{V}$). Then the Cartesian product of $u, v \in \mathcal{V}$ with respect to $\pi$, denoted by $u \times^{(\pi)} v$, is defined by

$$u \times^{(\pi)} v = \{ \pi(u', v') \mid u' \in u \land v' \in v \}.$$ 

The binary operation $\pi$ is said to be a pairing function if and only if the following two conditions hold for any $u, v, u', v' \in \mathcal{V}$:

1. $\pi(u,v) = \pi(u',v') \iff u = u' \land v = v'$,
2. $u \times^{(\pi)} v$ is a set in $\mathcal{V}$.

Let $\pi$ be a pairing function, and let $u$ be a set in the von Neumann cumulative hierarchy. We denote with $\bar{\pi}(u)$ the collection of the members of $u$ which are not pairs, with respect to the pairing function $\pi$.

The semantics for $\forall\pi^0$-formulae is given in terms of pair-aware set-theoretical interpretations. These are first-order interpretations $I$ whose domain is $\mathcal{V}$ (so that pure sets are assigned to constants and variables) and such that:

- the membership predicate $\in$ is interpreted as the membership relation among sets in $\mathcal{V}$;
- the pairing symbol $\cdot$ is interpreted by a pairing function, in the sense described above;
- $I[\pi](x) = \{ u \in Ix \mid (\forall v_1, v_2)(I[v_1, v_2] \neq u) \}$, for all $x \in \text{Vars} \cup \text{Consts}$.

Then, a $\forall\pi^0$-formula $\varphi$ is said to be satisfiable if and only if it admits a pair-aware set-theoretical interpretation which satisfies it (i.e., a pair-aware set-theoretical model). We use the notation $I \models_\pi \varphi$ to indicate that $I$ is a pair-aware set-theoretical model for the $\forall\pi^0$-formula $\varphi$.

The satisfiability test for $\forall\pi^0$-formulae reported in [3] relies on the existence of finite structures of bounded size, called skeletal representations, which witness the existence of certain particular models called realizations (see Definitions 1 and 2).

The definition of skeletal representation adopted in this paper differs slightly from the one presented in [3]. In particular, here we extend the notion of skeletal representation so as to also encapsulate the notion of $V$-extensionality and the technical condition (i) of Theorem 3 in [3]. It turns out that all the lemmas and theorems in [3] can be adapted to cope with this extended definition of skeletal representation. A reformulation of Theorem 3 in [3], which is central for the satisfiability problem for $\forall\pi^0$-formulae, is reported in Theorem 2 below.

In the following definition, we shall make use of the membership closure relation $\in_S^\pi$ of a set $S$ of atomic $\forall\pi^0$-formulae. This is the minimal transitive relation on $\text{Consts}(S)$ such that, for $a, b, c \in \text{Consts}(S)$, we have
– if \( S \models a \in \bar{\pi}(b) \) then \( a \in S^+ b \), and
– if \( S \models [a, b] \in c \) then \( a \in S^+ c \) and \( b \in S^+ c \).

**Definition 1 (Skeletal Representation).** Let \( \varphi \) be a \( \forall_0^\pi \)-formula and let \( V, T \) be two disjoint sets of constants. A skeletal representation \( S \) relative to \( (V, T) \) is a finite set of ground atomic \( \forall_0^\pi \)-formulae such that the following conditions hold:

\[ (S1) \] the membership relation induced by \( S \) is acyclic, i.e., \( a \not\in S^+ a \), for all \( a \in V \cup T \);
\[ (S2) \] if \( S \models a = b \), for some \( a \) and \( b \), then, for every \( \gamma \) in \( S \), \( \gamma[a \rightarrow b] \) and \( \gamma[b \rightarrow a] \) are both in \( S \);
\[ (S3) \] if \( S \not\models a = b \), for some \( a, b \in V \), then \( a \) and \( b \) must be distinguished in \( S \) by some constant \( c \), in the sense that \( S \models c \in \bar{\pi}(a) \) iff \( S \not\models c \in \bar{\pi}(b) \), or by some pair \([c, d]\), in the sense that \( S \models [c, d] \in a \) iff \( S \not\models [c, d] \in b \);
\[ (S4) \] statements in \( S \) of the form \( a = b \) can involve only constants occurring in \( V \);
\[ (S5) \] Consts(\( S \)) \( \subseteq V \cup T \).

Condition \((S1)\), which is closely related to the regularity axiom of set theory, guarantees that a skeletal representation \( S \) can be turned into a corresponding pair-aware set-theoretical interpretation (its realization; see below). Conditions \((S2)\) and \((S3)\) concern the extensionality axiom of set theory, which states that any two sets are equals if and only if they have the same members. Finally, \((S4)\) and \((S5)\) are technical conditions which, together with \((S1)-(S3)\) in Definition 1, guarantee some properties of realizations, as reported in Lemma 1 below.

In the following definition of realization of a skeletal representation, taken from from [3] (Definition 1), we shall make use of the family \( \{\pi_n\}_{n \in N} \) of pairing functions, recursively defined by

\[
\begin{align*}
\pi_0(u, v) & = \text{def} \{u, \{u, v\}\} \\
\pi_{n+1}(u, v) & = \text{def} \{\pi_n(u, v)\},
\end{align*}
\]

for \( u, v \in V \).

**Definition 2 (Realization [3]).** Let \( V \) and \( T = \{t_1, t_2, \ldots, t_n\} \), with \( n \in N \), be two finite, nonempty, and disjoint sets of constants, and let \( S \) be a skeletal representation relative to \( (V, T) \). Then the realization of \( S \) relative to \( (V, T) \) is the pair-aware set-theoretical model \( \mathcal{R} \) defined as follows:

\[
\begin{align*}
\mathcal{R}[u, v] & = \text{def} \pi_h(u, v) \quad \text{for all } u, v \in V \\
\mathcal{R}x & = \text{def} \{\mathcal{R}y \mid S \models y \in \bar{\pi}(x)\} \cup \{\mathcal{R}[y, z] \mid S \models [y, z] \in x\} \quad \text{for } x \in V \\
\mathcal{R}t_i & = \text{def} \{\mathcal{R}y \mid S \models y \in \bar{\pi}(t_i)\} \cup \{\mathcal{R}[y, z] \mid S \models [y, z] \in t_i\} \cup \{(k + 1, k, i)\} \quad \text{for } t_i \in T
\end{align*}
\]

where \( h = |V| + |T| \) and \( k = |V| \cdot (|V| + |T| + 3) \).\(^5\)

\(^5\) We are assuming that integers are represented à la von Neumann, namely \( 0 = \text{def} \emptyset \) and, recursively, \( n + 1 = \text{def} n \cup \{n\} \).

168
The following lemma is a direct consequence of Definition 2 and of Lemma 2 in [3].

**Lemma 1.** Let $V, T$ be two finite, nonempty, and disjoint sets of constants, $S$ a skeletal representation relative to $(V, T)$, and $R$ the realization of $S$ relative to $(V, T)$. Then

$$R \subseteq \{ R \; b \; | S \models b \in \pi(a) \} \cup \{ R \; [b, c] \; | S \models [b, c] \in a \},$$

for all $a \in V$. In addition, the following conditions hold, for all $a, b, c \in V \cup T$:

- $R \; a = R \; b$ iff $S \models a = b$;
- $R \; a \in R \; \pi(b)$ iff $S \models a \in \pi(b)$;
- $R \; [a, b] \in R \; c$ iff $S \models [a, b] \in c$.

We observe that, in view of Definition 2, the names of the variables in the set $T$ do not affect the realization of a skeletal representation relative to $(V, T)$. In other words, if $S$ is a skeletal representation relative to $(V, T)$, $t$ is a constant in $T$, and $t'$ is another constant not occurring in $V \cup T$, then the realization of $S$ relative to $(V, T)$ and the realization of $S[t \rightarrow t']$ (i.e., the set of atomic $\forall \pi_0$-formulae obtained from $S$ by replacing each occurrence of $t$ by $t'$) relative to $(V, T \setminus \{t\} \cup \{t'\})$ coincide. In addition, if $V, T$ are two disjoint set of constants, and $T'$ is a set of constants disjoint from $V$ and such that $T \subseteq T'$, then

- every skeletal representation $S$ relative to $(V, T)$ is a skeletal representation relative to $(V, T')$ as well, and
- the realization of $S$ relative to $(V, T)$ coincides with the realization of $S$ relative to $(V, T')$.

We conclude this section by restating Theorem 3 of [3], in view of the above observations and of Definition 1.

**Theorem 2.** Let $\varphi$ be a $\forall \pi_0$-formula, let $V = \text{Consts}(\varphi)$, and let $T$ be a set of constants disjoint from $V$ such that $|T| = 2 \cdot |V|$. Then $\varphi$ is satisfiable if and only if there exists a skeletal representation $S$ relative to $(V, T)$ such that the realization $R$ of $S$ relative to $(V, T)$ is a pair-aware set-theoretical model for $\varphi$. \hfill \Box

The decidability of the satisfiability problem for $\forall \pi_0$-formulae easily follows from Theorem 2, as the number of possible skeletal representations for any given $\forall \pi_0$-formula is finitely bounded, and it is effectively verifiable whether the realization of a skeletal representation is a pair-aware set-theoretical model for a $\forall \pi_0$-formula.

Next we define the *Disjunctive Datalog subset* $\forall \pi_0^{\text{D, V}}$ of $\forall \pi_0$ as the collection of the closed $\forall \pi_0$-formulae whose conjuncts are closed simple-prenex $\forall \pi_0$-formulae of the form

$$(\forall x_1 \in \pi(a_1)) \ldots (\forall x_n \in \pi(a_n)) (\forall [y_1, z_1] \in b_1) \ldots (\forall [y_m, z_m] \in b_m) \psi, \quad (4)$$
such that $\psi$ has the form

$$(\gamma_1 \land \ldots \land \gamma_l) \rightarrow (\sigma_1 \lor \ldots \lor \sigma_h),$$

where $l, h \geq 0$ and $\gamma_1, \ldots, \gamma_l, \sigma_1, \ldots, \sigma_h$ are $\forall \pi_0$-literals. Since (4) is closed, each variable $x$ which may occur in the head of the rule $\psi$ must be bound, so that $x$ occurs in at least one atom of the rule body, as required for Disjunctive Datalog rules, when restricted quantifiers in (4) are expanded as indicated in (2).

In the next section we present a reduction of the set-theoretic satisfiability for $\forall \pi_0$-formulae to Herbrand satisfiability.

### 4 Herbrand Satisfiability of $\forall \pi_0$-formulae

In this section we show how to reduce the satisfiability problem for the fragment $\forall \pi_0$ to the problem of first-order Herbrand satisfiability.

For this purpose, we introduce the function-free first-order language $L^*_0$ which involves, besides constants, also the following predicate symbols:

<table>
<thead>
<tr>
<th>binary</th>
<th>ternary</th>
<th>4-ary</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\in, P^\pi, P^\bar{\pi}, \text{dist}, \text{dist}_\pi$</td>
<td>$P^{\bar{1}}, \text{distBy}$</td>
<td>$\text{distBy}_\pi$</td>
</tr>
</tbody>
</table>

Next, we define a polynomial-time reduction $\varphi \mapsto \psi_\varphi$ from $\forall \pi_0$-formulae such that any $\forall \pi_0$-formula $\varphi$ is set-theoretically satisfiable if and only if its image $\psi_\varphi$ in $L^*_0$ is Herbrand-satisfiable.

In our reduction, the predicate $\tilde{\in}$ will be used to model the transitive closure of the membership relation among sets, whereas the predicates $\text{dist}, \text{dist}_\pi, \text{distBy}, \text{distBy}_\pi$ will be used to model the fact that two sets are distinct. In particular, the predicates $\text{dist}$ and $\text{distBy}$ will take care of the case in which two sets $x, y$ are distinguished by a set $z$ that is not a pair, whereas the predicates $\text{dist}_\pi$ and $\text{distBy}_\pi$ will take care of the case in which two sets $x, y$ are distinguished by a $[\nu_1, \nu_2]$. Finally, the predicate $P^\pi(\nu_1, \nu_2)$ will be used to model equality between $\nu_1$ and $\nu_2$, the predicate $P^{\bar{1}}(\nu_1, \nu_2)$ will stand for the set-theoretic formula $\nu_1 \in \tilde{\pi}(\nu_2)$, and $P^{\bar{1}}(\nu_1, \nu_2, \nu_3)$ will stand for $[\nu_1, \nu_2] \in \nu_3$, where $\nu_1, \nu_2, \nu_3 \in \text{Vars} \cup \text{Consts}$.

For the sake of clarity, with a slight abuse of notation, we shall write $\nu_1 = \nu_2$, $\nu_1 \in \tilde{\pi}(\nu_2)$, and $[\nu_1, \nu_2] \in \nu_3$ in place of $P^\pi(\nu_1, \nu_2), P^{\bar{1}}(\nu_1, \nu_2)$ and $P^{\bar{1}}(\nu_1, \nu_2, \nu_3)$, respectively, for all $\nu_1, \nu_2, \nu_3 \in \text{Vars} \cup \text{Consts}$. With such an understanding, $\forall \pi_0$ can be regarded as a sublanguage of $L^*_0$, and skeletal representations as Herbrand interpretations (over $L^*_0$) subject to the conditions reported in Definition 1.

In the rest of the paper, we shall refer to the subset of a Herbrand interpretation $\mathcal{H}$ consisting of the atomic $\forall \pi_0$-formulae in $\mathcal{H}$ as the $\forall \pi_0$-subset of $\mathcal{H}$.

As a first step in our reduction, we provide a polynomial-time procedure to construct an $L^*_0$-formula $\chi^{(V, T)}$ from two disjoint sets of constants $V$ and $T$, which enforces the conditions of Definition 1 on its models, in such a way that

- the $\forall \pi_0$-subset of every Herbrand model $\mathcal{H}$ for $\chi^{(V, T)}$ is a skeletal representation relative to $(V, T)$, and
every skeletal representation relative to \((V,T)\) can easily be extended to a Herbrand model for \(\chi(V,T)\).

Then, to complete the reduction of \(\forall_0^t\)-satisfiability to Herbrand satisfiability, we shall prove that
\[
S \models \chi(V,T) \land \varphi \iff R \models_s \varphi,
\]
where \(V, T\) are two disjoint sets of constants, \(\varphi\) is a \(\forall_0^t\)-formula such that \(\text{Consts}(\varphi) = V\), \(S\) is a skeletal representation relative to \((V,T)\), and \(R\) is the realization of \(S\) relative to \((V,T)\).

Thus, let \(V, T\) be two disjoint sets of constants. The formula \(\chi(V,T)\) is defined as follows:
\[
\chi(V,T) = \text{def} \chi_1 \land \chi_2 \land \chi_3^{(V,T)} \land \chi_4^{(V,T)},
\]
where
\[
\chi_1 = \text{def} (\forall x, y)(x \in \bar{\pi}(y) \to x \bar{\in} \bar{z})
\land (\forall x, y)(|x, y| \in z \to x \bar{\in} z)
\land (\forall x, y)(|x, y| \in z \to y \bar{\in} z)
\land (\forall x, y, z)(x \bar{\in} y \land y \bar{\in} z \to x \bar{\in} z)
\land (\forall x)\neg (x \bar{\in} x)
\]
\[
\chi_2 = \text{def} (\forall x, y)(x = y \to y = x)
\land (\forall x, y, z)(x = y \land y = z \to x = z)
\land (\forall x, y, z)(x \in \bar{\pi}(y) \land x = z \to x \in \bar{\pi}(y))
\land (\forall x, y, z)(x \in \bar{\pi}(y) \land y = z \to x \in \bar{\pi}(z))
\land (\forall x, y, z, v)(|x, y| \in z \land x = v \to [v, y] \in z)
\land (\forall x, y, z, v)(|x, y| \in z \land y = v \to [x, v] \in z)
\land (\forall x, y, z, v)(|x, y| \in z \land z = v \to [x, y] \in v)
\]
\[
\chi_3^{(V,T)} = \text{def} \bigwedge_{x, y \in V} \neg (x = y) \to \text{dist}(x, y) \lor \text{dist}_{\pi}(x, y)
\land (\forall x, y)(\text{dist}(x, y) \to \bigvee_{z \in V \cup T} (\text{dist}_{By}(x, y, z) \lor \text{dist}_{By}(x, y, z, z)))
\land (\forall x, y)(\text{dist}_{\pi}(x, y) \to \bigvee_{z, v \in V \cup T} (\text{dist}_{By}(x, y, z, v) \lor \text{dist}_{By}(x, y, z, v)))
\land (\forall x, y, z)(\text{dist}_{By}(x, y, z) \to (z \in \bar{\pi}(x) \land \neg (z \in \bar{\pi}(y)))
\land (\forall x, y, z, v)(\text{dist}_{By}(x, y, z, v) \to ([z, v] \in x \land \neg ([z, v] \in y)))
\]
\[
\chi_4^{(V,T)} = \text{def} \bigwedge_{x \in V \cup T, t \in T, x \neq t} \neg (x = t).
\]

It can be easily verified that \(\chi^{(V,T)}\) is a DATALOG\(^{\forall,-}\)-formula. The subformulae \(\chi_1, \chi_2, \chi_3^{(V,T)},\) and \(\chi_4^{(V,T)}\) formalize the conditions \((S1), (S2), (S3),\) and \((S4)\) of Definition 1, as clarified by the following lemma.
Lemma 2. Let $V, T$ be two disjoint sets of constants. Let $\mathcal{H}$ be a model for $\chi^{(V,T)}$ such that $\text{Consts}(\mathcal{H}) \subseteq V \cup T$, and let $S$ be the $\forall_0^-$-subset of $\mathcal{H}$. Then $S$ is a skeletal representation relative to $(V,T)$.

Proof. Plainly, $S$ is a set of atomic $\forall_0^-$-formulae, since it is the $\forall_0^-$-subset of $\mathcal{H}$. We must prove that $S$ satisfies all the conditions reported in Definition 1.

We first observe that the relation $\in^+ = \{[x,a]| [x,a] \in^+ \} \subseteq \mathcal{H}$ is acyclic, as $\mathcal{H}$ models correctly the conjunct $\chi_1$ of $\chi^{(V,T)}$. Hence, $\in^+_S$ must be acyclic as well, as required by Condition (S1), since $\in^+_S \subseteq \in^+_H$. Thus, we can conclude that $S$ satisfies (S1), since $S \subseteq \mathcal{H}$.

Next, suppose that $S \models x = y$, for some constants $x, y \in V \cup T$. Plainly, $\mathcal{H} \models x = y$. Let us consider any $\forall_0^-$-formula $\gamma$ in $S \subseteq \mathcal{H}$ involving either $x$ or $y$, say $x \in \pi(z)$. The definition of $S$ yields that $z \in V \cup T$, as we are assuming that $S \models x \in \pi(z)$. Then, $\mathcal{H} \models \gamma[x \rightarrow y]$, where $\gamma[x \rightarrow y]$ is the literal $y \in \pi(z)$, easily follows, since $\mathcal{H}$ models correctly the conjunct $(\forall x, y, z)(x \in \pi(y) \land x = z \rightarrow z \in \pi(y))$ of $\chi_2$. In addition, $S \models y \in \pi(z)$ holds as well, as $y \in \pi(z)$ is a $\forall_0^-$-formula with $y, z \in V \cup T$. Property (S2) for $S$ can be proved by reasoning in similar way, for all the $\forall_0^-$-formulae occurring in $S$.

Conversely, let $x, y \in V$ be such that $S \not\models x = y$ and, consequently, $\mathcal{H} \not\models x = y$. Thus, $\mathcal{H} \models \neg(x = y)$, so that $\mathcal{H} \models \text{dist}(x, y) \lor \text{dist}_\pi(x, y)$, as we are supposing that $\mathcal{H}$ is a model for $\chi_3^{(V,T)}$. It can easily be verified that if $\mathcal{H} \models \text{dist}(x, y)$, then $x$ and $y$ are distinguished in $S$ by a constant (in the sense of Definition 1), whereas, if $\mathcal{H} \models \text{dist}_\pi(x, y)$, then $x$ and $y$ are distinguished in $S$ by a pair term. Let us assume that $\mathcal{H} \models \text{dist}(x, y)$. Thus $\mathcal{H} \models \text{dist}By(x, y, z) \lor \text{dist}By(y, x, z)$, for some constant $z \in V \cup T$, so that either $\mathcal{H} \models z \in \pi(x)$ and $\mathcal{H} \not\models z \in \pi(y)$, or $\mathcal{H} \models z \in \pi(y)$ and $\mathcal{H} \not\models z \in \pi(x)$. Then $x$ and $y$ are distinguished in $\mathcal{H}$ by $z$, and, consequently, they are distinguished in $S$ by $z$ too, since $z \in \pi(x)$ and $z \in \pi(y)$ are $\forall_0^-$-formulae and $x, y \in V \cup T$. By reasoning as above, it can be proved that if $\mathcal{H} \models \text{dist}_\pi(x, y)$ then $x$ and $y$ are distinguished in $S$ by the pair $[z, v]$ if $\mathcal{H} \models \text{dist}(x, y)$, thus concluding the proof that $\mathcal{H}$ satisfies (S3).

Concerning (S4), by way of contradiction we prove that $S \models x = y$ implies that $x$ and $y$ are constants in $V$. Thus, let us assume that $y \not\in V$. Then $y$ must be in $T$, as $T$ and $V$ are disjoint, and $\text{Consts}(S) \subseteq \text{Consts}(\mathcal{H}) \subseteq V \cup T$. But $\mathcal{H}$ must model the conjunct $\neg(x = y)$ of $\chi_3^{(V,T)}$. Hence, we have $\mathcal{H} \not\models x = y$ and, a fortiori, $S \not\models x = y$, since $S \subseteq \mathcal{H}$, contradicting our initial assumption. Thus, $y \in V$. In addition, $S \models x = y$ yields that $S \models y = x$, as $\mathcal{H}$ models the conjunct $(\forall x)(x = y \rightarrow y = x)$ of $\chi_2$, and $x$ and $y$ are in $V \cup T$, and $y = x$ is a $\forall_0^-$-formula. Thus, $x \in V$ can be proved analogously.

Finally, (S5) holds as well, since, by assumption, $\text{Consts}(S) \subseteq \text{Consts}(\mathcal{H}) \subseteq V \cup T$.

In addition, every skeletal interpretation can be extended to comply with the constraints imposed by $\chi^{(V,T)}$.

---

6 The relation $\in^+_H$ does not necessarily coincide with $\in_H$, as the minimality of $\in_H$ is not enforced in $\chi_1$. 

172
Lemma 3. Given two disjoint sets of constants $V,T$ and a skeletal representation $S$ relative to $(V,T)$, there always exists a Herbrand model $H$ (over $L^0_0$) having $S$ as $\forall^n$-subset.

Proof. Let $S$ be a skeletal representation relative to $(V,T)$, where $V$ and $T$ are disjoint sets of constants. We can extend $S$ to a Herbrand model $H$ for $\chi^{(V,T)}$.

Initially, we put $H =_{\text{def}} S$ and then we enrich $H$ with new atomic formulae according to the following rules. If $x \in \pi(y)$ is in $S$, we add $x \in y$ to $H$. Likewise, if $[x,y] \in z$ is in $S$, we add the atoms $x \in z$ and $y \in z$ to $H$. Then we close transitively $H$ with respect to the $\hat{\varepsilon}$-atoms, i.e., for each sequence of atoms in $H$ of the form

$$x_1 \hat{\varepsilon} x_2, \ x_2 \hat{\varepsilon} x_3, \ldots, \ x_{n-1} \hat{\varepsilon} x_n,$$

we add the atom $x_1 \hat{\varepsilon} x_n$ to $H$.

Next, for every $x, y \in V$, if $S \not\models x = y$ and $x$ and $y$ are distinguished in $S$ by a $z \in V \cup T$, we add to $H$ the atoms $\text{dist}(x,y)$ and $\text{dist}(y,x)$ (resp., $\text{dist}(y,x)$ and $\text{dist}(x,y)$) provided that $z \in \pi(x)$ is in $S$ and $z \in \pi(y)$ is not in $S$ (resp., $z \in \pi(y)$ is in $S$ and $z \in \pi(x)$ is not in $S$). Otherwise, if $x$ and $y$ are distinguished in $S$ by a pair $[z,v]$ with $z, v \in V \cup T$, we add to $H$ the atoms $\text{dist}(x,y)$ and $\text{dist}(y,x)$ (resp., $\text{dist}(y,x)$ and $\text{dist}(x,y)$) provided that $[z,v] \in x$ is in $S$ and $[z,v] \in y$ is not in $S$ (resp., $[z,v] \in y$ is in $S$ and $[z,v] \in x$ is not in $S$).

In order to show that the set $H$ so obtained is a Herbrand model for $\chi^{(V,T)}$, we have to prove that $H$ is a Herbrand model for the formulae $\chi_1, \chi_2, \chi_3^{(V,T)}$, and $\chi_4^{(V,T)}$.

By the very construction process, $H$ satisfies the first four conjuncts of $\chi_1$. In addition, $H$ satisfies also the last conjunct $(\forall x)-(x \hat{\varepsilon} x)$ of $\chi_1$. Indeed, if this were not the case, $H$ would contain an atom of the form $z \hat{\varepsilon} z$, for some $z \in V \cup T$.

But this would be possible only if one of the following situations occurs:

(a) $S$ contains an atom of one of the following three forms

$$z \in \hat{\pi}(z), \ [z,y] \in z, \ [x,z] \in z,$$

for some $x, y \in V \cup T$;

(b) $H$ contains a sequence of atoms of the form

$$z \hat{\varepsilon} x_1, \ x_1 \hat{\varepsilon} x_2, \ldots, \ x_{n-1} \hat{\varepsilon} x_n, \ x_n \hat{\varepsilon} z,$$

with $x_i \in V \cup T$ for $1 \leq i \leq n$.

However, case (a) cannot occur, since, by condition (S1), the membership relation induced by $S$ is acyclic. We show that also case (b) cannot occur, thereby proving that $H$ cannot contain any atom of the form $z \hat{\varepsilon} z$. Indeed, if (b) were true, then $H$ would contain a maximal sequence of atoms of the form

$$x_0 \hat{\varepsilon} x_1, \ x_1 \hat{\varepsilon} x_2, \ldots, \ x_{n-1} \hat{\varepsilon} x_n, \ x_n \hat{\varepsilon} x_{n+1},$$

with $x_i \in V \cup T$ for $1 \leq i \leq n$.
where \( x_0 \) and \( x_{n+1} \) coincide. But then, for each \( i = 0, 1, \ldots, n \), \( S \) would contain at least an atom of one of the following types

\[
x_i \in \bar{\pi}(x_{i+1}), \quad [x_i, w] \in x_{i+1}, \quad [w, x_i] \in x_{i+1},
\]

with \( w \in V \cup T \), and therefore the membership relation \( \in^+_D \) induced by \( S \) would contain a cycle, contradicting condition \((S1)\).

Summing up, \( H \) satisfies also the last conjunct of \( \chi_1 \), and hence it satisfies the whole formula \( \chi_1 \).

To show that \( H \) satisfies \( \chi_2 \) and \( \chi_3^{(V,T)} \), it is enough to observe that \( S \subseteq H \) and that \( S \) satisfies conditions \((S2)\) and \((S3)\), respectively.

Finally, since (again) \( S \subseteq H \) and \( S \) satisfies condition \((S4)\), it follows that \( H \) satisfies also the formula \( \chi_4^{(V,T)} \), as no new atom of the form \( x = y \) can possibly be introduced into \( H \) during its construction.

Hence, in conclusion, \( H \) is a Herbrand model for the formula \( \chi^{(V,T)} \). \( \square \)

As already remarked, the following lemma exploits a semantical correspondence between pair-aware set-theoretical models and Herbrand models of \( \forall_0^\infty \)-formulae.

**Lemma 4.** Let \( V, T \) be two disjoint sets of constants and let \( H \) be a Herbrand interpretation such that \( H \models \chi^{(V,T)} \) and \( \text{Consts}(H) \subseteq V \cup T \). Let \( S \) be the \( \forall_0^\infty \)-subset of \( H \), and let \( R \) be the realization of \( S \) relative to \( V, T \). Then

\[
H \models \varphi \iff R \models_s \varphi,
\]

for every \( \forall_0^\infty \)-formula \( \varphi \) such that \( \text{Consts}(\varphi) \subseteq V \cup T \) and \( \text{Consts}_D(\varphi) \subseteq V \).

**Proof.** It will be enough to prove the lemma in the case of simple-prenex \( \forall_0^\infty \)-formulae, as general \( \forall_0^\infty \)-formulae are just finite conjunctions of simple-prenex \( \forall_0^\infty \)-formulae. Accordingly, in the rest of this proof we will assume that \( \varphi \) is a simple-prenex \( \forall_0^\infty \)-formula.

We proceed by induction on the quantifier prefix length of \( \varphi \). Preliminarily, we observe that

\[
H \models \gamma \iff R \models_s \gamma,
\]

for each atomic \( \forall_0^\infty \)-formula \( \gamma \) such that \( \text{Consts}(\gamma) \subseteq V \cup T \) (see Lemma 1), so that the lemma follows easily from propositional logic when \( \varphi \) is quantifier-free.

For the inductive case, let us first assume that \( \varphi = (\forall x \in \bar{\pi}(a))\psi \), for some \( x \in \text{Vars} \) and \( a \in \text{Consts} \), where \( \psi \) is a simple-prenex \( \forall_0^\infty \)-formula with one less quantifier than \( \varphi \). Since \( R \) is a realization, we plainly have \( R \bar{\pi}(a) = \{ Rb \mid H \models b \in \bar{\pi}(a) \} \), so that \( R \) models correctly \( (\forall x \in \bar{\pi}(a))\psi \) if and only if it correctly models all the formulae \( \psi[x \rightarrow b] \) such that \( H \models b \in \bar{\pi}(a) \). Observe that, for all the constants \( b \) such that \( H \models b \in \bar{\pi}(a) \), we have \( \text{Consts}_D(\psi[x \rightarrow b]) = \text{Consts}_D(\psi) \setminus \{a\} \subseteq V \), and \( \text{Consts}(\psi[x \rightarrow b]) \subseteq \text{Consts}(\psi) \cup \{b\} \subseteq \text{Consts}(\psi) \cup \{a\} \subseteq V \cup T \), as we have assumed that \( \text{Consts}(\psi) \cup \text{Consts}(H) \subseteq V \cup T \). Thus

\[
R \models_s \psi[x \rightarrow b] \iff H \models \psi[x \rightarrow b],
\]
for each \( b \) such that \( H \models b \in \bar{\pi}(a) \), and, consequently,

\[
\mathcal{R} \models_s (\forall x \in \bar{\pi}(a)) \psi \iff H \models (\forall x \in \bar{\pi}(a)) \psi
\]

follows by applying the inductive hypothesis to the formulae \( \psi[x \rightarrow b] \), for all \( b \) such that \( H \models b \in \bar{\pi}(a) \).

Next, let us consider the case in which \( \varphi \) has the form \((\forall x, y \in a)\psi\), with \( x, y \in \text{Vars} \), \( a \in \text{Consts} \), and \( \psi \) a simple-prenex \( \forall_0^n \)-formula with one less quantifier than \( \varphi \). In this case we must consider the set \( \mathcal{R}a \setminus \bar{\pi}(a) \) of the pair members of \( \mathcal{R}a \). But, \( \mathcal{R}a \setminus \bar{\pi}(a) = \{ \mathcal{R}[b, c] \mid H \models [b, c] \in a \} \), by Lemma 1, and thus \( \mathcal{R} \) models \((\forall x, y \in a)\psi\) if and only if it correctly models the formula \( \psi[x \rightarrow b][y \rightarrow c] \), for all \( a, b \) such that \( H \models [b, c] \in a \). Thus, the thesis follows by reasoning much as in the previous case, by applying the inductive hypothesis to the formulae \( \psi[x \rightarrow b][y \rightarrow c] \), for all \( a, b \) such that \( H \models [b, c] \in a \) \( \square \).

Finally, the next theorem states that the satisfiability of every \( \forall_0^n \)-formula \( \varphi \) can be decided by checking the Herbrand satisfiability of the corresponding \( \mathcal{L}_0^n \)-formula \( \chi^{(V, T)} \land \varphi \), thus concluding the verification of the correctness of our reduction.

**Theorem 3.** Let \( \varphi \) be a \( \forall_0^n \)-formula, let \( V = \text{Consts}(\varphi) \), and let \( T \) be any set of constants disjoint from \( V \) such that \( |T| = 2 \cdot |V| \). Then \( \varphi \) is satisfiable if and only if \( \chi^{(V, T)} \land \varphi \) is Herbrand-satisfiable.

**Proof.** To begin with, let us assume that \( \varphi \) is satisfiable. Then, by Theorem 2, there exists a skeletal representation \( \mathcal{S} \) (relative to \( (V, T) \)) such that its realization \( \mathcal{R} \) relative to \( (V, T) \) is a pair-aware set-theoretical model for \( \varphi \). In addition, by Lemma 2, there exists a Herbrand interpretation \( H \) with \( \mathcal{S} \) as its \( \forall_0^n \)-subset, which correctly models \( \chi^{(V, T)} \), and such that \( \text{Consts}(\mathcal{H}) = \text{Consts}(\mathcal{S}) \). Thus, \( \mathcal{H} \models \varphi \) directly follows from Lemma 4.

Conversely, let \( \mathcal{H} \) be a Herbrand model for \( \chi^{(V, T)} \land \varphi \), and let \( \mathcal{S} \) be the \( \forall_0^n \)-subset of \( \mathcal{H} \). We have \( \text{Consts}(\mathcal{H}) \subseteq V \cup T \), as \( \text{Consts}(\chi^{(V, T)}) = V \cup T \) and \( \text{Consts}(\varphi) = V \). In addition, from Lemma 2 and from \( \mathcal{H} \models \chi^{(V, T)} \), \( \mathcal{S} \) is a skeletal representation. Thus, by Lemma 4, the realization of \( \mathcal{S} \) relative to \( (V, T) \) must be a model for \( \varphi \). \( \square \)

We conclude this section by observing that, if \( \varphi \) is a \( \forall_0^0,\text{DV} \)-formula, then \( \chi^{(V, T)} \land \varphi \) is a \( \text{DATALOG}^{\forall,\text{-}n} \)-formula, as \( \chi^{(V, T)} \) is a \( \text{DATALOG}^{\forall,\text{-}n} \)-formula. Thus the satisfiability problem for \( \forall_0^0,\text{DV} \)-formulae can be reduced in polynomial time to the Herbrand satisfiability problem for \( \text{DATALOG}^{\forall,\text{-}n} \)-formulae.

**Corollary 1.** Let \( \varphi \) be a \( \forall_0^0,\text{DV} \)-formula. Let \( V = \text{Consts}(\varphi) \) and let \( T \) be any set of constants disjoint from \( V \) and such that \( |T| = 2 \cdot |V| \). Then \( \varphi \) is set-theoretically satisfiable if and only if the corresponding \( \text{DATALOG}^{\forall,\text{-}n} \)-formula \( \chi^{(V, T)} \land \varphi \) is satisfiable, in the sense of Disjunctive Datalog. \( \square \)
5 Conclusions and Future Works

In this paper we have identified a correspondence between the fragment of set-theory $\forall\pi_0$ and first-order logic (in particular Herbrand logic) by providing a polynomial-time reduction of $\forall\pi_0$-formulae to formulae in a first-order language, called $L_{\pi_0}$, suitable for this purpose. In addition, we have shown that if we limit ourselves to the Disjunctive Datalog restriction of $\forall\pi_0$ called $\forall\pi_0,D_{\lor}$, then our reduction maps formulae in this subfragment to DATALOG$_{\lor,\neg}$-formulae.

Such correspondence, and its consequences, has to be further investigated. For instance, applications of techniques and results devised in the context of logic programming, such as, for example, answer-set programming and negation-as-failure, to the $\forall\pi_0,D_{\lor}$ subfragment need to be studied. In view of our reduction, a satisfiability checker for $\forall\pi_0,D_{\lor}$-formulae can be implemented by reusing some machinery from logic programming, for example the Disjunctive Datalog system DLV described in [1].

We intend to develop analogous correspondences for other decidable fragments of set-theory, such as, for instance, the quantified fragment $\forall\pi_0,2$ whose decision procedure presented in [2] is based on a reduction to $\forall\pi_0$, and the unquantified fragment $\text{MLSS}^\times_{2,m}$ (see [4]), whose satisfiability problem can be in turn reduced to the satisfiability problem for $\forall\pi_0,2$.

References


7 http://www.dlvsystem.com/
Semantic Web Services for
Integrated Tourism in the Apulia Region

Francesca A. Lisi and Floriana Esposito

Dipartimento di Informatica, Università degli Studi di Bari “Aldo Moro”, Italy
{francesca.lisi,floriana.esposito}@uniba.it

Abstract. In this paper we report our ongoing work on Puglia@Service, a PON Research & Competitivity project aimed at creating an innovative service infrastructure for the Apulia Region, Italy. A sector of interest to the project for the application of the infrastructure is Integrated Tourism. In this sector, we have defined Semantic Web Services following the OWL-S approach. The services have been based on OWL ontologies in the domain of travel and tourism, which have been populated with data of Apulia. This will enable users and software agents to automatically discover, invoke, compose, and monitor Web resources offering services, under specified constraints, for Integrated Tourism in Apulia.

1 Introduction

Tourism activity is becoming more competitive, more extensive, more complicated, and more demanding of host communities and their culture and environment. Tourism planning has been beset by a number of new challenges such as the ones posed by the principles of sustainable development. In order for the tourism enterprise in any destination area to respond positively to these challenges, it is necessary for tourism planning to be practised in a fashion commensurate with the needs of the destination area and the nation. Many reasons are offered for tourism planning, not least the advocacy that planning is the best way of extending the vital life-cycle of a destination by providing a means of anticipating changes, adjusting to the demands of change, and exploring new opportunities. However, integrating tourism planning into official planning - whether economic, social, welfare, environmental, infrastructure, or cultural - has been slow, and remains unusual. The ideal model would be a national/regional/local comprehensive planning system in which tourism is an integral component. This model is rare, which is not surprising, as the various component strategies within tourism are seldom integrated. The goal of Integrated Tourism is twofold. For the various interests, requirements and needs the aim is to be fused together into a composite, integrated strategic tourism plan. For tourism the aim is to be planned with the intention of being fused into the social and economic life of a region and its communities. Although there is evidence that some tourism destinations have developed without conscious strategic and integrated planning, many of them have experienced unforeseen consequences (either physical, or human, or marketing or organizational impacts) which have led to their deterioration. Integrated...
Tourism has turned out to be crucial in the sustainable development of rural areas (so-called Integrated Rural Tourism) [14]. However, the integrated approach can be beneficial also to urban areas as testified by recent progress in Urban Tourism research [1]. Indeed, tourism is being seen as a cornerstone of a policy of urban development that combines a competitive supply able to meet visitors’ expectations with a positive contribution to the development of towns and cities and the well-being of their residents. Urban Tourism is complex, difficult to pin down and define, and depends on many factors such as the size of the town, its history and heritage, its morphology and its environment, its location, its image, etc.

**Contribution of the paper.** The application of Information and Communication Technology (ICT) to the tourism industry has been considered challenging since the very beginning due to the technical issues raised by interoperability. However, most research on so-called eTourism has been conducted by specializing technologies originally conceived for eCommerce (see [3] for a comprehensive yet not very recent review). We claim that ICTs for Integrated Tourism should go beyond the mere technological support for tourism marketing. For instance, identifying the most appropriate institutional structures and strategies to integrate the views and coordinate the actions of diverse tourism stakeholders is a key stage in the development of Integrated Tourism in rural and lagging areas. Bousset et al. [2] present a Decision Support System (DSS) which combines tools to assist in the analysis of the views, concerns and planned strategies of a wide range of tourism stakeholders in the face of given trends in tourists’ expectations. In this paper, we report our experience in supporting Integrated Tourism services with semantic technologies. The work has been conducted within *Puglia@Service*¹, an Italian PON Research & Competitivity project aimed at creating an innovative service infrastructure for the Apulia Region, Italy². As for the application to Integrated Tourism, the project addresses some of the issues analyzed in a report entitled “Sustainable Tourism and Local Development in Apulia Region” (2010)³ and prepared by the Local Economic and Employment Development (LEED) Programme and the Tourism Committee of the Organisation for Economic Co-operation and Development (OECD) in collaboration with Apulia Region. The region as a touristic destination needs a better management in spite of the recent growth of visitors and the high potential. In particular, the report emphasizes the lack of an adequate ICT infrastructure and little use of new technologies.

**Semantic Web Services** are among the semantic technologies which are going to be applied in *Puglia@Service*. Just like conventional web services, Semantic Web Services are the server end of a client-server system for machine-to-machine interaction via the World Wide Web (or simply, the Web) [7]. As a component of the Semantic Web, they are defined with mark-up languages which make data machine-readable in a detailed and sophisticated way. In particular, *OWL-S*⁴ is an ontology which provides a standard vocabulary that can be used together

---

⁴ [http://www.w3.org/Submission/OWL-S/](http://www.w3.org/Submission/OWL-S/)
with the other aspects of the Ontology Web Language (OWL)\textsuperscript{5} to create service descriptions. The use of OWL-S makes it easy for programmers to combine data from different sources and services without losing meaning. Web services can be activated "behind the scenes" when a web browser makes a request to a web server, which then uses various web services to construct a more sophisticated reply than it would have been able to do on its own. Semantic Web Services can also be used by automatic programs that run without any connection to a web browser. Overall, the interchange of semantic data allows to overcome some of the limits of conventional web services. Indeed, the mainstream XML standards for interoperation of web services specify only syntactic interoperability, not the semantic meaning of messages. For example, Web Services Description Language (WSDL)\textsuperscript{6} can specify the operations available through a web service and the structure of data sent and received but cannot specify semantic meaning of the data or semantic constraints on the data. This requires programmers to reach specific agreements on the interaction of web services and makes automatic web service composition difficult.

Structure of the paper. The paper is structured as follows. Section 2 summarizes the goals of the Puglia@Service project as for the application to Integrated Tourism. Section 3 shortly describes a domain ontology for Integrated Tourism, named OnTourism, which has been modeled for being used within Puglia@Service. Section 4 briefly presents a Web Information Extraction tool, named WIE-OnTour, which has been developed for populating OnTourism with data automatically retrieved from the Web. Section 5 illustrates some of the Semantic Web Services which have been defined on top of OnTourism for supporting Integrated Tourism in Apulia. Section 6 provides an overview of related work. Section 7 concludes the paper with final remarks and directions of future work. Appendix A provides further details of the OWL-S approach.

2 Integrated Tourism Services in Apulia

The research conducted in the Puglia@Service project falls within the area of Internet-based Service Engineering, i.e. it investigates methodologies for the design, development and deployment of innovative services. Concerning this area, the project will have an impact on the Apulia regional system at a strategic, organizational and technological level, with actions oriented to service innovation for the “sustainable knowledge society”. The reference market of Puglia@Service is represented by the so-called Knowledge Intensive Services (KIS), an emerging category of the advanced tertiary sector, and transversal to the other economic sectors, that is supposed to play a prominent role within the restructuring process which will follow the world economic crisis.

Objective of the project is to promote a new service culture over the Apulia region, marking a discontinuity point in the local development model, and guiding the transition of the region towards the “smart territory” paradigm where the

\textsuperscript{5} http://www.w3.org/TR/owl2-overview/

\textsuperscript{6} http://www.w3.org/TR/2007/REC-wsdl120-primer-20070626/
territory is intended to be a multiplayer system able to improve, by means of an adequate technological and digital infrastructure, its attitude to innovation as well as its skills in managing the knowledge assets of the regional stakeholders.

The project will pursue this goal through process and product innovations. In particular, it proposes to radically innovate the processes of service conceptualization, design, development and deployment, by assigning to the user a central role that anticipates his involvement. This will be obtained by applying a user-driven open innovation methodology, created at the US MIT Laboratories and adopted by the European countries, known as “Living-Lab”. The project also defines a set of methodologies and technologies for Internet-based Service Engineering starting from a next generation service model conceived to satisfy the needs for inclusion, participation and personalization. Finally, it is expected to produce not only a pervasive technological infrastructure (like a nervous system for the “smart territory”) but also:

- a qualified personnel, educated according to the Innovator and Entrepreneur Engineer profile, i.e. able to catch the opportunities offered by the new technologies and to transfer them into new business models in order to create economic and social value (Technological Entrepreneurship);
- a start-up company operating in the Service Engineering area and addressing the European Commission request for “service innovation leaders”.

The arrangement of the new service model into the regional context will regard the new generation services of the Public Administration and the Integrated Tourism. In particular, the application of Puglia@Service to Integrated Tourism (Puglia@Service.Tourism) encompasses an intervention on the Apulia tourism system, based on the definition of an Internet-based service model which increases the capability of KIS to create value for the region and for the tourist. Here, the tourist is not only “service user” but also “information supplier”. In particular, the application will require the development of methods and technologies enabling an interaction model between the tourist and the territory with the ultimate goal of local development along three directions: Culture, Environment and Economy. For the purposes of this paper, we shall focus only on the Environment dimension.

Puglia@Service.Tourism aims at promoting forms of tourism with a low environmental impact centered around the notion of eco-compatible mobility. This will contribute to the achievement of a twofold goal. On one side, the tourist will benefit from decision support facilities during his/her tours, e.g. he/she will receive suggestions about sites of interest and public transportation means suitable to reach a certain destination. On the other side, the territory will benefit from the environmental sustainability of local tourism. The reduced environmental impact of eco-mobility together with the need for a more efficient transportation system in touristic places can be obtained by combining sensoring tools and applications with rewarding mechanisms that encourage tourists and citizens to make eco-compatible choices. A possible scenario is described in the following. Once arrived in a touristic destination, the tourist could use his/her smartphone/PDA in order to obtain a suggestion about specific itineraries compliant with his/her profile and the information about the context. The tourist will be informed about the availability of alternative transportation means and
will be offered some credits for the green options (biking, trekking, car pooling, car sharing, etc.). In order to support this scenario, the Puglia@Service.Tourism infrastructure should deal with multi-dimensional information useful to suggest a touristic strategy which should meet users’ expectations and preferences (in culture, enogastronomy, shopping, relax, etc.); environmental conditions, both meteorological and natural; multi-modal transportation means; availability of car pooling and car sharing services; transfer time between sites of interest. The “fingerprint” of tourists visiting an area in a given time span can be anonymized and employed to improve continuously the user profiling with the choices made by tourists with the same profile. To this aim it is necessary to track the trajectories of citizens and tourists by means of localization and wireless communication technologies (traces from mobile phones, PDA, vehicles with GPS, etc.).

It is straightforward to notice that Internet-based Service Engineering for KIS in Integrated Tourism should strongly rely on Web technologies - such as Semantic Web Services - enabling an automated service composition. As shown in the rest of the paper, Web services in Puglia@Service.Tourism are enriched with semantic annotations starting from domain ontologies.

3 A Domain Ontology for Integrated Tourism

Domain ontologies for tourism are already available, e.g. the travel\textsuperscript{7} ontology is centered around the concept of Destination. However, it is not fully satisfactory from the viewpoint of Integrated Tourism. For instance, it lacks concepts modeling the reachability of places. In Puglia@Service.Tourism, we have decided to build a domain ontology, named OnTourism,\textsuperscript{8} more suitable for the project objectives and compliant with the OWL2 standard. It consists of 359 axioms, 196 logical axioms, 113 classes, 9 object properties, and 14 data properties, and has the expressivity of the DL $ALC\Box\mathcal{I}F(D)$.

The main concepts forming the terminology of OnTourism model the sites (class Site), the places (class Place), and the distances between sites (class Distance). Sites of interest include accommodations (class Accommodation), attractions (class Attraction), stations (class Station), and civic facilities (class Civic) as shown in Figure 1. The terminology encompasses the amenities (class Amenity with subclasses reported in Figure 2) and the services (class Service with subclasses reported in Figure 3) offered by hotels. Also, it models the official 5-star classification system for hotel ranking (class Rank with instances 1\_star, 2\_stars, and so on) as well as a user classification system for accommodation rating (class Rate with instances Excellent, Very\_Good, Average, Poor, Terrible). Finally, the terminology includes landscape varieties (class Landscape with instances City, Country, Lake, Mountain, River, and Sea) and transportation means (class Transportation\_Mean with instances Bike, Car, Foot, and Public\_transit). Distances are further classified into Distance\_by\_car and Distance\_on\_foot according to the transportation means used.

\textsuperscript{7} http://www.protege.cim3.net/file/pub/ontologies/travel/travel.owl
\textsuperscript{8} It significantly extends the Hotel ontology described in [10].
The object properties in OnTourism model the relationship between a site and a distance (hasDistance), the relationship between a distance and the two sites (isDistanceFor), and the relationship between a site and the place where the site is located at (isLocatedAt). Also, for each accommodation, it is possible to specify the amenities available (hasAmenity) and the services provided (provides). The user rating allows to classify accommodations into five categories (from Excellent_Accommodation to Terrible_Accommodation). In the case of hotels, the ranking (hasRank) is the starting point for the definition of five categories (from Hotel_1_Star to Hotel_5_Stars).

The data properties in OnTourism allow to refer to sites by name and to places by address, zipcode, city, and country. Details about accommodations are the number of rooms (numberOfRooms) and the average price of a room (hasPrice). Distances between sites have a numerical value in either length or time units (hasLengthValue/hasTimeValue). Note that each of these numerical values would be better modeled as attribute of a ternary relation. However, only binary relations can be represented in OWL. The concept Distance and the properties hasDistance, isDistanceFor and hasLengthValue/hasTimeValue are necessary to simulate a ternary relation by means of binary relations.

4 Extraction of Touristic Information from the Web

Information extraction (IE) is the task of automatically extracting structured information from unstructured and/or semi-structured machine-readable documents. In most of the cases this activity concerns processing human language texts by means of natural language processing (NLP). The proliferation of the Web has intensified the need for developing IE systems that help people to cope with the enormous amount of data that is available online, thus giving rise to Web Information Extraction (WIE) [4]. WIE tools typically exploit the HTML/XML tags and layout format that are available in online text. As a result, less linguistically intensive approaches have been developed for IE on the Web using wrappers, which are sets of highly accurate rules that extract a particular page’s content. Wrappers typically handle highly structured collections of web pages, such as product catalogues and telephone directories. They fail, however, when the text type is less structured, which is also common on the Web.

WIE-OnTour is a wrapper-based WIE tool implemented in Java and conceived for the population of OnTourism with data concerning accommodations (in particular, those in the categories “hotel” and “bed&breakfast”) available in the web site of TripAdvisor\[9\]. The tool is also able to compute distances of the extracted accommodations from sites of interest (e.g., touristic attractions) by means of Google Maps\[10\] API. Finally, the tool supports the user in the specification of sites of interest.

WIE-OnTour has been tested on several cities in the world. However, the main destinations of Urban Tourism in the Apulia Region are of interest to

\[9\] http://www.tripadvisor.com/
\[10\] http://maps.google.com/
the project. Therefore, as case studies, we have restricted our attention to capital towns of Apulia provinces (Andria, Bari, Barletta, Brindisi, Lecce, Taranto, Trani). A snapshot of WIE-ONTOUR performing the information extraction process for Bari, the capital city of Apulia Region, is shown in Figure 4. In this session (performed on May 13, 2014), the tool has extracted 46 hotels (instances of Hotel), 151 bed&breakfast (instances of Bed and Breakfast), 205 places (instances of Place), 1996 distances (instances of Distance) for a total of 2406 individuals. The distances have been computed with respect to the following sites of interest: Basilica di San Nicola\(^1\) and Cattedrale di San Sabino\(^2\) (both instances of Church), Museo Nicolaiano (instance of Museum), Porto di Bari (instance of Port), Aeroporto Karol Wojtyla (instance of Airport), and FS Bari Centrale (instance of Train_Station). The computation for this session has been completed in about 33 minutes.

5 Adding Semantics to Integrated Tourism Services

In Puglia@Service.Tourism, we have defined several services on top of two domain ontologies: travel and OnTourism. For example, destination_attractions_service is a service that returns the attractions located in a given destination. The semantic description of this service in OWL-S (shown in Figure 5) specifies that it is an atomic service with only one input and only one output where the parameter types for the input and the output are the classes Destination (belonging to travel) and Attraction (occurring in OnTourism) respectively. Several specializations of destination_attractions_service have been considered, one for each subclass of the parameter types. For example, city_churches_service is a service that returns the churches (output parameter of type Church) located in a given city (input parameter of type City). When executed for the city of, e.g., Bari, the service will query the underlying domain ontologies (more precisely, their instance level) to retrieve each Church that isLocatedAt some Place in Bari, e.g. Basilica di San Nicola and Cattedrale di San Sabino. Note that these instances will be returned also by destination_attractions_service because they are inferred to be instances of Attraction. As a further case, near_attraction_accomodations_service is a service that returns all the accommodations (output parameter of type Accommodation) near a given attraction (input parameter of type Attraction). Note that closeness can be defined on the basis of the distance between sites (class Distance) either in a crisp way (i.e., when the distance value is under a fixed threshold) or in a fuzzy way (i.e., through grades of closeness). In both ways, however, the computation should consider the transportation means used (Distance_by_car vs. Distance_on_foot) as well as the measure units adopted (hasLengthValue vs. hasTimeValue).

In Puglia@Service.Tourism, we have chosen to define only OWL-S atomic services in order to exploit the aforementioned advantages of the WSDL grounding. As an illustration, the WSDL grounding of destination_attractions_service is reported in Figure 6. Composite services can be automatically obtained by applying

---


service composition methods such as the one described in [15]. The simplest form of composite service is based on the control construct of Sequence. For example, the services city_churches_service and near_attraction_accomodations_service can be executed in sequence by having the output of the former as input to the latter. Note that the type mismatch is only apparent since Church is a subclass of Attraction. One such service composition satisfies, e.g., the user request of knowing the accommodations around Basilica di San Nicola and Cattedrale di San Sabino in Bari. Considering that Bari is a major destination of religious tourism in Apulia, this composite service effectively supports the demand from pilgrims who prefer to find an accommodation in the neighborhood of places of worship so that they can practise their own religions at any hour of the day. Also, if the suggested accommodations are easy to reach (i.e., at foot distance) from the site of interest, the service will bring benefit also to the city, by reducing the car traffic. In a more complex scenario, the pilgrim might need an accommodation accessible to disabled visitors. The service composition mechanism should then consider a further specialized service, say disabledfacilities_hotels_service, which returns the hotels (output parameter of type Hotel) with disabled facilities (input parameter of type Disabled_Facilities). Indeed, the resulting composite service can be considered compatible with the special needs of this user profile.

6 Related Work

The application of ICT to the tourism industry has been considered challenging since the very beginning due to the technical issues raised by interoperability. Werthner and Klein [18] defined interoperability as the provision of a well-defined and end-to-end service which is in a consistent and predictable way. This generally covers not merely technical features but also in the case of electronic market environments, contractual features and a set of institutional rules. Interoperability enables partners to interact electronically with each other by the most convenient method and to deliver the right information at the right time to the right user at the right cost. Using a domain ontology a mediator software system (such as HARMONISE [13,5]) effectively “translates” partners’ data and allows them to communicate electronically. Maedche and Staab [11,12] showed that semantic web technologies can be used for tourism applications to provide useful information on text and graphics, as well as generating a semantic description that is interpretable by machines. Dogac et al. [6] describe how to deploy semantically enriched travel Web services and how to exploit semantics through Web service registries. We also address the need to use the semantics in discovering both Web services and Web service registries through peer-to-peer technology. Hepp et al. [8] investigate the use of ontological annotations in tourism applications. They show, based on a quantitative analysis of Web content about Austrian accommodations, that even a perfect annotation of existing Web content would not allow the vision of the Semantic Web to become a short-term reality for tourism-related eCommerce. Also, they discuss the implications of these findings for various types of eCommerce applications that rely on the extraction of information from existing Web resource, and stress the importance of Semantic Web Services technology.
for the Semantic Web. Within the scope of the OnTour\textsuperscript{13} project, Siorpaes and Bachlechner [17] develop a system based on a fast and flexible Semantic Web backbone with a focus on e-tourism. The major benefits of the OnTour approach are its simplicity, modularity, and extensibility. In [9], Jakkilinki et al. describe the underlying structure and operation of a Semantic Web based intelligent tour planning tool. The proposed tour planner has inbuilt intelligence which allows it to generate travel plans by matching the traveller requirements and vendor offerings stored in conjunction with the travel ontology. Ricca et al. [16] present a successful application of logic programming for e-tourism: the iTravel system. The system exploits two technologies that are based on the state-of-the-art computational logic system DLV: (i) a system for ontology representation and reasoning, called OntoDLV; and, (ii) a semantic information-extraction tool. The core of iTravel is an ontology which models the domain of tourism offers. The ontology is automatically populated by extracting the information contained in the tourism leaflets produced by tour operators. A set of specifically devised logic programs is used to reason on the information contained in the ontology for selecting the holiday packages that best fit the customer needs. An intuitive web-based user interface eases the task of interacting with the system for both the customers and the operators of a travel agency.

7 Conclusions and Future Work

In this paper we have reported our ongoing work on the use of semantic technologies for supporting Integrated Tourism services in the Apulia region within the Puglia@Service project. More precisely, we have shortly described OnTourism, a domain ontology for Integrated Tourism. Also, we have briefly presented WIE-ONTOUR, a Web Information Extraction tool which has been developed for populating OnTourism with data automatically retrieved from the Web sites of TripAdvisor and Google Maps. Moreover, we have illustrated the semantic descriptions in OWL-S of some Integrated Tourism services built on top of OnTourism.

Though developed for the purposes of the project, the technical solutions here described are nevertheless general enough to be reusable for similar applications in other geographical contexts. Notably, they show the added value of having ontologies and ontology reasoning behind an Interned-based service infrastructure.

For the future we intend to apply Machine Learning tools such as FOIL-DL [10] to enhance the automated composition of OWL-S services. Notably, we shall consider the problem of learning from the feedback provided by specific user profiles. The idea is to use the ontology axioms induced by FOIL-DL in order to discard those compositions that do not reflect the preferences/expectations/needs of a certain user profile. Therefore, the axioms will act as composition rules to be integrated with other existing approaches to automated service composition.

Acknowledgements This work was partially funded by the Italian PON R&C 2007-2013 project PON02_00563_3489339 “Puglia@Service: Internet-based Service Engineering enabling Smart Territory structural development”.

\textsuperscript{13} The OnTour project should not be confused with our ontology OnTourism. The names are only accidentally very similar.
A The OWL-S approach

Besides being a service mark-up language, OWL-S is an upper ontology for services whose structuring is motivated by the need to provide three essential types of knowledge about a service (class Service): The service profile (class ServiceProfile), the service model (class ServiceModel), and the service grounding (class ServiceGrounding). Generally speaking, the service profile provides the information needed for an agent to discover a service, while the service model and the service grounding, taken together, provide enough information for an agent to make use of a service, once found. More specifically, the three components satisfy the following informative needs.

The service profile tells "what the service does", in a way that is suitable for a service-seeking agent (or matchmaking agent acting on behalf of a service-seeking agent) to determine whether the service meets its needs. This form of representation includes a description of what is accomplished by the service, limitations on service applicability and quality of service, and requirements that the service requester must satisfy to use the service successfully.

The service model tells a client how to use the service, by detailing the semantic content of requests, the conditions under which particular outcomes will occur, and, where necessary, the step by step processes leading to those outcomes. For services based on composite processes, this description may be used by a service-seeking agent in at least four different ways: (1) to perform a more in-depth analysis of whether the service meets its needs; (2) to compose service descriptions from multiple services to perform a specific task; (3) during the course of the service enactment, to coordinate the activities of the different participants; and (4) to monitor the execution of the service.

A service grounding specifies the details of how an agent can access a service. Typically a grounding will specify a communication protocol, message formats, and other service-specific details such as port numbers used in contacting the service. In addition, the grounding must specify, for each semantic type of input or output specified in the service model, the serialization techniques employed for exchanging data elements of that type with the service. The most commonly used grounding is WSDL due to the following reasons: (1) An OWL-S atomic process corresponds to a WSDL operation; (2) The inputs and outputs of an OWL-S atomic process correspond to WSDL messages; (3) The types of the inputs and outputs of an OWL-S atomic process correspond to WSDL abstract types.

References

Fig. 1. Taxonomy of sites in the OnTourism ontology.
Fig. 2. Taxonomy of amenities in the OnTourism ontology.
Fig. 3. Taxonomy of services in the OnTourism ontology.

Fig. 4. Web Information Extraction for the city of Bari, Italy, with WIE-OnTour.
Fig. 5. Semantic description of destination_attractions_service with OWL-S.
Fig. 6. Semantic description of destination_attractions_service with OWL-S (cont.).
A Dual Tableau-based Decision Procedure for a Relational Logic with the Universal Relation

Domenico Cantone\textsuperscript{1}, Marianna Nicolosi-Asmundo\textsuperscript{1}, and Ewa Orłowska\textsuperscript{2}

\textsuperscript{1} Università di Catania, Dipartimento di Matematica e Informatica  
email: cantone@dma.unict.it, nicolosi@dma.unict.it
\textsuperscript{2} National Institute of Telecommunications, Warsaw, Poland  
email: orlowska@itl.waw.pl

Abstract. We present a first result towards the use of entailment inside relational dual tableau-based decision procedures. To this end, we introduce a fragment of RL\textsubscript{(1)}, called (\{1, \cup, \cap\}; \_\_), which admits a restricted form of composition. We prove the decidability of the (\{1, \cup, \cap\}; \_\_) fragment by defining a dual tableau-based decision procedure with a suitable blocking mechanism and where the decomposition rules for compositional formulae are modified so as to deal with the constant 1 while preserving termination.

The (\{1, \cup, \cap\}; \_\_) fragment properly includes the logics presented in previous work and, therefore, it allows one to express, among others, the multi-modal logic K with union and intersection of accessibility relations and the description logic ALC with union and intersection of roles.

1 Introduction

The relational representation of various non-classical propositional logics has been systematically analyzed in the last decades [16]. A uniform relational framework based on the logic of binary relations RL\textsubscript{(1)}, presented in [15] and called relational dual tableau, showed to be an effective logical means to represent in a modular way three fundamental components of a formal system: its syntax, semantics, and deduction system. Relational systems have been defined for modal and intuitionistic logics, for relevant and many-valued logics, for reasoning in logics of information and data analysis, for reasoning about time and space, etc.

The formalization of non-classical logics in RL\textsubscript{(1)} is based on the fact that once the Kripke-style semantics of the logic under consideration is known, formulae can be treated as relations. In particular, since in Kripke-style semantics formulae are interpreted as collections of objects, in their relational representation they are seen as right ideal relations. In the case of binary relations this means that (R; 1) = R is satisfied, where ‘;’ is the composition operation on binary relations and ‘1’ is the universal relation.

One of the most useful features of the relational methodology is that, given a logic with a relational formalization, we can construct its relational dual tableau in a systematic and modular way.
Though the relational logic $RL(1)$ is undecidable, it contains several decidable fragments. In many cases, however, dual tableau proof systems are not decision procedures for decidable fragments of $RL(1)$. This is mainly due to the way decomposition and specific rules are defined and also to the strategy of proof construction.

Over the years, great efforts have been spent to construct dual tableau proof systems for various logics known to be decidable; little care has been taken, however, to design dual tableau-based decision procedures for them. On the other hand, it is well known that when a proof system is designed and implemented, it is important to have decision procedures for decidable logics. In [10], for example, an optimized relational dual tableau for $RL(1)$, based on Binary Decision Graphs, has been implemented. However, such an implementation turns out not to be effective for decidable fragments.

As far as we know, relational dual tableau-based decision procedures can be found in [16] for fragments of $RL(1)$ corresponding to the class of first-order formulae in prenex normal form with universal quantifiers only; in [12, 13] for the relational logic corresponding to the modal logic $K$; in [4, 5] for fragments of $RL$ characterized by some restrictions in terms of type $(R; S)$; in [11] for a class of relational logics admitting a single relational constant with the properties of reflexivity, transitivity, and heredity; and in [3] for a class of relational fragments extending the ones introduced in [11] by allowing a countable infinity of relational constants with the properties of reflexivity, transitivity, and heredity.

Throughout the paper terms of type $(R; S)$ will be referred to as compositional terms. Similarly, formulae with compositional terms will be referred to as compositional formulae.

In some cases, like in [11] and in [3], fragments with relational constants satisfying some fixed properties are considered. Therefore, dual tableau-based decision procedures are endowed with specific rules to treat relational constants and their properties. The design of specific rules often needs much care in order to guarantee termination of the related proof procedure. This task is delicate especially when the proof system provides several specific rules for different relational constants, and when the relational constants are related to one another.

An alternative way to treat properties of relational constants and variables, and of relations between them, is to use relational entailment. Relational entailment can be formalized in the logic $RL(1)$ as follows. Given relations $R, R_1, \ldots, R_n$, with $n \geq 1$, one has that $R_1 = 1, \ldots, R_n = 1$ imply $R = 1$ in a model if and only if $(1; (-(R_1 \cap \ldots \cap R_n)); 1) \cup R = 1$ holds. It means that entailment is expressible as a term of the language of logic $RL(1)$ and, as a consequence, any validity checker for $RL(1)$ can also be applied to entailment verification.

Introduction of entailment inside relational proof systems allows one to eliminate specific rules and, consequently, to keep the set of decomposition rules small. This approach can be convenient in implementations of automated theorem provers provided that decomposition rules and their application strategy are designed in a suitable way. In its relational formalization, however, entailment involves the universal constant $1$ on the left hand side and on the right hand side.
of compositional terms. Thus, the design of a relational dual tableau-based decision procedure where entailment is admitted is a challenging task that requires special care.

In this paper we present a first result towards the use of entailment inside relational dual tableau-based decision procedures. To this purpose, we introduce a fragment of RL(1), called \( \{1, \cup, \cap \}; \_ \), admitting a restricted form of composition where the left subterm \( R \) of any term of type \( (R; S) \) is allowed to be either the constant \( 1 \) or any term constructed from the relational variables by applying only the operators of relational intersection and union. Similarly, terms of type \( (R; 1) \) are admitted only if \( R \) is a Boolean term involving relational variables and the operators of intersection and union.

We prove that the \( \{1, \cup, \cap \}; \_ \)-fragment is decidable by defining a dual tableau-based decision procedure where a suitable blocking mechanism has been introduced and rules for compositional and complemented compositional formulae have been appropriately modified to deal with the constant \( 1 \) while preserving termination.

Such fragment properly includes the logics presented in [4] and, therefore, it can express the multi-modal logic \( K \) with union and intersection of accessibility relations and the description logic \( ALC \) with union and intersection of roles. Furthermore it can also express, via entailment, properties of the form \( 'r \subseteq -(s_1 \cup s_2)' \) and \( '(s_1 \cup s_2) \subseteq -r' \), where \( r, s_1, \) and \( s_2 \) are relational variables.

The rest of the paper is organized as follows. In Sect. 2 we briefly review the syntax and semantics of the relational logic RL(1) together with its dual tableau and in Sect. 3 we introduce some useful notions which will be used throughout the paper. Then in Sect. 4 we present the \( \{1, \cup, \cap \}; \_ \)-fragment and its dual tableau-based decision procedure. Finally, in Sect. 5, we draw our conclusions and give some hints for future work.

2 The Relational Logic RL(1) and its Dual Tableau

In this section we review the logic RL(1) and its dual tableau in full extent (see also [5] and [16]).

Let \( RV \) be a countably infinite set of relational variables \( p, q, r, s, \ldots \) and let \( 1 \) be a relational constant. Then, the set \( RT \) of relational terms of RL(1) is the smallest set of terms (with respect to inclusion) built from relational variables and the relational constant \( 1 \) with the relational operators \( \cap, \cup, \_ \) (binary) and \( '1', '\_1' \) (unary).

Let \( OV \) be a countably infinite set of object (individual) variables \( x, y, z, w, \ldots \). Then, RL(1)-formulae have the form \( xRy \), where \( x, y \in OV \) and \( R \in RT \). RL(1)-formulae of type \( x1y \) and \( xry \), with \( r \in RV \), are called atomic RL(1)-formulae. A literal is either an atomic formula or its complementation (namely a formula of type \( x(-1)y \) or \( x(-r)y \)). For a relational operator \( '2' \) other than \( '1' \), by a \( (2)-term \) we mean a relational term whose lead operator is \( '2' \), and by a \( (-2)-term \) we denote a complemented \( (2)-term \). For example, the term \( (r_1 \cup s) \cap (-r_2 ; s) \) is a \( (\cap)-term \) and has \( '\cap' \) as its lead operator, whereas \( -(r_1 \cup s) \cap (-r_2 ; s) \) is
a \((-\cap)\)-term. A \((\sharp)\)-formula (resp., \((-\sharp)\)-formula) is a formula whose relational term is a \((\sharp)\)-term (resp., \((-\sharp)\)-term). A Boolean term is a relational term built from relational variables with the Boolean operators \(\lnot\), \(\cup\), and \(\cap\). A positive Boolean term is a Boolean term in which the operator \(\lnot\) does not occur.

\(\text{RL}(1)\)-formulae are interpreted in \(\text{RL}(1)\)-models. An \(\text{RL}(1)\)-model is a structure \(\mathcal{M} = (U, m)\), where \(U\) is a nonempty universe and \(m : \mathcal{R} \to \mathcal{P}(U \times U)\) is a given map which is homomorphically extended to the whole collection \(\mathbb{RT}\) of relational terms as follows:

- \(m(1) = U \times U\);
- \(m(-R) = (U \times U) \setminus m(R)\);
- \(m(R \cup S) = m(R) \cup m(S)\);
- \(m(R \cap S) = m(R) \cap m(S)\);
- \(m(R ; S) = m(R) \cup m(S)\);
- \(m(R^\sim) = \{(a, b) \in U \times U : (a, c) \in m(R) \text{ and } (c, b) \in m(S), \text{ for some } c \in U\}\);
- \(m(S^\sim) = \{(b, a) \in U \times U : (a, b) \in m(R)\}\).

Let \(\mathcal{M} = (U, m)\) be an \(\text{RL}(1)\)-model. A valuation in \(\mathcal{M}\) is any function \(v : \bigvee \to U\). An \(\text{RL}(1)\)-formula \(xRy\) is satisfied by an \(\text{RL}(1)\)-model \(\mathcal{M} = (U, m)\) and by a valuation \(v\) in \(\mathcal{M}\) (in which case we write \(\mathcal{M}, v \models xRy\)) provided that \((v(x), v(y)) \in m(R)\). An \(\text{RL}(1)\)-formula \(xRy\) is (a) true in a model \(\mathcal{M} = (U, m)\), if \(\mathcal{M}, v \models xRy\), for every valuation \(v\) in \(\mathcal{M}\); (b) valid, if it is true in all \(\text{RL}(1)\)-models; (c) falsified by a model \(\mathcal{M} = (U, m)\) and by a valuation \(v\) in \(\mathcal{M}\), if \(\mathcal{M}, v \nmodels xRy\); (d) falsifiable, if there exist a model \(\mathcal{M}\) and a valuation \(v\) in \(\mathcal{M}\) such that \(\mathcal{M}, v \nmodels xRy\). An \(\text{RL}(1)\)-set is a finite set \(\{\varphi_1, \ldots, \varphi_n\}\) of \(\text{RL}(1)\)-formulae such that, for every \(\text{RL}(1)\)-model \(\mathcal{M}\) and for every valuation \(v\) in \(\mathcal{M}\), we have \(\mathcal{M}, v \models \varphi_i\), for some \(i \in \{1, \ldots, n\}\). Clearly, the first-order disjunction of the formulae in an \(\text{RL}(1)\)-set is valid in first-order logic.

Proof development in dual tableaux proceeds by systematically decomposing the (disjunction of the) formulae to be proved till a validity condition is detected, expressed in terms of axiomatic sets (see below). The method originated in [17] (see also [18]). Such an analytic approach is similar to Beth's tableau method [1], with the difference that the two systems work in a dual manner. Duality between tableaux and dual tableaux has been analyzed in depth in [14].

\(\text{RL}(1)\)-dual tableaux consist of decomposition rules, which allow one to analyze the structure of the formula to be proved valid, and of axiomatic sets, which specify closure conditions. The decomposition rules for \(\text{RL}(1)\) are listed in Table 1. In these rules, \(\lnot\), \(\lor\), and \(\land\) are interpreted respectively as disjunction and conjunction. A rule is \(\text{RL}(1)\)-correct provided that its premise is an \(\text{RL}(1)\)-set if and only if each of its consequents is an \(\text{RL}(1)\)-set. The rules presented in Table 1 have been proved \(\text{RL}(1)\)-correct in [16].

An \(\text{RL}(1)\)-axiomatic set is any set of \(\text{RL}(1)\)-formulae containing a subset of one of the following two forms: \(\textbf{Ax 1} \{xRy, x(-R)y\}\), \(\textbf{Ax 2} \{x1y\}\).

Clearly, an \(\text{RL}(1)\)-axiomatic set is also an \(\text{RL}(1)\)-set.

An \(\text{RL}(1)\)-proof tree for an \(\text{RL}(1)\)-formula \(xPy\) is an ordered tree whose nodes are labelled by disjunctive sets of formulae such that the following properties are satisfied:

- the root is labelled with \(\{xPy\}\);
### Table 1. RL(1) decomposition rules.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Rule</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>(∪)</td>
<td>[ x(R \cup S)y ]</td>
<td>[ x(-(R \cup S))y ]</td>
</tr>
<tr>
<td></td>
<td>[ xRy, xSy ]</td>
<td>[ x(-R)y, x(-S)y ]</td>
</tr>
<tr>
<td>(∩)</td>
<td>[ x(R \cap S)y ]</td>
<td>[ x(-(R \cap S))y ]</td>
</tr>
<tr>
<td></td>
<td>[ xRy</td>
<td>xSy ]</td>
</tr>
<tr>
<td>(--)</td>
<td>[ x(-R)y ]</td>
<td>[ x(-R)y ]</td>
</tr>
<tr>
<td>(◦)</td>
<td>[ x(R^-)y ]</td>
<td>[ x(-(R^-))y ]</td>
</tr>
<tr>
<td></td>
<td>[ yRx ]</td>
<td>[ y(-R)x ]</td>
</tr>
<tr>
<td>(z)</td>
<td>[ x(R ; S)y ]</td>
<td>[ x(-(R ; S))y ]</td>
</tr>
<tr>
<td></td>
<td>[ xRz, x(R ; S)y ]</td>
<td>[ zSy, x(R ; S)y ]</td>
</tr>
</tbody>
</table>

(z is any object variable)

(z is a new object variable)

– each node, except the root, is obtained from its predecessor node by applying a decomposition rule in Table 1 to one of the formulae labelling it;
– a node does not have successors (i.e., it is a leaf node) whenever its set of formulae is an axiomatic set or none of the rules of Table 1 can be applied to its set of formulae.

A branch \( \theta \) of a proof tree is any of its maximal paths; we denote with \( \bigcup \theta \) the set of all the formulae contained in the nodes of \( \theta \), and with \( W_\theta \) the collection of the object variables occurring in the formulae contained in the nodes of \( \theta \). A node of an RL(1)-proof tree is closed if its associated set of formulae is an axiomatic set. A branch is closed if one of its nodes is closed. A proof tree is closed if all of its branches are closed. An RL(1)-formula is RL(1)-provable if there is a closed RL(1)-proof tree for it, referred to as an RL(1)-proof.

A node of an RL(1)-proof tree is falsified by a model \( M = (U, m) \) and a valuation \( v \) in \( M \) if every formula \( xRy \) in its set of formulae is falsified by \( M \) and \( v \). A node is falsifiable if there exist a model \( M \) and a valuation \( v \) in \( M \) which falsify it.

Correctness and completeness of the RL(1)-dual tableau are proved in [16]. However, the logic RL(1) is undecidable. This follows from the undecidability of the equational theory of representable relation algebras discussed in [19].

### 3 Useful Notions and Properties

We introduce some useful notions and properties which are needed for the presentation of the results of the paper.

Let \( P \) be any relational term in RL(1). The following identities hold:

\[
\begin{align*}
(1 \cup P) & \equiv (P \cup 1) \equiv 1 \\
(1 \cap P) & \equiv (P \cap 1) \equiv P \\
(-1) & \equiv 1 \\
((-1) \cup P) & \equiv (P \cup (-1)) \equiv P \\
((-1) \cap P) & \equiv (P \cap (-1)) \equiv (-1)
\end{align*}
\]
Let \( H \) be a relational term in \( RL(1) \) and let \( H' \) be obtained from \( H \) by systematically simplifying \( H \) by means of the above identities. If the simplification is carried out in an inside-out way, the computational complexity of the transformation of \( H \) into \( H' \) is linear in the length of \( H \). Moreover, the following lemma holds (proof of Lemma 1 can be found in [6]).

**Lemma 1.** Let \( H \) be a relational term and let \( H' \) be constructed as outlined above. Then every Boolean subterm \( P \) of \( H' \) either is equal to \( 1 \), or is equal to \( -1 \), or it does not contain \( 1 \).

It is easy to check that \( m(H) = m(H') \) holds for every \( RL(1) \)-model \( M = (U, m) \) and for every \( H \in RL(1) \). Therefore we can restrict ourselves to relational terms simplified as described above.

**Parsing trees.** It is possible to associate a parsing tree \( S_P \) to each relational term \( P \) of \( RL(1) \), as with formulae of standard first-order logic (see [9] and [8] for details on the construction of parsing trees in first-order logic). Let \( S_P \) be the parsing tree for \( P \), and let \( \nu \) be a node of \( S_P \). We say that a relational term \( Q \) occurs within \( P \) at position \( \nu \) if the subtree of \( S_P \) rooted at \( \nu \) is identical to \( S_Q \). In this case we refer to \( \nu \) as an occurrence of \( Q \) in \( P \) and to the path from the root of \( S_P \) to \( \nu \) as its occurrence path.

An occurrence of a relational term \( Q \) within a relational term \( P \) is **positive** if its occurrence path deprived of its last node contains an even number of nodes labelled with \( \{ - \} \). Otherwise, the occurrence is said to be **negative**.

**Normal forms and term components.** Next we introduce the notion of complement normal form for Boolean relational terms, the notions of \( \text{Bool}_N \)-formula, of \( \text{Bool}_N \)-construction from \( N \), where \( N \) is a set of formulae, and of set of components of a relational term.

The **complement normal form** of a term \( R \) is a term \( \text{nf}-(R) \) obtained by successive applications of the De Morgan laws and of the law of double negation to \( R \).

A term is said to be in complement normal form whenever each occurrence of the complement operator in it acts only on relational variables or constants.

Clearly, for every Boolean relational term \( R \), the formulae \( xRy \) and \( x\text{nf}-(R)y \) are logically equivalent, that is \( \mathcal{M}, v \models xRy \) if and only if \( \mathcal{M}, v \models x\text{nf}-(R)y \), for every model \( \mathcal{M} = (U, m) \) and every valuation \( v \) in \( \mathcal{M} \).

Let \( N \) be a set of formulae, and let \( R, S \) be two Boolean relational terms. We define the notion of \( \text{Bool}_N \)-formulae as follows:

- every literal \( xRy \) in \( N \) is a \( \text{Bool}_N \)-formula;
- every formula of the form \( x(R \cap S)y \) is a \( \text{Bool}_N \)-formula, provided that either \( xRy \) is a \( \text{Bool}_N \)-formula and \( S \) is in complement normal form, or \( xSy \) is a \( \text{Bool}_N \)-formula and \( R \) is in complement normal form;
- every formula of the form \( x(R \cup S)y \) is a \( \text{Bool}_N \)-formula if both \( xRy \) and \( xSy \) are \( \text{Bool}_N \)-formulae.
Clearly, if \( xSy \) is a \( \text{Bool}_N \)-formula, then \( xSy \) is syntactically equal to \( x \text{nf}-(S) y \) and we write \( xSy = x \text{nf}-(S) y \). We say that a formula \( xRy \) has a Boolean-construction from \( N \) if \( x \text{nf}-(R) y \) is a \( \text{Bool}_N \)-formula.

For example, given the set of formulae \( N = \{ x(-r)z, xsz, x(-p)y, z(p \cup s)y \} \), we have that the formula \( x(((-r)\cup s)/q)z \) is a \( \text{Bool}_N \)-formula because \( x((-r)\cup s)z \) is a \( \text{Bool}_N \)-formula and \( xqz \) is in complement normal form. On the other hand the formula \( x(s \cap ((-q) \cup p))z \) is not a \( \text{Bool}_N \)-formula because although \( xsz \) is a \( \text{Bool}_N \)-formula, \( x((-q) \cap p)z \) is not in complement normal form. Both formulae, however, have a Boolean-construction from \( N \) because \( x(((-(r)\cup s)\cap q)z) \) is a \( \text{Bool}_N \)-formula and \( x \text{nf}-(s \cap ((-q) \cup p))z = x(s \cap ((-q) \cap p))z \) is a \( \text{Bool}_N \)-formula.

In this latter case, specifically, \( xsz \) is a \( \text{Bool}_N \)-formula and \( x((-q) \cap (-p))z \) is in complement normal form, although it is not a \( \text{Bool}_N \)-formula.

Given a term \( R \in \mathbb{RT} \), an object variable \( x \), and a set \( N \) of formulae, we define \( V(R, x, N) \) as the set of object variables \( z \) such that \( xRz \) has a Boolean-construction from \( N \).

Let \( P \) be a term in \( \mathbb{RT} \). We define recursively the set \( \text{cp}(P) \) of the components of the term \( P \) as follows:

- if \( P \) is the relational constant \( 1 \), or a relational variable, or their complements, then \( \text{cp}(P) = \{ P \} \);
- if \( P = -B \), then \( \text{cp}(P) = \{ P \} \cup \text{cp}(B) \);
- if \( P = B^\circ \), then \( \text{cp}(P) = \{ P \} \cup \text{cp}(B) \);
- if \( P = B \sharp C \) (resp., \( P = -(B \sharp C) \)), then \( \text{cp}(P) = \{ P \} \cup \text{cp}(B) \cup \text{cp}(C) \) (resp., \( \text{cp}(P) = \{ P \} \cup \text{cp}(-B) \cup \text{cp}(-C) \)), for every binary relational operator \( \sharp \).

Clearly \( \text{cp}(P) \) is finite, for any relational term \( P \).

4 The Fragment (\( \{ 1, \cup, \cap \}; \_ \)) and its Decision Procedure

4.1 The Fragment (\( \{ 1, \cup, \cap \}; \_ \))

Formulae of the fragment (\( \{ 1, \cup, \cap \}; \_ \)) of RL(1) are characterized by the fact that the left subterm \( R \) of any term of type \( (R; S) \) in them is only allowed to be either the constant \( 1 \) or a term constructed from the relational variables of \( \mathbb{RV} \) by applying only the ‘\( \cup \)’ and ‘\( \cap \)’ operators, whereas the right subterm \( S \) of \( (R; S) \) can involve all the relational operators of RL(1) but the converse operator ‘\( \_ \)’.

Formally, the set \( \mathbb{RT}_{(1, \cup, \cap); \_} \) of the terms allowed in (\( \{ 1, \cup, \cap \}; \_ \))-formulae is the smallest set of terms containing the constant \( 1 \) and the variables in \( \mathbb{RV} \), and such that if \( P, Q, B, H \in \mathbb{RT}_{(1, \cup, \cap); \_} \) and \( S \in \{ H, 1 \} \), with

- \( B \) a Boolean term containing neither \( 1 \) nor the complement operator, and
- \( H \) containing the constant \( 1 \) only inside terms of type \( (B; 1) \),

then \( (-P), (P \cup Q), (P \cap Q), (B; S), (1; S) \in \mathbb{RT}_{(1, \cup, \cap); \_} \).

Examples of formulae of the (\( \{ 1, \cup, \cap \}; \_ \))-fragment are: \( x((-((r_1 \cup s) \cup (p; 1))y, x(1; (((q \cup p) \cap r_1) \cup 1))y, x(1; (((r_1 \cup s) \cap r_2) \cup 1))y. The latter formula can be rewritten as \( x(1; ((-((r_1 \cup s) \cup r_2) \cup 1))y, where \( (-((r_1 \cup s) \cup r_2) \cup 1) \) is a relational term formalizing the property ‘\( (r_1 \cup s) \subseteq -r_2 \)’.
Table 2. Decomposition rules proper of the \((\{1, \cup, \cap\}; \_\_\_\_)-fragment.\)

\[
\begin{align*}
(\_\_\_\_a) & & \frac{x(B; S)y}{zSy, x(B; S)y}, \quad (\_\_\_\_b) & & \frac{x(1; S)y}{zSy, x(1; S)y} \\
& (z \text{ is an object variable in } V(-B, x, N)) & & (z \text{ is any object variable}) \\
(\_\_\_\_a) & & \frac{x(1; 1)y}{x(1; B)y} \quad (\_\_\_\_b) & & \frac{x(1; S)y}{z(1; S)y} \\
& (z \text{ is a new object variable}) & & (z \text{ is a new object variable})
\end{align*}
\]

4.2 A Dual Tableau Calculus for \((\{1, \cup, \cap\}; \_\_\_\_)-fragment\)

The decomposition rules for Boolean formulae of our dual tableau-based calculus are just the ones in Table 1. Concerning the decomposition rule for \((\_\_\_\_)-formulæ, it is convenient to distinguish between \((\_\_\_\_)-formulæ of type } x(B; S)y} and of type } x(1; S)y}. The rule for \((\_\_\_\_)-formulæ of type } x(B; S)y} is the \((\_\_\_\_a)\text{-rule of Table 2. There, } z} \text{ is an object variable belonging to } V(-B, x, N}, where } N \text{ stands for the current node. Notice that if } S = 1}, the node resulting from the decomposition step is axiomatic. In case of \((\_\_\_\_)-formulæ of type } x(1; S)y}, we apply the rule \((\_\_\_\_b)\text{ in Table 2. The variable } z \text{ used in rule } (\_\_\_\_b)\text{ is any variable on the current node, provided that the current branch does not already contain the formula } zSy}. Otherwise, } x(1; S)y} cannot be decomposed with } z}. If } S = 1, \text{ the same remark made for rule } (\_\_\_\_a), \text{ for the node resulting from the decomposition step, holds here as well.}

Concerning \((\_\_\_\_)-formulæ, we consider first the case of formulæ of type } x(-B; S)y}. If } S \neq 1, \text{ such formulæ are decomposed by means of the } (\_\_\_\_)-rule in Table 1. Otherwise, when } S = 1, \text{ we use the rule } (\_\_\_\_a) \text{ of Table 2. In the case of formulæ of type } x(-1; S)y, \text{ with } S \neq 1, \text{ we use instead the rule } (\_\_\_\_b) \text{ of Table 2, with } z \text{ an object variable new to the current node. The rule can be applied provided that the current branch does not contain any formula of the form } z'(-S)y, \text{ for any } ' \text{ new } z' \text{ (otherwise, the formula } x(-1; S)y \text{ cannot be decomposed). The formula } x(-1; 1)y \text{ is not decomposed.}

Some remarks on the rules \((\_\_\_\_a), (\_\_\_\_b), (\_\_\_\_a), \text{ and } (\_\_\_\_b) \text{ in Table 2 are in order. The idea behind the definition of the side condition of the rule } (\_\_\_\_a) \text{ takes inspiration from the side condition of expansion rules for universally quantified formulæ present in various well known tableau-based proof systems (see for instance [2]). The introduction of the set } V(-B, x, N) \text{ is motivated by the fact that our relational fragment admits compositional terms } (B; S), \text{ where } B \text{ may be a compound term. Observe that for every } RL(1)-\text{model } M = (U, m), m(1) = U \times U \text{ so that, } M, v \models x1z \text{ and } M, v \not\models x(1)z \text{ hold, for every valuation } v \text{ and object variables } x \text{ and } z. \text{ Thus, we shall assume without loss of generality that each node of any dual tableau for formulæ of the } (\{1, \cup, \cap\}; \_\_\_\_)-fragment contains implicitly all literals of type } x(1)z. \text{ This accounts for the fact that the decomposition rules } (\_\_\_\_a) \text{ and } (\_\_\_\_b) \text{ do not introduce } z(1)y \text{ and } x'(1)z, \text{ respectively, on the new node, and rule } (\_\_\_\_b) \text{ restricts } z \text{ to be any variable on the}
current node, rather than any possible variable. At any rate, we shall prove that such a restriction preserves the completeness of the procedure.

It is convenient to introduce the notion of deduction tree for RL(1)-formulae to give a step-by-step description of the proof tree construction process.

As proof trees, deduction trees are ordered trees whose nodes are labelled with disjunctive sets. However, deduction trees may have some leaf nodes that do not contain any axiomatic set and such that decomposition rules can still be applied to them. As will be clarified below, deduction trees can be seen as “approximations” of proof trees with the property that they can be completed to proof trees.

**Definition 1.** Let \( xPy \) be a \((\{1, \cup, \cap\}; \_)-formula. A deduction tree \( T \) for \( xPy \) is recursively defined as follows:

(a) the tree with only one node labelled with \( \{xPy\} \) is a deduction tree for \( xPy \) (initial deduction tree);

(b) let \( T \) be a deduction tree for \( xPy \) and let \( \theta \) be a branch of \( T \) whose leaf node \( N \) does not contain an axiomatic set.\(^3\) The tree obtained from \( T \) by applying to \( N \) either one of the decomposition rules in Table 1 (for Boolean formulae and for \((\_)-formulae of type \( x'-(B; S)y \), with \( S \neq 1 \)), or one of the decomposition rules in Table 2 (for \((\_)-formulae and for \((\_)-formulae of type \( x'-(B; 1)y \) and of type \( x-(1; S)y \) is a deduction tree for \( xPy \). More precisely, rules applications are described as follows:

- if a formula \( x'Qy \) occurs in \( N \) and a rule with a single conclusion set of formulae \( \Gamma \) (resp., a branching rule with the conclusion sets \( \Gamma_1 \) and \( \Gamma_2 \)) is applicable to \( x'Qy \), then we append the node \( N' = (N \setminus \{x'Qy\}) \cup \Gamma \) as the successor of \( N \) in \( \theta \) (resp., the node \( N'_1 = (N \setminus \{x'Qy\}) \cup \Gamma_1 \) as the left successor of \( N \) and the node \( N'_2 = (N \setminus \{x'Qy\}) \cup \Gamma_2 \) as the right successor of \( N \) in \( \theta \)).

Given a branch \( \theta \) of a deduction tree, each object variable in \( W_\theta \setminus \{x, y\} \) is generated by an application of a \((\_)-decomposition rule. We say that a variable \( w \) is an ancestor of degree \( n \) of a variable \( z \in W_\theta \setminus \{x, y\} \) if there is a sequence \( z_1, \ldots, z_n \) of variables in \( W_\theta \setminus \{x, y\} \), with \( z_n = z \) and \( n \geq 1 \), such that \( z_1 \) is generated by a \((\_)-formula \( w(-(B_0; S_0))y \), \( z_2 \) is generated by a \((\_)-formula \( z_1(-(B_1; S_1))y \), \( \ldots \), \( z_n \) is generated by a \((\_)-formula \( z_{n-1}(-(B_{n-1}; S_{n-1}))y \), where \( w(-(B_0; S_0))y, z_1(-(B_1; S_1))y, \ldots, z_{n-1}(-(B_{n-1}; S_{n-1}))y \) are formulae of \( \theta \). In such a case, we say that \( z_1 \) is a descendant of degree 1 of \( w \) and that \( z_n = z \) is a descendant of degree \( n \) of \( w \).

It is useful to define a total order \( <_\theta \) among variables in \( W_\theta \) such that:

- \( x <_\theta w \), for every \( w \in W_\theta \setminus \{x\}, \)
- \( x_1 <_\theta x_2 \), for every \( x_1, x_2 \in W_\theta \setminus \{x, y\} \), with \( x_1 \) introduced in \( \theta \) before \( x_2 \),
- \( y <_\theta z \), for every \( z \) descendant of \( y \),
- \( w <_\theta y \), for every \( w \) that is not a descendant of \( y \).

\(^3\)From now on, we identify nodes with the (disjunctive) sets labelling them.
Remark 1. Notice that the relationship ancestor/descendant is based on the literals of type \(x'(-r)z\) that are generated by applying either the \((-;\)\)-rule of Table 1 or the \((-;)_a\)-rule of Table 2, and, possibly, the Boolean decomposition rules.

Remark 2. By the construction of \(\mathbb{RT}_{\{\{1,\cup,\cap\};\langle\rangle\}}\), a deduction tree for a formula \(xPy\) may contain formulae of type \(x(1;S_1)y\) and of type \(x(-1;S_2)y\) only if their left variable is \(x\). This is motivated by the fact that each of these formulae can be obtained only by the Boolean decomposition of \(xPy\). Any variable \(z\) resulting from the decomposition of a \((-;)\)-formula of type \(x(-1;S))y\) is not a descendant of \(x\). However, according to the definition of the order \(<_\theta\), \(x <_\theta z\) holds.

The following notions will be used in the next section to turn our tableau calculus for \((\{1,\cup,\cap\};\langle\rangle)\) into a terminating proof tree construction procedure.

Let \(\theta\) be a branch of a deduction tree, and let \(z(-B;S)\) and \(z'(-B;S)\) be two \((-;)\)-formulae occurring in \(\theta\). We say that \(z'(-B;S)\) blocks \(z(-B;S)\) (and that \(z(-B;S)\) is blocked by \(z'(-B;S)\)), if the following conditions are satisfied:

- \(z(-B;S)\) and \(z'(-B;S)\) are identical with the exception of the left object variable,
- \(z'(-B;S)\) has been already decomposed in \(\theta\) using the variable \(w\),
- for every \(\langle\rangle\)-formula \(z(B_1;Q)\) occurring in \(\theta\) such that \(z(-B_1)w\) has a Boolean construction from the set of literals resulting from the Boolean decomposition of \(z(-B)w\), the \(\langle\rangle\)-formula \(z'(B_1;Q)\) occurs in \(\theta\) as well.

### 4.3 A Proof Tree Construction Procedure for \((\{1,\cup,\cap\};\langle\rangle)\)

Starting with an initial deduction tree \(T_0\) for a given formula \(xPy\), the following procedure constructs a proof tree for \(xPy\).

1. For every non-axiomatic branch \(\theta\) of the current deduction tree,
2. while \(\theta\) is non-axiomatic and is further expandable, let \(z\) be the smallest variable w.r.t. \(<_\theta\) such that formulae on \(\theta\) with left variable \(z\) have not been decomposed in \(\theta\). Apply to the formulae on \(\theta\) having left variable \(z\) the decomposition rules in the following order: Boolean rules, \((-;)\)-rules, rule \((\langle\rangle)_a\), and then apply rule \((\langle\rangle)_b\) to decompose the \(\langle\rangle\)-formulae of type \(x(1;S)\) \(y\) in \(\theta\) with the variable \(z\) in a systematic way under the following restrictions:
   a. all the rules can be applied at most once with the same premise;
   b. every formula of type \((-;)\), \(z(-B;S)\) \(y\) is not decomposed provided that it is blocked by a \(\langle\rangle\)-formula \(z'(-B;S)\) \(y\) occurring in \(\theta\).

   If \(z'(-B;S)\) \(y\) was decomposed in \(\theta\) with the variable \(w\), then for every literal \(z'(-r)w\in \bigcup\theta\) (obtained from the application of the Boolean rules to \(z'(-B)w\)) we store the literal \(z'(-r)w\) in \(Lit(-;1)\), a set (empty at the beginning of the execution of the procedure) collecting literals not explicitly occurring in \(\theta\) that are needed to construct \(M_\theta\) (see step 4).
3. If the branch $\theta$ is axiomatic and all the other branches on the current deduction tree are axiomatic, then the current deduction tree is a proof tree for $xPy$ and we terminate. Otherwise, if the branch $\theta$ is axiomatic and there are still non-axiomatic branches on the current deduction tree, return to step 1.

4. Otherwise, if $\theta$ is non-axiomatic, namely it is a non-axiomatic not further expandable branch, we construct from $\theta$ the model $M_\theta = (U_\theta, m_\theta)$ defined as follows. We put $U_\theta = W_\theta$. Next, let $\text{Lit}_\theta$ be the set of all literals occurring in $\theta$, and let $\text{Lit}_{\neg \cdot}$ be defined as in step 2. We define the interpretation $M_\theta$ by putting $(x', y') \in m_\theta(R)$ if and only if $x'Rx'' \in (\text{Lit}_\theta \cup \text{Lit}_{\neg \cdot})$. Let $v_\theta : \mathbb{O}V \to U_\theta$ be a valuation such that $v_\theta(x) = \text{Def}_x$, for every $x \in U_\theta$. We terminate returning $\theta, M_\theta$, and $v_\theta$.

The next lemma states two useful properties of the formulae occurring on the deduction trees constructed by the proof procedure above. Its proof can be carried out by induction on proof construction and by case distinction on the structure of $x'Rx''$.

**Lemma 2.** Let $T$ be a deduction tree for $xPy$ constructed by an execution of the procedure described above. If $x'Rx''$ is a formula of a branch $\theta$ of $T$, then (i) $R \in \text{cp}(P)$, and (ii) if $R$ contains the composition operator, then $x'' = y$.

**Termination of the procedure.** Let $T$ be a proof tree for a formula $xPy$ of the $\{\{1, \cup, \cap\}; \cdot\}$-fragment constructed according to our proof-tree construction procedure. To prove that our procedure always terminates, we show that any branch of $T$ can be constructed in a finite number of steps. We mainly focus on non-axiomatic not further expandable branches, since in the case of axiomatic branches the proof is straightforward. To begin with, we characterize a non-axiomatic not further expandable branch $\theta$ of $T$ as a non-axiomatic branch such that all the rules applicable to the formulas occurring on its nodes have been applied following the steps of the given decision procedure.

Next we state some preliminary lemmas and remarks useful to show that $\theta$ contains a finite number of formulae. Lemma 3 is a technical lemma used to prove Lemmas 4 and 5 which, in their turn, are used in Lemma 6 to show that $\cdot$-formulae, the only formulae that can be decomposed more than once, are decomposed a finite number of times. Lemma 4 is proved by showing that the set $V(B, w, N)$ is finite, where $N$ is the leaf node of $\theta$. The proof of Lemma 5 uses the fact that $\text{cp}(P)$ is finite (Lemma 2), the fact that for every $x' \in W_\theta$, $\bigcup \theta$ contains a finite number of formulae of type $x'Rx''$ (Lemma 3), and the blocking mechanism introduced in Sect. 4.2. The interested reader may find the proofs of Lemmas 3, 4, and 5 in [6].

**Lemma 3.** Let $\theta$ be a non-axiomatic not further expandable branch of a proof tree $T$ for a formula $xPy$. Then, for every $x' \in W_\theta$, $\bigcup \theta$ contains a finite number of formulae of type $x'Rx''$.

**Remark 3.** Variables generated by $\cdot$-formulae with left variable $y$ are finitely many because $\cdot$-formulae of type $y\cdot(B;S)y$ are finitely many too. Moreover
these variables are distinct from all the variables generated by the other \((-;\))-formulae because each application of the \((-;\))-rule introduces a new variable.

**Remark 4.** If a variable \(w\) is generated by a \((-;\))-formula \(x’(-(B;S))y\) with \(x’ \neq y\), then no literal of the form \(y(-r)w\) is in \(\theta\). In fact, by Lemma 3 we know that literals of type \(y(-r)z\), with \(z \neq y\), are introduced in \(\theta\) only after the decomposition of a \((-;\))-formula with left variable \(y\). But then \(z\) cannot be the same variable introduced by a \((-;\))-formula \(x’(-(B;S))y\) with \(x’ \neq y\).

**Remark 5.** Every \((;\))-formula \(w(B;S)y\) is decomposed only with the variables introduced by the decomposition of \((-;\))-formulae with left variable \(w\) and possibly with the variable \(y\).

**Lemma 4.** Every formula \(w(B;S)y\) in \(\theta\) is decomposed a finite number of times.

**Lemma 5.** \(W_\theta\) is finite.

Next, we define recursively the weight of a term by putting:

- \(weight(r) = weight(-r) = weight(1) = weight(-1) = 0;\)
- \(weight(A \; \sharp \; P) = weight(A) + weight(P) + 1\), for \(\sharp \in \{\cup, \cap, ;\};\)
- \(weight(-(A \; \sharp \; P)) = weight(-A) + weight(-P) + 1\), for \(\sharp \in \{\cup, \cap, ;\};\)
- \(weight(-P) = weight(P) + 1.\)

Then the weight of a formula \(xPy\) is defined as the weight of its term \(P\) and the weight of a node \(N\) is defined as the sum of the weights of the formulae in \(N\). In particular, the weight of every \((;\))-formula and the weight of every \((-;\))-formula that cannot be decomposed in \(N\), according to the decomposition rules and, in particular, to the conditions on rules application stated in step 2, is set to 0. It can be checked that the weight of a node \(N\) is 0 if and only if it contains only literals and formulae of types \((;\) and \((-;\)) that cannot be further decomposed, according to the definition of the decomposition rules and of the requirements on rules application in step 2 of our proof-tree construction procedure. Thus, a branch with leaf node of weight 0 is not further expandable.

**Lemma 6.** After a finite number of decomposition steps, a branch \(\theta\) of a deduction tree for \(xPy\) is prolonged to a branch which can be either axiomatic or non-axiomatic and whose leaf node has weight equal to 0.

**Proof.** Let \(\theta = \theta_1, \theta_2, \ldots\) be such that \(\theta_{i+1}\) is obtained from \(\theta_i\) by an application of a decomposition rule to the leaf node \(N_i\) of \(\theta_i\), for \(i = 1, \ldots\). If \(\theta_i\) happens to be an axiomatic branch, then the thesis immediately follows. Otherwise, we reason as shown next. For every \((;\))-formula \(\varphi\) of \(N_i\), of both types \(x’(B;S)y\) and \(x(1;S)y\), let \(\text{dec}(\varphi, N_i)\) be the number of times \(\varphi\) has been decomposed on the branch to which \(N_i\) belongs. If \(\varphi = x(1;S)y\), then \(\text{dec}(\varphi, N_i) \leq |W_\theta|\). By Lemma 5, \(|W_\theta|\) is finite and once \(\text{dec}(x(1;S)y, N_i)\) reaches it, \(weight(x(1;S)y)\) is set to 0. If \(\varphi = x’(B;S)y\), then \(\text{dec}(\varphi, N_i)\) is bounded as stated in Lemma 4. It turns out that, at each decomposition step, we have either
1. \(\text{weight}(N_i) > \text{weight}(N_{i+1})\), or
2. \(\sum_{\varphi \in N_i} \text{dec}(\varphi, N_i) < \sum_{\varphi \in N_{i+1}} \text{dec}(\varphi, N_{i+1})\).

The first condition holds when the decomposition rule applied to \(\theta_i\) to produce \(\theta_{i+1}\) is different from the \(\cdot\)-rule. In fact the decomposed formula is not introduced in the new node and the components have smaller weights. Moreover, each \(\cdot\)-formula that is blocked gets weight 0. The second condition, on the other hand, holds when the \(\cdot\)-rule is used. In this case, since the decomposed \(\cdot\)-formula \(\varphi\) is introduced in the new node, the weight of the new node does not decrease (it could increase), but \(\text{dec}(\varphi, N_i)\) increases and since it is bounded, after a finite number of steps \(\varphi\) is not decomposed anymore getting weight 0.

Since each node contains a finite number of formulae, after a finite number of steps we obtain a branch \(\theta_n\) whose leaf node has weight 0. This means that \(\theta_n\) is not further expandable. Moreover, if \(\theta_n\) is not closed, then it is a non-axiomatic not further expandable branch. In fact, all the Boolean formulae in \(\theta_n\) have been decomposed, and, in view of the conditions of step 2 all the \(\cdot\)-formulae either have been decomposed into formulae of smaller weight or have not been decomposed and their weight has been set to 0. Finally, all the \(\cdot\)-formulae in \(\theta_n\) have been decomposed, each finitely many times according to condition (a) of step 2.

Considering that our proof-tree construction procedure constructs any axiomatic branch and any non-axiomatic not further expandable branch of a proof tree for \(xPy\) in a finite number of decomposition steps and that each decomposition rule is finitely branching, we can state the following theorem.

**Theorem 1 (Termination).** The dual tableau procedure for the \(\{1, \cup, \cap\} \cdot\)-fragment always terminates.

**Soundness and completeness.** Correctness of our proof-tree construction procedure is proved by showing that when the input formula \(xPy\) is valid, the procedure yields a closed (axiomatic) dual tableau for \(xPy\), whereas if \(xPy\) is not valid, the procedure yields a non-axiomatic not further expandable branch \(\theta\) of a dual tableau for \(xPy\) and a model \(M_\theta\) that falsifies every formula on \(\theta\) (thus, in particular, \(xPy\) itself).

Lemma 7, stated below, is used in the proof of Theorem 2 to establish the first half of the correctness proof, and also later, in the proof of Theorem 3.

**Lemma 7.** Let \(T\) be a deduction tree for a formula \(xPy\) of the \(\{1, \cup, \cap\} \cdot\)-fragment, constructed as described in our proof-tree construction procedure. If the procedure terminates at step 4 yielding a non-axiomatic not further expandable branch \(\theta\), a model \(M_\theta = (U_\theta, m_\theta)\), and a valuation \(v_\theta\), then \(M_\theta\) and \(v_\theta\) falsify \(\theta\).

**Theorem 2.** If \(xPy\) is a valid formula of the \(\{1, \cup, \cap\} \cdot\)-fragment of RL(1), then our proof-tree construction procedure yields a closed proof tree for \(xPy\).
Lemma 8, presented next, states that each decomposition step performed by our proof-tree construction procedure preserves falsifiability. This result is needed later in the proof of Theorem 3, to establish the second half of the correctness proof. Proofs of Lemmas 7 and 8 and of Theorems 2 and 3 can be found in [6].

**Lemma 8.** Let \( \theta \) be a branch of a deduction tree for a formula \( xPy \) of the \( (\{1, \cup, \cap\}; \_ \) -fragment that is being constructed by our proof-tree construction procedure, and let \( \theta' \) be obtained from \( \theta \) by a decomposition step performed by the decision procedure. If \( \theta \) is a falsifiable branch, then \( \theta' \) is falsifiable too.

**Theorem 3.** Let \( xPy \) be a non valid relational formula of the \( (\{1, \cup, \cap\}; \_ \) -fragment. Then our proof-tree construction procedure yields a non-axiomatic not further expandable branch \( \theta \) of a dual tableau for \( xPy \) and a model \( M_\theta \) that falsifies every formula on \( \theta \) and, therefore, \( xPy \) itself.

Summing up, Theorems 2 and 3 yield the following result.

**Theorem 4.** The \( (\{1, \cup, \cap\}; \_ \) -fragment has a decidable validity problem.

5 Conclusions and Future Work

Relational entailment allows one to deal with properties of relational constants and of relational variables in dual tableau proofs without adding any specific rule to the basic set of decomposition rules. Using entailment in dual tableau-based decision procedures, however, can be tricky because the constant \( 1 \) occurs both on the left-hand side and on the right-hand side of composition.

We have presented a dual tableau-based decision procedure for a fragment of the logic RL(1) which can express simple forms of inclusion between relations. Specifically, we admit inside entailment only positive occurrences of Boolean terms and thus we can express inclusion properties of the form \( (r_1 \cup s) \subseteq r_2 \).

We plan to extend the expressibility of our relational fragment in order to make entailment widely applicable in dual tableau-based decision procedures. As a first step, we intend to include negative occurrences of Boolean terms inside entailment. In this way we will be able to formulate terms of type \( 1; (-(- (r_1 \cup s) \cup r_2) \cup 1) \) expressing the (positive) inclusion property \( (r_1 \cup s) \subseteq r_2 \).

Our further aim is to add, inside entailment, some restricted forms of composition so as to be able to express terms of type \( 1; (-(-s; s) \cup s) \cup 1 \) and of type \( 1; (-(-r; r) \cup r) \cup 1 \), stating, respectively, that the relational variables \( s \) and \( r \) are transitive (i.e., \( \langle s; s \rangle \subseteq s \)) and three-transitive (i.e., \( \langle r; r; r \rangle \subseteq r \)), respectively. Expressing these properties is important if one wants to use our dual tableau decision procedure with various non-classical logics such as, for instance, modal logics to reason with incomplete information [7].

We also intend to introduce the converse relation ‘\( \mapsto \)’ and the identity relation ‘\( 1' \)’ inside entailment for the purpose of dealing with properties such as symmetry and reflexivity.
Acknowledgments. Thanks are due to three anonymous referees for their helpful suggestions. This work was supported by Indam-GNCS, Progetto di ricerca “Automi Reattivi e loro Simulazione nell’Ambito del Non-Standard Secure Text Processing”.

References

Query Answering over Contextualized RDF Knowledge
with Forall-Existential Bridge Rules: Attaining
Decidability using Acyclicity

Mathew Joseph\textsuperscript{1,2}, Gabriel Kuper\textsuperscript{2}, and Luciano Serafini\textsuperscript{1}

\textsuperscript{1} DKM, FBK-IRST, Trento, Italy
\textsuperscript{2} DISI, University Of Trento, Trento, Italy

\{mathew,serafini\}@fbk.eu, kuper@disi.unitn.it

Abstract. The recent outburst of context-dependent knowledge on the Semantic Web (SW) has led to the realization of the importance of the quads in the SW community. Quads, which extend a standard RDF triple, by adding a new parameter of the ‘context’ of an RDF triple, thus informs a reasoner to distinguish between the knowledge in various contexts. Although this distinction separates the triples in an RDF graph into various contexts, and allows the reasoning to be decoupled across various contexts, bridge rules need to be provided for inter-operating the knowledge across these contexts. We call a set of quads together with the bridge rules, a quad-system. In this paper, we discuss the problem of query answering over quad-systems with expressive forall-existential bridge rules. It turns out the query answering over quad-systems is undecidable, in general. We derive a decidable class of quad-systems, namely context-acyclic quad-systems, for which query answering can be done using forward chaining. Tight bounds for data and combined complexity of query entailment has been established for the derived class.

Keywords: Contextualized RDF/OWL knowledge, Contextualized Query Answering, Quads, Forall-Existential Rules, Semantic Web, Knowledge Representation.

1 Introduction

One of the major recent changes in the SW community is the transformation from a triple to a quad as its primary knowledge carrier. As a consequence, more and more triple stores are becoming quad stores. Some of the popular quad-stores are 4store\textsuperscript{1}, Openlink Virtuoso\textsuperscript{2}, and some of the current popular triple stores like Sesame\textsuperscript{3} internally keep track of the context by storing arrays of four names \((c, s, p, o)\) (further denoted as \(c: (s, p, o)\)), where \(c\) is an identifier that stands for the context of the triple \((s, p, o)\). Some of the recent initiatives in this direction have also extended existing formats like N-Triples to N-Quads. The latest Billion triples challenge datasets (BTC 2012) have all been released in the N-Quads format.

\textsuperscript{1} http://4store.org
\textsuperscript{2} http://virtuoso.openlinksw.com/rdf-quad-store/
\textsuperscript{3} http://www.openrdf.org/
One of the main benefits of quads over triples are that they allow users to specify various attributes of meta-knowledge that further qualify knowledge [8], and also allow users to query for this meta-knowledge [30]. Examples of these attributes, which are also called context dimensions [12], are provenance, creator, intended user, creation time, validity time, geo-location, and topic. Having defined various contexts in which triples are dispersed, one can declare in another meta-context $mc$, statements such as $mc: (c_1, \text{creator}, \text{John})$, $mc: (c_1, \text{expiryTime}, \text{“jun-2013”})$ that talk about the knowledge in context $c_1$, in this case its creator and expiry time. Another benefit of such a contextualized approach is that it opens possibilities of interesting ways for querying a contextualized knowledge base. For instance, if context $c_1$ contains knowledge about Football World Cup 2014 and context $c_2$ about Football Euro Cup 2012. Then the query “who beat Italy in both Euro Cup 2012 and World Cup 2014” can be formalized as the conjunctive query:

$$c_1: (x, \text{beat}, \text{Italy}) \land c_2: (x, \text{beat}, \text{Italy}),$$

where $x$ is a variable.

As the knowledge can be separated context wise and simultaneously be fed to separate reasoning engines, this approach increases both efficiency and scalability. Besides the above flexibility, bridge rules [4] can be provided for inter-interoperating the knowledge in different contexts. Such rules are primarily of the form:

$$c: \phi(x) \rightarrow c': \phi'(x)$$

where $\phi, \phi'$ are both atomic concept (role) symbols, $c, c'$ are contexts. The semantics of such a rule is that if, for any $a$, $\phi(a)$ holds in context $c$, then $\phi'(a)$ should hold in context $c'$, where $a$ is a unary(binary) vector dependending on whether $\phi, \phi'$ are concept/role symbols. Although such bridge rules serve the purpose of specifying knowledge interoperability from a source context $c$ to a target context $c'$, in many practical situations there is the need of interoperating multiple source contexts with multiple target targets, for which the bridge rules of the form (1) is inadequate. Besides, one would also want the ability of creating new values in target contexts for the bridge rules.

In this work, we consider forall-existential bridge rules that allows conjunctions and existential quantifiers in them, and hence is more expressive than those, in DDL [4] and McCarthy et al. [28]. A set of quads together with such bridge rules is called a quad-system. The main contributions of this work can be summarized as:

1. We provide a basic semantics for contextual reasoning over quad-systems, and study contextualized conjunctive query answering over them. For query answering, we use the notion of a distributed chase, which is an extension of a standard chase [22, 1] that is widely used in databases and KR for the same.
2. We show that conjunctive query answering over quad-systems, in general, is undecidable. We derive a class of quad-systems called context acyclic quad-systems, for which query answering is decidable and can be done by forward chaining. We give both data and combined complexity of conjunctive query entailment for the same.

The paper is structured as follows: In section 2, we formalize the idea of contextualized quad-systems, giving various definitions and notations for setting the background.
In section 3, we formalize the query answering on quad-systems, define notions such as distributed chase that is further used for query answering, and give the undecidability results of query entailment for unrestricted quad-systems. In section 4, we present context acyclic quad-systems and its properties. We give an account of relevant related works in section 5, and conclude in section 6. A version of this paper with detailed proofs is available at [23].

2 Contextualized Quad-Systems

In this section, we formalize the notion of a quad-system and its semantics. For any vector or sequence $x$, we denote by $|x|$ the number of symbols in $x$, and by $\{|x|\}$ the set of symbols in $x$. For any sets $A$ and $B$, $A \to B$ denotes the set of all functions from set $A$ to set $B$. Given the set of URIs $U$, the set of blank nodes $B$, and the set of literals $L$, the set $C = U \cup B \cup L$ are called the set of (RDF) constants. Any $(s, p, o) \in C \times C \times C$ is called a generalized RDF triple (from now on, just triple).

A graph is defined as a set of triples. A Quad is a tuple of the form $c: (s, p, o)$, where $(s, p, o)$ is a triple and $c$ is a URI\(^4\), called the context identifier that denotes the context of the RDF triple. A quad-graph is defined as a set of quads. For any quad-graph $Q$ and any context identifier $c$, we denote by $\text{graph}_{Q,c}$ the set $\{(s, p, o)c: (s, p, o) \in Q\}$. We denote by $Q_{C}$ the quad-graph whose set of context identifiers is $C$. Let $V$ be the set of variables, any element of the set $C^V = V \cup C$ is a term. Any $(s, p, o) \in C^V \times C^V \times C^V$ is called a triple pattern, and an expression of the form $c: (s, p, o)$, where $(s, p, o)$ is a triple pattern, $c$ a context identifier, is called a quad pattern. A triple pattern $t$, whose variables are elements of the vector $x$ or elements of the vector $y$ is written as $t(x, y)$.

For any function $f: A \to B$, the restriction of $f$ to a set $A'$, is the mapping $f|_{A'}$ from $A' \cap A$ to $B$ s.t. $f|_{A'}(a) = f(a)$, for each $a \in A \cap A'$. For any triple pattern $t = (s, p, o)$ and a function $\mu$ from $V$ to a set $A$, $t[\mu]$ denotes $(\mu(s), \mu(p), \mu(o))$, where $\mu'$ is an extension of $\mu$ to $C$ s.t. $\mu'|_{C}$ is the identity function. For any set of triple patterns $G$, $G[\mu]$ denotes $\bigcup_{t \in G} t[\mu]$. For any vector of constants $\alpha = \langle a_1, \ldots, a_{|\alpha|}\rangle$, and vector of variables $x$ of the same length, $x/\alpha$ is the function $\mu$ s.t. $\mu(x_i) = a_i$, for $1 \leq i \leq |\alpha|$. We use the notation $t(a, y)$ to denote $t(x, y)[x/\alpha]$.

Bridge rules (BRs) Bridge rules (BR) enables knowledge propagation across contexts. Formally, a BR is an expression of the form:

$$\forall x \forall y \forall z \left[ c_1: t_1(x, z) \land \ldots \land c_n: t_n(x, z) \rightarrow \exists y \ c'_1: t'_1(x, y) \land \ldots \land c'_m: t'_m(x, y) \right] (1)$$

where $c_1, \ldots, c_n, c'_1, \ldots, c'_m$ are context identifiers, $x, y, z$ are vectors of variables s.t. $\{x\}, \{y\}$, and $\{z\}$ are pairwise disjoint. $t_1(x, z), \ldots, t_n(x, z)$ are triple patterns which do not contain blank-nodes, and whose set of variables are from $x$ or $z$. $t'_1(x, y), \ldots, t'_m(x, y)$ are triple patterns, whose set of variables are from $x$ or $y$, and also does not contain blank-nodes. For any BR, $r$, of the form (1), $\text{body}(r)$ is the set of quad patterns $\{c_1: t_1(x, z), \ldots, c_n: t_n(x, z)\}$, and $\text{head}(r)$ is the set of quad patterns $\{c'_1: t'_1(x, y), \ldots, c'_m: t'_m(x, y)\}$.

\(^4\) Although, in general a context identifier can be a constant, for the ease of notation, we restrict them to be a URI.
Definition 1 (Quad-System). A quad-system $QSC$ is defined as a pair $\langle QC, R \rangle$, where $QC$ is a quad-graph, whose set of context identifiers is $C$, and $R$ is a set of BRs.

For any quad-graph $QC$ (BR $r$), its symbols size $\|QC\|$ $(\|r\|)$ is the number of symbols required to print $QC$ ($r$). Hence, $\|QC\| \approx 4 \times |QC|$, where $|QC|$ denotes the cardinality of the set $QC$. Note that $|QC|$ equals the number of quads in $QC$. For a BR $r$, $\|r\| \approx 4 \times k$, where $k$ is the number of quad-patterns in $r$. For a set of BRs $R$, its size $\|R\|$ is given as $\Sigma_{r \in R} \|r\|$. For any quad-system $QSC = \langle QC, R \rangle$, its size $\|QSC\| = \|QC\| + \|R\|$.

Semantics In order to provide a semantics for enabling reasoning over a quad-system, we need to use a local semantics for each context to interpret the knowledge pertaining to it. Since the primary goal of this paper is a decision procedure for query answering over quad-systems based on forward chaining, we consider the following desiderata for the choice of the local semantics:

- there exists a set of inference rules and an operation $lclosure()$ that computes the deductive closure of a graph w.r.t to the local semantics using the inference rules.
- given a finite graph as input, the $lclosure()$ operation, terminates with a finite graph as output in polynomial time whose size is polynomial w.r.t. to the input set.

Some of the alternatives for the local semantics satisfying the above mentioned criterion are Simple, RDF, RDFS [19], OWL-Horst [20] etc. Assuming that a local semantics has been fixed, for any context $c$, we denote by $I^c = \langle \Delta^c, .^c \rangle$ an interpretation structure for the local semantics, where $\Delta^c$ is the interpretation domain, $.^c$ the corresponding interpretation function. Also $\models_{local}$ denotes the local satisfaction relation between a local interpretation structure and a graph. Given a quad graph $QC$, a distributed interpretation structure is an indexed set $\mathcal{I}^C = \{ I^c \}_{c \in C}$, where $I^c$ is a local interpretation structure, for each $c \in C$. We define the satisfaction relation $\models$ between a distributed interpretation structure $\mathcal{I}^C$ and a quad-system $QSC$ as:

Definition 2 (Model of a Quad-System). A distributed interpretation structure $\mathcal{I}^C = \{ I^c \}_{c \in C}$ satisfies a quad-system $QSC = \langle QC, R \rangle$, in symbols $\mathcal{I}^C \models QSC$, iff all the following conditions are satisfied:

1. $I^c \models_{local} graph_{QC}(c)$, for each $c \in C$;
2. $a^{c \sigma} = a^{c \sigma}$, for any $a \in C$, $c_i, c_j \in C$;
3. for each BR $r \in R$ of the form (1) and for each $\sigma \in V \rightarrow \Delta^c$, where $\Delta^c = \bigcup_{c \in C} \Delta^c$, if $I^c_1 \models_{local} t_1(x, z)[\sigma],..., I^c_m \models_{local} t_n(x, z)[\sigma]$, then there exists function $\sigma' \supseteq \sigma$, s.t.
   \begin{align*}
   I'^1 \models_{local} t'_1(x, y)[\sigma'] ,..., I'^m \models_{local} t'_n(x, y)[\sigma'].
   \end{align*}

Condition 1 in the above definition ensures that for any model $\mathcal{I}^C$ of a quad-graph, each $I^c \in \mathcal{I}^C$ is a local model of the set of triples in context $c$. Condition 2 ensures that any constant $c$ is rigid, i.e. represents the same resource across a quad-graph, irrespective of the context in which it occurs. Condition 3 ensure that any model of a quad-system satisfies each BR in it. Any $\mathcal{I}^C$ s.t. $\mathcal{I}^C \models QSC$ is said to be a model of $QSC$. A quad-system $QSC$ is said to be consistent if there exists a model $\mathcal{I}^C$, s.t. $\mathcal{I}^C \models QSC$, and otherwise said to be inconsistent. For any quad-system $QSC = \langle QC, R \rangle$, it can be
the case that graphQC\(c\) is locally consistent, for each \(c \in C\), whereas \(QS_C\) is not consistent. This is because the set of BRs \(R\) adds more knowledge to the quad-system, and restricts the set of models that satisfy the quad-system.

**Definition 3 (Quad-system entailment).** (a) A quad-system \(QS_C\) entails a quad \(c\) \((s, p, o)\), in symbols \(QS_C \models c\) \((s, p, o)\), iff for any distributed interpretation structure \(I^C\), if \(I^C \models QS_C\) then \(I^C \models \{c\ : \ (s, p, o)\}\). (b) A quad-system \(QS_C\) entails a quad-graph \(Q'_C\), in symbols \(QS_C \models Q'_C\), iff \(QS_C \models c\) \((s, p, o)\) for any \(c\) \((s, p, o)\) \(\in Q'_C\). (c) A quad-system \(QS_C\) entails a BR \(r\) iff for any \(I^C\), if \(I^C \models QS_C\) then \(I^C \models \{\emptyset, \{r\}\}\). (d) For a set of BRs \(R\), \(QS_C \models R\) iff \(QS_C \models r\) for every \(r \in R\). (e) Finally, a quad-system \(QS_C\) entails another quad-system \(QS_C'\) \(= \{Q'_C, R'\}\), in symbols \(QS_C \models QS_C'\), iff \(QS_C \models Q'_C\) and \(QS_C \models R'\).

We call the decision problems (DPs) corresponding to the entailment problems (EPs) in (a), (b), (c), (d), and (e) as quad EP, quad-graph EP, BR EP, BRs EP, and quad-system EP, respectively.

### 3 Query Answering on Quad-Systems

In the realm of quad-systems, the classical conjunctive queries or select-project-join queries are slightly extended to what we call **Contextualized Conjunctive Queries (CCQs)**. A CCQ \(CQ(x)\) is an expression of the form:

\[
\exists y_1(x, y) \land ... \land q_p(x, y)
\]

where \(q_i\), for \(i = 1, ..., p\) are quad patterns over vectors of free variables \(x\) and quantified variables \(y\). A CCQ is called a boolean CCQ if it does not have any free variables. For any CCQ \(CQ(x)\) and a vector \(a\) of constants s.t. \(\|x\| = \|a\|\), \(CQ(a)\) is boolean. A vector \(a\) is an answer for a CCQ \(CQ(x)\) w.r.t. structure \(I_C\), in symbols \(I_C \models CQ(a)\), iff there exists assignment \(\mu: \{y\} \rightarrow B\) s.t. \(I_C \models \bigcup_{i=1,...,p} g_i(a, y)[\mu]\). A vector \(a\) is a certain answer for a CCQ \(CQ(x)\) over a quad-system \(QS_C\), iff \(I_C \models CQ(a)\), for every model \(I_C\) of \(QS_C\). Given a quad-system \(QS_C\), a CCQ \(CQ(x)\), and a vector \(a\), \(DP\) of determining whether \(QS_C \models CQ(a)\) is called the CCQ EP. It can be noted that the other DPs over quad-systems that we have seen are reducible to CCQ EP. Hence, in this paper, we primarily focus on the CCQ EP.

**dChase of a Quad-System** In order to do query answering over a quad-system, we employ what has been called in the literature, a chase [22, 1], specifically, we adopt notion of the skolem chase given in Marnette [27] and Cuenca Grau et al [9]. In order to fit the framework of quad-systems, we extend the standard notion of chase to a distributed chase, abbreviated dChase. In the following, we show how the dChase of a quad-system can be constructed.

For any BR \(r\) of the form (1), the skolemization \(sk(r)\) is the result of replacing each \(g_i \in \{y\}\) with a globally unique Skolem function \(f'_r\), s.t. \(f'_r: C\|x\| \rightarrow B_{sk}\), where \(B_{sk}\) is a fresh set of blank nodes called skolem blank nodes. Intuitively, for every distinct vector \(a\) of constants, with \(\|a\| = \|x\|\), \(f'_r(a)\) is a fresh blank node, whose node id is a hash of \(a\). Let \(f^r = (f'_1, ..., f'_{\|y\|})\) be a vector of distinct Skolem functions; for any BR...
r the form (1), with slight abuse (Datalog notation) we write its skolemization sk(r) as follows:
\[ c_1 : t_1(x, z), \ldots, c_n : t_n(x, z) \rightarrow c'_1 : t'_1(x, f'^r), \ldots, c'_m : t'_m(x, f'^r) \]

Moreover, a skolemized BR r of the form (3) can be replaced by the following semantically equivalent set of formulas, whose symbol size is worst case quadratic w.r.t \(|r|\):
\[ \{ c_1 : t_1(x, z), \ldots, c_n : t_n(x, z) \rightarrow c'_1 : t'_1(x, f'^r), \ldots, c'_m : t'_m(x, f'^r) \} \]

Note that each BR in the above set has exactly one quad pattern with optional function symbols in its head part. Also note that a BR with out function symbols can be replaced with a set of BRs with single quad-pattern heads. Hence, w.l.o.g, we assume that any BR in a skolemized set \( sk(R) \) of BRs is of the form (4). For any quad-graph \( Q_C \) and a skolemized BR r of the form (4), application of r on \( Q_C \), denoted by \( r(Q_C) \), is given as:
\[ r(Q_C) = \bigcup_{\mu \in v \rightarrow c} \{ c'_1 : t'_1(x, f'^r)[\mu] \mid c_1 : t_1(x, z)[\mu] \in Q_C, \ldots, c_n : t_n(x, z)[\mu] \in Q_C \} \]

For any set of skolemized BRs \( R \), application of \( R \) on \( Q_C \) is given by: \( R(Q_C) = \bigcup_{r \in R} r(Q_C) \). For any quad-graph \( Q_C \), we define:
\[ \text{lclosure}(Q_C) = \bigcup_{c \in C} \{ c : (s, p, o) \mid (s, p, o) \in \text{lclosure}(graph_{Q_C}(c)) \} \]

For any quad-system \( QS_C = (Q_C, R) \), generating BRs \( R_F \) is the set of BRs in \( sk(R) \) with function symbols, and the non-generating BRs is the set \( R_I = sk(R) \setminus R_F \).

Let \( dChase_0(QS_C) = \text{lclosure}(Q_C) \); for \( i \in \mathbb{N}, dChase_{i+1}(QS_C) = \text{lclosure}(dChase_i(QS_C) \cup R_I(dChase_i(QS_C))) \), if \( R_I(dChase_i(QS_C)) \not\subseteq dChase_i(QS_C) \); otherwise:

The dChase of \( QS_C \), denoted \( dChase(QS_C) \), is given as:
\[ dChase(QS_C) = \bigcup_{i \in \mathbb{N}} dChase_i(QS_C) \]

Intuitively, \( dChase_i(QS_C) \) can be thought of as the state of \( dChase(QS_C) \) at the end of iteration \( i \). It can be noted that, if there exists \( i \) s.t. \( dChase_i(QS_C) = dChase_{i+1}(QS_C) \), then \( dChase(QS_C) = dChase_i(QS_C) \). An iteration \( i \), s.t. \( dChase_i(QS_C) \) is computed by the application of the set of (resp. non-)generating BRs \( R_F \) (resp. \( R_I \)), on \( dChase_{i-1}(QS_C) \) is called a (resp. non-)generating iteration. The dChase \( dChase(QS_C) \) of a consistent quad-system \( QS_C \) is a universal model [10] of the quad-system, i.e. it is a model of \( QS_C \), and for any model \( I_C \) of \( QS_C \), there is a homomorphism from
dChase(QSC) to IC. Hence, for any boolean CCQ CQ(), QSC |= CQ() iff there exists a map μ: V(CQ) → C s.t. \{CQ()\}[µ] ⊆ dChase(QSC). We call the sequence dChase_0(QSC), dChase_1(QSC), ..., the dChase sequence of QSC. The following lemma shows that in a dChase sequence of a quad-system, the result of a single generating iteration and a subsequent number of non-generating iterations causes only an exponential blow up in size.

Lemma 1. For a quad-system QSC = (QC, R), the following holds: (i) if i ∈ N is a generating iteration, then \|dChase_i(QSC)\| = O(\|dChase_{i-1}(QSC)\|^{\|R\|}) , (ii) suppose i ∈ N is a generating iteration, and for any j ≥ 1 , i+1, ... , i+j are non-generating iterations, then \|dChase_{i+j}(QSC)\| = O(\|dChase_{i-1}(QSC)\|^{\|R\|}), (iii) for any iteration k, dChase_k(QSC) can be computed in time O(\|dChase_{k-1}(QSC)\|^{\|R\|}).

Proof (sketch). (i) R can be applied on dChase_{i-1}(QSC) by grounding R to the set of constants in dChase_{i-1}(QSC), the number of such groundings is of the order O(\|dChase_{i-1}(QSC)\|^{\|R\|}).

Since lclosure only increases the size polynomially, \|dChase_i(QSC)\| = O(\|dChase_{i-1}(QSC)\|^{\|R\|}).

(ii) From (i) we know that \|R(dChase_{i-1}(QSC))\| = O(\|dChase_{i-1}(QSC)\|^{\|R\|}).

Since no new constant is introduced in any subsequent non-generating iterations, and since any quad contains only four constants, the set of constants in any subsequent dChase iteration is O(4 * \|dChase_{i-1}(QSC)\|^{\|R\|}). Since only these many constants can appear in positions c, s, p, o of any quad generated in the subsequent iterations, the size of dChase_{i+j}(QSC) can only increase polynomially, which means that \|dChase_{i+j}(QSC)\| = O(\|dChase_{i-1}(QSC)\|^{\|R\|}).

(iii) Since any dChase iteration k involves the following two operations: (a) lclosure(), and (b) computing R(dChase_{k-1}(QSC)). (a) can be done in PTIME w.r.t to its input. (b) can be done in the following manner: ground R to the set of constants in dChase_{k-1}(QSC); then for each grounding g, if body(g) ⊆ dChase_{k-1}(QSC), then add head(g) to R(dChase_{k-1}(QSC)). Since, the number of such groundings is of the order O(\|dChase_{k-1}(QSC)\|^{\|R\|}), and checking if each grounding is contained in dChase_{k-1}(QSC), can be done in time polynomial in \|dChase_{k-1}(QSC)\|, the time taken for (b) is O(\|dChase_{k-1}(QSC)\|^{\|R\|}). Hence, any iteration k can be done in time O(\|dChase_{k-1}(QSC)\|^{\|R\|}).

Although, we now know how to compute the dChase of a quad-system, which can be used for deciding CCQ EP, it turns out that for the class of quad-systems whose BRs are of the form (1), which we call unrestricted quad-systems, the dChase can be infinite. This raises the question if there are other approaches that can be used, for instance similar problem arises in DLs with value creation, due to the presence of existential quantifiers, whereas the approaches like the one in Glim et al. [16] provides an algorithm for CQ entailment based on query rewriting.

Theorem 1. The CCQ EP over unrestricted quad-systems is undecidable.

Proof (sketch). We show that the well known undecidable problem of non-emptiness of intersection of context-free grammars (CFGs) is reducible to the CCQ entailment.

216
problem. Given two CFGs, \( G_1 = \langle V_1, T, S_1, P_1 \rangle \) and \( G_2 = \langle V_2, T, S_2, P_2 \rangle \), where \( V_1, V_2 \) are the set of variables, \( T \) s.t. \( T \cap (V_1 \cup V_2) = \emptyset \) is the set of terminals, \( S_1 \in V_1 \) is the start symbol of \( G_1 \), and \( P_1 \) are the set of PRs of the form \( v \to w \), where \( v \in V \), \( w \) is a sequence of the form \( w_1\ldots w_n \), where \( w_i \in V_1 \cup T \). Similarly \( s_2, P_2 \) is defined. Deciding whether the language generated by the grammars \( L(G_1) \) and \( L(G_2) \) have non-empty intersection is known to be undecidable [18].

Given two CFGs \( G_1 = \langle V_1, T, S_1, P_1 \rangle \) and \( G_2 = \langle V_2, T, S_2, P_2 \rangle \), we encode grammars \( G_1, G_2 \) into a quad-system \( QS_c = \langle Q_c, R \rangle \), with only a single context identifier \( c \). Each PR \( r = v \to w \in P_1 \cup P_2 \), with \( w = w_1w_2w_3\ldots w_n \), is encoded as a BR of the form: \( c: (x_1,w_1,x_2)c: (x_2,w_2,x_3),\ldots\ldots c: (x_n,w_n,x_n+1) \to c: (x_1,v,x_n+1) \), where \( x_1,\ldots,x_{n+1} \) are variables. For each terminal symbol \( t_i \in T \), \( R \) contains a BR of the form: \( c: (x, \text{rdf:type}, C) \to \exists y c: (x,t_i,y), c: (y, \text{rdf:type}, C) \) and \( Q_c \) is the singleton: \( c: (a, \text{rdf:type}, C) \). It can be observed that:

\[
QS_c \models \exists y c: (a,S_1,y) \land c: (a,S_2,y) \iff L(G_1) \cap L(G_2) \neq \emptyset
\]

We refer the reader to [23] for the complete proof. □

4 Context Acyclic Quad-Systems: A decidable class

In the previous section, we saw that query answering on unrestricted quad-systems is undecidable, in general. We in the following define a class of quad-systems for which query entailment is decidable. The class has the property that algorithms based on forward chaining, for deciding query entailment, can straightforwardly be implemented (by minor extensions) on existing quad stores. It should be noted that the technique we propose is reminiscent of the Weak acyclicity [13, 11] technique used in the realm of Datalog+.

Consider a BR \( r \) of the form: \( c_1: t_1(x, z), c_2: t_2(x, z) \to \exists y c_3: t_3(x, y), c_4: t_4(x, y) \). Since such a rule triggers propagation of knowledge in a quad-system, specifically triples from the source contexts \( c_1, c_2 \) to the target contexts \( c_3, c_4 \) in a quad-system.

As shown in Fig. 1, we can view a BR as a propagation rule across distinct compartments of knowledge, divided as contexts. For any BR of the form (1), each context in the set \( \{c_1,\ldots,c_m\} \) is said to depend on the set of contexts \( \{c_1,\ldots,c_n\} \). In a quad-system \( QS_c = \langle Q_c, R \rangle \), for any \( r \in R \), of the form (1), any context whose identifier is in the set \( \{ c | c: (s, p, o) \in \text{head}(r), s \text{ or } p \text{ or } o \text{ is an existentially quantified variable} \} \), is called a triple generating context (TGC). One can analyze the set of BRs in a quad-system \( QS_c \).
using a context dependency graph, which is a directed graph, whose nodes are context identifiers in C, s.t. the nodes corresponding to TGCs are marked with a *, and whose edges are constructed as follows: for each BR of the form (1), there exists an edge from each $c_i$ to $c'_j$, for $i = 1, ..., n, j = 1, ..., m$. A quad-system is said to be context acyclic, iff its context dependency graph does not contain cycles involving TGCs.

**Example 1.** Consider a quad-system, whose set of BRs $R$ are:

$$c_1: (x_1, x_2, U_1) \rightarrow \exists y_1 \ c_2: (x_1, x_2, y_1), c_3: (x_2, \text{rdf:type}, \text{rdf:Property})$$

$$c_2: (x_1, x_2, z_1) \rightarrow c_1: (x_1, x_2, U_1)$$

$$c_3: (x_1, x_2, x_3) \rightarrow c_1: (x_1, x_2, x_3)$$

where $U_1$ be a URI, whose corresponding dependency graph is shown in Fig. 2. Note that the node corresponding to the triple generating context $c_2$ is marked with a ‘*’ symbol. Since the cycle $(c_1, c_2, c_1)$ in the quad-system contains $c_2$ which is a TGC, the quad-system is not context acyclic.

In a context acyclic quad-system $QS_C$, since there exists no cyclic path through any TGC node in the context dependency graph, there exists a set of TGCs $C^{'} \subseteq C$ s.t. for any $c \in C^{'}$, there exists no incoming path\(^5\) from a TGC to $c$. We call such TGCs, level-1 TGCs. In other words, a TGC $c$ is a level-1 TGC, if for any $c' \in C$, there exists an incoming path from $c'$ to $c$, implies $c'$ is not a TGC. For $l \geq 1$, a level-$l$ TGC $c$ is a TGC that has an incoming path from a level-$l$ TGC, and for any incoming path from a level-$l'$ TGC to $c$, is s.t. $l' \leq l$. Extending the notion of level also to the non-TGCs, we say that any non-TGC that does not have any incoming paths from a TGC is at level-0; we say that any non-TGC $c \in C$ is at level-$l$, if there exists an incoming path from a level-$l$ TGC to $c$, and for any incoming path from a level-$l'$ TGC to $c$, is s.t. $l' \leq l$. Hence, the set of contexts in a context acyclic quad-system can be partitioned using the above notion of levels.

**Definition 4.** For a quad-system $QS_C$, a context $c \in C$ is said to be saturated in an iteration $i$, iff for any quad of the form $c: (s, p, o), c: (s, p, o) \in d\text{Chase}(QS_C)$ implies $c: (s, p, o) \in d\text{Chase}_i(QS_C)$.

Intuitively, context $c$ is saturated in the dChase iteration $i$, if no new quad of the form $c: (s, p, o)$ will be generated in any $d\text{Chase}_k(QS_C)$, for any $k > i$. The following lemma gives the relation between the saturation of a context and the required number of dChase iterations, for a context acyclic quad-system.

**Lemma 2.** For any context acyclic quad-system, the following holds: (i) any level-0 context is saturated before the first generating iteration, (ii) any level-1 TGC is saturated after the first generating iteration, (iii) any level-$k$ context is saturated before the $k + 1$th generating iteration.

\(^5\) assume that paths have at least one edge
Proof. Let $QS_C = \langle Q_C, R \rangle$ be the quad-system, whose first generating iteration is $i$.

(i) for any level-0 context $c$, any BR $r \in R$, and any quad-pattern of the form $c: (s, p, o)$, if $c: (s, p, o) \in head(r)$, then for any $c'$ s.t. $c': (s', p', o')$ occurs in $body(r)$ implies that $c'$ is a level-0 context and $r$ is a non-generating BR. Also, since $c'$ is a level-0 context, the same applies to $c'$. Hence, it turns out that only non-generating BRs can bring triples to any level-0 context. Since at the end of iteration $i - 1$, $dChase_{i-1}(QS_C)$ is closed w.r.t. the set of non-generating BRs (otherwise, by construction of $dChase$, $i$ would not be a generating iteration). This implies that $c$ is saturated before the first generating iteration $i$.

(ii) for any level-1 TGC $c$, any BR $r \in R$, and any quad-pattern $c: (s, p, o)$, if $c: (s, p, o) \in head(r)$, then for any $c'$ s.t. $c': (s', p', o')$ occurs in $body(r)$ implies that $c'$ is a level-0 context (Otherwise level of $c$ would be greater than 1). This means that only contexts from which triples get propagated to $c$ are level-0 contexts. From (i) we know that all the level-0 contexts are saturated before $i$th iteration, and since during the $i$th iteration $R_F$ is applied followed by the $\text{lclosure}()$ operation ($R_I$ need not be applied, since $dChase_{i-1}(QS_C)$ is closed w.r.t. $R_I$), $c$ is saturated after iteration $i$, the 1st generating iteration.

(iii) can be obtained from generalization of (i) and (ii), and from the fact that any level-$k$ context can only have incoming paths from contexts whose levels are less than or equal to $k$. □

Example 2. Consider the dependency graph in Fig. 3a, where .. indicates part of the graph that is not under the scope of our discussion. The TGCs nodes $c_1$ and $c_3$ are marked with a *. It can be seen that both $c_2$ and $c_4$ are level-0 contexts, since they do not have any incoming paths from TGCs. Since the only incoming paths to context $c_1$ are from $c_2$ and $c_4$, which are not TGCs, $c_1$ is a level-1 TGC. Context $c_3$ is a level-2 TGC, since it has an incoming path from the level-1 TGC $c_1$, and has no incoming path.
from a TGC whose level is greater than 1. Since the level-0 contexts only have incoming
paths from level-0 contexts and only appear on the head part of non-generating BRs,
before first generating iteration, all the level-0 TGCs becomes saturated, as the set of
non-generating BRs \( R_1 \) has been exhaustively applied. This situation is reflected in
Fig. 3b, where the saturated nodes are shaded with gray. Note that after the first and
second generating iterations \( c_1 \) and \( c_3 \) also become saturated, respectively.

The following lemma shows that for context acyclic quad-systems, there exists a finite
bound on the size and computation time of its dChase.

**Lemma 3.** For any context acyclic quad-system \( QSC = (Q_C, R) \), the following holds:
(i) the number of dChase iterations is finite, (ii) size of the dChase \( |d\text{Chase}(QSC)| \) = \( O(2^{2|QSC|}) \), (iii) computing \( d\text{Chase}(QSC) \) is in 2EXPTIME, (iv) if \( R \) and the set of schema triples in \( Q_C \) is fixed, then \(|d\text{Chase}(QSC)|\) is a polynomial in \(|QSC|\), and computing \( d\text{Chase}(QSC) \) is in PTIME.

**Proof.** (i) Since \( QSC \) is context-acyclic, all the contexts can be partitioned according
to their levels. Also, the number of levels \( k \) is s.t. \( k \leq |C| \). Hence, applying lemma 1,
before the \( k + 1 \)th generating iteration all the contexts becomes saturated, and \( k + 1 \)th
generating iteration do not produce any new quads, terminating the dChase computation
process.

(ii) In the dChase computation process, since by lemma 1, any generating iteration and a sequence of non-generating iterations can only increase the dChase size exponentially in \(|R|\), the size of the dChase before \( k + 1 \)th generating iteration is \( O(|d\text{Chase}_0(QSC)||R|^k) \), which can be written as \( O(|QSC||R|^k) \) (\( \dagger \)). As seen in (i),
there can only be \(|C|\) generating iterations, and a sequence of non-generating iterations.
Hence, applying \( k = |C| \) to (\( \dagger \)), and taking into account the fact that \(|C| \leq |QSC|\), the size of the dChase \( |d\text{Chase}(QSC)| \) = \( O(2^{2|QSC|}) \).

(iii) Since in any dChase iteration except the final one, atleast one new quad should
be produced and the final dChase can have at most \( O(2^{2|QSC|}) \) quads (by ii), the total number of iterations are bounded by \( O(2^{2|QSC|}) \) (\( \dagger \)). Since from lemma 1, we know
that for any iteration \( i \), computing \( d\text{Chase}_i(QSC) \) is of the order \( O(|d\text{Chase}_{i-1}(QSC)| ||R||) \). Since, \(|d\text{Chase}_{i-1}(QSC)|\) can at most be \( O(2^{2|QSC|}) \), computing \( d\text{Chase}_i(QSC) \) is of the order \( O(2^{2|QSC|}) \). Also since \(|R|| \leq |QSC|\), any iteration requires
\( O(2^{2|QSC|}) \) time (\( \dagger \)). From (\( \dagger \)) and (\( \dagger \)), we can conclude that the time required for computing dChase is in 2EXPTIME.

(iv) In (ii) we saw that the size of the dChase before \( k + 1 \)th generating iteration is
given by \( O(|QSC||R|^k) \) (\( \circ \)). Since by hypothesis \(|R|| \) is a constant and also the
size of the dependency graph and the levels in it. Hence, the expression \(|R|^k \) in (\( \circ \))
amounts to a constant \( z \). Hence, \( |d\text{Chase}(QSC)| = O(|QSC|^z) \). Hence, the size of
dChase \( |QSC| \) is a polynomial in \(|QSC|\).

Also, since in any dChase iteration except the final one, atleast one quad should be
produced and the final dChase can have at most \( O(|QSC|^z) \) quads, the total number
of iterations are bounded by \( O(|QSC|^z) \) (\( \dagger \)). Also from lemma 1, we know
that any dChase iteration \( i \), computing \( d\text{Chase}_i(QSC) \) involves two steps: (a) computing \( R(d\text{Chase}_{i-1}(QSC)) \), and (b) computing \( \text{lclosure()} \), which can be done in
PTIME in the size of its input. Since computing \( R(dChase_{i−1}(QS_C)) \) is of the order \( O(||dChase_{i−1}(QS_C)||||R||) \), where \(|R|\) is a constant and \( ||dChase_{i−1}(QS_C)|| \) is a polynomial is \( ||QS_C|| \), each iteration can be done in time polynomial in \( ||QS_C|| \). From \((i)\) and \((\dagger)\), it can be concluded that \( dChase \) can be computed in \( \text{PTIME} \).

**Lemma 4.** For any context acyclic quad-system, the following holds: (i) data complexity of CCQ entailment is in \( \text{PTIME} \) (ii) combined complexity of CCQ entailment is in \( 2\text{EXPTIME} \).

**Proof.** For a context acyclic quad-system \( QS_C = \langle QC, R \rangle \), since \( dChase(QS_C) \) is finite, a boolean CCQ \( CQ() \) can naively be evaluated by grounding the set of constants in the chase to the variables in the \( CQ() \), and then checking if any of these groundings are contained in \( dChase(QS_C) \). The number of such groundings can at most be \( ||dChase(QS_C)||||CQ()|| \).

(i) Since for data complexity, the size of the BRs \( ||R|| \), the set of schema triples, and \( ||CQ()|| \) is fixed to constant. From lemma 3 (iv), we know that under the above mentioned settings the \( dChase \) can be computed in \( \text{PTIME} \) and is polynomial in the size of \( QS_C \). Since \( ||CQ()|| \) is fixed to a constant, and from \((\dagger)\), binding the set of constants in \( dChase(QS_C) \) on \( CQ() \) still gives a number of bindings that is worst case polynomial in the size of \( QS_C \). Since membership of these bindings can be checked in the polynomially sized \( dChase \) in \( \text{PTIME} \), the time required for CCQ evaluation is in \( \text{PTIME} \).

(ii) Since in this case \( ||dChase(QS_C)|| = O(2^{||QS_C||}) \), from \((\dagger)\) and \((\ddagger)\), binding the set of constants in \( dChase(QS_C) \) to variables in \( CQ() \) amounts to \( O(2^{||CQ()||+2^{||QS_C||}}) \) bindings. Since the size of \( dChase \) is double exponential in \( ||QS_C|| \), checking the membership of each of these bindings can be done in \( 2\text{EXPTIME} \). Hence, the combined complexity is in \( 2\text{EXPTIME} \). \( \square \)

**Theorem 2.** For any context acyclic quad-system, the following holds: (i) The data complexity of CCQ entailment is \( \text{PTIME} \)-complete, (ii) The combined complexity of CCQ entailment is \( 2\text{EXPTIME} \)-complete.

For \( \text{PTIME} \)-hardness of data complexity, it can be shown that the well known problem of 3HornSat, the satisfiability of propositional Horn formulas with atmost 3 literals, and for \( 2\text{EXPTIME} \)-hardness for the combined complexity, it can be shown that the word problem of a double exponentially time bounded deterministic turing machine, which is a well known \( 2\text{EXPTIME} \)-hard problem, is reducible to the CCQ entailment problem (see [23] for detailed proof).

Reconsidering the quad-system in example 1, which is not context acyclic. Suppose that the contexts are enabled with RDFS inferencing, i.e \( \text{lclosure()} = \text{rdfsclousure()} \). During \( dChase \) construction, since any application of rule (5) can only create a triple in \( c_2 \) in which the skolem blank node is in the object position, where as the application of rule (6), does not propagate constants in object position to \( c_1 \). Although at a first look, the \( dChase \) might seem to terminate, but since the application of the following RDFS inference rule in \( c_2: (s, p, o) \rightarrow (o, \text{rdf:type}, \text{rdfs:Resource}) \), derives a quad of the form \( c_2: (\cdot, \cdot, \text{rdf:type}, \text{rdfs:Resource}) \), where \( \cdot, \cdot \) is the skolem blank-node created by the application of rule (5). Now by application of rule (6) leads
to $c_1 : (b, \text{rdf:}\text{type}, U_1)$. Since rule (5) is applicable on $c_1 : (b, \text{rdf:}\text{type}, U_1)$, which again brings a new skolem blank node to $c_2$, and hence the dChase construction doesn’t terminate. Hence, as seen above the notion of context acyclicity can alarm us about such infinite cases.

5 Related Work

Contexts and Distributed Logics
The work on contexts began in the 80s when McCarthy [21] proposed context as a solution to the generality problem in AI. After this various studies about logics of contexts mainly in the field of KR was done by Guha [29], Distributed First Order Logics by Ghidini et al. [14] and Local Model Semantics by Giunchiglia et al. [15]. Primarily in these works contexts are formalized as a first order/propositional theories, and bridge rules were provided to inter-operate the various contexts. Some of the initial works on contexts relevant to semantic web were the ones like Distributed Description Logics [4] by Borgida et al., E-connections [26] by Kutzi et al., Context-OWL [5] by Bouquet et al., and the work of CKR [31, 24] by Serafini et al. These were mainly logics based on DLs, which formalized contexts as OWL KBs, whose semantics is given using a distributed interpretation structure with additional semantic conditions that suits varying requirements. Compared to these works, the bridge rules we consider are much more expressive with conjunctions and existential variables that supports value/blank-node creation.

$\forall\exists$ rules, TGDs, Datalog++ rules
Query answering over rules with universal existential quantifiers in the context of databases/KR, where these rules are called tuple generating dependencies (TGDs)/Datalog++ rules, was done by Beeri and Vardi [3] even in the early 80s, where the authors show that the query entailment problem in general is undecidable. However, recently many classes of such rules have been identified for which query answering is decidable. These includes (a) fragments s.t. the resulting models have bounded tree widths, called bounded treewidth sets (BTS), such as Weakly guarded rules [7], Frontier guarded rules [2], (b) fragments called finite unification sets (FUS), such as ‘sticky’ rules [6, 17], and (c) fragments called finite extension sets (FES), where sufficient conditions are enforced to ensure finiteness of the chase and its termination. The approach used for query answering in FUS is to rewrite the input query w.r.t. to the TGDs to another query that can be evaluated directly on the set of instances, s.t. the answers for the former query and latter query coincides. The approach is called the query rewriting approach. FES classes uses certain termination guarantying tests that check whether certain sufficient conditions are satisfied by the structure of TGDs. A large number of classes in FES are based on tests that detects ‘acyclicity conditions’ by analyzing the information flow between the TGD rules. Weak acyclicity [13, 11], was one of the first such notions, and was extended to joint acyclicity [25], super weak acyclicity [27], and model faithful acyclicity [9]. The most similar approach to ours is the weak acyclicity technique, where the structure of the rules is analyzed using a dependency graph that models the propagation of constants across various predicates positions, and restricting the dependency graph to be acyclic. Although this technique can be used in our scenario by translating a quad-system to a set of TGDs; if the obtained translation is weakly acyclic, then one could use existing algorithms for chase computation for the TGDs to compute the chase, the query entailment check can
be done by querying the obtained chase. However, our approach has the advantage of straightforward implementability on existing quad-stores.

### 6 Summary and Conclusion

In this paper, we study the problem of query answering over contextualized RDF knowledge. We show that the problem in general is undecidable, and present a decidable class called context acyclic quad-systems. Table 1 summarizes the main results obtained. We can show that the notion of context acyclicity, introduced in section 4 can be used to extend the currently established tools for contextual reasoning to give support for expressive BRs with conjunction and existentials with decidability guarantees. We view the results obtained in this paper as a general foundation for contextual reasoning and query answering over contextualized RDF knowledge formats such as Quads, and can straightforwardly be used to extend existing Quad stores to incorporate for-all existentials of the form (1).

### References


Revising Description Logic Terminologies to Handle Exceptions: a First Step

Roberto Micalizio, Gian Luca Pozzato

Dipartimento di Informatica - Università degli Studi di Torino
roberto.micalizio@unito.it, gianluca.pozzato@unito.it

Abstract. We propose a methodology to revise a Description Logic knowledge base when detecting exceptions. Our approach relies on the methodology for debugging a Description Logic terminology, addressing the problem of diagnosing incoherent ontologies by identifying a minimal subset of axioms responsible for an inconsistency. In the approach we propose, once the source of the inconsistency has been localized, the identified axioms are revised in order to obtain a consistent knowledge base including the detected exception. To this aim, we make use of a non-monotonic extension of the Description Logic \( \mathcal{ALC} \) based on the combination of a typicality operator and the well established nonmonotonic mechanism of rational closure, which allows to deal with prototypical properties and defeasible inheritance.

1 Introduction

Exceptions do exist. Intuitively, we can define an exception as an individual (or a group of individuals) that has, or has not, a property in contrast with other individuals of the same class. For instance, a penguin can be considered as an exceptional bird since, although equipped with wings, is unable to fly. We can find numerous examples of exceptions in nature, but also in natural languages (e.g., irregular verbs), medicine (e.g. situs inversus is an anomaly in which the major visceral organs are reversed or mirrored from their normal positions, so that the heart is positioned in the right-hand side of the chest), and many other real-world scenarios.

Dealing with exceptions can be tricky for an ontology engineer. All the exceptions should be known in advance by the ontology engineer; i.e., when the ontology is being designed. Unfortunately, in many real-world scenarios, exceptions are discovered just while the system is running, and the ontology is actually used. Whenever exceptions are discovered at runtime, the resulting ontology becomes incoherent (i.e., at least one concept is mapped to an empty set of individuals). Some recent works [16, 17, 15, 8, 9, 13] have addressed the problem of diagnosing incoherent ontologies by identifying a minimal subset of axioms responsible for the inconsistency. The idea of these works is that once the source of the inconsistency has been localized, the ontology engineer can intervene and revise the identified axioms in order to restore the consistency. These approaches presuppose that the ontology has become incoherent due to the introduction of errors; as for instance when two ontologies are merged together. It is worth noting that, albeit an exception has the same effect of an error (i.e., it causes an ontology to become incoherent), an exception is not an error. An exception is
rather a piece of knowledge that partially contradicts what is so far known about
a portion of the ontology at hand. Thus, on the one hand, ignoring exceptions
would be deleterious as the resulting ontology would not reflect the applicative
domain correctly. On the other hand, accommodating exceptions requires the
exploitation of some form of defeasible reasoning that allows us to revise some
of concepts in the ontology.

In this paper we propose a methodology to revise a Description Logic (for
short: DL) knowledge base when detecting exceptions. Our approach relies on the
above mentioned methodology of [16, 17, 15] for detecting exceptions by identify-
ing a minimal subset of axioms responsible for an inconsistency. Once the source
of the inconsistency has been localized, the identified axioms are revised in order
to obtain a consistent knowledge base including the detected exception. To this
aim, we use a nonmonotonic extension of the DL $\mathcal{ALC}$ recently presented by
Giordano and colleagues in [7]. This extension is based on the introduction of a
typicality operator $T$ in order to express typical inclusions. The intuitive idea is
to allow concepts of the form $T(C)$, whose intuitive meaning is that $T(C)$ selects
the typical instances of a concept $C$. It is therefore possible to distinguish between
properties holding for all instances of a concept $C$ ($C \sqsubseteq D$), and those only holding
for the typical instances of $C$ ($T(C) \sqsubseteq D$). For instance, a knowledge base
can consistently express that birds normally fly ($T(Bird) \sqsubseteq Fly$), but penguins
are exceptional birds that do not fly ($Penguin \sqsubseteq Bird$ and $Penguin \sqsubseteq \neg Fly$).

The $T$ operator is intended to enjoy the well-established properties of rational
logic, introduced by Lehmann and Magidor in [12] for propositional logic. In order
to reason about prototypical properties and defeasible inheritance, the semantics
of this nonmonotonic DL, called $\mathcal{ALC}_{max}^R$, is based on rational models and
exploits a minimal models mechanism based on the minimization of the rank of
domain elements. This semantics corresponds to a natural extension to DLs of
Lehmann and Magidor’s notion of rational closure [12].

The paper is organized as follows: in Section 2 we recall extensions of DLs for
prototypical reasoning, focusing on the approach based on a typicality operator
$T$ and rational closure [7]; in Section 3 we recall and extend the notions intro-
duced in [17, 13] for diagnosing incoherent ontologies; in Section 4 we present
our methodology for revising a DL terminology to handle exceptions by means
of $T$; we conclude with some pointers to future issues in Section 5.

2 Description Logics and Exceptions

The family of Description Logics [1] is one of the most important formalisms of
knowledge representation. DLs are reminiscent of the early semantic networks
and of frame-based systems. They offer two key advantages: (i) a well-defined
semantics based on first-order logic and (ii) a good trade-off between expressiv-
ity and computational complexity. DLs have been successfully implemented by a
range of systems, and are at the base of languages for the Semantic Web such as
OWL. In a DL framework, a knowledge base (KB) comprises two components:
(i) a TBox, containing the definition of concepts (and possibly roles) and a spec-
ification of inclusion relations among them; (ii) an ABox, containing instances
of concepts and roles, in other words, properties and relations of individuals. Since the primary objective of the TBox is to build a taxonomy of concepts, the need of representing prototypical properties and of reasoning about defeasible inheritance of such properties easily arises.

In the recent years, a large amount of work has been done in order to extend the basic formalism of DLs with nonmonotonic reasoning features. The traditional approach is to handle defeasible inheritance by integrating some kind of nonmonotonic reasoning mechanisms [5, 2, 10, 3, 4]. A simple but powerful nonmonotonic extension of DLs is proposed in [6]. In this approach, “typical” or “normal” properties can be directly specified by means of a “typicality” operator $T$ enriching the underlying DL; the typicality operator $T$ is essentially characterized by the core properties of nonmonotonic reasoning, axiomatized by preferential logic $P$ in [11].

In this work we refer to the most recent approach proposed in [7], where the authors extend $ALC$ with $T$ by considering rational closure as defined by Lehman and Magidor [12] for propositional logic. Here the $T$ operator is intended to enjoy the well-established properties of rational logic $R$. Even if $T$ is a nonmonotonic operator (so that for instance $T(Bird) \subseteq Fly$ does not entail $T(Bird \cap Penguin) \subseteq Fly$), the logic itself is monotonic. Indeed, in this logic it is not possible to monotonically infer from $T(Bird) \subseteq Fly$, in the absence of information to the contrary, that also $T(Bird \cap Black) \subseteq Fly$. Nor it can be nonmonotonically inferred from $Bird(tweety)$, in the absence of information to the contrary, that $T(Bird)(tweety)$. Nonmonotonicity is achieved by adapting to $ALC$ with $T$ the propositional construction of rational closure. This nonmonotonic extension allows to infer typical subsumptions from the TBox. Intuitively, and similarly to the propositional case, the rational closure construction amounts to assigning a rank (a level of exceptionality) to every concept; this rank is used to evaluate typical inclusions of the form $T(C) \subseteq D$: the inclusion is supported by the rational closure whenever the rank of $C$ is strictly smaller than the rank of $C \cap \neg D$. From a semantic point of view, nonmonotonicity is achieved by defining, on the top of $ALC$ with typicality, a minimal model semantics where the notion of minimality is based on the minimization of the ranks of the domain elements. The problem of extending rational closure to $ABox$ reasoning is also taken into account: in order to ascribe typical properties to individuals, the typicality of an individual is maximized. This is done by minimizing its rank (that is, its level of exceptionality). Let us recall the resulting extension $ALC^{R_{\min}T}$ in detail.

**Definition 1.** We consider an alphabet of concept names $C$, of role names $R$, and of individual constants $O$. Given $A \in C$ and $R \in R$, we define:

- $C_R := A | \top | \bot | \neg C_R | C_R \cap C_R | C_R \cup C_R | \forall R.C_R | \exists R.C_R$
- $C_L := C_R | T(C_R)$

A knowledge base is a pair $(TBox, ABox)$. $TBox$ contains a finite set of concept inclusions $C_L \subseteq C_R$. $ABox$ contains assertions of the form $C_L(a)$ and $R(a,b)$, where $a, b \in O$.

We define the semantics of the monotonic $ALC + T_R$, formulated in terms of rational models: ordinary models of $ALC$ are equipped with a preference relation
Definition 2 ([7]). A model $\mathcal{M}$ of $\mathcal{ALC} + \mathcal{T}_R$ is any structure $\langle \Delta, <, I \rangle$ where: $\Delta$ is the domain; $<$ is an irreflexive, transitive and modular (if $x < y$ then either $x < z$ or $z < y$) relation over $\Delta$; $I$ is the extension function that maps each concept $C$ to $C^I \subseteq \Delta$, and each role $R$ to $R^I \subseteq \Delta \times \Delta$ as follows: $\top^I = \Delta$, $\bot^I = \emptyset$, $\neg C^I = \Delta \setminus C^I$, $(C \cap D)^I = C^I \cap D^I$, $(C \cup D)^I = C^I \cup D^I$, $(\exists R. C)^I = \{x \in \Delta \mid \exists y. (x, y) \in R^I \}$, $(\forall R. C)^I = \{x \in \Delta \mid \forall y. (x, y) \in R^I \}$, $(\exists R.C)^I = \{x \in \Delta \mid \exists y. (x, y) \in R^I \}$ and $y \in C^I \}$, $(\forall R.C)^I = \{u : u \in \Delta \}$. Furthermore, $<$ satisfies the Well–Foundedness Condition, i.e., for all $S \subseteq \Delta$, for all $x \in S$, either $x \in \text{Min}_<(S)$ or $\exists y \in \text{Min}_<(S)$ such that $y < x$.

Given an $\mathcal{ALC} + \mathcal{T}_R$ model $\mathcal{M} = (\Delta, <, I)$, we say that (i) $\mathcal{M}$ satisfies an inclusion $C \sqsubseteq D$ if it holds $C^I \subseteq D^I$, (ii) $\mathcal{M}$ satisfies an assertion $C(a)$ if $a^I \in C^I$ and (iii) $\mathcal{M}$ satisfies an assertion $R(a, b)$ if $(a^I, b^I) \in R^I$. Given $K = (\text{TBox}, \text{ABox})$, we say that $\mathcal{M}$ satisfies TBox if $\mathcal{M}$ satisfies all inclusions in TBox, $\mathcal{M}$ satisfies ABox if $\mathcal{M}$ satisfies all assertions in ABox and $\mathcal{M}$ satisfies $K$ if it satisfies both its TBox and its ABox.

Given a knowledge base $K$, an inclusion $C_L \sqsubseteq C_R$ and an assertion $C_L(a)$, with $a \in \mathcal{O}$, we say that the inclusion $C_L \sqsubseteq C_R$ is derivable from $K$, written $K \models_{\mathcal{ALC}+\mathcal{T}_R} C_L \subseteq C_R$, if $C_L^I \subseteq C_R^I$ holds in all models $\mathcal{M} = (\Delta, <, I)$ satisfying $K$. Moreover, we say the assertion $C_L(a)$ is derivable from $K$, written $K \models_{\mathcal{ALC}+\mathcal{T}_R} C_L(a)$, if $a^I \in C_L^I$ holds in all models $\mathcal{M} = (\Delta, <, I)$ satisfying $K$.

As already mentioned, although the typicality operator $\top$ itself is nonmonotonic (i.e. $\top(C \sqsubseteq D)$ does not imply $\top(C \sqcap E \subseteq D)$), the logic $\mathcal{ALC} + \mathcal{T}_R$ is monotonic: what is inferred from $K$ can still be inferred from any $K'$ with $K \subseteq K'$. This is a clear limitation in DLs. As a consequence of the monotonicity of $\mathcal{ALC} + \mathcal{T}_R$, one cannot deal with irrelevance, for instance. So, from the knowledge base of birds and penguins, one cannot derive that $K \models_{\mathcal{ALC}+\mathcal{T}_R} \mathcal{T}(\text{Penguin} \sqcap \text{Black}) \subseteq \neg \text{Fly}$, even if the property of being black is irrelevant with respect to flying. In the same way, if we added to $K$ the information that Tweety is a bird (Bird(tweety)), in $\mathcal{ALC} + \mathcal{T}_R$ one cannot tentatively derive, in the absence of information to the contrary, that $\mathcal{T}(\text{Bird}(\text{tweety}))$ and $\text{Fly}(\text{tweety})$.

In order to tackle this problem, in [7] the definition of rational closure introduced by Lehmann and Magidor [12] for the propositional case has been extended to the DL $\mathcal{ALC} + \mathcal{T}_R$. Due to space limitations, we omit all definitions of the rational closure of a DL knowledge base, reminding to [7].

From a semantic point of view, in [7] it is shown that minimal rational models that minimize the rank of domain elements can be used to give a semantical reconstruction of this extension of rational closure. The rank $k_M$ of a domain element $x$ is the length of the longest chain $x_0 < \cdots < x$ from $x$ to a minimal $x_0$ (i.e. such that there is no $x'$ such that $x' < x_0$). The idea is as follows: given two
models of $K$, one in which a given domain element $x$ has rank $x_1$ and another in which it has rank $x_2$, with $x_1 > x_2$, then the latter is preferred, as in this model the element $x$ is “more normal” than in the former.

Given a knowledge base $K=(\text{TBox},\text{ABox})$, in [7] it is shown that an inclusion $C \subseteq D$ (respectively, an assertion $C(a)$) belongs to the rational closure of $K$ if and only if $C \subseteq D$ (resp., $C(a)$) holds in all minimal models of $K$ of a “special” kind, named canonical models. The rational closure construction for $\mathcal{ALC}$ is inexpensive, since it retains the same complexity of the underlying logic, and thus a good candidate to define effective nonmonotonic extensions of DLs. More precisely, the problem of deciding whether a typical inclusion belongs to the rational closure of the TBox is in ExpTime as well as the problem of deciding whether an assertion $C(a)$ belongs to the rational closure over the ABox.

3 Explaining Incoherent Terminologies

In this paper, we propose a methodology to deal with exceptions that builds up on the methodology for debugging a DL terminology proposed by Schlobach et al. since their seminal work [17].

Let us first introduce the notion of incoherent terminology. Given a TBox $\mathcal{T}$, we say that it is coherent if there is no unsatisfiable concept, in other words there is at least a model of $\mathcal{T}$ in which the extensions of all concepts are not empty.

To explain incoherences in terminologies, Schlobach et al. propose a methodology based on two steps: first, axiom pinpointing excludes axioms which are irrelevant to the incoherence; second, concept pinpointing provides a simplified definition highlighting the exact position of a contradiction within the axioms previously selected. In this paper we are interested in the axiom pinpointing step, which identifies debugging-relevant axioms. Intuitively, an axiom is relevant for debugging if, when removed, a TBox becomes coherent, or at least one previously unsatisfiable concept turns satisfiable. The notion of subset of relevant axioms is captured by the following definition.

Definition 3 (MUPS, Definition 3.1 [17]). Let $C$ be a concept which is unsatisfiable in a TBox $\mathcal{T}$. A set $\mathcal{T}' \subseteq \mathcal{T}$ is a minimal unsatisfiability-preserving sub-TBox (MUPS) of $\mathcal{T}$ if $C$ is unsatisfiable in $\mathcal{T}'$, and $C$ is satisfiable in every sub-TBox $\mathcal{T}'' \subset \mathcal{T}'$.

In the following, $\text{mups}(\mathcal{T},C)$ is used to denote the set of MUPS for a given terminology $\mathcal{T}$ and a concept $C$. Intuitively, each set of axioms in $\text{mups}(\mathcal{T},C)$ represents a conflict set; i.e., a set of axioms that cannot all be satisfied. From this point of view, it is therefore possible to infer a diagnosis for the concept $C$ by applying the Hitting-Set tree algorithm proposed by Reiter [14]. However, the set $\text{mups}(\mathcal{T},C)$ is sufficient for our purpose of dealing with exceptions.

A drawback of the debugging approach in [16,17] is that it is restricted to unfoldable TBoxes, only containing unique, acyclic definitions. An axiom is called a definition of $A$ if it is of the form $A \subseteq C$, where $A \in C$ is an atomic concept. An
axiom \( A \sqsubseteq C \) is unique if the KB contains no other definition of \( A \). An axiom is acyclic if \( C \) does not refer either directly or indirectly (via other axioms) to \( A \) [1]. This restriction is too strong for our objective of representing and reasoning about defeasible inheritance in a natural way. As an example, the TBox expressing that students are not tax payers, but working students do pay taxes, can be naturally expressed by the following, not unfoldable, TBox: \( \{ \text{Student} \sqsubseteq \neg \text{TaxPayer}, \text{Student} \sqcap \text{Worker} \sqsubseteq \text{TaxPayer} \} \). In order to overwhelm this gap, in [13,8] axiom pinpointing is extended to general TBoxes, more suitable to our purposes. As a further difference, Schlobach and colleagues limit their attention to the basic \( \mathcal{ALC} \), whereas Parsia and colleagues in [8] are able to deal also with the more expressive \( \mathcal{SHOIN} \), corresponding to OWL-DL. A set of algorithms for computing axiom pinpointing, in particular to compute the set of MUPS for a given terminology \( T \) and a concept \( A \), is also provided.

In [16], Schlobach also proposes a methodology for explaining concept subsumptions. The idea is to reduce the structural complexity of the original concepts in order to highlight the logical interplay between them. To this aim, Schlobach proposes to exploit the structural similarity of concepts, that can be used to simplify terminological concepts, and hence the subsumption relations.

The structural similarity is based on the notion of qualified subconcepts; namely, variants of those concepts a knowledge engineer explicitly uses in the modeling process, and where the context (i.e., sequence of quantifiers and number of negations) of this use is kept intact. Schlobach specifies the notion of qualified subconcepts in two ways: Generalized Qualified Subconcepts (\( \text{gqs} \)), and Specialized Qualified Subconcepts (\( \text{sqs} \)) which are defined by induction as follows:

**Definition 4 (Generalized and Specialized Qualified Subconcepts, [16]).**

Given concepts \( A, C \) and \( D \), we define:

\[
\begin{align*}
gqs(A) &= \text{sqs}(A) = \{A\} \quad \text{if } A \text{ is atomic} \\
gqs(C \cap D) &= \{C', D', C' \cap D' | C' \in gqs(C), D' \in gqs(D)\} \\
gqs(C \sqcup D) &= \{C' \sqcup D' | C' \in gqs(C), D' \in gqs(D)\} \\
gqs(\exists r.C) &= \{\exists r.C' | C' \in gqs(C)\} \\
gqs(\forall r.C) &= \{\forall r.C' | C' \in gqs(C)\} \\
gqs(\neg C) &= \{\neg C' | C' \in \text{sqs}(C)\} \\
gqs(C \cap D) &= \{C' \cap D' | C' \in \text{sqs}(C), D' \in \text{sqs}(D)\} \\
gqs(C \sqcup D) &= \{C', D', C' \sqcup D' | C' \in \text{sqs}(C), D' \in \text{sqs}(D)\} \\
gqs(\exists r.C) &= \{\exists r.C' | C' \in \text{sqs}(C)\} \\
gqs(\forall r.C) &= \{\forall r.C' | C' \in \text{sqs}(C)\} \\
gqs(\neg C) &= \{\neg C' | C' \in \text{gqs}(C)\}
\end{align*}
\]

As Schlobach himself notes, a simple consequence of this definition is that \( \models C \sqsubseteq C' \) for every \( C' \in \text{gqs}(C) \), and \( \models D' \sqsubseteq D \) for each \( D' \in \text{sqs}(D) \).

We slightly extend the base case of Definition 4 as follows:

**Definition 5 (Generalized and Specialized Qualified Subconcepts extended).** We define \( \text{sqs}(C) \) and \( \text{gqs}(C) \) by adding to clauses in Definition 4 the following ones:
Thus, we also take into account the axioms (i.e., concept inclusions) defined in a given TBox. This generalization allows us to move upward (by means of gqs), and downward (by means of sqs) in the hierarchy of concepts defined by a given TBox T. Relying on the notions of sqs and gqs, we can define a partial ordering relation between concepts as follows:

**Definition 6.** Let C and D be two concepts in a given TBox T, we say that C is more specific than D, denoted as C ⊑ D, iff at least one of the following relations holds: (i) C ∈ sqs(D), or (ii) D ∈ gqs(C).

It is easy to see that ⊑ is irreflexive, antisymmetric, and transitive; however, it is just partial because ⊑ is not defined for any pair of concepts; i.e., there may exist two concepts C and D such that neither C ⊑ D nor D ⊑ C holds. As we will discuss later, the methodology we propose for determining which concepts represent properties to be made “typical” relies on the fact that concepts are considered in order from the most specific to the most general. In those situations where two concepts are not directly comparable with one another by means of ⊑, we need some further tools. For this reason, we introduce the notions of Concept Network and depth of concepts. First of all, let S be the set of concepts (and subconcepts) occurring in a given TBox.

**Definition 7 (Concept Network).** Given a TBox T, a concept network for T is a graph CN(T) : (V, E) where V is a set of vertexes, each of which corresponds to a concept C ∈ S, and E is a set of oriented edges of the form ⟨C, D⟩, where C and D belong to V; an edge ⟨C, D⟩ is in E iff C ⊑ D.

**Definition 8 (Depth of a concept).** Given a TBox T, its corresponding concept network CN(T), and a concept C ∈ S, the depth of a concept C ∈ T, denoted as depth(C), corresponds to the length of the longest path from ⊤ to C.

In principle, any two concepts C and D at the same depth can be considered by our methodology in either orderings C ⊑ D or D ⊑ C. However, when both C and ¬C are at the same depth, we use a further criteria of preference based on the distance between concepts.

**Definition 9 (Distance between concepts).** Given a TBox T, its corresponding concept network CN(T), and two concepts C, D ∈ S, the distance between C and D in T, denoted as dist(C, D) is given by the shortest path in CN(T) from C to D, if C ⊑ D; ∞, otherwise.

**Example 1.** Let us consider a simple example in which a TBox Tstudent contains the following concept definitions:

- Student ⊑ ¬ TaxPayer
- Student ⊓ Worker ⊑ TaxPayer
Of course, $T_{\text{student}}$ is consistent only if there are no working students, and in the following we will show how it can be repaired by means of the typicality operator. For the time being, we are just interested in determining the depth of the concepts in $T_{\text{student}}$. By applying Definition 6 we obtain:

$$\text{Student} \sqcap \text{Worker} \prec \text{Student} \prec \neg \text{TaxPayer}; \text{Student} \sqcap \text{Worker} \prec \text{TaxPayer}.$$  

It is easy to see that $\text{Student} \sqcap \text{Worker}$ is the deepest concept in our terminology. In fact, we have that both $\text{TaxPayer}$ and $\neg \text{TaxPayer}$ are at the same depth 1, as well as $\text{Worker}$; $\text{Student}$ is at depth 2; whereas $\text{Student} \sqcap \text{Worker}$ is at depth 3. However, since $\text{dist}(\text{Student} \sqcap \text{Worker}, \text{TaxPayer}) < \text{dist}(\text{Student} \sqcap \text{Worker}, \neg \text{TaxPayer})$, we will prefer to consider $\text{TaxPayer}$ before $\neg \text{TaxPayer}$.

Definition 5 above does not take into account that the terminology may be inconsistent, and hence some of the concepts generated via the generalization (specialization) rules might be trivially contradictory (e.g., having the form $D \sqcap \neg D$). Moreover, the set of concepts within the $gqs$ ($sqs$) of a given concept $C$ are not necessarily consistent with one another. For instance, let us consider again $T_{\text{student}}$: it is easy to see that $gqs(\text{Student} \sqcap \text{Worker})$ is the set:

$$\{\text{Student} \sqcap \text{Worker}, \text{Student}, \text{TaxPayer}, \neg \text{TaxPayer}, \neg \text{Student} \sqcap \text{Worker}, \text{Worker}, \neg \text{TaxPayer} \sqcap \text{Worker}, \neg \text{Student} \sqcap \text{Worker} \sqcap \text{Worker}, \neg \text{Worker}, \text{TaxPayer}\}.$$

Both $\text{TaxPayer}$ and $\neg \text{TaxPayer}$ are members of $gqs(\text{Student} \sqcap \text{Worker})$, but only one of them will be correct, the other will have to be pruned. As we have already mentioned above, $\text{TaxPayer}$ will be preferred as it is closer to $\text{Student} \sqcap \text{Worker}$. In the next section we propose a pruning technique that discards those concepts that, albeit generalize a concept $C$ (i.e., belong to $gqs(C)$), contradict the terminology $T$. Our pruning technique relies on a compact representation of $gqs$. That is, rather than explicitly computing all concepts in $gqs(C)$ by recursively unfolding every $gqs(C')$ for each $C' \in gqs(C)$, we consider $gqs(C)$ as a list of concepts (not necessarily atomic), and $gqss$.

For example, $gqs(\text{Student} \sqcap \text{Worker})$ can be compactly represented as the list of elements $\{\text{Student} \sqcap \text{Worker}, gqs(\text{Student}) \sqcap gqs(\text{Worker}), gqs(\text{Student}), gqs(\text{Worker})\}$. Intuitively, $gqs(\text{Student} \sqcap \text{Worker})$ is given by the union of the concepts that are directly mentioned in the list (e.g., $\text{Student} \sqcap \text{Worker}$), or that belong to one of the $gqs$ mentioned in the list itself (e.g., $gqs(\text{Student})$).

Note that we keep the list associated with $gqs(C)$ ordered according to the depth (and distance when necessary) of the mentioned concepts. Also in this case, deepest concepts come first. However we have to pay some attention in dealing with $gqs$. In fact, if $D \prec E$, then also $gqs(D) \prec E$, and hence $gqs(D) \prec gqs(E)$. However, if both $D$ and $gqs(D)$ are in $gqs(C)$, then $D \prec gqs(D)$. On the other hand, if $D$ and $F$ have the same depth, we put first the concept which is closer to $C$. In case, $D$ and $F$ are at the same distance from $C$ than their relative ordering is arbitrary. The idea is that $gqs(D)$ abstracts a set of concepts which are at least as specific as $D$, but possibly more general than $D$. Therefore, any concept more specific than $D$ is also more specific than $gqs(D)$. Vice versa, if $D \prec E$, then also $gqs(D) \prec E$ as $gqs(D)$ contains at least $D$. 

232
4 Revising Terminologies Including Exceptions

In this section we present the methodology for automatically modifying a TBox $\mathcal{T}$ whenever a detected inconsistency can be treated as an exception. The basic idea of our proposal is first to isolate a subset of $\mathcal{T}$ which is relevant for the inconsistency, and for this purpose we will adopt the notion of MUPS introduced in Definition 3. Then we reconsider all the concepts within a MUPS and try to identify which concepts describe characterizing properties to be made “typical”. The intuition is that all the properties labeled as typical hold for all “normal” members of a given class $C$, but not necessarily for all members in $C$. That is, we admit that a subset of individuals in $C$ do not present the typical properties of the class, and hence are considered as exceptional.

Let us start by considering a coherent terminology $\mathcal{T}$, and assume we discover a new subsumption relation $\text{new}C \sqsubseteq D$ that once added to $\mathcal{T}$ introduces an incoherence. As mentioned above, the first step consists in restricting our attention to only those concepts that are relevant for the incoherence. These concepts are identified by $\text{mups}(\mathcal{T}, \text{new}C)$ [18]. For the sake of discussion, we will assume that $\text{mups}(\mathcal{T}, \text{new}C)$ will only contain a set of incoherent axioms, the approach can be easily extended to consider the more general situation in which $\text{mups}(\mathcal{T}, \text{new}C)$ is a set of sets.

In principle, we have to identify which concept subsumptions in $\text{mups}(\mathcal{T}, \text{new}C)$ are to be modified by introducing the $\text{T}$ operator on the left-hand side of an axiom. The main issue in this process is that the terminology $\mathcal{T}$ might hide implicit knowledge (i.e., implicit subsumptions), and these implicit subsumptions are still hidden in $\text{mups}(\mathcal{T}, \text{new}C)$. Inferring implicit subsumptions is therefore essential in order to correctly identify the most specific properties to be made typical.

To reach this result, we propose a search process that is based on the following intuition: Given a concept $C$ in $\text{mups}(\mathcal{T}, \text{new}C)$, all the implicit subsumptions we are looking for are already contained in $\text{gqs}(C)$. In other words, for any concept $D \in \text{gqs}(C)$ (either atomic or not), the subsumption $C \sqsubseteq D$ must hold by definition of $\text{gqs}$. However, we have to be cautious when dealing with $\text{gqs}(C)$ as it potentially contains subsumptions which are irrelevant, or even incorrect, for the specific case under consideration. First of all, $\text{gqs}(C)$ considers all the possible concept definitions in $\mathcal{T}$, but we are just interested in those concepts mentioned within $\text{mups}(\mathcal{T}, \text{new}C)$. Thus, in considering the concepts mentioned within $\text{gqs}(C)$ we have to restrict ourselves only to those concepts which are also mentioned $\text{mups}(\mathcal{T}, \text{new}C)$; any other concept in $\text{gqs}(C)$ will be ignored as irrelevant. In addition, not all the generalizations suggested by $\text{gqs}(C)$ are consistent with the concepts in $\text{mups}(\mathcal{T}, \text{new}C)$. As we have already noted, $\text{mups}(\mathcal{T}, \text{new}C)$ might contain the subsumption relation $C \sqsubseteq D$; at the same time, the concept $\neg D$ appears in $\text{gqs}(C)$: implying that $C \sqsubseteq \neg D$. Of course, this second subsumption is in direct contrast with a subsumption in $\text{mups}(\mathcal{T}, \text{new}C)$; we say that $C \sqsubseteq \neg D$ syntactically contradicts $C \sqsubseteq D$. (Note that such a contradiction is just syntactic, and hence it can be checked by inspecting concepts in $\text{mups}(\mathcal{T}, \text{new}C)$.)

Another critical issue is that we are interested in finding the most specific properties that have to be considered as typical. This is also the reason why we
look for implicit subsumptions: we want to discover what properties are inherited by the concepts in $\text{mups}(T, newC)$. Thus, when we look for inclusions, we have to proceed from the most specific concepts in $\text{mups}(T, newC)$ to the most general ones. In this way, we can progressively build a new set of subsumption relations $S$, in which the typicality operator is added to only those specific properties that are relevant for dealing with the exception introduced by $newC$.

4.1 Algorithms
In this section we give a detailed description of the methodology we propose for automatically managing, by means of the typicality operator, a concept $newC$ with exceptional properties $D$. We give for granted that $\text{mups}(T, newC)$ has already been computed by means of one of the calculi proposed in [8]. Thus, $\text{mups}(T, newC)$ identifies a minimal subset of concept inclusions that are relevant for the inconsistency arising when $T$ is extended with $newC \sqsubseteq D$.

Algorithm 1 FindSubsumptionsMain($\text{mups}(T, newC)$)

1: $S \leftarrow \emptyset$
2: for all depth $l$ of concepts in $\text{mups}(T, newC)$, from the highest to the lowest do
3: for all $C_i$ in $\text{mups}(T, newC)$ such that $\text{depth}(C_i) = l$ do
4: $S_i \leftarrow \text{FindSubsumption}(C_i, gqs(C_i), S)$
5: end for
6: if $S \cup \bigcup S_i$ is coherent then
7: $S \leftarrow S \cup \bigcup S_i$
8: else
9: for all $S_i$ do
10: for all $C \sqsubseteq D \in S_i$, such that $C$ does not occur in $D$ do
11: $S \leftarrow S \cup \{T(C) \sqsubseteq D\}$
12: end for
13: end for
14: end if
15: end for
16: return $S$

FindSubsumptionsMain To restore the consistency in $T$, we propose to rewrite the inclusion relations in $\text{mups}(T, newC)$ by characterizing some of them with the typicality operator. As said above, we reach this goal by considering all concepts mentioned in $\text{mups}(T, newC)$ in depth ordering, starting from the deepest ones up to the most generic concepts. Algorithm 1 simply shows this loop over the concepts in $\text{mups}(T, newC)$. At each iteration, a set $S$ of inclusions is incrementally extended with the new relations discovered by function FindSubsumptions. In case several concepts $C_i$ have the same depth, a further consistency check among their corresponding sets $S_i$s is performed to deal with inconsistent properties, coming from different $S_i$s, and inherited by a given concept $F$. For instance, let us assume that $F \sqsubseteq C \cap D \in S$, $C$ and $D$ have the same depth and distance from $F$, thus we cannot prefer one to the other. In addition, by invoking FindSubsumptions over $C$ and $D$ we get, respectively, $C \sqsubseteq E \in S_C$.
and $D \sqsubseteq \neg E \in S_D$. Since $F$ inherits both $E$ and $\neg E$, we do not conclude anything about $F$ having, or not, property $E$; however, to build a coherent $S$, $T(C) \sqsubseteq E$ and $T(D) \sqsubseteq \neg E$ are added to $S$.

**FindSubsumptions** Algorithm 2 sketches the main steps of function *FindSubsumptions*: it takes as inputs a concept $C_i$ mentioned in $\text{mups}(T, newC)$, the corresponding $\text{gqs}(C_i)$ given as a list in compact form, and the set $S$ of inclusions found so far. The intuition is that all implicit inclusions we are looking for have the form $C_i \sqsubseteq D$, where $D \in \text{gqs}(C_i)$.

First of all, the algorithm initializes some local structures: $S_i$ will contain the subsumptions discovered in this invocation, $\text{BlackList}$ will only contain the $\text{gqs}$ of concepts that have been pruned: future occurrences of concepts belonging to these $\text{gqs}$ have to be discarded; $\text{Queue}$ will either contain concept nodes or complex nodes: a concept node represents a concept (not necessarily atomic), whereas a complex node represents a structure where at least a $\text{gqs}$ is mentioned (i.e., a complex node is an abstraction of a number of concepts). All nodes in $\text{Queue}$ are kept ordered from the most specific to the most general according to the depth of the concepts they contain. At the beginning of the algorithm (see lines 3-12), $\text{Queue}$ is initialized with the elements in $\text{gqs}(C)$.

After these preliminary steps, the algorithm loops over the elements in $\text{Queue}$. At each iteration, the first node $n$ is removed from the head of $\text{Queue}$. The algorithm then checks whether $n$ is a concept node or a complex node.

**Handling concept nodes** (lines 15-28). If $n$ is a concept node, for instance representing concept $D$, the algorithm has discovered a possible inclusion $C_i \sqsubseteq D$ to be added to $S_i$. However, if $C_i \sqsubseteq \neg D$ is already in $S \cup S_i$, relation $C_i \sqsubseteq D$ has to be discarded. Moreover, since we know that $D$ is not acceptable, then any other concept which generalizes $D$ is not acceptable, too; thereby we remove from $\text{Queue}$ any node $e$ belonging to $\text{gqs}(D)$ (line 18), and then add $\text{gqs}(D)$ in $\text{BlackList}$ (line 19). Otherwise ($C_i \sqsubseteq \neg D$ is not in $S \cup S_i$), $C_i \sqsubseteq D$ represents a new piece of knowledge that we can add to $S_i$. If $S \cup S_i \cup \{C_i \sqsubseteq D\}$ is coherent, the new relation is added to $S_i$ as it is (line 22). Conversely, we have discovered one of the most specific properties of $C_i$ that have to be made typical. That is, in order to restore the consistency in $T$ when we also consider the exceptional properties of $\text{newC}$, the inclusion $C_i \sqsubseteq D$ has to be rewritten as $T(C_i) \sqsubseteq D$ (line 24).

Note that, since we add either $C_i \sqsubseteq D$ or $T(C_i) \sqsubseteq D$, we can conclude that any other concept in $\text{gqs}(\neg D)$ will not be acceptable as it would introduce an inconsistency. Thus, we can remove from $\text{Queue}$ any node $e$ which is a generalization of $\neg D$ (line 26), and then add $\text{gqs}(\neg D)$ in $\text{BlackList}$ (line 27).

**Handling complex nodes** (lines 29-32). In case the head node $n$ is a complex node, then it must mention at least a $\text{gqs}$, which abstracts a number of concepts. Dealing with a complex node means generating a number of child-nodes by expanding one of the $\text{gqs}$ mentioned in $n$. The $\text{ExpandComplexNode}$ function is shown in Algorithm 3. The set of new nodes $\text{NewNodes}$ returned by $\text{ExpandComplexNode}$ are enqueue in $\text{Queue}$ in the specificity ordering.
Algorithm 2 FindSubsumptions($C_i, gqs(C_i), S$)
1: $S_i \leftarrow \emptyset$
2: $BlackList \leftarrow \emptyset$
3: $Queue \leftarrow \emptyset$
4: for all element $e \in gqs(C_i)$, in the specificity ordering do
5: if $e$ is a concept $E$ then
6: create a concept node $n[E]$
7: enqueue $n[E]$ in $Queue$ in the ordering
8: else $\triangleright e$ is a generalization of a concept $E$; i.e., $gqs(E)$
9: create a complex node $n[gqs(E)]$
10: enqueue $n[gqs(E)]$ in $Queue$ in the ordering
11: end if
12: end for
13: while $Queue$ is not empty do
14: $n \leftarrow \text{Head}(Queue)$
15: if $n$ is a concept node then
16: Let $D$ be the concept in $n$; then $S_i$ is possibly extended to include $C_i \sqsubseteq D$
17: if $C_i \sqsubseteq \neg D \in S \cup S_i$ then
18: remove from $Queue$ any node $e$ such that $e \in gqs(D)$
19: enqueue $gqs(D)$ in $BlackList$
20: else $\triangleright C_i \sqsubseteq \neg D \not\in S$
21: if $S \cup S_i \cup \{C_i \sqsubseteq D\}$ is coherent then
22: $S_i \leftarrow S_i \cup \{C_i \sqsubseteq D\}$
23: else
24: $S_i \leftarrow S_i \cup \{T(C_i) \sqsubseteq D\}$
25: end if
26: remove from $Queue$ any node $e$ such that $e \in gqs(\neg D)$
27: enqueue $gqs(\neg D)$ in $BlackList$
28: end if
29: else $\triangleright n$ is a complex node
30: $NewNodes \leftarrow \text{ExpandComplexNode}(n, BlackList)$
31: enqueue each node $n' \in NewNodes$ in $Queue$ in the ordering
32: end if
33: end while
34: return $S_i$

Algorithm 3 ExpandComplexNode($n, BlackList$)
1: $List \leftarrow \emptyset$
2: Let $gqs(D)$ be one of the most specific $gqs$ mentioned in $n$
3: for all element $e \in gqs(D)$ such that $e \not\in gqs(F)$, $\forall gqs(F) \in BlackList$ do
4: create a node $n'$ by substituting $gqs(D)$ with $e$ in $n$
5: if $n'$ is an inconsistent concept then
6: discard $n'$
7: else
8: put $n'$ in $List$
9: end if
10: end for
11: return $List$
ExpandComplexNode

The expansion of a complex node is outlined in Algorithm 3. It takes as inputs the complex node $n$ that has to be expanded, and the BlackList containing the gqs of those concepts that have been pruned so far. The algorithm returns a list of new nodes, either concept or complex.

After having initialized List, (line 1), the algorithm selects one of the most specific gqs in $n$. In fact, since $n$ is a complex node, then it must mention at least one gqs (line 2). Let us assume that gqs($D$) has been selected, remember that gqs($D$) is compactly encoded as a list of concepts or gqs. The algorithm therefore loops over the elements $e$ in the list gqs($D$), if $e$ belongs to any gqs($F$) $\in$ BlackList, then $e$ can be pruned off as it has already been determined that $F$, and then any other concept in gqs($F$), is not consistent with some axioms in $S$ (line 3). Otherwise, $e$ can be used to create a new node $n'$. More precisely, $n'$ is obtained by substituting gqs($D$) with $e$ in $n$. Of course, $n'$ can either be a concept node or a complex node (line 4). As noted above, the generalizations of a given concept $D$ might contain contradictory concepts since the initial set of axioms in mups($T$, newC) is inconsistent. Thus, before adding $n'$ to List, we first check whether $n'$ is contradictory, in which case we discard it (line 5). Otherwise, $n'$ is added to List (line 8). The algorithm terminates by returning the, possibly empty, list of new nodes (line 11).

Let newC $\sqsubseteq D$ be the subsumption relation that, once added to a TBox $T$, generates an inconsistency, and let mups($T$, newC) the minimal subset of axioms in $T$ preserving the inconsistency, then we can prove the following properties.

**Proposition 1.** The set $S$ of concept inclusions generated by Algorithm 1 with the invocation FindSubsumptionsMain(mups($T$, newC)) is coherent.

Intuitively, $S$ is only modified either in line 7 or in line 11 of algorithm FindSubsumptionsMain. In the first case, we already know that all the sets $S_i$s inferred for all concepts at a given depth $l$ are altogether coherent. In the second case, instead, we know that an inconsistency has been detected; thereby we “correct” the axioms in $S_i$s by means of $T$, yielding the consistency of $S$. In fact, for the properties $\mathcal{ALC}^R_{\text{min}}T$ in [7], since $T$ is nonmonotonic, we have that $S$ is coherent.

**Proposition 2.** Let $S$ be the set of concept inclusions generated by Algorithm 1, then either newC $\sqsubseteq D$ or $T$(newC) $\sqsubseteq D$ belongs to $S$.

This follows directly by the fact that newC $\sqsubseteq D$ has generated an inconsistency in $T$, and then mups($T$, newC) necessarily includes such a relation. Algorithm 1 simply reconsiders all the axioms in mups($T$, newC), and builds a new set $S$ of axioms which contains, at least, the same axioms as in mups($T$, newC), but at least one has been modified by the typicality operator.

**Theorem 1.** Let $T'$ be a new terminology obtained as $T' = T \setminus \text{mups} (T, \text{newC}) \cup S$. The new terminology $T'$ is coherent.

This allows us to conclude that once we have computed $S$, we have a means for correcting the initially incoherent terminology $T$. The simplest way to do that is
to substitute the axioms in \( \text{mups}(\mathcal{T}, \text{newC}) \) with \( \mathcal{S} \). Further details and proofs are omitted due to space limitations.

**Example 2.** Let us consider again the terminology \( \mathcal{T}_{\text{student}} \) presented in Example 1. Algorithm 1 considers the concepts in the ordering: \( \text{Student} \sqcap \text{Worker} \), \( \text{Student} \), \( \text{TaxPayer} \), \( \neg \text{TaxPayer} \), \( \text{Worker} \). The first invocation of \( \text{FindSubsumptions} \) (Algorithm 2) is therefore over the inputs: \( \text{Student} \sqcap \text{Worker} \sqsubseteq \text{Student} \); \( \text{Student} \sqcap \text{Worker} \sqsubseteq \text{TaxPayer} \). The first invocation is the set \( \mathcal{S} = \{ \text{Student} \sqcap \text{Worker} \sqsubseteq \text{Student} \}; \text{Student} \sqcap \text{Worker} \sqsubseteq \text{TaxPayer} \}. \) The second invocation of \( \text{FindSubsumptions} \) is over the inputs: \( \text{Student}, \text{gqs(Student)} \). In this case, the algorithm will find the subsumption \( \text{Student} \sqsubseteq \neg \text{TaxPayer} \), which is incoherent with the subsumptions already in \( \mathcal{S} \), thus, \( \neg \text{TaxPayer} \) is considered as the typical property of \( \text{Student} \); in other words, \( \mathcal{S} \) is modified by adding \( \mathcal{T}(\text{Student}) \sqsubseteq \neg \text{TaxPayer} \). It is easy to see that no further subsumptions are added in \( \mathcal{S} \) when \( \text{FindSubsumptions} \) is invoked over the remaining concepts \( \text{TaxPayer} \), \( \text{Worker} \), and \( \neg \text{TaxPayer} \). In conclusion, the original terminology \( \mathcal{T}_{\text{student}} \), can be rewritten as \( \mathcal{T}_{\text{student}}' \):

\[
\text{Student} \sqcap \text{Worker} \sqsubseteq \text{Student} \\
\text{Student} \sqcap \text{Worker} \sqsubseteq \text{TaxPayer} \\
\text{Student} \sqcap \text{Worker} \sqsubseteq \text{Worker} \\
\mathcal{T}(\text{Student}) \sqsubseteq \neg \text{TaxPayer}
\]

By the properties of the DL \( \mathcal{ALC}_{\mathcal{R}} \), from the resulting knowledge base above we are able to reason about defeasible inheritance. For instance, if we have \( \text{Student(franco)} \) (Franco is a student), we are able to infer \( \neg \text{TaxPayer(franco)} \) (Franco does not pay taxes), however this conclusion is retracted if we further discover that \( \text{Worker(franco)} \) (Franco is also a worker), from which we obtain \( \text{TaxPayer(franco)} \). We are also able to deal with irrelevance: for instance, \( \mathcal{T}(\text{Student} \sqcap \text{Fat}) \sqsubseteq \neg \text{TaxPayer} \) can be inferred, and the above conclusions still hold even if we further know \( \text{Fat(franco)} \) (Franco is fat).

**Example 3.** Let us consider a simplified variant of the Pizza ontology distributed together with Protégé. In particular, let us assume that a \( \text{TBox} \) \( \mathcal{T}_{\text{tops}} \) contains the following subsumption relations:

\[
\text{ax}_1 : \text{FatToppings} \sqsubseteq \text{PizzaToppings}, \\
\text{ax}_2 : \text{CheesyToppings} \sqsubseteq \text{FatToppings}, \\
\text{ax}_3 : \text{VegetarianToppings} \sqsubseteq \neg \text{FatToppings}.
\]

Now, let us assume that we discover that there also exists the \text{tofu} cheese, which is made of curdled soymilk. Thus, we change \( \mathcal{T}_{\text{tops}} \) by adding the following:

\[
\text{ax}_4 : \text{CheesyVegetarianToppings} \sqsubseteq \text{CheesyToppings} \sqcap \text{VegetarianToppings}.
\]

Let the \( \text{ABox} \) \( \mathcal{A}_{\text{tops}} \) contain \( \text{CheesyVegetarianToppings(tofu)} \). The resulting knowledge base is inconsistent, respectively, the knowledge base obtained by adding \( \text{CheesyVegetarianToppings(tofu)} \) to the initial ABox is inconsistent, since \( \text{FatToppings} \) and \( \text{DieteticToppings} \) are disjunct concepts, whereas \( \text{CheesyVegetarianToppings} \) falls in their intersection.
The tofu cheese is an exception, and the first step to tackle is to compute \( mups(T_{tops}, \text{CheesyVegetarianToppings}) = \{ax_2, ax_3, ax_4\} \). Concept inclusions (even implicit) are taken into account: possibly, some of them will be “corrected” by means of the typicality operator in order to accommodate the exceptional CheesyVegetarianToppings concept. In particular, the inclusions discovered are:

\[
S_{tops} = \{
\text{CheesyVegetarianToppings} \subseteq \text{CheesyToppings},
\text{CheesyVegetarianToppings} \subseteq \text{VegetarianToppings},
\text{T}(\text{CheesyToppings}) \subseteq \text{FatToppings},
\text{T}(\text{VegetarianToppings}) \subseteq \neg\text{FatToppings}
\}
\]

The concept inclusions in \( mups(T_{tops}, \text{CheesyVegetarianToppings}) \) can be now substituted in \( T_{tops} \) with the set \( S_{tops} \). Theorem 1 ensures that the new \( T_{top} \) so obtained is now coherent and correctly addresses the exceptional concept CheesyVegetarianToppings. By the properties of \( \mathcal{ALC}_{min}^R \), from the resulting knowledge base, we will be able to reason about defeasible inheritance: for instance, if we know \( \text{CheesyToppings}(\text{brie}) \) (Brie is a cheesy topping), we conclude that \( \text{FatToppings}(\text{brie}) \) (Brie is a fat topping), whereas for tofu we say nothing about being a fat topping or not. We are also able to deal with irrelevance: for instance, it can be inferred that, normally, cheesy potatoes toppings are fat toppings, i.e. \( \text{T}(\text{Cheesy} \odot \text{Potatoes}) \subseteq \text{FatToppings} \).

5 Conclusions and Future Issues

We have presented some preliminary results in defining a methodology for automated revision of a DL terminology in presence of exceptions. We exploit techniques and algorithms proposed in [8], which are also extended to more expressive DLs such as \( \mathcal{SHOIN} \), corresponding to ontology language OWL-DL. On the one hand, we aim at extending our approach to such expressive DLs. On the other hand, we intend to develop an implementation of the proposed algorithms, by considering the integration with existing tools for manually modifying ontologies when inconsistencies are detected.

Acknowledgements

The authors are supported by the twin projects ODIATI#1 and ODIATI#2: “Ontologie, DIAgnosi e TIpicalità nelle logiche descrittive” of the local research funds 2013 by the Università degli Studi di Torino - part B, supporting young researchers.
References


240
Abstract. This paper deals with assurance of logical agent systems via runtime self-monitoring and checking. We adopt temporal-logic-based special constraints to be dynamically checked at a certain (customizable) frequency. These constraints are based upon a simple interval temporal logic particularly tailored to the agent realm, A-ILTL (‘Agent-Oriented Interval LTL’, LTL standing as customary for ‘Linear Temporal Logic’).

1 Introduction

Certification and assurance of agent systems constitute crucial and far-from-trivial issues, as agents represent a particularly complex case of dynamic, adaptive and reactive software systems. Certification is aimed at producing evidence indicating that deploying a given system in a given context involves the lowest possible level of risk of adverse consequences (which level of risk can be considered sufficiently “low” depends upon the application at hand). Assurance is related to dependability, i.e., to ensuring (or at least obtaining a reasonable confidence) that system users can rely upon the system. The issue is nicely discussed in [1], where it is noted that:

The term [assurance] is used in a broad (and somewhat imprecise) sense. Where there is a clear specification (which is not always the case!) then we can use the two standard terms “verification” and “validation”. Verification in this context refers to checking whether software meets its specification, and validation refers to checking whether the specification meets the user’s requirements.

It is widely acknowledged that industrial adoption of agents systems finds a serious obstacle in the stakeholders lack of confidence about reliability of runtime behavior of such systems. Citing [2],

… the use of adaptive systems for greater resilience create situations where runtime verification and monitoring could be particularly valuable. … Within suitable new frameworks, some of the evidence required for certification can be achieved by runtime monitoring - by analogy with runtime verification, this approach can, somewhat provocatively, be named “runtime certification”.
In this paper, we propose methods for runtime monitoring of agent systems. These methods are not in alternative but rather complementary to the many existing verification and testing methodologies.

Pre-deployment assurance and certification techniques for agent systems include verification and testing. Since we do not have room for an extensive illustration we can provide just few pointers to recent literature, so we invite the reader to refer to the recent book [3] and to the references therein. Most verification methods rely upon model-checking, and some (e.g., [4]) upon theorem proving. Among recent interesting work about agent systems (pre-deployment) assurance we particularly mention [1] which proposes (though in a preliminary way) a method that alternates the application of testing with formal verification techniques applied within a “Shallow Scope”, i.e, with a limited scope of variable values. The outcome of each phase should be taken as a guidance for the other phase. Thus, different techniques are used in synergy so as to improve the overall level of assurance. About fault detection and recovery a particularly interesting work is that of [5], that opens a new promising direction in model-checking-based verification techniques. This approach allows for CTL specifications that express injection and eventual recovery from a fault.

For formalizing and implementing runtime self-checking in logical agents while coping with unanticipated circumstances, we propose temporal-logic-based special constraints to be dynamically checked at a certain (customizable) frequency. These constraints are based upon a simple interval temporal logic particularly tailored to the agent realm, A-ILTL (‘Agent-Oriented Interval LTL’, LTL standing as customary for ‘Linear Temporal Logic’). In this setting, properties can be defined that should hold according to events that have happened and to events which are supposed to happen or not to happen in the future. This also considering partially specified event sequences, unexpected events or event order. The adoption of an interval logics allows for the specification of time-bounded properties: it makes it possible to specify that some property should occur within a certain time frame or before/after a certain time, where the interval can also be conditionally defined. A-ILTL constraints are contextual, i.e., they can be specified in a general form and each time they are checked they are instantiated (via suitable preconditions) to the present agent’s state.

In [2], it is advocated that for adaptive systems (of which agents are clearly a particularly interesting case) assurance methodologies should whenever possible imply not only detection but also recovery from software failures. In fact, though (at least in principle) a certified software should not fail, in practice serious software-induced incidents have been observed in certified critical systems. In [2] examples are produced concerning airplane and air traffic control, where failures are often due on the one hand to incomplete specifications and on the other hand to unpredictability of the environment.

In [6], which discusses medical robotic applications in human telesurgery, it is emphasized how such systems should be fail safe in the sense that, in the event of failure, should proactively respond so as to limit harm to other devices or danger to users.

Our methods in fact provide the possibility of correcting and/or improving agent’s functioning: the behavior can be corrected whenever an anomaly is detected, but can also be improved whenever it is acceptable, yet there is room for getting a better perfor-
mance. Counter measures can be object-level, i.e., related to the application, or meta-
level, e.g., replacing (as suggested in [2]) a software component by a diverse alternate.

A-ILTL constraints are defined over formulas of any underlying logic language $L$, and
are rooted in the Evolutionary Semantics of agent programs [7]. We thus obtain a
fairly general setting, that could be adopted in several logic agent-oriented languages
and formalisms, such as, e.g., AgentSpeak (cf. [8, 9] and the references therein), DALI
[10–12], GOAL [13, 14], and 3APL [15, 16].

In this paper, we show how A-ILTL temporal constraints may be used to check
for critical situations and to enforce suitable reaction patterns for achieving recovery.
The novelty of the approach is in the following aspects. (i) A-ILTL temporal constraints
constitute a device for run-time self-monitoring which can be completely integrated into
agent programs and their semantics. I.e., there is no separate monitor which examines
a “trace” of observations performed on the agent’s behavior. (ii) Self-recovery/repair
is encompassed in the approach. (iii) The semantic integration into the Evolutionary
semantics is devised such that there is no need to implement a full temporal-logic in-
ference engine, at least if keeping the expressions to be checked reasonably simple. (iv)
Consequently, the complexity of check is reasonably low.

The paper is organized as follows. In Section 2 we recall the Evolutionary Seman-
tics. In Sections 3–4 we introduce the A-ILTL logic, also in relation to the Evolutionary
Semantics. In Section 5 we illustrate A-ILTL constraints and show by means of ex-
amples how such constraints can be exploited for runtime monitoring and self-repair
of agent systems. In Section 6 we briefly discuss the complexity related to run-time
constraint checking. Finally, in Section 7 we discuss related work and propose some
concluding remarks.

2 Evolutionary Semantics

The Evolutionary semantics (introduced in [7]) is meant at providing a high-level gen-
eral account of evolving agents, trying to abstract away from the details of specific
agent-oriented frameworks. We define, in very general terms, an agent as the tuple $Ag
= \langle P_A, E \rangle$ where $A$ is the agent name and $P_A$ (that we call “agent program”, but
can be in turn a tuple) describes the agent according to some agent-oriented formalism
$L$. $E$ is the set of the events that the agent is able to recognize or determine (so, $E$
includes actions that the agent is able to perform), according to the specific agent-oriented
framework.

Let $H$ be the history of an agent as recorded by the agent itself (in a form that
will depend upon the specific agent-oriented framework), i.e., $H$ includes agent’s per-
ceptions and memories. For instance, in DALI the history consists of: the set $Ev$ of
external and internal events, that represent respectively events that the agent presently
perceives of its environment, and events that the agent has raised by its own internal
reasoning processes; the set $Act$ of the actions that the agent is enabled to perform at
its present stage of operation; the set $P$ of most recent versions “past events”, which
include: previously perceived events, but also actions that the agent has performed (no-
tice that elements of $Ev$ and $Act$ will be transferred into $P$ at the next stage); the set
$PNV$ of previous instances of past events (e.g., $P$ may contain the last measurements
of temperature while $PNV$ may contain older ones), plus past constraints that specify interaction between $P$ and $PNV$.

We assume that program $P_A$ as written by the programmer is in general transformed into an initial agent program $P_0$ by means of an initialization step. When agent $A$ is activated $P_0$ will go into execution, and will evolve according to events that either happen or are generated internally, to actions which are performed, etc., i.e., according to the evolution of $H$.

Evolution in this setting is represented via program-transformation steps, each one transforming $P_i$ into $P_{i+1}$ according to $H_i$, which is the partial history up to stage $i$. The choice of which elements of $H_i$ do actually trigger an evolution step is part of the definition of a specific agent framework.

Thus, we obtain a Program Evolution Sequence $PE = [P_0, \ldots, P_n, \ldots]$. The program evolution sequence will imply a corresponding Semantic Evolution Sequence $ME = [M_0, \ldots, M_n, \ldots]$ where $M_i$ is the semantics of $P_i$ according to $L$. Notice in fact that the approach is parametric w.r.t $L$.

**Definition 1 (Evolutionary semantics).** Let $Ag$ be an agent. The evolutionary semantics $\varepsilon^{Ag}$ of $Ag$ is a tuple $\langle H, PE, ME \rangle$, where $H$ is the history of $Ag$, and $PE$ and $ME$ are respectively its program and semantic evolution sequences.

The next definition introduces the notion of instant view of $\varepsilon^{Ag}$, at a certain stage of the evolution (which is in principle of unlimited length).

**Definition 2 (Evolutionary semantics snapshot).** Let $Ag$ be an agent, with evolutionary semantics $\varepsilon^{Ag} = \langle H, PE, ME \rangle$. The snapshot at stage $i$ of $\varepsilon^{Ag}$ is the tuple $\langle H_i, P_i, M_i \rangle$, where $H_i$ is the history up to the events that have determined the transition from $P_{i-1}$ to $P_i$.

In [7], program transformation steps associated with DALI language constructs are defined in detail. They can easily be adapted to AgentSpeak [8, 9] as the two languages share a number of similarities. More generally however, in the specific agent setting under consideration an evolution step will occur at least whenever new events are perceived, reacted to, and recorded, and whenever an agent proactively undertakes measures to pursue its goals. An evolution step will possibly determine an update of the history, which is a part of the agent’s belief base\(^1\). Thus, each evolution step affects the belief or “mental” state of an agent. The evolutionary semantics may express for instance the notion of trace of a GOAL agent [13, 14] where agent program $P_i$ encompasses the agent’s mental state and each evolution step, which in GOAL is called computation step is determined by a conditional action. For 3APL [15, 16], agent program $P_i$ encompasses the agent’s initial configuration, and the related sets GR of goal rules, PR of plan rules, IR of interactive rules; the evolutionary semantics corresponds to a 3APL agent computation run, and evolution steps are determined by the 3APL transition system.

The semantics presented in [17] for Reactive Answer Set Programming, based upon “incremental logic programs” and “online progression”, brings some conceptual similarity with the (pre-existing) Evolutionary Semantics.

\(^1\) Equivalently, according to the specific agent framework with its own terminology, one may talk of an agent’s knowledge base.
3 A-ILTL

For defining properties that are supposed to be respected by an evolving system, a well-established approach is that of Temporal Logic, and in particular of Linear-time Temporal Logics (LTL, cf., e.g., [18]). These logics are called ‘linear’ because (in contrast to ‘branching time’ logics) they evaluate each formula with respect to a vertex-labeled infinite path (or “state sequence”) $s_0s_1\ldots$ where each vertex $s_i$ in the path corresponds to a point in time (or “time instant” or “state”). In what follows, we use the standard notation for the best-known LTL operators.

An interval-based extension to the well-known linear temporal logic LTL is formally introduced in [19] where it is called A-ILTL for ‘Agent-Oriented Interval LTL’. Though, as discussed in [19], several “metric” and interval temporal logic exist, the introduction of A-ILTL is useful in the agent realm because the underlying discrete linear model of time and the complexity of the logic remains unchanged with respect to LTL. This simple formulation can thus be efficiently implemented, and is nevertheless sufficient for expressing and checking a number of interesting properties of agent systems.

Formal syntax and semantics of A-ILTL operators (also called below “Interval Operators”) are fully defined in [19]. A-ILTL expressions are (like plain LTL ones) interpreted in a discrete, linear model of time. Formally, this structure is represented by $M = \langle N, I \rangle$ where, given countable set $\Sigma$ of atomic propositions, interpretation function $I : N \mapsto 2^\Sigma$ maps each natural number $i$ (representing state $s_i$) to a subset of $\Sigma$. Given set $F$ of formulas built out of classical connectives and of LTL and A-ILTL operators (where however nesting of A-ILTL operators is not allowed), the semantics of a temporal formula is provided by a satisfaction relation: for $\varphi \in F$ and $i \in N$ we write $M, i \models \varphi$ if, in the satisfaction relation, $\varphi$ is true w.r.t. $M, i$. We can also say (leaving $M$ implicit) that $\varphi$ holds at $i$, or equivalently in state $s_i$, or that state $s_i$ satisfies $\varphi$. A structure $M = \langle N, I \rangle$ is a model of $\varphi$ if $M, i \models \varphi$ for some $i \in N$.

Some among the A-ILTL operators are the following.

**Definition 3.** Let $\varphi \in F$ and let $m, n$ be positive integer numbers.

- $F_{m,n} \varphi$ (eventually (or “finally”) in time interval). $F_{m,n} \varphi$ states that $\varphi$ has to hold sometime on the path from state $s_m$ to state $s_n$. I.e., $M, i \models F_{m,n} \varphi$ if there exists $j$ such that $j \geq m$ and $j \leq n$ and $M, j \models \varphi$. Can be customized into $F_m$, bounded eventually (or “finally”), where $\varphi$ should become true somewhere on the path from the current state to the $(m)$-th state after the current one.

- $G_{m,n} \varphi$ (always in time interval). $G_{m,n} \varphi$ states that $\varphi$ should become true at most at state $s_m$ and then hold at least until state $s_n$, i.e., $M, i \models G_{m,n} \varphi$ if for all $j$ such that $j \geq m$ and $j \leq n$ and $M, j \models \varphi$. Can be customized into $G_m$, bounded always, where $\varphi$ should become true at most at state $s_m$.

- $N_{m,n} \varphi$ (never in time interval). $N_{m,n} \varphi$ states that $\varphi$ should not be true in any state between $s_m$ and $s_n$, i.e., $M, i \models N_{m,n} \varphi$ if there not exists $j$ such that $j \geq m$ and $j \leq n$ and $M, j \models \varphi$. 

S. Costantini, G. De Gasperis. Runtime Self-Checking via Temporal (Meta-)Axioms for Assurance of Logical AS
4 A-ILTL and Evolutionary Semantics

In this section, we refine A-ILTL so as to operate on a sequence of states that corresponds to the Evolutionary Semantics defined before. In fact, states in our case are not simply intended as time instants. Rather, they correspond to stages of the agent evolution. Time in this setting is considered to be local to the agent, where with some sort of “internal clock” is able to time-stamp events and state changes. We borrow from [20] the following definition of \textit{timed state sequence}, that we tailor to our setting.

**Definition 4.** Let \( \sigma \) be a (finite or infinite) sequence of states, where the \( ith \) state \( e_i, e_i \geq 0 \), is the semantic snapshots at stage \( i \varepsilon_{Ag}^i \) of given agent \( Ag \). Let \( T \) be a corresponding sequence of time instants \( t_i, t_i \geq 0 \). A timed state sequence for agent \( Ag \) is the couple \( \rho_{Ag} = (\sigma, T) \). Let \( \rho_i \) be the \( i-th \) state, \( i \geq 0 \), where \( \rho_i = (e_i, t_i) = (\varepsilon_{Ag}^i, t_i) \).

We in particular consider timed state sequences which are monotonic, i.e., if \( e_{i+1} \neq e_i \), then \( t_{i+1} > t_i \). In our setting, it will always be the case that \( e_{i+1} \neq e_i \) as there is no point in semantically considering a static situation: as mentioned, a transition from \( e_i \) to \( e_{i+1} \) will in fact occur when something happens, externally or internally, that affects the agent.

Then, in the above definition of A-ILTL operators, it is immediate to let \( s_i = \rho_i \). This requires however a refinement: in fact, in a writing \( Op_m \) or \( Op_{m,n} \) occurring in an agent program parameters \( m \) and \( n \) will not necessarily coincide with time instants of the above-defined timed state sequence. To fill this gap, in [19] a suitable approximation is introduced.

We need to adapt the interpretation function \( I \) of LTL to our setting. In fact, we intend to employ A-ILTL within agent-oriented languages, where we restrict ourselves to logic-based languages for which an evolutionary semantics and a notion of logical consequence can be defined. Thus, given agent-oriented language \( L \) at hand, the set \( \Sigma \) of propositional letters used to define an A-ILTL semantic framework will coincide with all ground expressions of \( L \) (an expression is \textit{ground} if it contains no variables, and each expression of \( L \) has a possibly infinite number of ground versions). A given agent program can be taken as standing for its (possibly infinite) ground version, as it is customarily done in many approaches. Notice that we have to distinguish between logical consequence in \( L \), that we indicate as \( \models L \), from logical consequence in A-ILTL, indicated above simply as \( \models \). However, the correspondence between the two notions can be quite simply stated by specifying that in each state \( s_i \) the propositional letters implied by the interpretation function \( I \) correspond to the logical consequences of agent program \( P_i \):

**Definition 5.** Let \( L \) be a logic language. Let \( Expr_L \) be the set of ground expressions that can be built from the alphabet of \( L \). Let \( \rho_{Ag} \) be a timed state sequence for agent \( Ag \), and let \( \rho_i = \langle \varepsilon_{Ag}^i, t_i \rangle \) be the \( i-th \) state, with \( \varepsilon_{Ag}^i = \langle H_i, P_i, M_i \rangle \). An A-ILTL formula \( \tau \) is defined over sequence \( \rho_{Ag} \) if in its interpretation structure \( M = (\mathbb{N}, I) \), index \( i \in \mathbb{N} \) refers to \( \rho_i \), which means that \( \Sigma = Expr_L \) and \( I : \mathbb{N} \rightarrow 2^{\Sigma} \) is defined such that, given \( p \in \Sigma, p \in I(i) \) iff \( P_i \models_L p \). Such an interpretation structure will be indicated with \( M^{Ag} \). We will thus say that \( \tau \) holds/does not hold w.r.t. \( \rho_{Ag} \).
A-ILTL properties will be verified at run-time, and thus they act as constraints over the agent behavior\(^2\). In an implementation, verification may not occur at every state (of the given interval). Rather, sometimes properties need to be verified with a certain frequency, that can be specifically tuned to the various cases. Then, we have introduced a further extension that consists in defining subsequences of the sequence of all states: if \(Op\) is any of the operators introduced in A-ILTL and \(k > 1\), \(Op^k\) is a semantic variation of \(Op\) where the sequence of states \(\rho_{Ag}\) of given agent is replaced by the subsequence \(s_0, s_{k_1}, s_{k_2}, \ldots\) where for each \(k_r, r \geq 1\), \(k_r \mod k = 0\), i.e., \(k_r = g \times k\) for some \(g \geq 1\).

A-ILTL formulas to be associated to given agent can be defined within the agent program, though they constitute an additional but separate layer, composed of formulas \(\{\tau_1, \ldots, \tau_l\}\). Agent evolution can be considered to be “satisfactory” if it obeys all these properties.

**Definition 6.** Given agent \(Ag\) and given a set of A-ILTL expressions \(A = \{\tau_1, \ldots, \tau_l\}\), timed state sequence \(\rho_{Ag}\) is coherent w.r.t. \(A\) if A-ILTL formula \(G\zeta\) with \(\zeta = \tau_1 \land \ldots \land \tau_l\) holds.

Notice that the expression \(G\zeta\) is an invariance property in the sense of [21]. In fact, coherence requires this property to hold for the whole agent’s “life”. In the formulation \(G_{m,n}\zeta\) that A-ILTL allows for, one can express temporally limited coherence, concerning for instance “critical” parts of an agent’s operation. Or also, one might express forms of partial coherence concerning only some properties.

An “ideal” agent will have a coherent evolution, whatever its interactions with the environment can be, i.e., whatever sequence of events arrives to the agent from the external “world”. However, in practical situations such a favorable case will seldom be the case, unless static verification has been able to ensure total correctness of agent’s behavior. Instead, violations will occasionally occur, and actions should be undertaken so as to attempt to regain coherence for the future.

A-ILTL rules may imply asserting and retracting rules or sets of object rules (“modules”). In this setting, assert and retract can be considered as special A-ILTL operators, for which a formal semantics is provided (cf. [19]).

## 5 A-ILTL for Monitoring Liveness and Safety Properties

In this section we illustrate the usefulness of A-ILTL constraints for defining and verifying liveness and safety properties in agent systems. In software engineering, liveness properties concern the progress that an agent makes and express that a (good) state eventually will be reached, while safety properties express that some (bad) state will never be entered. This implies that liveness is concerned with the evolution of a system, while in general safety is not: notice in fact that, paradoxically, doing nothing prevents bad states from being reached. Notice however that in our setting we restricted ourselves to monotonic state sequences based upon the evolutionary semantics, so that our agents evolve by definition. Notice that, if violated, liveness properties are violated in infinite

\(^2\) By abuse of notation we will indifferently talk about A-ILTL rules, expressions, or constraints.
time (a good state not yet reached might be in principle reached in the future) while safety properties are violated in finite time, in case a “bad” state is reached. It is widely acknowledged (cf., e.g., [22]) that any property can be expressed as a conjunction of a safety and a liveness property. In agents, “bounded” liveness is often more interesting than “pure” liveness: in fact, sometimes it does not suffice that a certain state might be reached in an indefinite future, as agents are situated real-time working entities that operate with limited computational resources and within deadlines. Bounded liveness properties are equivalent to safety properties that are violated whenever the desirable state is not reach withing the deadline. However, expressing such properties in the form of liveness properties is often more intuitive. A-ILTL operators can be defined either on finite intervals and then, to any practical extent, they define safety properties, or to infinite intervals (with no upper bound) thus defining liveness properties.

We employ in the examples a pragmatic form for A-ILTL expressions related to logic agent-oriented languages. In particular, we represent an A-ILTL expression in the form $OP(m, n; k) \varphi$ where: $m, n$ define the time interval where (or since when, if $n$ is omitted) expression $OP \varphi$ is required to hold, and $k$ (optional) is the frequency (in terms of states, or time instants) for checking whether the expression actually holds.

For instance, $EVENTUALLY (m, n; k) \varphi$ states that $\varphi$ should become true at some point between time instants (states) $m$ and $n$.

In rule-based logic programming languages, we may reasonably restrict $\varphi$ to be a conjunction of literals. In pragmatic A-ILTL formulas, $\varphi$ must be ground when the formula is checked. In fact, we allow variables to occur in an A-ILTL formula, to be instantiated via a context $\chi$ (we then talk about contextual A-ILTL formulas). Notice that, for the evaluation of $\varphi$ and $\chi$, we rely upon the procedural semantics of the ‘host’ language.

In the following, a contextual A-ILTL formula $\tau$ will implicitly stand for the ground A-ILTL formula obtained via evaluating the context. In [19] it is specified how to operationally check whether such a formula holds. This by observing that A-ILTL operators defined over finite intervals there is a crucial state where it is definitely possible to assess whether a related formula holds or not in given state sequence, by observing the sequence up to that point and ignoring the rest.

In runtime self-checking, as discussed above, an issue of particular importance in case of violation of a property is that of undertaking suitable measures in order to recover or at least mitigate the critical situation. Actions to be undertaken in such circumstances can be seen as an internal reaction to criticalities. More effective reaction can be defined if complex reactive features are available in the underlying language. In non-trivial cases, the issue of runtime recovery has a significant intersection (that had not been identified so far) with “Complex Event Processing” (CEP), which is an emergent relevant new field of software engineering and computer science [23]. In fact, a lot of practical applications have the need to actively monitor vast quantities of event data to make automated decisions and take time-critical actions [24–27] (cf. also the Proceedings of the RuleML Workshop Series). Many of the current approaches to CEP are declarative and based on rules, and often on logic-programming-like languages and semantics: for instance, [24] is based upon a specifically defined interval-based Event Calculus [28]. In logical agents, [29–31] tackled the issue of complex reactivity, by
considering the possibility of choosing among different possible reactive patterns also
by means of complex preferences. In the present paper, we show by means of examples
how kinds of A-ILTL constraints exploiting complex reactivity can be useful in runtime
recovery. For lack of space reactive patterns will be discussed informally in relation to
examples.

Below is the general form of an A-ILTL constraint with a reactive component that
we call recovery pattern.

Definition 7. A reactive A-ILTL rule is of the form (where $M, N, K$ can be either vari-
ables or constants)

$$\text{OP}(M, N; K) \varphi :: \chi \div \rho$$

where: (i) $\text{OP}(M, N; K) \varphi :: \chi$ is a contextual A-ILTL formula, called the monitoring
condition, that should involve the observation of either external or internal events; (ii)
$\rho$ is called the recovery component of the rule, and it consists of a complex reactive
pattern.

Whenever the monitoring condition (automatically checked at frequency $K$) is vi-
olated (i.e., it does not hold) within given interval, then the recovery component $\rho$ is
executed. Syntax and semantics of reactive patterns usable in the recovery component
will depend upon the underlying language $L$. In the examples, we adopt a sample syntax
suitable for logic-programming-based settings.

Consider for instance the example of a controller agent that has to keep the tem-
perature in a certain time frame (say between 8 a.m. and 5 p.m.) in the range 19–21
(Celsius degrees). In this case, the measure temperature implies sens-
ing actions to be performed with a sampling period by the agent. If the condition is
violated, a reaction should try to restore the wished-for situation. We assume in fact to
be in a smart building, where the temperature is monitored by intelligent agents, and
where each agent tries to select, in order to modify the temperature, the best suitable
energy source: for instance, according to present circumstances, an agent might select
the less expensive font of energy or, in case of a measure significantly different from
wished-for values, the font which guarantees the most efficient correction. Notice that
in the course of time different fonts of energy can be deemed to be the best choice. At
each check (where in fact the A-ILTL constraints is dynamically checked at the speci-
fied frequency, or at a default frequency in case none is provided) we assume that the
best choice can determined by means of an application-dependent decision procedure.
So, in given interval, the monitoring condition will sometimes succeed (the temperature
is within range, then nothing is done) and will sometimes fail. In the latter case, the font
of energy $S$ which is deemed more effective (in terms of cost and/or efficiency) in that
moment is determined, and used in order to suitably affect the temperature and try to
keep it within the specified range (where $\text{modify} \_\text{temperature}_G(S)$ is a goal, involv-
ing appropriate actions). In A-ILTL, this can be formalized as follows by exploiting
complex preferences introduced in [32]. As there are no variables, context is omitted.

$$\text{ALWAYS}(8 : 00 \text{ a.m.}, 5 : 00 \text{ p.m.; } 10 \text{m}) 19 \leq \text{temperature} \leq 21 \div
\text{modify} \_\text{temperature}_G(S),
S \text{ IN } \{\text{external\_electricity, gas, solar\_panel\_electricity : most\_effective}\}$$
The next example is a meta-statement expressing the capability of an agent to modify its own behavior. In case a goal \( G \) has not been achieved (in a certain context) because the allotted time has elapsed, then the recovery component implies replacing the planning module (assuming that more than one is available) and retrying the goal. We suppose that the possibility of achieving a goal \( G \) is evaluated w.r.t. a module \( M \) that represents the context for \( G \) (notation \( P(G, M) \), \( P \) standing for 'possible'). Necessity and possibility evaluation within a reasonably complex framework has been discussed in [30]. In case the goal is still deemed to be possible but has not been achieved before a certain deadline, the reaction consists in substituting the present planning module and re-trying the goal.

\[
\text{NEVER } \text{goal}(G), \text{eval}_{\text{context}}(G, M), P(G, M), \text{timed}_{\text{out}}(G), \text{not achieved}(G) \Rightarrow \text{replace_pn} \Rightarrow \text{retry}(G)
\]

It can be useful to define properties to be checked upon arrival of event sequences, of which however only relevant events (and their order) should be considered. To this aim we introduce a new kind of A-ILTL constraints, that we call Evolutionary A-ILTL Expressions. To define partially known sequences of any length, on the line of dynamic logic [33] we admit for event sequences a syntax reminiscent of regular expressions so as to specify irrelevant/unknown events, and repetitions. In particular, event expressions (and, analogously, action expressions) may be primitive events \( e \), sequences of event expressions \( e_1; e_2; \ldots \), zero or more iterations of an event expression \( e^* \), or a choice among event expressions \( e_1 + e_2 + \ldots \). We also admit “wild cards”, i.e., variables (starting with uppercase) to stand for unknown events/actions.

**Definition 8 (Evolutionary A-ILTL Expressions).** Let \( S^{Evp} \) be a sequence of past events, and \( S^F \) and \( J^J \) be sequences of events. Let \( \tau \) be a contextual A-ILTL formula \( Op \varphi \rightarrow \chi \). An Evolutionary LTL Expression \( \varpi \) is of the form \( S^{Evp} : \tau :: S^F :: J^J \) where: (i) \( S^{Evp} \) denotes the sequence of relevant events which are supposed to have happened, and in which order, for the rule to be checked; i.e., these events act as preconditions: whenever one or more of them happen in given order, \( \tau \) will be checked; (ii) \( S^F \) denotes the events that are expected to happen in the future without affecting \( \tau \); (iii) \( J^J \) denotes the events that are expected not to happen in the future; i.e., whenever any of them should happen, \( \varphi \) is not required to hold any longer, as these are “breaking events”.

An Evolutionary LTL Expression can be evaluated w.r.t. a state \( s_i \) which includes among its components the history of the agent, i.e., the list of past events perceived by the agent. A history \( H \) satisfies an event sequence \( S \) whenever all events in \( H \), in the order specified by \( S \) itself.

**Definition 9.** An Evolutionary A-ILTL Expression \( \varpi \), of the form specified in Definition 8: (1) holds in state \( s_i \) whenever (i) history \( H_i \) satisfies \( S^{Evp} \) and \( S^F \) and does not include any event in \( J^J \), and \( \tau \) holds or (ii) \( H_i \) includes any event occurring in \( J^J \) (the expression is broken); (2) is violated in state \( s_i \) whenever \( H_i \) satisfies \( S^{Evp} \) and \( S^F \) and does not include any event in \( J^J \), and \( \tau \) does not hold.
Operationally, an Evolutionary A-ILTL Expression can be finally deemed to hold if either the critical state has been reached and \( \tau \) holds, or an unwanted event has occurred. Instead, an expression can be deemed *not* to hold (or, as we say, to be *violated* as far as it expresses a wished-for property) whenever \( \tau \) is false at some point without the occurrence of breaking events.

The following is an example of Evolutionary A-ILTL Expression that might occur in an agent program installed on an autonomous robot working on batteries, and able to check its own charge level. The robot moves in some environment to perform some task. The following A-ILTL axiom states that after a battery recharge (indicated as a past event, postfix \( 'P' \)) at time \( T \), the charge level should be sufficient for 6 hours despite a sequence of actions which can be considered to be 'normal' in relation to the robot’s task. These actions may for instance involve moving around, cleaning rubbish, delivering packages, etc. Instead, the charge level can be expected to be low in case of extensive usage actions, for instance in case of an exceptional unexpected event that requires the robot to increase its activities (e.g., drying water in case of a flooding from a broken pipe). There is a classification of what should be intended by 'normal' and 'extensive' usage.

\[
\text{recharge\_battery}_{P}: T \ :
\begin{align*}
\text{ALWAYS}(T, T + 6\text{hour}) \ & \text{charge\_level}(L), L > \text{low} \\
::: \ & \text{normal\_usage\_action}(Act)* \ & \\
::: \ & \text{extensive\_usage\_action}(Act)*
\end{align*}
\]

The above expression should be combined with another A-ILTL expression forcing recharge every six hours. The latter should state that if the last battery recharge \( \text{recharge\_battery}_{P} \) has occurred at time \( T \) which is more than six hours different from present time \( \text{now} \), then as a recovery the goal \( \text{recharge\_battery}_{G} \) must be set. Achieving this goal may require, for instance, reaching the nearest recharge station. Notice that, in this case, we have used an A-ILTL constraint as a programming construct, which however has a role in terms of assurance since it forces the agent to respect a timing which is essential for the system good functioning.

\[
\begin{align*}
\text{ALWAYS} \ & \text{recharge\_battery}_{P}: T, \text{now} - T > 6\text{hour} \ \div \ \text{recharge\_battery}_{G}
\end{align*}
\]

Whenever an Evolutionary A-ILTL expression is either violated or broken, a reaction can be attempted aiming at recovering a desirable or at least acceptable agent’s state.

**Definition 10.** An evolutionary LTL expression with repair \( \varpi^r \) is of the form \( \varpi |\eta_1| |\eta_2 \) where \( \varpi \) is an Evolutionary LTL Expression adopted in language \( L \), and \( \eta_1, \eta_2 \) are atoms of \( L \). \( \eta_1 \) will be executed (according to \( L \)'s procedural semantics) whenever \( \varpi \) is violated, and \( \eta_2 \) will be executed whenever \( \varpi \) is broken. \( \eta_1 \) and \( \eta_2 \) are called countermeasures.

In previous example, whenever the robot detects a low level of charge, countermeasure \( \eta_1 \), taken in case of low battery under normal usage, may for instance imply alerting the user, as a fault either in the battery or in the recharge station can be hypothesized. Instead \( \eta_2 \), taken in case of low battery under exceptional usage, will simply imply the robot to resort to the recharge station. The overall expression will take the form:
recharge_battery P : T :
ALWAYS(T, T + 6 hour) charge_level(L), L > low
| alert_user_possible_fault A || recharge_battery G

6 Complexity of Check and Discussion

In this section we synthetically analyze the complexity of checking A-ILTL expressions. For lack of space, we cannot provide a detailed account. We make the simplifying assumption that all expressions are checked at the same frequency: i.e., the agent devotes with a certain periodicity some amount time to perform the check. Here we evaluate this amount. Let us assume to have \( f \) A-ILTL expressions, and that the time for retrieving each expression from the computer memory is \( m \). Thus, retrieving all expressions to be evaluated is \( O(f \times m) \). Let \( k \) be the number of the different A-ILTL operator occurring in the \( f \) expressions. Let \( if_{eval} \) be the time needed in order to understand whether each expression needs to be evaluated at the present state: this includes checking w.r.t. the crucial state and, in case of Evolutionary A-ILTL Expressions, checking the event sequence \( S^E \) w.r.t. current agent’s history. Let \( max_{eval} \) be the maximum time needed for the evaluation of each contextual A-ILTL formula \( Op \varphi :: \chi \). Let \( if_{viol or broken} \) be the maximum time needed to state whether each Evolutionary A-ILTL Expressions is either violated or broken: this implies checking event sequences \( S^P \) and \( J^D \) w.r.t. current agent’s history.

Therefore, the total time to be spent for checking all A-ILTL Expression (in the worst case, where all of them are of the Evolutionary kind, and each of them needs to be evaluated at the present state) can be estimated to:

\[
O((f \times m) + (f \times (if_{eval} + max_{eval} + if_{viol or broken})))
\]

Then, for each expression which is either violated or broken, there will be a time spent in the recovery and countermeasure actions.

The relatively low complexity of check (which however requires to keep the number of A-ILTL expressions as low as possible, and to tune frequency carefully, according to the environment change rate) is due to the definition of A-ILTL in relation to the Evolutionary semantics: in fact, it is not needed to implement a temporal logic inference engine, rather to periodically check \( Op \varphi :: \chi \). This in the case of simple non-nested A-ILTL expressions. Introducing more complex expressions is a subject of future work.

7 Related Work and Concluding Remarks

In this paper, we have proposed A-ILTL runtime constraints for agents’ self-checking and monitoring. We have shown how to express via these constraints a number of useful liveness and safety properties. We have provided a semantic framework general enough for accommodating a number of agent-oriented languages, so as to allow A-ILTL constraints to be adopted in different settings. This work has been influenced by [34, 19, 35, 36].

We may easily notice similarities between A-ILTL constraints and event-calculus formulations [28]. Also, approaches based on abductive logic programming such as,
SCIFF (cf. [37] and the references therein) allow one to model dynamically upcoming events, and specify positive and negative expectations, and the concepts of fulfillment and violation of expectations. Reactive Event Calculus (REC) stems from SCIFF [38] and adds more flexibility by reacting to new events by extending and revising previously computed results. However, these approaches have been devised for static or dynamic checking when performed by a third party. Event sequences, the concepts of violated and broken expressions, complex reaction patterns, and independence of the underlying logic are however distinguished features of the proposed approach.

A well-established line of work concerning the use of temporal logic in order to define run-time monitors is discussed in [39] and the references therein. However, this work is not related to agents, and does not concern self-checking: in fact, they propose a rule-based temporal language for defining “monitors” which examine either on-line or off-line some kind of “observable trace” generated by the program under check. There is no notion of recovery in case malfunctioning should be detected.

The proposed approach has been experimented in the context of energy management in smart buildings [40]. Such intelligent control is dynamic by nature, and must fulfill real-time requirements: in fact, each building has its own dynamical thermo-physical behavior and is immersed in a dynamic environment where weather events change its energy footprint in function of time. The outcome of the experiments is encouraging, in the sense that adopting agents equipped with the proposed features allows for not only general but also local (room-by-room or area-by-area) control of energy saving according to user comfort requirements and preferences.

Future work includes refining A-ILTL constraints to adapt to different self-checking issues and contexts. As suggested in [2], a very interesting line of investigation concerns automated synthesis of runtime constraints from specifications but also from test results, extracting invariants expressing correct or critical situations.

References


254
Complex Events and Actions in Logical Agents

Stefania Costantini and Regis Riveret

1 Università di L’Aquila, Italy, stefania.costantini@univaq.it
2 Imperial College London, UK, r.riveret@imperial.ac.uk

Abstract. Complex Event Processing (CEP) has emerged as a relevant new field of software engineering and computer science. Many of the current approaches to CEP are declarative and based on rules, and often on logic-programming-like languages and semantics. Some work on CEP is situated within the field of logical agents. Usually, event processing is based upon Event-Condition-Action rules, which are however able to manage only simple events or unstructured sets of events. In this paper, we refine the notion of “complex event”, which is an event that cannot be detected directly, but rather can be identified (not necessarily in a deterministic way) by interrelating sets of simple events. We propose a formalization and a possible implementation of such notion, that we extend to complex actions.

1 Introduction

“Complex Event Processing” (CEP) has emerged as a relevant new field of software engineering and computer science [1]. In fact, a lot of practical applications have the need to actively monitor vast quantities of event data to make automated decisions and take time-critical actions (the reader may refer, e.g., to the Proceedings of the RuleML Workshop Series). Complex Event Processing is discussed in depth in [2]. With cloud computing, the importance of event processing is even more visible, and a connection of “event pattern languages” with ontologies and with the semantic web is envisaged. Many of the current approaches are declarative and based on rules, and often on logic-programming-like languages and semantics: for instance, [3] is based upon a specifically defined interval-based Event Calculus [4].

Complex Event Processing is particularly important in software agents. Naturally, most agent-oriented languages architectures and frameworks are to some extent event-oriented and are able to perform event-processing. The issue of event processing agents (EPAs) is of growing importance in the industrial field, since agents and multi-agent systems are able to manage rapid change and thus to allow for scalability in applications aimed at supporting the ever-increasing level of interaction. In particular this paper is concerned with logical agents, i.e., agents whose syntax and semantics is rooted in Computational Logic. There are several approaches to logical agents (for a recent survey cf., e.g., [5]). For lack of space, we are not able here to properly discuss and compare their event-processing features. Rather, we recall only the approaches that have more strongly influenced the present work.

A recent but well-established and widely used approach to CEP in computational logic is ETALIS [6–8], which is an open source plug-in for Complex Event Processing
implemented prolog which runs in many prolog systems. ETALIS is in fact based on
a declarative semantics, grounded in Logic Programming. Complex events can be de-
duced from simpler events by means of deductive rules. ETALIS, in addition, supports
reasoning about events, context, and real-time complex situations, and has a nice repre-
sentation of time and time intervals aimed at stream reasoning. Relations among events
can be expressed via several operators, reminiscent of causal reasoning and Event Cal-
culus.

In logical agents, some relevant work about CEP is related to DALI, a logic agent-
oriented language (first appeared in 1999 [9]) that extends prolog with reactive and
proactive features [9–12] and is fully implemented and publicly available [13]. DALI,
like virtually all agent-oriented languages, is equipped with “event-condition-action”
rules (ECA rules) for defining the behavior of an agent in consequence of perception
of external events. The feature that makes DALI proactive and strongly event-oriented
is however the internal events construct: i.e., the programmer can indicate internal con-
ditions to be interpreted as events, to which a reaction can be defined. Management of
events which occur “together” (with the possibility to specify in which time interval),
priorities between events, and aggregation of simple events into complex ones via the
internal event construct are other useful features. Related to (the pre-existing) DALI
approach is for instance the work of [14], also providing ECA rules and a “snapshot”
semantics similar to the one already introduced in [12]. Work on CEP in DALI is pre-
sented in [15] and [16], which discuss the issue of of selecting different reactive patterns
by means of simple preferences, then extended to more complex forms of preferences in
[17]. Such preferences can be also defined in terms of “possible worlds” elicited from
a declarative description of a current or hypothetical situation, and can depend upon
past events, and the specific sequence in which they occurred. [18] and [19, 20] discuss
event-based memory-management, and temporal-logic-based constraints for complex
dynamic self-checking and reaction.

Teleo-Reactive Computing by Kowalski and Sadri [21–23] is an attempt to recon-
cile and combine conflicting approaches in logic programming, production systems,
active and deductive databases, agent programming languages, and the representation
of causal theories in AI. In this approach, enhanced reactive rules determine the inter-
action of an agent with the environment in a logical but not necessarily “just” deductive
way. The semantics relies upon an infinite Herbrand-like model which is incrementally
constructed.

The motivation of the present work relies in the observation that a complex event
cannot always be defined and recognized from simple deterministic incremental aggre-
gation of simple events. Rather, complex events sometimes require a more involved
and not necessarily deterministic description. In fact, in order to be flexible and ver-
satile instead of brittle and rigid an agent should be able to possibly interpret a set of
simple events in different ways, and to assign/learn the plausibility and reliability of
each interpretation (for a discussion of brittleness w.r.t. versatility in agents, cf. [24–
26]). For instance, in diagnosis a number of symptoms, if occurring together, might
lead to hypothesize the presence of one or more illness/fault, possibly with some cer-
tainty/probability. Another reason why event patterns should be carefully described is to
make an agent able to detect unknown events or wrong patterns, and take the necessary
counter-measures (see, e.g., [19, 20]).

In this paper, we propose a novel conceptual view of complex events and a possible
formalization of the new concepts. In our view, an agent should be able to possibly
interpret a set of simple events in different ways, and to assign/learn a plausibility and
reliability of each interpretation. We also consider complex actions, which we con-
sider as agent-generated events. To this aim, we propose to equip agents with specific
modules, that we call Event-Action modules, describing complex events and complex
actions. Such modules receives as “input” a set of simple events, either perceived by the
agent at its present stage of operation (external events), or generated internally by the
agent itself. Each module returns: (i) possible interpretations of a set of simple events
in terms of complex events; (ii) an ordering of such interpretations (if more than one is
possible) in terms of preferences and/or of certainty factors; (iii) detection of anomalies;
(iv) (sets of) actions to perform in response. Modules are (automatically) re-evaluated
whenever new instances of the “triggering” events become available.

From a practical point of view, for both providing a formal semantics and an imple-
mentation of modules we have presently adopted Answer Set Programming (ASP, cf.,
among many, [27, 28]). ASP is a well-established logic programming paradigm, where
a program may have several (rather than just one) “model”, called “answer set”, each
one representing a possible interpretation of the situation described by the program.
So, in the proposed framework, a logical agents can be seen as composed of a “main
program” including event processing, planning, action making, etc. and a set of Event-
Action modules (implemented in ASP) for event/action recognition, generation, aggre-
gation and management. We propose to exploit, in order to discern among alternatives,
both preferences and probabilities, refined via reinforcement learning. The approach
is encompassed into the general declarative semantics for logical agents introduced in
[12] (and summarized below). This makes our proposal immediately applicable to many
agent-oriented logic languages/framework.

To define Event-Action modules we presently refer to a quite simple logical setting
which can be in fact translated into ASP. However, in principle such modules might
be based upon more expressive logics (e.g., modal and/or probabilistic logics which
would certainly be suitable to express event recognition/aggregation). The structure
of our proposal is such that the basic concept, i.e. modules for (non-deterministic, defea-
sible) reasoning about event aggregation, might be easily rephased upon another logic.
Clearly, in such case a suitable semantic and execution model (other than ASP) should
then be provided.

The paper is organized as follows. In Section 2 we provide some background about
the building blocks of the proposed approach. In Section 3 we present the proposal,
and in Sections 4 and 5 its formalization, and some considerations about learning. In
Section 6 we conclude. This paper is the extended version of [29].

2 Background

In this section, we recall notions that define some basic foundation elements of the
proposed approach.
2.1 Evolutionary Semantics of Logical Agent-Oriented Languages

The Evolutionary semantics (introduced in [12]) is meant to provide a high-level general account of evolving agents, trying to abstract away from the details of specific agent-oriented frameworks. As seen below, the Evolutionary semantics in fact generalizes the specific semantic approaches underlying well-known logical agent-oriented languages. Actually, the Evolutionary semantics can be seen as a meta-semantics, as it accounts for agent evolution in terms of transformation steps. The precise definition of an agent and how a transformation step is determined and described attains to the specific agent-oriented language.

We define in fact, in very general terms, an agent as the tuple \( Ag = < P_A, E > \) where \( A \) is the agent name and \( P_A \) (that we call “agent program”, but can be in turn a tuple) describes the agent according to some agent-oriented formalism \( L \). \( E \) is the set of the events that the agent is able to recognize or determine (so, \( E \) includes actions that the agent is able to perform), according to the specific agent-oriented framework.

Let \( H \) be the history of an agent as recorded by the agent itself (in a form that will depend upon the specific agent-oriented framework), i.e., \( H \) includes agent’s perceptions and memories. We assume that events that either happen externally or are generated internally, actions which are performed, and other relevant agent’s activities are recorded in \( H \).

We assume that program \( P_A \) as written by the programmer is in general transformed into an initial agent program \( P_0 \) by means of an initialization phase (possibly doing nothing). When agent \( A \) is activated \( P_0 \) will go into execution, and will evolve according to the evolution of \( H \).

Evolution is represented via program-transformation steps, each one transforming \( P_i \) into \( P_{i+1} \) according to \( H_i \), which is the partial history up to stage \( i \). The choice of which elements of \( H_i \) do actually trigger an evolution step is part of the definition of a specific agent framework.

Thus, we obtain a Program Evolution Sequence \( PE = [P_0, \ldots, P_n, \ldots] \). The program evolution sequence will imply a corresponding Semantic Evolution Sequence \( ME = [M_0, \ldots, M_n, \ldots] \) where \( M_i \) is the semantics of \( P_i \) according to \( L \). Notice in fact that the approach is parametric w.r.t \( L \).

Definition 1 (Evolutionary semantics). Let \( Ag \) be an agent. The evolutionary semantics \( \varepsilon^{Ag} \) of \( Ag \) is a tuple \( (H, PE, ME) \), where \( H \) is the history of \( Ag \), and \( PE \) and \( ME \) are respectively its program and semantic evolution sequences.

The next definition introduces the notion of instant view of \( \varepsilon^{Ag} \), at a certain stage of the evolution (which is in principle of unlimited length).

Definition 2 (Evolutionary semantics snapshot). Let \( Ag \) be an agent, with evolutionary semantics \( \varepsilon^{Ag} = (H, PE, ME) \). The snapshot at stage \( i \) of \( \varepsilon^{Ag} \) is the tuple \( (H_i, P_i, M_i) \), where \( H_i \) is the history up to the events that have determined the transition from \( P_{i-1} \) to \( P_i \).

In [12], program transformation steps associated with DALI language constructs [10, 11] are defined in detail. They can easily be adapted to AgentSpeak (cf. [30] and

S. Costantini and R. Riveret. Complex Events and Actions in Logical Agents

259
the references therein). The evolutionary semantics may also express the notion of trace of a GOAL agent (cf. [31] and the references therein) where agent program $P_i$ encompasses the agent’s mental state and each evolution step, which in GOAL is called computation step and is determined by a conditional action. For 3APL (cf. [32] and the references therein), agent program $P_i$ encompasses the agent’s initial configuration, and the related sets GR of goal rules, PR of plan rules, IR of interactive rules; the evolutionary semantics corresponds to a 3APL agent computation run, and evolution steps are determined by the 3APL transition system.

In Section 4.1 we provide an integration of the proposed approach to Complex Event Processing into the Evolutionary semantics, in order to make it applicable to many agent-oriented logic programming languages among which the above-mentioned ones.

2.2 Background on Answer Set Semantics

The answer set semantics is used in this paper to provide both a formal semantics and an execution model for the proposed modules. In fact, this logic gives rise to answer set programming (ASP), which is a very well-established and fully logical programming paradigm. Many efficient solvers have become freely available for ASP, [33], each one proposing many extension to aid practical programming.

In the answer set semantics (originally named “stable model semantics”), a (logic) program $\Pi$ (cf., [34]) is a collection of rules of the form $H \leftarrow A_1, \ldots, A_n, \neg B_1, \ldots, \neg B_m$ where the $A_i$s $(i \leq n$ and the $B_j$s $(j \leq m$ are atoms, and in literals $\neg B_j$ not is default negation. A rule with empty body is called a fact. A rule with empty head is a constraint, where a constraint $\leftarrow L_1, \ldots, L_n$. states that literals $L_1, \ldots, L_n$ cannot be all true in an answer set.

The answer sets semantics is a view of a logic program as a set of inference rules (more precisely, default inference rules), or, equivalently, a set of constraints on the solution of a problem: each answer set represents a solution compatible with the constraints expressed by the program. In simple program $\{q \leftarrow \neg p \ p \leftarrow \neg q\}$, the first rule is read as “assuming that $p$ is false, we can conclude that $q$ is true.” This program has two answer sets. In the first one, $q$ is true while $p$ is false; in the second one, $p$ is true while $q$ is false. Thus, this program represent choice among two alternatives.

In the ASP (Answer Set Programming) paradigm, each answer set is seen as a solution of given problem, encoded as an ASP program. An ASP program is basically composed by program fragments like the above ones that generate the search space by defining available alternatives, and by constraints which prune the generated space thus selecting solutions. To find these solutions, one employs one of the several existing ASP solvers [33]. The expressive power of ASP and its computational complexity have been deeply investigated (cf., e.g., [35]).

3 Complex Event Processing in Logical Agent Languages

In this section we present our approach to advanced Complex Event Processing in logical agents. First, we recall how event processing and aggregation can be performed in most existing frameworks. Then, we argue that new methods and tools would be useful, and introduce our proposal.
3.1 Complex Events Recognition and Evaluation: Simple Event Aggregation

Event processing is traditionally based upon Event-Condition-Action (ECA) rules, of the form “IF Event then Reaction” where sometimes reaction is limited to performing a sequence of actions, sometimes is enlarged so as to allow forms of reasoning and various kinds of constraints. Most (virtually all) agent-oriented languages provide ECA rules of some form.

For instance, the following sample ECA rules defined in DALI evaluate medical symptoms, stating that to cope with high temperature one should assume an antipyretic, and to cope with cough one should take a syrup (clearly, we do not aim at medical precision). External events are indicated with postfix $E$, actions with postfix $A$, and connective $>$ indicates reaction. I.e., $>$ is a specific connective used to define ECA rules. Since DALI is an extension of prolog, the comma represents conjunction.

```
high_temperatureE > take_antipyreticA.
intense_coughE > take_cough_syrupA.
```

Plain ECA rules can hardly provide more than this kind of simple reaction. Agent languages may offer empowering mechanisms that allow for some form of CEP. In DALI for instance, by means of the “internal event” construct evaluation can become more involved. In the example we may, e.g., consider the possibility that a combination of symptoms suggests the occurrence of a more serious illness. For illustration purposes we assume to hypothesize the presence of either pneumonia, or just flu, or both. In DALI external events are, a short while after perception (that may or may not imply reaction), recorded as past events (postfix $P$), with a time-stamp. By $high\_temperatureP : [4\text{days}]$ it is intended that at least two past events of the form $high\_temperatureP : T1$ and $high\_temperatureP : T2$ have been recorded, where $T2 - T1$ is no less than four days. By default, the most recent records are considered, so the interval refers to the last and to the immediately previous measures of temperature (if the time-stamp of a past event is omitted, the most recent version is implicitly referred to).

The first rule below “reasons” about what has happened: if in fact both high temperature and cough have occurred, and high temperature has lasted for at least four days, one may conclude that occurrence of pneumonia should be suspected. Therefore, from the occurrence of a set of simple events, the occurrence of a complex event is inferred, as an internal event $suspect\_pneumoniaI$. Once inferred, such an event can be reacted to exactly like external ones by an ECA rule, the one below stating that the patient should take an antibiotic and consult a lung doctor.

```
suspect\_pneumoniaI :-
  high\_temperatureP : [4\text{days}],
  intense\_coughP,
  compatible(pneumonia, anamnesis).
suspect\_pneumoniaI > take\_antibioticA,
  consult\_lung\_doctorA.
```

Internal events are periodically re-evaluated at a certain (customizable) frequency. Between one attempt and the next one, the agent’s belief base will evolve as new events will have happened, reacted to and recorded. Thus, at each attempt an internal event can be either generated (and reacted to) or not, according to the agent’s belief state. In
the example, the suspect of pneumonia will not raise immediately, but rather after four days of high temperature measurements.

Clearly, the proposed example might be formalized also in ETALIS and in many other agent-oriented frameworks, each one offering its own improvements over simple ECAs in view of CEP. In summary, by means of ECA rules little can be done for CEP. By means of suitable empowering mechanisms such as the internal events, complex events can possibly be generated (and then reacted to) by reasoning on simple external events.

We believe however that such a simple pattern is often not sufficient, as there are many cases where the interpretation of sets of simple events is not univocal or straightforward. Therefore, different hypotheses about the meaning of a situation at hand should be generated and evaluated. We then propose a novel view and a novel formalization of event aggregation.

3.2 Complex Events Recognition and Evaluation: Event-Action Modules

Below we introduce the notion of Event and Action modules. For lack of space we illustrate these modules by means of two examples that we consider as representatives of significant situations. The first example concerns complex event recognition, the second one concerns complex actions generation. We do not intend to propose here a specific syntax for Event and Action modules: rather, we intend to point out what should be the elements and functions of such modules. In the sample syntax, adopted only for illustration purposes, as before postfix ‘P’ indicates past events, i.e., events which have occurred (after the ‘:’ there is the time-stamp or the interval of occurrence). Postfix ‘A’ indicates actions. Connective :> indicated Event-Condition-Action rules, while :- is the usual prolog if.

Event Interpretation

The following example illustrates an Event-Action Module that re-elaborates the previously-introduced example concerning medical diagnosis. This should allow the reader to appreciate the improvements over the simple formulation.

An Event-Action Module will be (re-)evaluated whenever the triggering events occur within a certain time interval, and according to specific conditions: in the example, the module is evaluated whenever in the last two days both high temperature and intense cough have been recorded.

EVENT-ACTION-MODULE

TRIGGER
(high_temperatureP AND intense_coughP) : [2days]

COMPLEX_EVENTS
suspect_flu OR suspect_pneumonia

suspect_flu :- high_temperatureP.

suspect_pneumonia :- high_temperatureP : [4days], intense_coughP.
PREFERENCES
suspect_flu :- patient_is_healthy.
suspect_pneumonia :- patient_is_at_risk.

ACTIONS
suspect_flu :-> stay_in_bedA.
suspect_flu, high_temperatureP : [4days], not suspect_pneumonia :-> take_antibioticA.
suspect_pneumonia :->
	take_antibioticA, consult_lung_doctorA.

MANDATORY
suspect_pneumonia :- high_temperatureP : [4days],
	suspect_fluP,
	take_antibioticP : [2days].

From given symptoms, either a suspect flu or a suspect pneumonia or both can be concluded, though for suspecting pneumonia high temperature should have lasted for at least four days, accompanied by intense cough. This is stated in the COMPLEX EVENTS section, where each of the listed complex events (in the example, suspect_flu and suspect_pneumonia) can be inferred, though according to the specified conditions. Notice that the whole agent’s belief base (including the history) is implicitly included in the definition of an Event-Action module. Explicit preferences are expressed in the PREFERENCES section, here stating that hypothesizing a flu should be preferred in case the patient is healthy, while pneumonia is more plausible for risky patients. If either none or both options hold, then they are equally preferred. More involved forms of preferences might be specified, that for lack of space we do not discuss here (cf., e.g., [17] and the references therein). Actions to undertake in the two cases are specified. As mentioned, the module is re-evaluated at subsequent new occurrences of high temperature and intense cough. Re-evaluation is performed on the (possibly) updated belief base.

Actions will be actually performed depending upon the outcome that the agent will prefer to choose. In particular, if a flu is suspected then the patient should stay in bed, and if the high temperature persists then an antibiotic should also be assumed (even if pneumonia is not suspected). In case of suspect pneumonia, an antibiotic is mandatory, plus a consult with a lung doctor.

The MANDATORY section of the module includes constraints, that may be of various kinds: in this case, it specifies which complex events must be mandatorily inferred in module (re)evaluations if certain conditions occur. Specifically, pneumonia is to be assumed mandatorily whenever flu has been previously assumed, but high temperature persists despite at least two days of antibiotic therapy.

Answer set modules for possibility and necessity (cf. [16]) find a fruitful integration in the present approach. In the example, it may be for instance that clinical history and conditions of a patient force to assume a pneumonia. A constraint such as the following might be added:
MANDATORY suspect_pneumonia :-
	Necessary(clinical_history, suspect_pneumonia).
Generating Complex Actions

We have noted before that event interpretation is not necessarily deterministic and straightforward. The same happens for actions: devising which actions and agent should perform can be highly context-dependent, and can be subjected to various kinds of uncertainty. To avoid brittleness, an agent should in our opinion be able to flexibly choose what to do in specific circumstances, and to dynamically adapt to changes of context/role/situation. This is why we propose to adapt previously introduced modules to action generation.

We illustrate the approach by means of an example related to what happens when two persons meet. The representation is simplistic and is meant only to illustrate purposes. A “real” encoding of the module below might be much more involved, and imply various kinds of knowledge representation methods, possibly including a theory of emotions.

In the sample representation, we assume that the person/agent who first gets sight of a person/agent (s)he knows possibly smiles, and then either simply waves or stops to shake hands. Section RELATED EVENTS specifies, as a boolean combination, events that may occur contextually to the triggering ones. There are some conditions, for instance that one may smil and/or waves if (s)he is neither in a bad temper nor angry at the other. Also, one who is in a hurry just waves, while good friends or people who meet each other in a formal setting should shake hands. Actions simply consist in returning what the other one does, and it is anomalous not doing so (e.g., if one smiles and the other does not smile back). In the formalization below, the expression meet_friend(A,F) means that agent A meets agent F; then, each one possibly makes some actions and the other one will normally respond. This module is totally revertible, in the sense that it manages both the case where “we” meet a friend and the case where vice versa somebody else meets us. This is the reason why in some module sections events have no postfixes. In fact, meet_friend(A,F), smile, wave and shake_hand are present events if a friend meets “us”, and are actions if “we” meet a friend. Postfixes appear in the ACTIONS and ANOMALY sections, where all elements (whatever their origin) have become past events to be coped with. The PRECONDITIONS section expresses action preconditions, via connective :<. Section MANDATORY defines obligations, here via a rule stating that it is mandatory to shake hands in a formal situation. The anomaly management section (left undefined here) may include counter-measures to be taken in case of unexpected behavior, that in the example may go from asking for explanation to getting angry, etc.

EVENT-ACTION-MODULE

TRIGGER meet_friend(A,F),
RELATED EVENTS
smile(A,F)OR (wave(A,F) XOR shake_hand(A,F))
4 Formalization of Event-Action Modules

In this section we provide a formalization of Event-Action modules based upon previously-introduced building blocks. To ensure integration of modules within the most common agent-oriented programming languages, we suitably merge them into the Evolutionary semantics. To provide (as a proof of concept) a formal semantics and an execution model, we define a translation of Event-Action modules into ASP modules.

4.1 Evolutionary Semantics Extended

We now briefly illustrate how to refine the Evolutionary semantics so as encompass the proposed approach. The agent program \(P_A\) becomes now a tuple including at least the “main” agent program \(P_M\), and the available Event-Action modules \(E_{A_1}, \ldots, E_{A_k}\).

**Definition 3 (Evolutionary Semantics).** Let \(Ag\) be an agent, defined by agent program

\[
P_A = (P_M, \langle E_{A_1}, \ldots, E_{A_k} \rangle, \ldots)
\]

where \(P_M\) is the main agent program, and \(E_{A_1}, \ldots, E_{A_k}, k \geq 0\), are the available Event-Action modules.
Its evolutionary semantics is \( \varepsilon_{\text{Ag}} = (H, P_{\text{Ext}}, M_E) \). The program evolution sequence, indicated by \( P_{\text{Ext}} \), stems from an Extended initial program \( P_{\text{Ext}_0} \) obtained from \( P_M \) by means of a a suitable initialization phase. In particular, \( P_{\text{Ext}_0} \) includes the following elements:

- Program \( P_0 \), obtained by adding to given main agent program the rules included in sections ACTIONS, ANOMALY_MANAGEMENT, ACTIONS and PRECONDITIONS of \( EA_1, \ldots, EA_k \).
- Tuple of ASP modules \( \langle \Pi_1, \ldots, \Pi_k \rangle \), where each \( \Pi_i \) is obtained by translating Event-Action module \( EA_i \) into ASP.

At the i-th evolution step, the agent’s history in general evolves, as new events/knowledge items may be recorded/removed. This determines an evolution of both the main agent program \( P_i \) and of the ASP modules: in fact, the main program and the modules implicitly encompass the agent’s history as the set of given facts. This implies that both the main program and modules semantics needs to be re-evaluated at each step. Each module will admit none, one or several answer sets, among which just one has to be selected. Such answer set will encompass a number of inferred complex events and actions, as well as possibly a number of anomalies, that have to be added to the agent’s history in order to be respectively reacted to (for events) or executed (for actions) or coped with (for anomalies). Precisely, at each step we have:

**Definition 4 (Evolutionary Semantics Snapshot).** The snapshot at stage \( i \) of \( \varepsilon_{\text{Ag}}^i \) is the tuple \( \langle H_i, P_{\text{Ext}_i}, M_i \rangle \), where \( M_i \) is in turn a tuple, i.e.,

\[
M_i = \langle F_i, S_1, \ldots, S_r \rangle
\]

where \( F_i \) is the semantics of \( P_i \), and \( S_i \) (\( i \leq r \leq k \)) is the set composed of the answer sets of each \( \Pi_i \) which has been (re-)evaluated at stage \( i \). The evolution step to stage \( i + 1 \) will imply: (i) the choice of one answer set \( A_i \) for each \( \Pi_i \) (selected among the elements of \( S_i \)); (ii) and the addition to \( H_{i+1} \) of all the complex events and actions and anomalies included in \( A_i \).

Therefore, at step \( i + 1 \), complex events, actions and anomalies generated at step \( i \) by each ASP module will be coped with as specified in the original corresponding Event-Action module, and then recorded as past events in the agent’s history.

In summary, the integration of Event-Action modules within a logical agent’s basic functioning can be described as follows.

- At every agent’s evolution step, the TRIGGER headline of each Event-Action module has to be checked, in order to state whether the module is to be re-evaluated. We cannot treat here complexity of this check, that will depend upon complexity of expressions involved.
- ASP modules corresponding to Event-Action modules that are to be (re-)evaluated will be fed to an answer set solver. A module will admit as a result of evaluation none, one or more answer sets.
If the module admits answer sets, each one will encompass a number of complex events, actions and anomalies. One answer set will be selected, according to preferences (if expressed within the module), or by random choice, or by informed choice deriving from a learning process, as discussed in Section 5. The rules for coping with the inferred complex events, actions and anomalies are defined in sections ACTIONS, ANOMALY MANAGEMENT, ACTIONS and PRECONDITIONS, which can be found in the main agent program.

4.2 ASP Representation of Modules

Each Event-Action module can be translated in a fully automated way into an ASP module. This except sections ACTIONS, ANOMALY MANAGEMENT, ACTIONS and PRECONDITIONS, whose content as seen before has to be added to the main agent program.

The answer set program (module) $\Pi$ corresponding to a given Event-Action module will be obtained by translating into ASP the contents of sections COMPLEX EVENTS, CHECK, RELATED EVENTS, ANOMALIES and MANDATORY. The translation is straightforward and can be fully automated.

For the sake of simplicity we outline a translation into basic ASP, thus not considering the various additional useful constructs that have been introduced in the literature and in the implementations. It can be noted that extensions of ASP oriented to stream reasoning have been defined (cf., e.g., [36]). We observe however that we do not perform stream reasoning, rather we perform periodical re-evaluation of modules, so at present we do not deem it necessary to resort to such approach.

For lack of space we cannot properly describe how to cope with time intervals: however, our method consists in representing time-stamps as additional arguments of ASP predicates representing events. Intervals are then computed by trivial arithmetic constraints.

The translation of (the relevant sections of) an Event-Action module into ASP can be performed by means of the following guidelines (a formal definition of the translation, including the treatment of time intervals, is deferred to an extended version of this paper).

- **conj** In ASP, the conjunction among a number of elements $a_1, \ldots, a_n$ is simply expressed as $\text{conj} \leftarrow a_1, \ldots, a_n$.

- **or-xor** Disjunction among two elements $a$ and $b$ is expressed by the cycle $a \leftarrow b \ b \leftarrow a$. This disjunction is not exclusive, since either $a$ or $b$ or both might be derived elsewhere in the program. To obtain exclusive disjunction, a constraint $\leftarrow a, b$ must be added. A constraint in ASP can be read as *it cannot be that...*. In the case of exclusive disjunction, it cannot be that both $a$ and $b$ belong to the same answer set. Disjunction (also exclusive) can be expressed also on several elements.

- **choice** Choice, or possibility, or hypothesis, expressing that some element $a$ may or may not be included in an answer set, can be expressed by means of a cycle involving a fresh atom, say $na$. The cycle is of the form $a \leftarrow na \ na \leftarrow a$. Therefore, an answer set will contain either $a$ or $na$, the latter signifying the absence of $a$. 

S. Costantini and R. Riveret. Complex Events and Actions in Logical Agents

267
A choice that can be done (by pattern choice) on element a only if certain conditions Conds are satisfied, is expressed by a choice pattern plus a rule c ← Conds and a constraint ← a, not c. The constraint states that a cannot be hypothesized in an answer set if c does not hold, i.e., if Conds are not implied by that answer set.

Mandatory presence in an answer set of atom a defined by rule a ← Body whenever Body is implied by that answer set can be obtained as follows. In addition to the defining rule a ← Body, a constraint must be added of the form ← not a, Body stating that it cannot be that an answer set implies Body but does not contain a. The constraint is necessary for preventing a to be ruled out by some other condition occurring elsewhere in the program.

Events in the RELATED EVENTS section can be expressed by means of the choice pattern, and their combinations via the conj and or-xor patterns. Constraints in the MANDATORY can be expressed by means of the mand pattern.

Section COMPLEX EVENTS is coped with by the choice and choyf patterns.

Sections CHECK and ANOMALIES can be translated by a plain transposition of their rules into ASP, possibly exploiting the conj and or-xor patterns.

5 Evaluating Outcomes and Reinforcement Learning

Outcomes of an Event-Action Module are often not univocal: thus, at each stage the agent face a choice among the answer sets of corresponding ASP module Π. As we have seen in the example, preferences may help the agent in choosing an outcome rather than another one. However, knowledge can be incomplete or partial about reliability of such preferences, and in general about plausibility of the choice. This choice is akin to the “multi-armed bandit problem” [37], and thus machine learning techniques can be used so that an agent will learn over time to make the ‘best’ choice over the answer sets of each module Π (i.e., over the outcomes of each Event-Action Module).

Therefore, a self-improving process is in order, and for this purpose, an agent can be endowed with a simple reinforcement learning mechanism [38]: at each evolution step, occurring at a time t, an agent: (i) senses its environment; (ii) re-evaluates Π obtaining the answer sets A₁, …, Aₖ, k > 0; (iii) adopt one of them, say A, and evaluates its “quality”, denoted with Q(A). In order to evaluate the quality of an answer set, we assume that at stage t, a numerical value denoted V(A) is associated to it: this value can be either epistemic or practical. Its expression shall be dependent on the application so we leave it unspecified in the description of the present general framework. At time t, the quality of the selected answer A will be computed as a discounted moving average of its value over time:

\[ Q^{t+1}(A) = Q^t(A) + \alpha \cdot (V^t(A) - Q^t(A)) \]

where α is the discount rate. Then, an agent draws an answer set A amongst all the possible answer sets A₁, …, Aₖ of Π with probability P(A). This probability can be computed using a Gibbs-Boltzmann distribution:

\[ P^t(M) = e^{Q^t(A)/\tau} / \sum_i e^{Q^t(A_i)/\tau} \]
where $\tau$ is a ‘learning temperature’ to balance the exploitation and the exploration of possible models.

6 Concluding Remarks

In summary, in this paper we have introduced kinds of Event-Action modules that allow an agent to: aggregate simple events into complex ones, also according to constraints; check events that occur w.r.t. expectations; cope with events possibly occurring contextually to certain other events; detect anomalies; decide actions to be performed in both normal or anomalous cases, according to a number of issues among which we may include context, role, circumstances, past experience, etc.

In future work, other event aggregation and recognition patterns might be introduced. The approach might be extended to a more involved definition of complex actions, and to the choice among possible action patterns. The proposed simple learning mechanism might be refined based on experimentations on suitable test cases. We may notice that the approach is basically independent of the underlying logic, so that more expressive (e.g., modal) logic might be employed in future extensions. Clearly, a semantics and execution model going beyond ASP should then be provided.

We intend to introduce forms of ‘deep’ learning of Event-Action modules. In our intention, an Event-Action Modules might be initially defined in a tentative or embryonic form: then, module elements might be learnt via a training phase, and refined via reinforcement learning during the agent’s ‘life’ thus adding to agent’s flexibility and adaptability. To this aim, we have been developing suitable argumentation-based learning techniques [39].

We do not intend to claim that all events can be recognized and reacted to, and all actions generated, only in a logic-based way. Nevertheless, also based upon relevant literature, we claim that event/actions recognition, generation and management often require forms of (even non-trivial) reasoning. Overall, any really “intelligent” agent capable of flexible interaction with complex real-world environment will presumably result from a graceful integration of several kind of components, possibly based upon different approaches.

References


24. Brachman, R.J.: (AA)AI more than the sum of its parts. AI Magazine 27(4) (2006) 19–34


Argumentation for Propositional Logic and Nonmonotonic Reasoning

Antonis Kakas, Francesca Toni, and Paolo Mancarella

1 University of Cyprus, Cyprus
antonis@cs.ucy.ac.cy
2 Imperial College London, UK
ft@imperial.ac.uk
3 Università di Pisa, Italy
paolo@di.unipi.it

Abstract. Argumentation has played a significant role in understanding and unifying under a common framework different forms of defeasible reasoning in AI. Argumentation is also close to the original inception of logic as a framework for formalizing human argumentation and debate. In this context, the purpose of this paper is twofold: to draw a formal connection between argumentation and classical reasoning (in the form of Propositional Logic) and link this to support defeasible, Non-Monotonic Reasoning in AI. To this effect, we propose Argumentation Logic and show properties and extensions thereof.

1 Introduction

Over the past two decades argumentation has played a significant role in understanding and unifying under a common framework defeasible Non-Monotonic Reasoning (NMR) in AI [16, 10, 3]. Moreover, a foundational role for argumentation has emerged in the context of problems requiring human-like commonsense reasoning, e.g. as found in the area of open and dynamic multi-agent systems to support context-dependent decision making, negotiation and dialogue between agents (e.g. see [14, 9]). This foundational role of argumentation points back to the original inception of logic as a framework for formalizing human argumentation.

This paper reexamines the foundations of classical logical reasoning from an argumentation perspective, by formulating a new logic of arguments, called Argumentation Logic (AL), and showing how this relates to Propositional Logic. AL is formulated by bringing together argumentation theory from AI and the syllogistic view of logic in Natural Deduction (ND). Its definition rests on a reinterpretation of Reductio ad Absurdum (RA) through a suitable argumentation semantics. One consequence of this is that in AL the implication connective behaves like a default rule that still allows a form of contrapositive reasoning. The reasoning in AL is such that the ex-falso rule where everything can be derived from an inconsistent theory does not apply and hence an inconsistent part of a theory does not trivialize the whole reasoning with that theory.
The main motivation for studying this argumentation perspective on logical reasoning is to examine its use to bring together classical reasoning and non-monotonic commonsense reasoning into a single unified framework. The paper presents a preliminary investigation into building such a NMR framework based on AL that integrates into the single representation framework of AL classical reasoning, as in Propositional Logic including forms of Reductio ad Absurdum, with defeasible reasoning. In particular, we study, in the context of examples, the possible use of preferences over sentences of an AL theory to capture NMR defeasible reasoning and naturally combine this with the classical reasoning of AL. Our vision is for all forms of reasoning to be captured in the argumentation reasoning of AL and its extensions with preferences.

This paper is an extract of [13].

2 Preliminaries on Natural Deduction

Let $\mathcal{L}$ be a Propositional Logic language and $\vdash$ denote the provability relation of Natural Deduction (ND) in Propositional Logic. Throughout the paper, theories and sentences will always refer to theories and sentences wrt $\mathcal{L}$. We assume that $\bot$ stands for $\phi \land \neg \phi$, for any $\phi \in \mathcal{L}$.

**Definition 1.** Let $T$ be a theory and $\phi$ a sentence. A direct derivation for $\phi$ (from $T$) is a ND derivation of $\phi$ (from $T$) that does not contain any application of RA. If there is a direct derivation for $\phi$ (from $T$) we say that $\phi$ is directly derived (or derived modulo RA) from $T$, denoted $T \vdash_{MRA} \phi$.

**Example 1.** Let $T_1 = \{\alpha \rightarrow (\beta \rightarrow \gamma)\}$. The following is a direct derivation for $\alpha \land \beta \rightarrow \gamma$ (from $T_1$):

\[
\begin{array}{c}
\hline
\lbrack \alpha \land \beta \rbrack \quad \text{hypothesis} \\
\alpha \quad \land E \\
\alpha \rightarrow (\beta \rightarrow \gamma) \text{ from } T \\
\beta \rightarrow \gamma \quad \rightarrow E \\
\beta \quad \land E \\
\gamma ] \\
\hline \\
\alpha \land \beta \rightarrow \gamma \\
\rightarrow I \\
\end{array}
\]

Thus, $T_1 \vdash_{MRA} \alpha \land \beta \rightarrow \gamma$ (and, trivially, $T_1 \vdash \alpha \land \beta \rightarrow \gamma$). Consider now $T_2 = \{\neg(\alpha \lor \beta)\}$. The following

\[
\begin{array}{c}
\hline
\lbrack \alpha \rbrack \quad \text{hypothesis} \\
\alpha \lor \beta \quad \lor I \\
\neg(\alpha \lor \beta) \text{ from } T \\
\bot ] \\
\hline \\
\neg \alpha \\
\land I \\
\hline \\
\neg \alpha \\
\rightarrow RA \\
\end{array}
\]

is a ND derivation of $\neg \alpha$ that is not direct. Since there is no direct derivation of $\neg \alpha$, $T_2 \vdash \neg \alpha$ but $T_2 \not\vdash_{MRA} \neg \alpha$.

\(^4\) See the appendix for a review of the ND rules that we use.
Definition 2. A theory $T$ is classically inconsistent iff $T \vdash \bot$. A theory $T$ is directly inconsistent iff $T \vdash_{MRA} \bot$. A theory $T$ is (classically or directly) consistent iff it is not (classically or directly, respectively) inconsistent.

Trivially, if a theory is classically consistent then it is directly consistent. However, a directly consistent theory may be classically inconsistent, as the following example shows.

Example 2. Consider $T_1 = \{\alpha \rightarrow \bot, \neg \alpha \rightarrow \bot\}$. $T_1$ is classically inconsistent, since $T_1 \vdash \bot$ as follows:

$$
\begin{array}{c}
\hypothesis \alpha \\
\alpha \rightarrow \bot \ \text{from } T \\
\bot \ \rightarrow E \\
\neg \alpha \ \RA \\
\hypothesis \neg \alpha \\
\neg \alpha \rightarrow \bot \ \text{from } T \\
\bot \ \rightarrow E \\
\alpha \ \RA \\
\bot \ \land I
\end{array}
$$

However, $T_1$ is directly consistent, since $T_1 \not\vdash_{MRA} \bot$. Consider instead $T_2 = \{\alpha, \neg \alpha\}$. $T_2$ is classically and directly inconsistent, since $T_2 \not\vdash_{MRA} \bot$, as follows:

$$
\begin{array}{c}
\alpha \ \text{from } T \\
\neg \alpha \ \text{from } T \\
\bot \ \land I
\end{array}
$$

We will use a special kind of ND derivations, that we call Reduction ad Absurdum Natural Deduction (RAND). These are ND derivations with an outermost application of RA:

Definition 3. A RAND derivation of $\neg \phi \in \mathcal{L}$ from $T$ is a ND derivation of $\neg \phi$ from $T$ of the form

$$
\begin{array}{c}
\hypothesis \phi \\
\vdots \\
\bot \vdots \\
\neg \phi \ \RA
\end{array}
$$

A sub-derivation (of some $\psi \in \mathcal{L}$) of a derivation $d$ is a RAND sub-derivation of $d$ iff it is a RAND derivation.

The ND derivation of $\neg \alpha$ given $T_2$ in example 1 is a RAND derivation. Also, given $T_1$ in example 2, the sub-derivations\(^5\)

$$
\begin{array}{c}
\alpha \\
\alpha \rightarrow \bot \\
\bot \\
\neg \alpha \\
\alpha
\end{array}
\quad
\begin{array}{c}
\neg \alpha \\
\neg \alpha \rightarrow \bot \\
\bot \\
\alpha
\end{array}
$$

of the derivation (d) of $\bot$ are RAND sub-derivations (of d).

---

\(^5\) If clear from the context, we omit to give the ND rules used.
3 Argumentation Logic Frameworks

Given a propositional theory we will define a corresponding argumentation framework as follows.

**Definition 4.** The argumentation logic (AL) framework corresponding to a theory $T$ is the triple $(\text{Args}^T, \text{Att}^T, \text{Def}^T)$ defined as follows:

- $\text{Args}^T = \{T \cup \Sigma | \Sigma \subseteq \mathcal{L}\}$ is the set of all extensions of $T$ by sets of sentences in $\mathcal{L}$;
- given $a, b \in \text{Args}^T$, with $a = T \cup \Delta$, $b = T \cup \Gamma$, such that $\Delta \neq \emptyset$, $(b, a) \in \text{Att}^T$ iff $a \models_{\text{MRA}} b$;
- given $a, d \in \text{Args}^T$, with $a = T \cup \Delta$, $(d, a) \in \text{Def}^T$ iff
  1. $d = T \cup \{\neg \phi\}$ ($d = T \cup \{\phi\}$) for some sentence $\phi \in \Delta$ (respectively $\neg \phi \in \Delta$), or
  2. $d = T \cup \emptyset$ and $a \not\models_{\text{MRA}} b$.

In the remainder, $b$ attacks $a$ (wrt $T$) stands for $(b, a) \in \text{Att}^T$ and $d$ defends or is a defence against $a$ (wrt $T$) stands for $(d, a) \in \text{Def}^T$.

Note also that, since $T$ is fixed, we will often equate arguments $T \cup \Sigma$ to sets of sentences $\Sigma$. So, for example, we will refer to $T \cup \{\} = T$ as the empty argument. Similarly, we will often equate a defence to a set of sentences. In particular, when $d = T \cup \Gamma$ defends/is a defence against $a = T \cup \Delta$ we will say that $\Gamma$ defends/is a defence against $\Delta$ (wrt $T$).

The attack relation between arguments is defined in terms of a direct derivation of inconsistency. Note that, trivially, for $a = T \cup \Delta$, $b = T \cup \Gamma$, $(b, a) \in \text{Att}^T$ iff $T \cup \Delta \cup \Gamma \not\models_{\text{MRA}} \bot$. The following example illustrates our notion of attack:

**Example 3.** Given $T_1 = \{\alpha \rightarrow (\beta \rightarrow \gamma)\}$ in example 1, $\{\alpha, \beta\}$ attacks $\{\neg \gamma\}$ (and vice-versa), $\{\alpha, \neg \gamma\}$ attacks $\{\beta\}$ (and vice-versa), $\{\alpha, \neg \alpha\}$ attacks $\{\gamma\}$ (and vice-versa) as well as any non-empty set of sentences (and vice-versa).

Note that the attack relationship is symmetric except for the case of the empty argument. Indeed, for $a, b$ both non-empty, it is always the case that $a$ attacks $b$ iff $b$ attacks $a$. However, the empty argument cannot be attacked by any argument (as the attacked argument is required to be non-empty), but the empty argument can attack an argument. For example, given $T_2 = \{\alpha, \neg \alpha\}$ in example 2, $\{\}$ attacks $\{\alpha\}$ and $\{\}$ attacks $\{\beta\}$ (for any $\beta \in \mathcal{L}$), since $T \not\models_{\text{MRA}} \bot$. Finally, note that our notion of attack includes the special case of attack between a sentence and its negation, since, for any theory $T$, $\{\phi\}$ attacks $\{\neg \phi\}$ (and vice-versa), for any $\phi \in \mathcal{L}$.

The notion of defence is a subset of the attack relation. In the first case of the definition we defend against an argument by adopting the complement$^6$ of some sentence in the argument, whereas in the second case we defend against any directly inconsistent set using the empty argument. Then, in example 3, $\{\neg \alpha\}$

$^6$ The complement of a sentence $\phi$ is $\neg \phi$ and the complement of a sentence $\neg \phi$ is $\phi$. 

275
defends against the attack \(\{\alpha, \beta\}\) and \(\{\}\) defends against the (directly inconsistent) attack \(\{\alpha, \neg \alpha\}\). Note that the empty argument cannot be defended against if \(T\) is directly consistent. Finally, note that when \(T\) is directly inconsistent the attack and defence relationships trivialise, in the sense that any two non-empty arguments attack each other, the empty argument attacks any argument, and any argument can be defended against by the empty argument.

4 Argumentation Logic

In this section we assume that \(T\) is directly consistent.

As conventional in argumentation, we define a notion of acceptability of sets of arguments to determine which conclusions can be dialectically justified (or not) from a given theory. Our definition of acceptability and non-acceptability is formalised in terms of the least fix point of (monotonic) operators on the cartesian product of the set of arguments/sentences in \(L\), as follows:

Definition 5. Let \(\langle \text{Args}^T, \text{Att}^T, \text{Def}^T \rangle\) be the AL framework corresponding to a directly consistent theory \(T\), and \(R\) the set of binary relations over \(\text{Args}^T\).

- The acceptability operator \(A_T: R \rightarrow R\) is defined as follows: for any \(\text{acc} \in R\) and \(a, a_0 \in \text{Args}^T\):
  \[(a, a_0) \in A_T(\text{acc})\] if
  - \(a \subseteq a_0\), or
  - for any \(b \in \text{Args}^T\) such that \(b\) attacks \(a\) wrt \(T\),
    - \(b \nsubseteq a_0 \cup a\), and
    - there exists \(d \in \text{Args}^T\) that defends against \(b\) wrt \(T\) such that \((d, a_0 \cup a) \in \text{acc}\).

- The non-acceptability operator \(N_T: R \rightarrow R\) is defined as follows: for any \(\text{nacc} \in R\) and \(a, a_0 \in \text{Args}^T\):
  \[(a, a_0) \in N_T(\text{nacc})\] if
  - \(a \nsubseteq a_0\), and
  - there exists \(b \in \text{Args}^T\) such that \(b\) attacks \(a\) wrt \(T\) and
    - \(b \subseteq a_0 \cup a\), or
    - for any \(d \in \text{Args}^T\) that defends against \(b\) wrt \(T\), \((d, a_0 \cup a) \in \text{nacc}\).

These \(A_T\) and \(N_T\) operators are monotonic wrt set inclusion and hence their repeated application starting from the empty binary relation will have a least fixed point.

Definition 6. \(\text{ACC}^T\) and \(\text{NACC}^T\) denote the least fixed points of \(A_T\) and \(N_T\) respectively. We say that \(a\) is acceptable wrt \(a_0\) in \(T\) iff \(\text{ACC}^T(a, a_0)\), and \(a\) is not acceptable wrt \(a_0\) in \(T\) iff \(\text{NACC}^T(a, a_0)\).

Thus, given the AL framework \(\langle \text{Args}^T, \text{Att}^T, \text{Def}^T \rangle\) (for \(T\) directly consistent) and \(a, a_0 \in \text{Args}^T\):
\[\text{ACC}^T(a, a_0), \text{iff} \]
\[\bullet a \subseteq a_0, \text{ or} \]
\[\bullet \text{for all } b \in \text{Args}^T \text{ such that } b \text{ attacks } a:\]
\[\quad \ast b \not\subseteq a_0 \cup a, \text{ and} \]
\[\quad \ast \text{there exists } d \in \text{Args}^T \text{ such that } d \text{ defends against } b \text{ and } \text{ACC}^T(d, a_0 \cup a);\]

\[\text{NACC}^T(a, a_0), \text{iff} \]
\[\bullet a \not\subseteq a_0 \text{ and} \]
\[\bullet \text{there exists } b \in \text{Args}^T \text{ such that } b \text{ attacks } a \text{ and} \]
\[\quad \ast b \subseteq a_0 \cup a, \text{ or} \]
\[\quad \ast \text{for all } d \in \text{Args}^T \text{ such that } d \text{ defends against } b \text{ it holds that } \text{NACC}^T(d, a_0 \cup a).\]

We will often equate the (non-)acceptability of an argument \(T \cup \Delta\) wrt an argument \(T \cup \Delta_0\) to the (non-)acceptability of the set of sentences \(\Delta\) wrt the set of sentences \(\Delta_0\).

Note that non-acceptability, \(\text{NACC}^T(a, a_0)\), is the same as the classical negation of \(\text{ACC}^T(a, a_0)\), i.e. \(\text{NACC}^T(a, a_0) = \neg \text{ACC}^T(a, a_0)\). We choose to give the definition of non-acceptability explicitly to aid readability. We will use these two versions of non-acceptability interchangeably.

Note that the empty argument is always acceptable, wrt any other argument. Note also that the “canonical” attack of a sentence on its complement (i.e. of \(T \cup \{\phi\}\) on \(T \cup \{\neg \phi\}\) and vice-versa) does not affect the acceptability relation as it can always be defended against by this complement itself.

The following examples illustrate non-acceptability.

\[
\begin{align*}
\{\neg \beta\} & \quad (\text{since } T \cup (\neg \beta) + MRA \perp) \\
\emptyset & \quad (\text{since } T \cup (\alpha) \cup (\beta) + MRA \perp) \\
\{\beta\} & \quad (\text{since } T \cup (\neg \beta) + MRA \perp) \\
\{\neg \beta\} & \quad (\text{since } T \cup (\neg \beta) + MRA \perp) \\
\emptyset & (\text{since } T \cup (\neg \beta) + MRA \perp)
\end{align*}
\]

\textbf{Fig. 1.} Illustration of \(\text{NACC}^T(\{\neg \beta\}, \emptyset)\) (left) and \(\text{NACC}^T(\{\alpha\}, \emptyset)\) (right), for example 4.

\textbf{Example 4.} Let \(T = \{\alpha \land \beta \rightarrow \bot, \neg \beta \rightarrow \bot\}\). \(T\) is classically and directly consistent, \(T \cup \{\neg \beta\}\) is classically and directly inconsistent, and \(T \cup \{\alpha\}\) is classically inconsistent but directly consistent. It is easy to see that \(\text{NACC}^T(\{\neg \beta\}, \emptyset)\) holds, as illustrated in figure 1 (left)\(^\dagger\), since \(\{\neg \beta\} \not\subseteq \emptyset\), \(b = \emptyset\) attacks \(\{\neg \beta\}\)

\(^\dagger\) Here and throughout the paper we adopt the following graphical convention: \(\uparrow\) denotes an attack and \(\uparrow\uparrow\) denotes a defence.
and $\{} \subseteq \{\neg \beta\}$. Also, $NACC^T(\{\alpha\}, \{})$ holds, as illustrated in figure 1 (right).
Indeed:

- since $\{\alpha\} \not\subseteq \{}$, $b = \{\beta\}$ attacks $\{\alpha\}$ and $\{\neg \beta\}$ is the only defence against $b$, to prove that $NACC^T(\{\alpha\}, \{})$ it suffices to prove that $NACC^T(\{\neg \beta\}, \{\alpha\})$;
- since $\{\neg \beta\} \not\subseteq \{\alpha\}$, $b = \{\} \attacks \{\neg \beta\}$ and $\{} \subseteq \{\alpha, \neg \beta\}$, it follows that $NACC^T(\{\neg \beta\}, \{\alpha\})$ holds as required.

Note that if an argument $a$ is attacked by the empty argument, then it is acceptable wrt $a_0$ iff $a \subseteq a_0$, since there is no defence against the empty argument.

This observation is used in the following example.

**Example 5.** Given $T = T_1 = \{\alpha \rightarrow \bot, \neg \alpha \rightarrow \bot\}$ in example 2, $NACC^T(\{\alpha\}, \{})$ and $NACC^T(\{\neg \alpha\}, \{})$ both hold. Indeed, for $NACC^T(\{\alpha\}, \{}), \{\alpha\}$ is attacked by $\{\}$.

The following example shows a case of non-acceptability making use of a non-empty attack for the base case.

**Example 6.** Let $T = \{\alpha \land \neg \beta \rightarrow \bot, \beta \land \gamma \rightarrow \bot, \alpha \land \beta \land \neg \gamma \rightarrow \bot\}$. $T$ is classically (and directly) consistent, and $T \cup \{\alpha\}$ is classically inconsistent but directly consistent. $NACC^T(\{\alpha\}, \{})$ holds, as illustrated in figure 2. Indeed:

\[
\begin{align*}
\{\alpha\} & \quad (\text{since } T \cup \{\alpha\} \cup \{\neg \beta\} \models MRA \bot) \\
\{\neg \beta\} & \\
\{\beta\} & \quad (\text{since } T \cup \{\beta\} \cup \{\gamma\} \models MRA \bot) \\
\{\gamma\} & \\
\{\neg \gamma\} & \quad (\text{since } T \cup \{\alpha, \beta\} \cup \{\neg \gamma\} \models MRA \bot) \\
\{\alpha, \beta\} & 
\end{align*}
\]

**Fig. 2.** Illustration of $NACC^T(\{\alpha\}, \{})$ for example 6.

- since $\{\alpha\} \not\subseteq \{}$, $b = \{\neg \beta\}$ attacks $\{\alpha\}$ and $\{\beta\}$ is the only defence against $b$, to prove that $NACC^T(\{\alpha\}, \{})$ it suffices to prove that $NACC^T(\{\beta\}, \{\alpha\})$;
- since $\{\beta\} \not\subseteq \{\alpha\}$, $b' = \{\gamma\}$ attacks $\{\beta\}$ and $\{\neg \gamma\}$ is the only defence against $b'$, to prove $NACC^T(\{\beta\}, \{\alpha\})$ it suffices to prove $NACC^T(\{\neg \gamma\}, \{\alpha, \beta\})$;
- since $\{\neg \gamma\} \not\subseteq \{\alpha, \beta\}$, $b'' = \{\alpha, \beta\}$ attacks $\{\neg \gamma\}$ and $b'' \subseteq \{\alpha, \beta, \neg \gamma\}$,

$NACC^T(\{\neg \gamma\}, \{\alpha, \beta\})$ and so $NACC^T(\{\beta\}, \{\alpha\})$ and $NACC^T(\{\alpha\}, \{})$ hold.

The following example illustrates non-acceptability in the case of an empty (and thus classically consistent) theory.

278
Example 7. For \( T = \{ \} \), \( \text{NACCT}(\{\neg(\beta \lor \neg \beta)\}, \{\}) \) holds, as illustrated in figure 3. Also, trivially, \( \text{NACCT}(\{\beta \land \neg \beta\}, \{\}) \) holds, since it is attacked by the empty argument.

\[
\begin{align*}
&\{\neg(\beta \lor \neg \beta)\} \\
&\{\neg \beta\} \\
&\{\beta\} \\
&\{\neg(\beta \lor \neg \beta)\}
\end{align*}
\]

Fig. 3. Illustration of \( \text{NACCT}(\{\neg(\beta \lor \neg \beta)\}, \{\}) \) for example 7.

A novel, alternative notion of *entailment* can be defined for theories that are directly consistent in terms of the (non-) acceptability semantics for AL frameworks, as follows:

**Definition 7.** Let \( T \) be a directly consistent theory and \( \phi \in \mathcal{L} \). Then \( \phi \) is AL-entailed by \( T \) (denoted \( T \models_{AL} \phi \)) iff \( \text{ACC}^T(\{\phi\}, \{\}) \) and \( \text{NACCT}(\{\neg \phi\}, \{\}) \).

This is motivated by the argumentation perspective, where an argument is held if it can be successfully defended and it cannot be successfully objected against.

In the remainder of the paper we will study properties of \( \models_{AL} \) and discuss extensions thereof to support NMR.

5 Basic Properties

The following result gives a core property of the notion of AL-entailment wrt the notion of direct derivation in Propositional Logic, for directly consistent theories.

**Proposition 1.** Let \( T \) be a directly consistent theory and \( \phi \in \mathcal{L} \) such that \( T \models_{MRA} \phi \). Then \( T \models_{AL} \phi \).

**Proof:** Let \( a = T \cup \Delta \) be any attack against \( \{\phi\} \), i.e. \( T \cup \{\phi\} \cup \Delta \models_{MRA} \bot \). Since \( T \models_{MRA} \phi \), then \( T \cup \Delta \models_{MRA} \bot \). Since \( T \) is directly consistent, \( \Delta \neq \{\} \). Hence any such \( a \) can be defended against by the empty argument. Since \( \text{ACC}^T(\{\}, \Sigma) \), for any \( \Sigma \subseteq \mathcal{L} \), then \( \text{ACC}^T(\{\phi\}, \{\}) \) holds. Moreover, since \( T \models_{MRA} \phi \), necessarily \( T \cup \{\neg \phi\} \models_{MRA} \bot \). Hence the empty argument attacks \( \{\neg \phi\} \) and thus \( \text{NACCT}(\{\neg \phi\}, \{\}) \) holds.

QED
Theorem 1. Let $T$ be a directly consistent theory and $\phi \in \mathcal{L}$. If $NACC^T(\{\neg\phi\}, \{\})$ holds then there exists a RAND derivation of $\neg\phi$ from $T$.\footnote{The other half of this result shows how (under some conditions) a RAND derivation of $\neg\phi$ implies $NACC^T(\{\phi\}, \{\})$, proven in [12].}

\[\begin{array}{c}
[\alpha] \\
[\neg\beta] \\
\bot \\
\neg\beta \\
\beta \\
\alpha \land \beta \\
\bot \\
\neg\alpha \\
\bot \\
\end{array} \quad \begin{array}{c}
[\alpha] \\
[\beta] \\
c(\alpha) \\
\alpha \land \beta \\
\bot \\
\neg\beta \\
\bot \\
\end{array}\]

\textbf{Fig. 4.} Two RAND derivations of $\neg\alpha$ in example 4: $d_1$ (left) and $d_2$ (right).

For example, the RAND derivation corresponding to the proof of $NACC^T(\{\alpha\}, \{\})$ in figure 1 is $d_1$ in figure 4.\footnote{The proof of this theorem is included in [13].} Here, the inner RAND derivation in $d_1$ corresponds to the non-acceptability of the defence $\{\neg\beta\}$ against the attack $\{\beta\}$ against $\{\alpha\}$. Derivation $d_2$ in figure 1 is an alternative RAND of $\neg\alpha$, but this cannot be obtained from any proof of $NACC^T(\{\alpha\}, \{\})$, because there is a defence against the attack $\{\beta\}$ given by the empty set (in other words, $d_2$ does not identify a useful attack, that cannot be defenced against, for proving non-acceptability).

6 AL for Propositional Logic

The following result gives a core property of the notion of non-acceptability for \textit{classically consistent} theories.

Proposition 2. Let $T$ be classically consistent and $\phi \in \mathcal{L}$. If $NACC^T(\{\neg\phi\}, \{\})$ holds then $ACC^T(\{\phi\}, \{\})$ holds.

\textbf{Proof:} By theorem 1, since $NACC^T(\{\neg\phi\}, \{\})$, then $T \vdash \phi$. Suppose, by contradiction, that $ACC^T(\{\phi\}, \{\})$ does not hold. Then $NACC^T(\{\phi\}, \{\})$ holds (since $NACC^T(\{\phi\}, \{\})=\neg ACC^T(\{\phi\}, \{\})$) and by theorem 1 there is a RAND derivation of $\neg\phi$ from $T$ and thus $T \vdash \neg\phi$. This implies that $T$ is classically consistent.
inconsistent: contradiction. Hence $\text{ACC}^T(\{\phi\},\{\})$ holds. QED

Thus, in Propositional Logic, trivially AL-entailment reduces to the notion of non-acceptability:

**Corollary 1.** Let $T$ be a classically consistent theory and $\phi \in \mathcal{L}$. Then $T \models_{AL} \phi$ iff $N\text{ACC}^T(\{\neg \phi\},\{\})$.

The following property sanctions that AL-entailment implies classical derivability:

**Corollary 2.** Let $T$ be a classically consistent theory and $\phi \in \mathcal{L}$. If $T \models_{AL} \phi$ then $T \vdash \phi$.

**Proof:** If $N\text{ACC}^T(\{\neg \phi\},\{\})$, then, by theorem 1, there is a RAND derivation of $\neg \neg \phi$ from $T$ and thus $T \vdash \phi$. QED

This corollary gives that consequences of a classically consistent theory under $\models_{AL}$ are classical consequences too. Proposition 1 sanctions that direct consequences are not lost by $\models_{AL}$. However, in general not all classical consequences are retrieved by $\models_{AL}$, namely the converse of corollary 2 does not hold, as the following example shows.

**Example 8.** Let $T = \{\neg \alpha\}$. We show that $T \models_{AL} \alpha \rightarrow \beta$ by showing that $N\text{ACC}^T(\{\neg(\alpha \rightarrow \beta)\},\{\})$ does not hold. A standard ND derivation of $\alpha \rightarrow \beta$ from $T$ is:

$$
\begin{align*}
[\alpha] \\
[\neg \beta] \\
[\neg \alpha \text{ from } T] \\
\bot \\
\neg \neg \beta \quad \text{RA} \\
\beta \\
\alpha \rightarrow \beta \\
\text{I}
\end{align*}
$$

This does not help with determining $N\text{ACC}^T(\{\neg(\alpha \rightarrow \beta)\},\{\})$. This is related to the fact that the inconsistency in the inner RAND derivation of $\neg \beta$ is derived without the need of the hypothesis, $\neg \beta$, of this RAND derivation. In general, any RAND derivation of $\neg(\neg(\alpha \rightarrow \beta)$ (and hence of $\alpha \rightarrow \beta$) from this theory, $T$, contains such a RAND sub-derivation relying on the inconsistency of the copy of $\alpha$ from a ($\rightarrow$I) sub-derivation, with $\neg \alpha$ from $T$. This means that $N\text{ACC}^T(\{\neg(\alpha \rightarrow \beta)\},\{\})$ cannot hold, since, otherwise, by theorem 1, we would have a RAND derivation of $\neg(\neg(\alpha \rightarrow \beta)$ without such a sub-derivation. This is because by construction of the corresponding RAND derivation given by theorem 1 the existence of such a RAND sub-derivation would violate the non-acceptability of some defence in the assumed non-acceptability of $\neg(\alpha \rightarrow \beta)$.

This example shows, in particular, that implication is not material implication under $\models_{AL}$.
7 AL for Non-Monotonic Reasoning-Discussion

Here we present a first investigation on how AL can be used as a basis for NMR unifying classical and defeasible reasoning, in the context of the well known *tweety* example. Our examination is based on the (expected) need to extend AL with preferences and the observation that when a theory is (directly) inconsistent we have the possibility to reason with its sub-theories, considering these as arguments that support their conclusions under AL. For the illustration we use the following (abbreviations of) sentences:

\[
\begin{align*}
\phi_{bf} &= \text{[bird(tweety) } \rightarrow \text{ fly(tweety)]} \\
\phi_{pf} &= \text{[penguin(tweety) } \rightarrow \text{ \neg fly(tweety)]} \\
\phi_{pb} &= \text{[penguin(tweety) } \rightarrow \text{ bird(tweety)]} \\
\phi_{\neg f} &= \text{[\neg fly(tweety)]} \\
\phi_{p} &= \text{[penguin(tweety)]} \\
\phi_{\neg b} \neg p &= \text{[\neg bird(tweety) } \rightarrow \text{ \neg penguin(tweety)]}
\end{align*}
\]

*Example 9.* Let \(T = \{ \phi_{bf}, \phi_{pb}, \phi_{\neg f} \}\) (\(T\) is classically consistent). It is easy to see that \(T \models_{AL} \neg \text{bird(tweety)}\) as \(\{\}\) attacks \(\{\text{bird(tweety)}\}\) and therefore \(\text{NACC}^T(\{\text{bird(tweety)}\}, \{\})\). Similarly, \(T \models_{AL} \neg \text{penguin(tweety)}\). In absence of other information, we believe that these conclusions are legitimate/desirable.

Note that AL does not distinguish default rules and facts and it supports contrapositive reasoning with the single form of implication it allows. In example 9, default logic [19] would derive the same conclusions only by labelling \(T\) as facts, but would not derive either conclusion if the first sentence were labelled as a default rule, as conventional.

*Example 10.* Let \(T = \{ \phi_{bf}, \phi_{pb}, \phi_{\neg f}, \phi_{p} \neg f \}\) (\(T\) classically consistent, obtained by adding \(\phi_{p} \neg f\) to \(T\) in example 9). As in example 9, \(T \models_{AL} \neg \text{bird(tweety)}\) and \(T \models_{AL} \neg \text{penguin(tweety)}\). From a commonsense reasoning perspective, this is counter-intuitive, as it disregards the newly added sentence and the alternative possibility for \(\neg \text{fly(tweety)}\) it supports, namely \(\text{penguin(tweety)}\).

By comparison, default logic with the first and last sentences in \(T\) labelled as default rules (as conventional) would (sceptically) derive no conclusion as to whether *tweety* is (or not) a bird or penguin. Arguably, this is too sceptical a behaviour. Note that we have the same counter-intuitive behaviour of deriving \(\neg \text{penguin(tweety)}\) when the sentence \(\neg \text{fly(tweety)}\) is deleted from the theory of example 10. In order to accommodate within AL the intuitive kind of reasoning pointed out for these examples, we can extend AL with priorities over sentences, so that, in particular, exceptions may override rules, in the spirit of prioritised default logic [4, 5] and other approaches to supporting reasoning with priorities [7]. In our illustration, these priorities may be drawn from the partial order \(\phi_{\neg f}, \phi_{p}, \phi_{pb}, \phi_{\neg b} \neg p > \phi_{p} \neg f > \phi_{bf}\). The challenge is to incorporate these priorities without imposing a separation amongst sentences (as done instead in prioritised and standard default logic) and without imposing a specific structure on the defeasible knowledge (the default rules) so as to achieve, e.g., the behaviour of
AL in example 9. In example 10, the given priorities may be used to identify the sub-theory \(\{\phi_{pb}, \phi_{pf}, \phi_{bf}\}\) as the strongest and thus entail penguin(tweety).

By introducing priorities we can also use preference-based argumentation, as in e.g. [14, 18], to distinguish between strengths of AL-entailment from sub-theories, and, in particular, allow for stronger sub-theories to dominate, as illustrated by the following example:

**Example 11.** Let \(T = \{\phi_{bf}, \phi_{pf}, \phi_{pf}, \phi_{bf}\}\) (\(T\) is directly but not classically consistent). Then, correctly, in absence of other information, \(T \not\models_{AL} \text{bird(tweety)}\) and \(T \not\models_{AL} \text{bird(tweety)}\). The sub-theories \(T_1 = \{\phi_{bf}, \phi_{pf}\}\) and \(T_2 = \{\phi_{pf}, \phi_{bf}\}\) AL-entail \(\text{b(tweety)}\) and \(\text{bird(tweety)}\) respectively and hence dispute each other. If we now take into account \(\phi_{bf} > \phi_{pf}\), then, under a preference-based argumentation approach, \(T_2\) would dominate \(T_1\) and thus \(T\) would correctly entail \(\text{bird(tweety)}\).

The core technical challenge of using priorities over sentences is to understand how these could influence the reasoning by contradiction afforded by RA in AL. In our illustrative setting we want the priorities (especially \(\phi_{pf} > \phi_{bf}\)) to restrict the application of RA. There are other cases, however, where RA gives intuitive results and should not be restricted. For example, from the theory \(\{\text{bird(tweety)}, \phi_{pb}, \phi_{pf}, \phi_{bf}\}\) with \(\phi_{pb} > \phi_{pf} > \phi_{bf}\) we expect that \(\neg\text{penguin(tweety)}\) is entailed since \(\text{fly(tweety)}\) is an intuitive default conclusion of this theory and then, by RA, \(\text{penguin(tweety)}\) cannot be entailed (as otherwise through the stronger sentence of \(\phi_{pf}\), the sentence \(\neg\text{fly(tweety)}\) would follow). Similarly, given the theory \(\{\text{fly(tweety)}, \phi_{pb}, \phi_{pf}, \phi_{bf}\}\) with \(\phi_{pb} > \phi_{pf} > \phi_{bf}\), we expect that \(\neg\text{penguin(tweety)}\) is entailed as \(\text{penguin(tweety)}\) would give \(\neg\text{fly(tweety)}\) due to the higher strength of \(\phi_{pf}\). To accommodate such cases it may be necessary to use the priority information more tightly within the definition of AL, i.e. within the definition of (non-)acceptability.

8 Related Work

AL is based on a notion of acceptability of arguments which is in the same spirit as that in [8,11] for capturing the semantics of negation as failure in Logic Programming. These notions of acceptability are global in the sense that acceptable and non-acceptable arguments are all defined at the same time. This view has also recently been taken in [6,20] where the argumentation semantics is defined through the notion of a global labelling of arguments as IN, OUT or UNDECIDED.

The link of argumentation to NMR has been the topic of extensive study for many years. Most of these studies either separate in the language the classical reasoning from the defeasible part of the theory (e.g. in Default Logic) or restrict the classical reasoning (e.g. in LP with NAF) or indeed as in the case of circumscription [17] the theory is that of classical logic but a complex prescription of model selection is imposed on top of the classical reasoning.
Recently, [2] proposed an argumentation framework based upon classical logic with the aim (that we share) to use argumentation to reason with possibly inconsistent classical theories, beyond the realms of classical logic. In their approach, arguments are defined in terms of sub-theories of a given (typically inconsistent) theory and they have minimal and consistent supports (wrt the full classical consequence relation). Attacks are defined in terms of a notion of canonical undercut that relies on arguments for the negation of the support of attacked argument. Further, the evaluation of arguments is given through a related tree structure of defeated or undefeated nodes.

Other works that aim for a tighter link between classical and defeasible reasoning include the work of Amgoud and Vesic [1], studying the problem of handling inconsistency using argumentation with priorities over sentences, and [21], who have adapted the approach of [2] to Description Logic and have proposed an argumentation-based operator to repair inconsistencies. Our approach differs from these works in that it starts with providing an alternative understanding of Propositional Logic in argumentation terms on which to base any further development of reasoning with inconsistent or defeasible theories. In comparison with our approach, these other works can be seen more as a form of belief revision, based on argumentation, for classically inconsistent theories rather than a re-examination of classical logic through argumentation to provide a uniform basis for classical and defeasible reasoning.

9 Conclusion and Future Work

We have presented Argumentation Logic (AL) and shown how it allows us to understand classical reasoning in Propositional Logic in terms of argumentation. Its definition rests on capturing semantically the Reductio ad Absurdum rule through a suitable notion of acceptability of arguments. One property of the ensuing AL is that the interpretation of implication is different from that of material implication. Further results on the relationship between AL and Propositional Logic including how AL can completely capture the entailment of PL are given in [12].

Given the significant role that argumentation has played in understanding under a common framework NMR in AI we have examined the problem of how we could unify classical reasoning and NMR within the framework of AL. In this context, we have considered the following questions: How could we use AL as the underlying logic to build a NMR framework? Can AL with its propositional language provide a single representation framework for classical and defeasible reasoning without any distinctions on the type of sentences allowed in a given theory? In particular, can we understand AL as a NMR framework with sentences that would behave as default rules but also as classical rules, with a form of contrapositive reasoning with these rules allowed? In this paper we have identified this problem and the challenges it poses, and studied these questions in the context of examples.
Our preliminary investigation suggests the need for an extension of AL to accommodate preferences amongst sentences. Many existing frameworks for NMR use, either explicitly or implicitly, preferences to capture defeasible reasoning, e.g. [4, 5] for Default Logic [19]. Also many frameworks of argumentation rely on some form of preference between arguments, e.g. [14, 15, 18] to capture a notion of (relative) strength of arguments through which the attack relation between arguments can be realized. One way therefore to study this problem of integrating classical and defeasible reasoning is to use some form of preference on the sentences of AL theories, and adapt existing approaches of reasoning with preferences to AL.

A Appendix: Natural Deduction

We use the following rules, for any $\phi, \psi, \chi \in L$:

- $\land I : \phi, \psi \vdash \phi \land \psi$
- $\land E : \phi \land \psi \vdash \phi$
- $\land E : \phi \land \psi \vdash \psi$
- $\lor I : \phi \lor \psi \vdash \phi$
- $\lor I : \phi \lor \psi \vdash \psi$
- $\rightarrow I : \phi \rightarrow \psi \vdash \phi \rightarrow \psi$
- $\neg I : \phi \vdash \neg \phi$
- $\neg E : \neg \neg \phi \vdash \phi$
- $\lor E : \phi, \phi \lor \chi, \phi \lor \chi \vdash \phi \lor \chi$
- $\lor E : \phi, \phi \lor \chi, \phi \lor \chi \vdash \phi \lor \chi$

where $[\zeta,...]$ is a (sub-)derivation with $\zeta$ referred to as the hypothesis. $\neg I$ is also called Reduction ad Absurdum (RA).

References

10. Dung, P.: On the acceptability of arguments and its fundamental role in non-
monotonic reasoning, logic programming and n-person games. Artif. Intell. 77, 
11. Kakas, A.C., Mancarella, P., Dung, P.M.: The acceptability semantics for logic 
of Computer Science, University of Cyprus, Cyprus (April 2012)
13. Kakas, A., Toni, F., Mancarella, P.: Argumentation for propositional logic and 
nonmonotonic reasoning. In: Working notes of the 11th International Symposium 
on Logical Formalizations of Commonsense Reasoning (2013)
14. Kakas, A.C., Moraitis, P.: Argumentation based decision making for autonomous 
agents. In: The Second International Joint Conference on Autonomous Agents & 
Multiagent Systems, AAMAS 2003, July 14-18, 2003, Melbourne, Victoria, Aus-
16. Lin, F., Shoham, Y.: Argument systems: A uniform basis for nonmonotonic rea-
soning. In: Brachman, R.J., Levesque, H.J., Reiter, R. (eds.) Proceedings of the 1st 
International Conference on Principles of Knowledge Representation and Reason-
(1989)
17. McCarthy, J.: Circumscription - a form of non-monotonic reasoning. Artificial In-
Artificial Intelligence (2012), in Press
(1980)
in Logic 3(4), 12–29 (2010)
The representation of Boolean algebras in the spotlight of a proof checker

Rodica Ceterchi, Eugenio G. Omodeo, Alexandru I. Tomescu

1 Facultatea de Matematică și Informatică, Universitatea din București
email: rceterchi@gmail.com

2 Dipartimento di Matematica e Geoscienze, Università di Trieste,
Via Valerio 12/1, I-34127 – Trieste, Italy
email: eomodeo@units.it

3 Helsinki Institute for Information Technology HIIT,
Department of Computer Science, University of Helsinki,
P.O. 68 (Gustaf Hallström latin 2b), FI-00014 – Helsinki, Finland
email: tomescu@cs.helsinki.fi

Abstract. We report on a proof-checked version of Stone’s result on the representability of Boolean algebras via the clopen sets of a totally disconnected compact Hausdorff space. Our experiment is based on a proof verifier based on set theory, whose usability can in its turn benefit from fully formalized proofs of representation theorems akin to the one discussed in this note.

Key words: Theory-based automated reasoning; proof checking; Referee aka ÆtnaNova; Boolean rings; Boolean algebras; Stone spaces.

"Boolean algebras have an almost embarrassingly rich structure." [7, p. 10]

Introduction

This paper reports on a formally verified proof of Stone’s celebrated results [20,21,22] on the representability of Boolean algebras as fields of sets, carried out along the lines found in [4, pp. 41–43] by means of Jacob T. Schwartz’s proof-checker Referee, aka ÆtnaNova, to be simply called ‘Ref’ for brevity in the ongoing. The proof of Zorn’s lemma, formalized much earlier (cf. [19, Chapter 7]) and previously exploited in the compactness proof for classical propositional logic [16], played again a crucial role in our present scenario.

A website reporting on our experiment is at http://www2.units.it/eomodeo/StoneReprScenario.html. In its final form, the script-file supporting it and leading from first principles to the topological version of Stone’s result, comprises 42 definitions and proves 210 theorems altogether, organized in 19 THEORY’s, including the background THEORY Set_theory. Its processing takes about 25 seconds.

* Work partially supported by the INdAM/GNCS 2013 project “Specifica e verifica di algoritmi tramite strumenti basati sulla teoria degli insiemi” and by the Academy of Finland under grant 250345 (CoECGR)
1 Boolean algebras and rings

A ring is said to be Boolean when every one of its elements is self-inverse w.r.t. addition and idempotent w.r.t. multiplication:

\[ X + X = 0, \quad X \cdot X = X. \]

The former of these laws makes it superfluous to postulate the commutativity of addition (for, it implies it); the latter implies the commutativity of multiplication. When endowed with multiplicative identity, 1, a Boolean ring is called a Boolean algebra. (To avoid trivialities, we will require that 0 ≠ 1).

An aspect of the richness of Boolean rings is that the relation

\[ X \leq Y \iff X \cdot Y = X \]

is a partial order, in which every pair \( X, Y \) of elements has greatest common lower bound \( X \sqcap Y = X \cdot Y \) and least common upper bound \( X \sqcup Y = X \cdot Y + X + Y \). In this ordering 0 acts as the minimum and—when present—1 acts as the maximum. Historically, Boolean algebras were first studied as lattices endowed with peculiar properties (namely, distributivity and complementedness).\(^4\) The salient operations, from this viewpoint, were \( \sqcap, \sqcup, \overline{\phantom{0}} \); the algebraic kinship with the rings of numbers remained unnoticed for quite a while (cf.\([9, p. 208]\)).

From the ring-based point of view—the one which will prevail in these pages—the complementation operation turns out to be:

\[ X = \overline{1 + X}. \]

\(^4\) From the lattice-based point of view, the simplest available characterization of the structure ‘Boolean algebra’ is the one proposed by Herbert Robbins, ca. 1933 (cfr. \([12,10]\)). Ignoring the construct \( \sqcap \), which can be introduced by way of shortening notation, the Robbins laws are:

\[ X \sqcup Y = Y \sqcup X, \quad X \sqcup (Y \sqcup Z) = (X \sqcup Y) \sqcup Z, \quad \overline{X \sqcup Y \sqcup Z} = X \sqcup Y \sqcup Z. \]

Fig. 1. A few derived Boolean laws

\[ X \in B \rightarrow \overline{X} \in B \& \overline{\overline{X}} = X \& \overline{1}_B = 0_B \& \overline{0}_B = 1_B \]

\[ \{X, Y\} \subseteq B \rightarrow \overline{X} \sqcup X = 1_B \& Y \cdot X + Y \cdot \overline{X} = Y \& (Y \cdot X) \cdot (Y \cdot \overline{X}) = 0_B \]

\[ \{X, Y\} \subseteq B \rightarrow \overline{X} \sqcup Y = X \cdot Y + \overline{X} \cdot Y \]

\[ \forall \, X \in B \rightarrow \overline{X} \neq X \& \{X \notin \{0_B, 1_B\} \rightarrow \overline{X} \in B \setminus \{0_B, 1_B\}\} \]

\[ \{X, Y\} \subseteq B \rightarrow (X \cdot Y = 1_B \rightarrow X = 1_B \& Y = 1_B) \]

\[ \{U, V, X, Y\} \subseteq B \rightarrow U \cdot \overline{X} + V \cdot \overline{Y} = (U \cdot \overline{X} + V \cdot \overline{Y}) \cdot \overline{X} \cdot \overline{Y} \]

\[ X \sqcup Y = Y \sqcup X, \quad X \sqcup (Y \sqcup Z) = (X \sqcup Y) \sqcup Z, \quad \overline{X \sqcup Y \sqcup Z} = X \sqcup Y \sqcup Z. \]
2 Fields of sets

Starting with a non-void set $S$, let us construct the following families of sets:

- $\mathcal{P}(S) = \{ x : x \subseteq S \}$, the family of all subsets of $S$;
- $\mathcal{F}(S) = \{ x \subseteq S | |x| \in \mathbb{N} \}$, the family of all finite subsets of $S$;
- $\mathcal{B}(S) = \mathcal{F}(S) \cup \{ S \setminus x : x \in \mathcal{F}(S) \}$, the family formed by those subsets of $S$ each of which is either finite or has a finite complement (relative to $S$).

If $S$ is finite, $\mathcal{P}(S) = \mathcal{F}(S) = \mathcal{B}(S)$ holds; otherwise, we get three families.

To get Boolean rings out of these, it suffices to define:

- $X \cdot Y = X \cap Y$ (intersection),
- $X + Y = (X \cup Y) \setminus (X \cap Y)$ (symmetric difference).

One readily sees that $\mathcal{P}(S)$ and $\mathcal{B}(S)$ thus become Boolean algebras, whose additive identity, 0, and multiplicative identity, 1, are $\emptyset$ and $S$; as for $\mathcal{F}(S)$, it lacks 1 when $S$ is infinite.

By generalizing the case of $\mathcal{P}(S)$ and $\mathcal{B}(S)$, one calls FIELD OF SETS any family $\mathcal{B}$ which

- is closed under the operations of intersection and symmetric difference;
- has $\bigcup \mathcal{B}$ among its members, viz., owns a maximum w.r.t. set inclusion;
- differs from $\{ \emptyset \}$.

This clearly is an instance of a Boolean algebra. How general? A renowned theorem by Marshall H. Stone [20] gives us the answer:

_Every Boolean algebra is isomorphic to a field of sets._

This field is not always of the form $\mathcal{P}(S)$ (this holds only for particular Boolean algebras, among which the ones whose underlying domain is finite\(^5\)): in fact it is trivial that the field $\mathcal{B}(\mathbb{N})$, whose cardinality equals the one of $\mathbb{N}$, cannot be isomorphic to $\mathcal{P}(S)$ for any $S$.\(^7\)

3 Stone spaces

We define a BASE OF A TOPOLOGICAL SPACE to be a pair $(\mathcal{X}, \beta)$ such that:

(i) $\mathcal{X} \neq \emptyset$; (ii) $\beta \subseteq \mathcal{P}(\mathcal{X})$; (iii) $\beta$ enjoys, w.r.t. dyadic intersection, $\cap$, and to monadic union, $\bigcup$, the following closure properties:

\(^5\) We designate by $|X|$ the cardinality of a set $X$ and by $\mathbb{N}$ the set $\{0, 1, 2, \ldots \}$ of all natural numbers (which is, in its turn, a cardinal number—the first infinite one).

\(^6\) More generally, the Boolean algebras which are isomorphic to $\mathcal{P}(S)$ for some $S$ are the ones which are _completely distributive_; [2, pp. 221–222] credits this result to Alfred Tarski.

\(^7\) Indeed, $|\mathcal{P}(S)| \in \mathbb{N}$ (i.e., $|\mathcal{P}(S)|$ is smaller than the cardinal number $\mathbb{N}$) when $|S| \in \mathbb{N}$; whereas $|\mathcal{P}(S)|$ exceeds $\mathbb{N}$—because $|\mathcal{P}(S)|$ exceeds $|S|$—when $S$ is infinite.
The topological space generated by such a base is, by definition, the pair $(\mathcal{X}, \tau)$ where

$$\tau = \{ \bigcup a : a \subseteq \beta \}.$$ 

A topological space hence is a pair $(\mathcal{X}, \tau)$ such that:

1) $\mathcal{X} \in \tau$,
2) $A, B \in \tau \rightarrow (A \cap B) \subseteq \bigcup \{ c \in \beta \mid c \subseteq (A \cap B) \}$,
3) $A \in \mathcal{P}(\tau) \rightarrow \bigcup A \in \tau$,

whence it plainly follows that $A, B \in \tau \rightarrow A \cap B \in \tau$ and, more generally, that $A \in \mathcal{P}(\tau) \rightarrow \bigcap A \in \tau$, under the proviso that the intersection $\bigcap \emptyset$ equals $\mathcal{X}$.

The open and the closed sets of such a space are, respectively, the members of $\tau$ and their complements $\mathcal{X} \setminus A$ (with $A \in \tau$). A subset of $\mathcal{X}$ which is open and closed is called a clopen set: examples are $\emptyset$ and $\mathcal{X}$. It is apparent that clopen sets form a field of sets.

A topological space is said to be a **Hausdorff space** if: for every pair of distinct members $p, q \in \mathcal{X}$, there exist disjoint open sets $P, Q \subseteq \mathcal{X}$ such that $p \in P$, $q \in Q$.

A topological space is said to be **compact** if: every family of open sets whose union is $\mathcal{X}$ includes a finite subfamily whose union is $\mathcal{X}$.

One calls **Stone space** any topological space which, in addition to enjoying the two properties just stated, also

– owns a base—namely a $\beta \subseteq \tau$ such that every open set is the union of a subfamily of $\beta$—entirely formed by clopen sets.

By relying on these concepts, one enhances the claim of Stone’s theorem as follows (cf. [21,22]):

**Every Boolean algebra is isomorphic to a field of sets consisting of the clopen sets of a Stone space.**

In sight of the proof of this theorem, it is worthwhile to recall another way of stating the compactness property, dual to the one proposed above:

A topological space $(\mathcal{X}, \tau)$ is compact if and only if every (non-void) family $\mathcal{C}$ of closed sets whose intersection $\bigcap \mathcal{C}$ is void has some finite subfamily whose intersection is void: $\mathcal{F} \subseteq \mathcal{C}$, $|\mathcal{F}| \in \mathbb{N} \setminus \{0\}$, $\bigcap \mathcal{F} = \emptyset$.

This definition implies that in a Hausdorff space every singleton subset of $\mathcal{X}$ is closed.
4 Boolean ideals and their maximal enlargements

The study of an abstract algebraic structure forcibly leads one to investigate the associated homomorphisms. In the Boolean case at hand, to move resolutely in the direction that best suits our purposes, we will just consider those homomorphisms which translate the operations of a Boolean algebra (the homomorphism’s domain) into the set-operations $\cap, \triangle$ of intersection and symmetric difference. The images of such a homomorphism will hence form a field of sets; particularly worth of consideration, among the homomorphisms of interest, are the ones whose values form the special field $\mathbb{2} = (\emptyset, \{\emptyset\}, \cap, \triangle, \emptyset)$. Let us refer by $B = (B, \cdot, +, 1_B, 0_B)$ to a Boolean algebra which, tacitly, will act as domain of the homomorphisms that will enter into play. A homomorphism of $B$ into $\mathbb{2}$ is fully characterized by either one of the two subsets of the underlying domain $B$ which are counter-images, respectively, of $0 = \emptyset$ and of $1 = \{\emptyset\}$. Not all subsets of $B$ can play the role of counter-images of the minimum via a Boolean homomorphism; let us hence figure out which conditions a set must meet in order to qualify for such a role. In investigations of this nature, algebraists tend to focus on the counter-images of the minimum, the so-called ideals, or ‘kernels’; logicians, on the opposite, tend to focus on the counter-images of the maximum, the so-called filters, or ‘shells’. We will conform to the algebraic habit; moreover, since we must concentrate mainly on homomorphisms into $\mathbb{2}$, we will tribute special attention to ideals which are maximal w.r.t. to set inclusion.

**Definition 1.** An ideal is a subset of the underlying domain $B$ which is closed with respect to addition, as well as to multiplication of its elements by elements of $B$, and which is not one of the (exceedingly trivial) sets $\emptyset, \{0_B\}, B$.

Three theorems about ideals play a crucial role in the proof of Stone’s results:

a) Every element $x$ of $B$ which is neither $1_B$ nor $0_B$ belongs to at least one ideal: the least such ideal, named a principal ideal, is simply formed by the multiples of $x$. It hence follows, save in the case when $B = \{0_B, 1_B\}$, that there is at least one ideal.

b) To each ideal $I$ and each $x \neq 1_B$ not belonging to $I$, there corresponds an ideal $J \supset I$ one of whose elements is $x$: this is $\{a \cdot x + y : a \in B, y \in I\}$.

c) Every ideal $I$ is included in an ideal which is maximal w.r.t. $\subseteq$.

Checking the first two of these is very plain; the third can be proved by means of Zorn’s lemma, after observing that every chain of ideals is closed w.r.t. union.

5 1st Stone’s representation theorem

Let us recall first the algebraic version of Stone’s theorem:

**Theorem 1 (Stone’s algebraic representation).** Every Boolean algebra $B = (B, \cdot, +, 1_B, 0_B)$ is isomorphic to a field $\mathbb{H}$ of sets whose underlying domain is included in $\mathcal{P}(\mathcal{H})$, where $\mathcal{H}$ is the set of all homomorphisms from $B$ into $\mathbb{2}$. 

Proof. Associate with each \( x \in B \) the set \( \tilde{x} \) of those homomorphisms in \( H \) which send \( x \) to \( \{\emptyset\} \); thus, clearly, \( \tilde{0} = \emptyset \) and \( \tilde{1} = H \). Moreover \( \tilde{x} \cdot \tilde{y} = \tilde{x} \cap \tilde{y} \) holds: in fact, when \( h \) is a homomorphism, \( h(x \cdot y) = h(x) \cap h(y) \) equals \( \{\emptyset\} \) if and only if \( h(x) = h(y) = \{\emptyset\} \), i.e., \( h \in \tilde{x} \cap \tilde{y} \). By an analogous argument, \( h(x + y) = h(x) \Delta h(y) \) and \( x + y = \tilde{x} \Delta \tilde{y} \). Take \( H \) to be the image-set of the function \( x \mapsto \tilde{x} \).

In order to see the injectivity of this function, consider the difference \( e_0 = x_0 + x_0 \cdot x_1 \) between two elements \( x_0, x_1 \in B \) such that \( x_0 \cdot x_1 \neq x_0 \) (whence \( e_0 \neq 0_B \)). We will show that there is an \( h \in H \) sending \( e_0 \) to \( 1_B \); accordingly, since \( 1_B = h(e_0) = h(x_0 + x_0 \cdot x_1) = h(x_0) \Delta (h(x_0) \cap h(x_1)) \), we will have \( h(x_0) = \{\emptyset\}, h(x_1) = \emptyset \), and therefore \( h \in \tilde{x}_0 \setminus \tilde{x}_1 \) as desired. We readily get the sought \( h \) if \( B = \{0_B, 1_B\} \); otherwise we pick a \( y_0 \in B \setminus \{0_B, 1_B\} \), choosing \( y_0 = e_0 \) if \( e_0 \neq 1_B \). This \( y_0 \) belongs to a principal ideal and hence to a maximal ideal \( M \), and it is plain that the opposite \( h = 1 - \chi_M \) of the characteristic function of \( M \), manifestly a homomorphism from \( B \) to \( 2 \), does to our case. \( \dashv \)

Call set-representation of the algebra \( B \) the field of sets just built. We will see next that this \( H \) generates a very peculiar topology on \( H \).

6 2\textsuperscript{nd} Stone’s representation theorem

We are now ready for the topological version of Stone’s theorem:

**Theorem 2 (Stone’s topological representation).** The field of sets by which we have represented a Boolean algebra \( B \) is the base—as well as the family of all clopen sets—of a topology on the set \( H \) of all homomorphisms from \( B \) into \( 2 \). Once endowed with such a topology, \( H \) turns out to be a Stone space.

**Proof.** To see that \( H \) (constructed as in the preceding proof) is the base of a topology on \( H \), we can directly check that \( H \subseteq \mathcal{P}(H) \) and \( \bigcup H = H \neq \emptyset \) hold, and that \( \bigcap F \) belongs to \( H \) for every finite non-void subset \( F \) of \( H \). Indeed, \( H \) is the image-set of an injective function \( x \mapsto \tilde{x} \) from \( B \) into \( \mathcal{P}(H) \), where \( B \) is at least doubleton; hence, readily, \( H \neq \emptyset, H \subseteq \mathcal{P}(H) \), and \( \bigcup H \subseteq H \) hold. The last inclusion is in fact an equality, because \( H \), which is \( 1_B \), belongs to \( H \). Then we get that \( H \) is closed under intersection through the remark, made above, that \( \tilde{x} \cap \tilde{y} = \tilde{x \cdot y} \).

Knowing, at this point, that \( H \) qualifies as the base for a topology on \( H \), let us notice that all sets in \( H \) are clopen in the topology \( \tau \) generated by it; in fact, since \( H \setminus \tilde{x} = H \triangle \tilde{x} = \tilde{1_B} \triangle \tilde{x} = \tilde{1_B} + \tilde{x} \), the complement of a set in the base belongs to the base in its turn. This remark readily gives us that \( (H, \tau) \) is a Hausdorff space. In fact, when \( f, g \in H \) differ, there is an \( x \in B \) such that \( f(x) = 1 \leftrightarrow g(x) \neq 1 \); thus, since \( H \setminus \tilde{x} = \tilde{f} \), we can find sets \( u, v \in \tau \) such that \( f \in u, g \in v \), and \( u \cap v = \emptyset \), by also insisting that \( \{u, v\} = \{\tilde{f}, \tilde{g}\} \).
It remains to be shown that the space is compact, which will also yield that every clopen set of \( \tau \) belongs to \( H \). One easily sees that the closed sets in \( \tau \) are: \( H \) and all intersections of non-void subsets of \( \{ \tilde{x} : x \in B \} \); compactness will hence readily follow if we manage to show that whenever \( \emptyset \neq O \subseteq H \) and \( \bigcap O = \emptyset \) hold, there is an \( F \subseteq O \) such that \( |F| \in \mathbb{N} \setminus \{0\} \) and \( \bigcap F = \emptyset \). Equivalently, assuming that \( \emptyset \neq O \subseteq H \) and that \( \bigcap F \neq \emptyset \) holds for every finite non-void subset \( F \) of \( O \), we will show that \( \bigcap O \neq \emptyset \).

Notice that the subset

\[
B_0 = \text{Def} \left\{ x : F \subseteq O \mid |F| \in \mathbb{N} \setminus \{0\} \& \tilde{x} = \bigcap F \right\} \cup \{1_B\}
\]

of \( B \) meets the conditions:
1) \( x \cdot y \in B_0 \) for all \( x, y \in B_0 \),
2) \( 0_B \notin B_0 \setminus \{1_B\} \),

implying that \( \{ a \cdot \tilde{x} : a \in B, x \in B_0 \} \) is an ideal of \( B \). By enlarging this into a maximal ideal, we get the kernel of a homomorphism \( h \) sending all complements of el’ts of \( B_0 \) to \( \emptyset \), hence sending all elements of \( B_0 \) to \( \{0\} \). Thus, \( h \in \bigcap O \). –

### 7 Formalization of Stone’s representation theorems in Ref

This section offers glimpses of our formal development of Stone’s result on the representability of Boolean algebras via the clopen sets of a totally disconnected, compact Hausdorff space. In carrying out this task, we relied on a proof checker: Ref. Our experiment culminated in a rather elaborate series of mathematical claims, shown not in this section, but in the appendix.

Organization and rationale of our automated proof assistant are extensively discussed in [19]; but so far, due to the short time elapsed from its implementation, Ref is not widely known. Hence, putting aside our report on our experiment for a moment, we devote a quick subsection to introducing Ref itself.

#### 7.1 Brief presentation of the proof-checking framework

Ref is a proof assistant for mathematics based on a variant of Zemel-Fraenkel set theory, which is “hardwired” in Ref’s proof-checking abilities. From the user Ref receives script files, called scenarios, consisting of successive definitions, theorems, and auxiliary commands, which it either certifies as constituting a valid sequence or rejects as defective. In the case of rejection, the verifier attempts to pinpoint the troublesome locations within a scenario, so that errors can be

---

9 To derive from compactness that \( \bigcup O \in H \) holds when \( O \subseteq H \) and \( \bigcup O \) is closed, we argue as follows. Assuming, w.l.o.g., that \( \emptyset \neq O \), since \( \emptyset = (\bigcup O) \cap (H \setminus \bigcup O) = (\bigcup O) \cap \bigcap \{H \setminus u : u \in O\} \) is the intersection of a family of closed sets, we can pick an \( F \) such that \( \emptyset \neq F \subseteq O \), \( |F| \in \mathbb{N} \), and \( (\bigcup O) \cap \bigcap \{H \setminus u : u \in F\} \neq \emptyset \). Thus \( \bigcup O = \bigcap \{H \setminus u : u \in F\} \in H \), because \( H \) is closed relative to complementation and to finite intersection. 

293
located and repaired. Step timings are produced even for correct proofs, to help
the user in spotting places where appropriate modifications could speed up proof processing.

The bulk of the text normally submitted to the verifier consists of theorems and proofs. Some theorems (and their proofs) are enclosed within so-called theories, whose external conclusions are justified by these internal theorems. This lets scenarios be subdivided into modules, which increases the readability and supports proof reuse.

Many theorems are not enclosed within a user-defined theory; when this happens, they belong on their own right to the underlying “big” theory identified by the name Set theory.

Of all relationships treated within Set theory, the most fundamental is membership, ∈, which is supposed to be well founded in the sense that no infinite sequence $x_0, x_1, x_2, \ldots$ of sets can satisfy $x_{i+1} \in x_i$ for every $i$. The well foundedness of ∈ is witnessed by the built-in arbitrary selection operator, arb, meeting the conditions $arb(x) \in x$ and $arb(x) \cap x = \emptyset$ for every set $x$ other than the null set $\emptyset$ (about which the equality $arb(\emptyset) = \emptyset$ is assumed).

At their simplest definitions are merely abbreviations which concentrate attention on interesting constructs by assigning them names which shorten their syntactic form (an example of this kind is the definition of finitude that we will soon meet). Beyond this simple level, Ref offers a primitive scheme of ∈-recursive definition legitimatized by the well foundedness of ∈ and illustrated, e.g., by the following specification:

$$\text{Def} : \text{filum}(X) = \text{Def} \{ X \} \cup \text{arb}(\{ \text{filum}(y) : y \in X \mid X = \{ y \} \})$$

(This collects together into $\text{filum}(X)$ all elements of the finitely many singletons “spreading”—in a quite definite sense—from $\{X\}$.)

This example also shows the availability, in the formal language of Ref, of perspicuous set-formers such as $\{ \text{filum}(y) : y \in X \mid X = \{ y \} \}$. Just in order to see a few other set-formers at work, consider the following shorthand definitions:

$$\bigcup Y = \text{Def} \{ z : x \in Y, z \in x \},$$

$$\text{edges}(V,E) = \text{Def} \{ \{x,y\} : x \in V, y \in V \mid x \neq y \} \cap E,$$

$$\text{Connected}(E) \leftrightarrow \text{Def} \{ b : b \subseteq E \mid \bigcup b \cap (\bigcup (E \setminus b) = \emptyset) \subseteq \{0,E\} \}.$$

A very small Ref scenario, consisting of one definition (involving yet another set-former) and two theorems with their proofs, is shown in Fig. 2. As one sees there, each inference step in a Ref proof has two components, separated by the ‘⇒’ sign: on the right of which a logical statement (sometimes hidden behind one of the keywords AUTO, Qed) appears; while, on the left, there is a justification of the statement, namely an indication of which inference method enables its derivation from the preceding part of the proof.

Last but not least, Ref supports proof reuse through a costruct named theory, essentially a second-order form of Skolemization. Fig. 3 shows the interface of a specific theory, named finite image, which has two input parameters: $s_0$, a
\textbf{Def \(P\):} [Family of all subsets of a given set] \(P \subseteq \{x : x \subseteq S\}\)

\textbf{Thm \(\text{pow}_0\):} [No set equals its own powerset] \((X \supseteq Y \leftrightarrow Y \in \mathcal{P}X) \& X \neq \mathcal{P}X\). \textbf{Proof:}

Suppose \(\text{not}(x_0, y_0) \Rightarrow\) \text{Auto}

Use \(\text{def}(\mathcal{P}x_0) \Rightarrow\) \text{Auto}

Suppose \(\Rightarrow x_0 = \mathcal{P}x_0\)

\(\text{ELEM} \Rightarrow\) \(\text{Stat0} : x_0 \notin \{y : y \subseteq x_0\}\)

\(\langle x_0 \rangle \Rightarrow \text{Stat0} \Rightarrow\) false; \text{Discharge} \(\Rightarrow\) \text{Auto}

\(\text{EQUAL} \Rightarrow\) \(\text{Stat1} : X \supseteq Y \neq y_0 \notin \{y : y \subseteq x_0\}\)

Suppose \(\Rightarrow\) \(\text{Stat2} : y_0 \notin \{y : y \subseteq x_0\}\)

\(\langle y_1 \rangle \Rightarrow \text{Stat2}(\text{Stat1}*) \Rightarrow\) false; \text{Discharge} \(\Rightarrow\) \(\text{Stat3} : y_0 \notin \{y : y \subseteq x_0\}\)

\(\langle y_0 \rangle \Rightarrow \text{Stat3}(\text{Stat1}*) \Rightarrow\) false; \text{Discharge} \(\Rightarrow\) \text{QED}

\textbf{Thm \(\text{pow}_1\):} [Monotonicity of powerset] \(S \supseteq X \rightarrow \mathcal{P}X \cup \{\emptyset, X\} \subseteq \mathcal{P}S\). \textbf{Proof:}

Suppose \(\text{not}(x_0, x_0) \Rightarrow\) \text{Auto}

\(\text{Set\_monot} \Rightarrow\) \(\{x : x \subseteq x_0\} \subseteq \{x : x \subseteq s_0\}\)

Use \(\text{def}(\mathcal{P}) \Rightarrow\) \(\text{Stat1} : \emptyset \notin \{x : x \subseteq s_0\} \lor x_0 \notin \{x : x \subseteq s_0\}\)

\(\langle \emptyset, x_0 \rangle \Rightarrow \text{Stat1} \Rightarrow\) false; \text{Discharge} \(\Rightarrow\) \text{QED}

\textbf{Thm \(\text{pow}_2\):} [Powerset of null set and of singletons] \(\emptyset = \{\emptyset\} \& \mathcal{P}\{X\} = \{\emptyset, \{X\}\}\). \textbf{Proof:}

Suppose \(\text{not}(x_0) \Rightarrow\) \text{Auto}

\(\emptyset \neq \{}\emptyset\} \Rightarrow\) \(\text{Topow}_1 \Rightarrow\) \(\text{Stat0} : \emptyset \not\subseteq \{\emptyset\}\)

\(\langle y_0 \rangle \Rightarrow \text{Stat0}(\text{Stat0}*) \Rightarrow\) \(\text{Stat1} : y_0 \notin \emptyset \land y_0 \notin \{\emptyset\}\)

\(\langle \emptyset, y_0 \rangle \Rightarrow \text{Topow}(\text{Stat1}*) \Rightarrow\) false; \text{Discharge} \(\Rightarrow\) \(\mathcal{P}\{x_0\} \neq \emptyset, \{x_0\}\}

\(\{x_0\} \Rightarrow \text{Topow}_1 \Rightarrow\) \(\text{Stat2} : \mathcal{P}\{x_0\} \not\subseteq \{\emptyset, \{x_0\}\}\)

\(\{x_0\} \Rightarrow \text{Stat2} \Rightarrow\) \(\text{Stat3} : y_0 \notin \mathcal{P}\{x_0\} \land y_0 \notin \{\emptyset, \{x_0\}\}\)

\(\langle x_0, y_1 \rangle \Rightarrow \text{Topow}(\text{Stat3}*) \Rightarrow\) false; \text{Discharge} \(\Rightarrow\) \text{QED}

\textbf{Fig. 2.} Tiny sample of a \texttt{Ref} scenario

Finite set, and \(g\), a global function. Inside \texttt{finite\_image}, from the assumed finiteness of \(s_0\) the user has derived that \(\{g(x) : x \in s_0\}\) is a finite set; moreover, (s)he has defined the output parameter \(f_\emptyset\) so as to insure that \(f_\emptyset\) be an \(\subseteq\)-minimal subset of \(s_0\) such that \(f_\emptyset\) and \(s_0\) are sent by \(g\) to the same image.

\begin{verbatim}
THEORY finite_image (s_0, g(X))
  Finite(s_0)
  ⇒
  (f_\emptyset)
  Finite( {g(x) : x ∈ s_0} )
  f_\emptyset ⊆ s_0 \& ( ∀ t ⊆ f_\emptyset | g(t) = g(s_0) ⇔ t = f_\emptyset )
END finite_image
\end{verbatim}

\textbf{Fig. 3.} Interface of a \textbf{THEORY} specifying an induction principle
7.2 Excerpts from our proof scenario of Stone’s theorems

As a warm-up exercise, we developed with the assistance of Ref the THEORY pord displayed in Fig. 4, showing that every partially ordered set is isomorphic to a family of sets partially ordered by inclusion. The assumptions of pord state that Le must be a partial ordering of dd. By sending each element \( x \) of dd to the set consisting of those elements of \( \theta \) which are smaller than or equal to \( x \), we get an order monomorphism between (dd, Le) and \( (\mathcal{P}(dd), \subseteq) \) whose name \( \text{polso}_\theta \), as indicated by the subscript \( \theta \), is specified—along with its definition—inside the THEORY.

**Fig. 4.** Interface of a representation THEORY about partial orderings

Our next step consisted in developing a theory of Boolean rings:

**THEORY** booleanRing(\( bb, \cdot, \div \))

\( bb \neq \emptyset \)

\( \langle \forall x, y \mid \{ x, y \} \subseteq bb \rightarrow x \cdot y \in bb \rangle \)

\( \langle \forall x, y \mid \{ x, y \} \subseteq bb \rightarrow x \div y \in bb \rangle \)

\( \langle \forall x, y, z \mid \{ x, y, z \} \subseteq bb \rightarrow x \cdot (y \div z) = (x \cdot y) \div z \rangle \)

\( \langle \forall x, y, z \mid \{ x, y, z \} \subseteq bb \rightarrow x \div (y \cdot z) = (x \div y) \cdot z \rangle \)

\( \langle \forall x, y \mid \{ x, y \} \subseteq bb \rightarrow x \div x = y \div y \rangle \)

\( \langle \forall x, y \mid \{ x, y \} \subseteq bb \rightarrow x \div (y \div x) = y \rangle \)

\( \langle \forall x \mid x \in bb \rightarrow x \cdot x = x \rangle \)

\( \Rightarrow \ (zz) \)

\( zz = \text{arb}(bb) \div \text{arb}(bb) \)

\( \langle \forall x \mid (x \in bb \rightarrow x \div x = zz) \& x \div zz = x \& zz \div x = x \& zz \in bb \rangle \)

**END** booleanRing

296
Here $\Theta$ designates the additive identity. Notice, among the internally derived claims, the commutativity laws.

Due to its entirely algebraic character, this Theory could have been developed somewhat more easily with an autonomous theorem prover oriented to the treatment of equality: as announced in [6, Sec. 3], we plan to implement interfaces between Ref and outer automated proof assistants.

Our next Theory, presupposing the definition of symmetric difference, shows that rings of sets match the assumptions of the Theory booleanRing:

```plaintext
Theory protoBoolean(dd)

0 ≠ \bigcup dd
\forall x, y \mid \{x, y\} \subseteq dd \to x \cap y \in dd
\forall x, y \mid \{x, y\} \subseteq dd \to x \triangle y \in dd
\Rightarrow
dd ≠ \emptyset
\forall x \in dd, y \in dd, z \in dd \mid x \cap (y \cap z) = (x \cap y) \cap z
\forall x \in dd, y \in dd, z \in dd \mid x \triangle (y \triangle z) = (x \triangle y) \triangle z
\forall x \in dd, y \in dd, z \in dd \mid (x \triangle y) \cap z = z \cap y \cap z \cap x
\forall x \in dd, y \in dd \mid x \triangle x = y \triangle x
\forall x \in dd \mid x \cap x = x
End protoBoolean
```

Two claims, proved inside the background Theory, namely Set theory, and presupposing the definition of symmetric difference, shows that rings of sets match the assumptions of the Theory booleanRing:

```plaintext
Thm. W ≠ \emptyset \& \{X, Y\} \subseteq \mathcal{P} W \to \{X \cap Y, X \triangle Y\} \subseteq \mathcal{P} W \& \bigcup \mathcal{P} W \neq \emptyset.
Thm. W ≠ \emptyset \& D = \{s \subseteq W \mid \text{Finite}(s) \lor \text{Finite}(W \setminus s)\} \& \{X, Y\} \subseteq D \to \{X \cap Y, X \triangle Y\} \subseteq D \& \bigcup D \neq \emptyset.
```

Another Theory, akin to the preceding one, introduces a slightly more specific algebraic variety than the one treated by protoBoolean:

```plaintext
Theory archeoBoolean(dd)

0 ≠ \bigcup dd
\forall x, y, z \mid \{x, y\} \subseteq dd \& z \subseteq x \cup y \to z \in dd
\Rightarrow
\forall x, y \mid \{x, y\} \subseteq dd \to x \cap y \in dd
\forall x, y \mid \{x, y\} \subseteq dd \to x \triangle y \in dd
dd ≠ \emptyset
End archeoBoolean
```

After switching back to the background Set theory level, one proves that there are fields of sets which are instances of protoBoolean but are not instances...
of archeoBoolean. Indeed, the collection of all finite and cofinite subsets of an infinite set is not closed with respect to inclusion.

\[
\text{Thm} . \neg \text{Finite}(W) \land D = \{ s \subseteq W \mid \text{Finite}(s) \lor \text{Finite}(W \setminus s) \} \rightarrow W \in D \land (\exists z \subseteq W \mid z \notin D).
\]

Surprisingly enough, it is unnecessary to resort to a theory of cardinals of any sophistication in order to get the result just cited: the distinction between finite sets and sets which are not finite more than suffices for that purpose, where the following definition applies:

\[
\text{Def : [Finitude]} \text{Finite}(F) \leftrightarrow \forall g \in \mathcal{P}(F) \setminus \{\emptyset\}, \exists m \mid g \cap \mathcal{P}m = \{m\}
\]

Last but not least, we developed the Theory booleanAlgebra whose interface is shown in the Appendix.

Conclusions and future work

Proof-verification can highly benefit from representation theorems of the kind illustrated by Stone’s results on Boolean algebras. On the human side, such results disclose new insights by shedding light on a discipline from unusual angles; on the technological side, they enable the transfer of proof methods from one realm of mathematics to another.

Examples of this can be found in various recent proofs concerning connected claw-free graphs: thanks to a convenient choice on how to represent those graphs, Milanič and Tomescu [13] proved with relative ease two classical propositions, namely that any such graph owns a near-perfect matching and has a Hamiltonian cycle in its square; a proof of the somewhat deeper theorem [8] that all connected claw-free graphs have a vertex-pancyclic square was also attained cheaply through the same representation [23]. Specifically, the facilitation stems from transferring those results to the special class of the membership digraphs, whose set of vertices is a hereditarily finite set and whose arcs precisely reflect the membership relation between vertices. Under this change of perspective, a fully formal reconstruction of the first two results became affordable and, once carried out, was certified correct with the Ref proof-checker [15,17,18].

This motivated us in undertaking the formal development, with Ref, of proofs of various representation theorems (see also [1]). An envisaged continuation of the present work will be in the direction of MV-algebras (cf. [14,5,3,11]).

10 A graph is said to be claw-free if no induced subgraph of its is isomorphic to the graph, called the claw: \(K_{1,3} = \{\{w, x, y, z\}, \{y, x\}, \{y, z\}, \{y, w\}\}\).
References


A  The main theory in our scenario on Boolean algebras

**THEORY** booleanAlgebra\((B, \cdot, \div, 1_B)\)

\[1_B \in B\]
\[1_B \neq 1_B \div 1_B\]
\[\forall x, y \mid \{x, y\} \subseteq B \rightarrow x \cdot y \in B\]
\[\forall x, y, z \mid \{x, y, z\} \subseteq B \rightarrow x \cdot (y \cdot z) = (x \cdot y) \cdot z\]
\[\forall x, y, z \mid \{x, y, z\} \subseteq B \rightarrow x \cdot (y \div z) = (x \div y) \div z\]
\[\forall x, y \mid \{x, y\} \subseteq B \rightarrow x \div x = y \div y\]
\[\forall x \mid x \in B \rightarrow x \cdot x = x\]
\[\Rightarrow (0_\varnothing, (\dashv)_\varnothing, \text{Ideal}_\varnothing, \text{BooHom}_\varnothing, \mathcal{H}_\varnothing, \varphi_\varnothing)\]
\[0_\varnothing = \text{arb}(B) \div \text{arb}(B)\]
\[\forall x \mid x \in B \rightarrow x \div x = 0_\varnothing \& x \div 0_\varnothing = x \& 0_\varnothing \div x = x\] and \(\& 0_\varnothing \in B\)
\[\forall x, y \mid \{x, y\} \subseteq B \rightarrow x \div y = y \div x\]
\[\forall x \mid x \in B \rightarrow (x \div y) = 0_\varnothing \& (x \div 0_\varnothing) = 0_\varnothing\]
\[\forall x, y \mid \{x, y\} \subseteq B \rightarrow x \div y \div z = y \div x \div z\]
\[\forall x \mid x \in B \rightarrow (x \div y) = (y \div z) \div x\]
\[\forall x \mid x \in B \rightarrow x \cdot (y \div x) = y\]
\[\forall x \mid x \in B \rightarrow 1_B \div x = x\]

300
\[ H_\Theta = \{ h \subseteq B \times 2 \mid \text{BooHom}_\Theta(h) \} \]
\[ \forall h \mid h \in H_\Theta \rightarrow h|\emptyset_\Theta = \emptyset \land h|1_B = 1 \]
\[ \forall h \mid h \in H_\Theta \land \{ x, y \} \subseteq B \land h|(x \setminus x \cdot y) = 1 \land h|y = \emptyset \rightarrow h|x = 1 \]
\[ \forall x, m \mid x \notin m \land x \in B \land \langle \forall j \mid \text{Ideal}_\Theta(j) \land m \subseteq j \leftrightarrow j = m \rangle \rightarrow (\times)_\Theta \in m \]
\[ \forall m \mid \langle \forall j \mid \text{Ideal}_\Theta(j) \land m \subseteq j \leftrightarrow j = m \rangle \rightarrow \{ [x, \text{if } x \in m \text{ then } \emptyset \text{ else } 1, 1] : x \in B \} \in H_\Theta \]
\[ B \subseteq \{ \emptyset_\Theta, 1_B \} \rightarrow \{ [\emptyset_\Theta, 0], [1_B, 1] \} \in H_\Theta \]
\[ \forall x \in B \setminus \{ \emptyset_\Theta \} \mid \{ h \in H_\Theta \mid h|x = 1 \} \neq \emptyset \land H_\Theta \neq \emptyset \]
\[ \varphi_\Theta = \{ [b, \{ h \in H_\Theta \mid h|b = 1 \}] : b \in B \} \]
\[ \forall x \in B \mid \varphi_\Theta|x = \{ h \in H_\Theta \mid h|x = 1 \} \]
\[ \forall x, y \mid \{ x, y \} \subseteq B \rightarrow \varphi_\Theta|(x \cdot y) = \varphi_\Theta|x \cap \varphi_\Theta|y \land \varphi_\Theta|(x \setminus y) = \varphi_\Theta|x \setminus \varphi_\Theta|y \]
\[ \forall x, y \mid \{ x, y \} \subseteq B \land x \cdot y \neq x \leftrightarrow \varphi_\Theta|x \neq \varphi_\Theta|y \]
\[ H_\Theta = \bigcup_{\varphi_\Theta \subseteq B} \{ \varphi_\Theta|0_\Theta = \emptyset \land \varphi_\Theta|1_B \neq \varphi_\Theta|0_\Theta \land \varphi_\Theta|1_B = H_\Theta \}
\[ \forall x \in B \mid \varphi_\Theta|(x)_\Theta = H_\Theta \setminus \varphi_\Theta|x \}
\[ 1 \rightarrow 1(\varphi_\Theta) \land \text{domain}(\varphi_\Theta) = B \]
\[ \text{BooHom}_\Theta(\varphi_\Theta) \]
\[ \text{range}(\varphi_\Theta) \subseteq \{ x : x \subseteq H_\Theta \} \land \emptyset \in \text{range}(\varphi_\Theta) \land H_\Theta \in \text{range}(\varphi_\Theta) \]
\[ \forall u \in \text{range}(\varphi_\Theta) \mid H_\Theta\setminus u \in \text{range}(\varphi_\Theta) \}
\[ \forall f, g \mid \{ f, g \} \subseteq H_\Theta \& f \neq g \rightarrow \exists u \in \text{range}(\varphi_\Theta) \cdot v \in \text{range}(\varphi_\Theta) \mid f \in u \land g \in v \land u \setminus v = \emptyset \} \]
\[ \forall f \subseteq \text{range}(\varphi_\Theta) \mid f \neq \emptyset \& \text{Finite}(f) \rightarrow \bigcap f \in \text{range}(\varphi_\Theta) \}
\[ \forall k \subseteq \text{range}(\varphi_\Theta) \mid k \neq \emptyset \& \bigcap k = \emptyset \rightarrow \exists f \subseteq k \mid f \neq \emptyset \& \text{Finite}(f) \rightarrow \bigcap f = \emptyset \}
\]
Short papers
A framework for the verification of parameterized infinite-state systems

Francesco Alberti¹,³, Silvio Ghilardi², Natasha Sharygina¹

¹ University of Lugano, Lugano, Switzerland
² Università degli Studi di Milano, Milan, Italy
³ Verimag, Grenoble, France

Abstract. We present our tool, developed for the analysis and verification of parameterized infinite-state systems. The framework has been successfully applied in the verification of programs handling unbounded data-structures. In such application domain, being able to infer quantified invariants is a mandatory requirement for successful results. We will describe the techniques implemented in our system and discuss how they contribute in achieving important results in the analysis of parameterized distributed and timed systems, as well as of programs with arrays of unknown length.

1 Introduction

Efficient and automatic static analysis of imperative programs is still an open challenge. A promising line of research investigates the use of model-checking coupled with abstraction-refinement techniques [11, 20, 27, 30, 35, 36] including Lazy Abstraction [12, 32] and its later improvements that use interpolants during refinement [34]. An intrinsic limitation of the approaches based on Lazy Abstraction is that they manipulate quantifier-free formulæ to symbolically represent states. However, when defining properties over arrays, universal quantified formulæ are needed, e.g., as in specifying the property “the array is sorted”. The tool we present MCMT (in the new version 2.5) is based on a novel approach [4], in which Lazy Abstraction is used in combination with the backward reachability analysis of array-based systems [28]; recent acceleration techniques for arrays [8, 9] have also been significantly (although not yet completely) included in the latest version of the tool.

2 The Tool

MCMT takes as input a transition system \((v, \tau(v, v'), \iota(v))\) representing the encoding of an array-based system [28]; this can be a parameterized distributed system, a network of timed automata, an imperative program, etc. The essential nature of the specification system is its parametricity: a finite (but unspecified)
number of components takes part in it, the components being interpreted as single processes, agents, array cells, etc. Formally, the array based system is specified by fixing a tuple $\mathbf{v}$ of state variables, a formula $\iota(\mathbf{v})$ describing initial states and a formula $\tau(\mathbf{v}, \mathbf{v}')$ relating current variables $\mathbf{v}$ with their updated counterparts $\mathbf{v}'$. State variables are typed and some of them represent arrays, modeled as free function symbols from a sort of indexes $\text{INDEX}$ to some elements $\text{ELEM}_1, \ldots, \text{ELEM}_k$ sorts. The type of indexes is natural numbers, whereas the types of elements can be integers, Boolean, reals, enumerated data-types, etc. A set of formulæ $\{U_k(\mathbf{v})\}$ representing unsafe states is also given to the tool; each $U_k$ represents an undesired property, e.g. a violation of an assertion in the code.

Next we describe the main features of the tool.

**Symbolic Reachability Analysis** - This module implements a classical backward reachability analysis [1–3]. Starting from the set of unsafe states, it repeatedly computes the pre-images with respect to the transition relation. It halts once it finds (the negation of a) safe inductive invariant $\mathcal{S}$ for the input system or when a run from an initial state to an unsafe state is found. The symbolic reachability search is based on the safety and the covering tests: the former checks the violation of an assertion while the latter tests fix-points.

**Lazy Abstraction** - The search for a safe inductive invariant on the original (concrete) system may require a lot of resources or it cannot be computed because of possible divergence. To mitigate this problem, MCMT extends the Lazy Abstraction paradigm by allowing existentially quantified formulæ to represent states. Moreover, MCMT is able to introduce new quantified predicates on the fly, by means of Term Abstraction or Acceleration, see below.

**Acceleration** - Acceleration is a well established technique in model-checking: the acceleration (i.e. the transitive closure) of a relation encoding systems evolution (like loops in programs) allows us to compute ‘in one shot’ the reachable set of states after an arbitrary but finite number of execution steps. This has the great advantage of keeping under control sources of (possible) divergence arising in the reachability analysis. Definability results for accelerations are well known in numerical domains like difference bounds constraints [17, 21], octagons [14] and finite monoid affine transformations [26] (the paper [16] presents a general approach covering all these domains); however, little is known for more complex data structures like arrays (but see [15]). In [8,9] it is shown that the acceleration of relations corresponding to some classes of guarded assignments over arrays lead to formulæ in decidable fragments of the theory of arrays; as a consequence, some common classes of imperative programs over arrays (including those implementing searching, initializing, finding, copying, comparing functions) have decidable reachability problems. Acceleration for arrays is another way of introducing quantifiers in formulæ describing reachable states; it is only partially implemented in MCMT, where it is exploited for over-approximations in abstraction/refinements loops.

**Quantifier Handling** - The presence of quantified formulæ imposes particular attention during the satisfiability tests: available SMT-Solvers might not be able to deal automatically with such quantified formulæ. MCMT provides a
specific instantiation procedure, adapted from [29] to address this issue. To be effective, this procedure implements caching of information inside of specific data-structures used to represent formulæ. On one hand the caching increases the amount of space, on the other hand it cuts the number of instantiations due to constant-time checks.

**Refinement** - This module receives an abstract counterexample and it checks first if the counterexample has a concrete counterpart. If so a feasible execution violating an assertion corresponding to some satisfied $U_k$ is returned to the user. Otherwise the formulæ representing the states along the abstract execution trace have to be strengthened, possibly by adding new predicates, in order to rule out spurious executions. In the current implementation, refinement is performed by means of a form of interpolation guided by term abstraction.

**Term Abstraction** - Term Abstraction [4] is a novel technique applied during the abstraction phase to select the “right” over-approximation to be computed, and during the refinement phase to “lift” the concrete infeasible counterexample to a more abstract level, by eliminating some terms. Term Abstraction (implemented also in the SAFARI tool [5]) is the main heuristic which distinguishes MCMT from other tools based on abstraction-refinement. It works as follows. Suppose we are given a list of undesired terms $t_1, \ldots, t_n$ (called term abstraction list). The underlying idea is that terms in this list should be abstracted away for achieving convergence of the model checker. Iteratively, these terms are abstracted out (if possible) from formulæ over-approximating sets of reachable states; one way to do this is to replace them by fresh free constants, so that they are likely not to occur anymore in interpolants or in formulæ to which quantifier elimination is applied. MCMT retrieves automatically from the input system a list of terms to be abstracted. The terms to be abstracted are usually set to iterators or variables representing the lengths of the arrays or the bounds of loops. The user can also suggest terms to be added to the list.

**Specification Syntax** - MCMT has its own specification language (roughly, an extension of the specification language of the underlying SMT-Solver YICES). Even though it has been significantly improved, it is still rather low level. Such a language is exploited by the BOOSTER verifying compiler.

## 3 Implementation and Related Work

MCMT is written in C and can be downloaded from [http://users.mat.unimi.it/users/ghilardi/mcmt/](http://users.mat.unimi.it/users/ghilardi/mcmt/). Information on the usage of the tool and a full description of all the options can be found on the User Manual that can be downloaded from MCMT’s website. The use of the appropriate options is crucial not only for performances, but also for convergence (some benchmarks can be solved by plain backward search, in some cases run-time invariant search can be exploited to speed up the tool, for imperative programs it is essential to use abstraction/refinement mode or acceleration or both). MCMT relies on

---

4The interested reader is pointed to the BOOSTER web-page, [http://inf.usi.ch/phd/alberti/prj/booster](http://inf.usi.ch/phd/alberti/prj/booster) for more information about it.
the SMT-Solver Yices (see http://yices.csl.sri.com/) to decide satisfiability queries; the solver can be linked to the model-checker on a client/server architecture via API. mcmt distribution includes about 100 examples files taken from different sources (cache coherence and mutual exclusion problems, timed and fault tolerant systems, imperative programs, etc); more examples related to specific case studies [6, 7, 18, 19] can be reached from mcmt web-page. A Table covering few experimental results is attached in the Appendix below. 

Current literature on infinite state model checking is extremely large, however it is much more limited if we restrict to papers and tools handling parametric specifications. For distributed systems, the best performing tool (resulting from an extension and a re-implementation on a parallel architecture of mcmt framework) is probably Cubicle [22, 23]. In software model checking, unbounded arrays problems have been attacked from various viewpoints, including abstract interpretation [24, 25, 31], program transformations [10], predicate abstraction [35, 36] and template/constraints generation [13, 33, 37]. An experimental tool comparison is difficult for various practical reasons; however, since most benchmarks are taken from common sources, from the results reported in the above mentioned papers, it seems that mcmt compares well, both in terms of performances and in terms of solved instances.

References

A Some experimental data

We report in the Table below some experimental data on benchmark problems; we made a selection including both easy well known problems and more challenging ones. The experiments were run on a laptop Intel(R) Core(TM) i3 CPU 2.27GHz with 4GB RAM running Linux Ubuntu 12.04.

In the second column we indicate the class of the problem: (M) mutual exclusion, (C) cache coherence, (D) other distributed protocols (timed, fault tolerant, etc.), (S) sequential program for arrays, (S+) sequential program for arrays with nested loops. In column 3-8 we respectively give the depth of the search tree, the number of nodes generated by the tool in the search tree, the number of subsumed or subcovered nodes, the number of calls to the SMT solver, the number of invariants found by the tool in forward search and the number of refinements applied in abstraction/refinement mode. In the last column we put the total time in seconds and in the last-but-one column the options used (A=acceleration, AR=abstraction/refinement, I=invariant search). For each problem we reported the result in the best configuration we found for the tool.

<table>
<thead>
<tr>
<th>Problem</th>
<th>kind</th>
<th>d</th>
<th>#n</th>
<th>#del</th>
<th>#SMT</th>
<th>#inv</th>
<th>#ref</th>
<th>heur</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Illinois</td>
<td>(C)</td>
<td>4</td>
<td>8</td>
<td>0</td>
<td>212</td>
<td>0</td>
<td>0</td>
<td></td>
<td>-0.06</td>
</tr>
<tr>
<td>German</td>
<td>(C)</td>
<td>26</td>
<td>2121</td>
<td>255</td>
<td>117121</td>
<td>0</td>
<td>0</td>
<td></td>
<td>60.76</td>
</tr>
<tr>
<td>German_buggy</td>
<td>(C)</td>
<td>16</td>
<td>1300</td>
<td>263</td>
<td>24275</td>
<td>0</td>
<td>0</td>
<td></td>
<td>14.28</td>
</tr>
<tr>
<td>Bakery</td>
<td>(M)</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>29</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0.00</td>
</tr>
<tr>
<td>Szymanski</td>
<td>(M)</td>
<td>11</td>
<td>17</td>
<td>5</td>
<td>1092</td>
<td>12</td>
<td>0</td>
<td>I</td>
<td>0.21</td>
</tr>
<tr>
<td>Szymanski_atomic</td>
<td>(M)</td>
<td>19</td>
<td>63</td>
<td>7</td>
<td>5470</td>
<td>32</td>
<td>0</td>
<td>I</td>
<td>1.82</td>
</tr>
<tr>
<td>Distributed_Lamport</td>
<td>(M)</td>
<td>23</td>
<td>248</td>
<td>42</td>
<td>19622</td>
<td>7</td>
<td>0</td>
<td>I</td>
<td>27.18</td>
</tr>
<tr>
<td>Crash</td>
<td>(D)</td>
<td>13</td>
<td>113</td>
<td>21</td>
<td>1731</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0.75</td>
</tr>
<tr>
<td>Fischer</td>
<td>(D)</td>
<td>10</td>
<td>16</td>
<td>2</td>
<td>363</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0.08</td>
</tr>
<tr>
<td>Fischer_buggy</td>
<td>(D)</td>
<td>6</td>
<td>16</td>
<td>0</td>
<td>307</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0.06</td>
</tr>
<tr>
<td>Lynch-Shavit_full</td>
<td>(D)</td>
<td>25</td>
<td>1103</td>
<td>99</td>
<td>56638</td>
<td>0</td>
<td>0</td>
<td></td>
<td>33.39</td>
</tr>
<tr>
<td>Strcpy</td>
<td>(S)</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>48</td>
<td>0</td>
<td>0</td>
<td>A</td>
<td>0.01</td>
</tr>
<tr>
<td>Strcmp</td>
<td>(S)</td>
<td>6</td>
<td>10</td>
<td>4</td>
<td>128</td>
<td>0</td>
<td>0</td>
<td>A</td>
<td>0.02</td>
</tr>
<tr>
<td>Max_in_array</td>
<td>(S)</td>
<td>7</td>
<td>13</td>
<td>6</td>
<td>166</td>
<td>0</td>
<td>0</td>
<td>A</td>
<td>0.04</td>
</tr>
<tr>
<td>Reverse</td>
<td>(S)</td>
<td>4</td>
<td>8</td>
<td>5</td>
<td>101</td>
<td>0</td>
<td>0</td>
<td>A</td>
<td>0.03</td>
</tr>
<tr>
<td>Palindrome</td>
<td>(S)</td>
<td>4</td>
<td>7</td>
<td>4</td>
<td>107</td>
<td>0</td>
<td>0</td>
<td>A</td>
<td>0.04</td>
</tr>
<tr>
<td>AllDifferent</td>
<td>(S+)</td>
<td>7</td>
<td>49</td>
<td>39</td>
<td>871</td>
<td>0</td>
<td>8</td>
<td>A + AR</td>
<td>0.40</td>
</tr>
<tr>
<td>BubbleSort</td>
<td>(S+)</td>
<td>5</td>
<td>14</td>
<td>10</td>
<td>200</td>
<td>0</td>
<td>0</td>
<td>A</td>
<td>0.07</td>
</tr>
<tr>
<td>InsertionSort</td>
<td>(S+)</td>
<td>18</td>
<td>98</td>
<td>56</td>
<td>3874</td>
<td>0</td>
<td>2</td>
<td>AR</td>
<td>1.43</td>
</tr>
<tr>
<td>SelectionSort</td>
<td>(S+)</td>
<td>8</td>
<td>101</td>
<td>77</td>
<td>6059</td>
<td>8</td>
<td>11</td>
<td>AR + I</td>
<td>4.98</td>
</tr>
</tbody>
</table>

\(^5\)Corresponding command line options: \(-Z\) gives acceleration, \(-i1, -i2, -i3, -a, -I\) give different invariant searches in normal mode, \(-AN\) gives abstraction-refinement (with max \(N\) refinements per node), \(-CN\) gives abstraction-refinement (with max \(N\) refinements per node) together with a specific form of invariant search.
On Relating Voting Systems and Argumentation Frameworks

Irene Benedetti⋆, Stefano Bistarelli⋆⋆, and Paolo Piersanti

Dipartimento di Matematica e Informatica, Università di Perugia
[irene.benedetti,bista]@dmi.unipg.it
paolopiers@gmail.com

Abstract. In the modern world formal voting theories are becoming established and can be used to determine if a Voting System (VS) is fair or not in order to preserve democracy. The Argumentation Framework (AF) is based on the exchange and the evaluation of interacting arguments which may represent information of various kinds. We define a bijective mapping between the two theories and investigate how fairness criteria defined for voting systems can be re-interpreted inside the Argumentation Frameworks.

1 Introduction

The analysis of voting methods and their properties start from the pioneering work of Arrow [1]. Using his impossibility theorem results classical and nowadays voting and election systems can be analyzed and some fairness judgements can be expressed.

The formal study of argumentation has come to be increasingly central as a core study within Artificial Intelligence and it is also of interest in several disciplines, such as Logic, Philosophy and Communication Theory [7]. Argumentation [4, 5] is based on the exchange and the evaluation of interacting arguments which may represent information of various kinds, especially beliefs or goals. Many theoretical and practical developments are built on Dung’s seminal theory of argumentation.

We define a voting system VS as a function that associates to the given set of votes, the elected candidate(s). We then define a map from VSs to AFs and study the known semantics (ground extension in particular) as voting methods.

2 Argumentation framework

Definition 1. An Argumentation Framework (AF) is a pair \((\mathcal{A}_{rgs}, R)\) of a set \(\mathcal{A}_{rgs}\) of arguments and a binary relation \(R\) on \(\mathcal{A}_{rgs}\) called the attack relation. \(\forall a_i, a_j \in \mathcal{A}_{rgs}, a_i R a_j\) means that \(a_i\) attacks \(a_j\). An AF may be represented by a directed graph (the interaction graph) whose nodes are arguments and edges represent the attack relation. A set of arguments \(\mathcal{B}\) attacks an argument \(a\) if \(a\) is attacked by an argument of \(\mathcal{B}\). A set of arguments \(\mathcal{B}\) attacks a set of arguments \(\mathcal{C}\) if there is an argument \(b \in \mathcal{B}\) which attacks an argument \(c \in \mathcal{C}\).

⋆ partially supported by the Italian Research Project GNAMPA 2014: “Metodi Topologici: sviluppi ed applicazioni a problemi differenziali non lineari”.
⋆⋆ Partially supported by the italian MIUR project PRIN "Metodi logici per il trattamento dell’informazione"
The “acceptability” of an argument [5] depends on its membership to some sets, called extensions. These extensions characterize collective “acceptability”. Dung [5] gave several semantics to “acceptability”. These various semantics produce none, one or several acceptable sets of arguments, called extensions. In Def. 2 we define the concepts of conflict-free and stable extensions:

Definition 2. A set \( B \subseteq \text{Args} \) is conflict-free iff no two arguments \( a \) and \( b \) in \( B \) exist such that \( a \) attacks \( b \). A conflict-free set \( B \subseteq \text{Args} \) is a stable extension iff for each argument which is not in \( B \), there exists an argument in \( B \) that attacks it.

The other semantics for “acceptability” rely upon the concept of defense:

Definition 3. An argument \( b \) is defended by a set \( B \subseteq \text{Args} \) (or \( B \) defends \( b \)) iff for any argument \( a \in \text{Args} \), if \( a \) attacks \( b \) then \( B \) attacks \( a \).

An admissible set of arguments according to Dung must be a conflict-free set which defends all its elements. Formally:

Definition 4. A conflict-free set \( B \subseteq \text{Args} \) is admissible iff each argument in \( B \) is defended by \( B \).

Besides the stable semantics, three semantics refining admissibility have been introduced by Dung [5]:

Definition 5. A preferred extension is a maximal (w.r.t. set inclusion) admissible subset of \( \text{Args} \). An admissible \( B \subseteq \text{Args} \) is a complete extension iff each argument which is defended by \( B \) is in \( B \). The least (w.r.t. set inclusion) complete extension is the grounded extension.

A stable extension is also a preferred extension and a preferred extension is also a complete extension. Stable, preferred and complete semantics admit multiple extensions whereas the grounded semantics ascribes a single extension to a given argument system. Since the grounded extension is proven to be unique, and we are going to define a new voting systems using Argumentation Semantics, this semantics will be our best candidate (see Section 4).

3 Voting Systems

The process of cooperative decision making has been formalized using formal social choice theory and formal game theory, see e.g. [3, 8]. A voting system enforces rules to ensure valid voting, and how votes are counted and aggregated to yield a final result.

More formally, a voting system specifies the form of the ballot, the set of allowable votes, and the tallying method, an algorithm for determining the outcome. This outcome may be a single winner, or may involve multiple winners such as in the election of a legislative body. We focus our study on the non-preferential voting methods such as the block voting.

Example 1 (Block voting). This non-preferential voting method is used to elect \( n \) options from a group of \( m \) options (\( m > n \)). The voter has to point out \( l \) with \( n \geq l \) preferences between the \( m \) available. Consider five candidates \( A, B, C, D \) and \( E \) and suppose each of them vote three options between the five proposed. Let’s suppose the result yielded by the election is as in Table 1. This situation leads to a tie because there are four candidates, each one with three votes received (or to elect all of them). □
Table 1: Selecting the winner with block voting.

<table>
<thead>
<tr>
<th>Voter/ Candidate</th>
<th>Vote</th>
<th>Votes received</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B, A, D</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>C, D, E</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>A, B, C</td>
<td>3</td>
</tr>
<tr>
<td>D</td>
<td>D, E, A</td>
<td>3</td>
</tr>
<tr>
<td>E</td>
<td>C, E, B</td>
<td>2</td>
</tr>
</tbody>
</table>

Different voting systems may give very different results, particularly in cases where there is no clear majority preference. Many fairness criteria were defined; We remind here the five basic criteria of fairness proposed by Arrow in 1950 [1] and revised in 1963 [2].

**Definition 6 (Arrow Fairness Criteria).**

1. **Universal admissibility (unrestricted domain):** Voting systems should not place any restrictions other than transitivity on how voters can rank the candidates in an election.

2. **Monotonicity:** if an election is held and a winner is declared, this winning candidate should remain the winner in any revote in which all preference changes are in favor of the winner of the original election.

3. **Independence of irrelevant alternatives (IIA) (binary independence):** If an election is held and a winner is declared, this winning candidate should remain the winner in any recalculation of votes as a result of one or more of the losing candidates dropping out.

4. **Condition of citizens sovereignty (non imposition):** Voting systems should not be imposed in any way. That is, there should never be a pair of candidates in an election, say A and B, such that A is preferred over or tied with B in the resulting social preference order regardless of how any of the voters vote.

5. **Condition of non-dictatorship:** Voting systems should not be dictatorial. That is, there should never be a voter v such that, for any pair of candidates A and B, if v prefers A over B, then society will also prefer A over B.

**Theorem 1 (Arrow [1,2]).** If there are at least three candidates, any (preferential) voting method satisfying criteria 1,2 and 3 must be either imposed or dictatorial.

### 4 Main results

In this section we formally define a voting system and the mapping between Voting Systems and Argumentation Frameworks.

**Definition 7 (Ballots and Voting Systems).** A Ballot B is a pair $B = \langle \text{candidates}, \text{voters}, V P \rangle$ of a set $\text{candidates}$ of Candidates, a set $\text{voters}$ of Voters and a Voting Procedure $V P$ representing a binary relation on $\text{voters} \times \mathcal{P}(\text{candidates})$ (where given a set A with $\mathcal{P}(A)$ we denote the power set of A) that associates to each voter $v \in \text{voters}$ her votes to the candidates $C \subseteq \text{candidates}$. A Voting System $\text{vs} : \text{candidates} \cup \text{voters} \times V P \rightarrow \mathcal{P}(\text{candidates})$ is a function assigning to a ballot $B = \langle \text{candidates} \cup \text{voters}, V P \rangle$ a set (or more sets in case of ties) of winning candidates $W \subseteq \mathcal{P}(\text{candidates})$. 
In the rest of this paper we assume the set of Candidates \( C \) and the set of Voters \( V \) to coincide in an unique set of Options \( O = C = V \) (as in many real social elections).

The first of our results is to show that using a suitable mapping between VSs and AFs, the Semantics of an argumentation framework can be used to define a voting system with interesting properties. More in detail, we map every option (representing candidates or voters) to an argument and the relation ‘a votes for b’ to the attacks \( a \rightarrow b \) for any \( b \neq b \) (to support in this way b).

**Definition 8 (from VS to AF and back).** We define a mapping \( m \) from VSs (more specifically from a ballot \( B \)) to AFs \( m : O \times V P \rightarrow \mathcal{A}_{rgs} \times \mathcal{R} \) such that

- for each option \( o \in O \) we consider an argument \( a = m(o) \),
- for each vote \( \langle o, C \rangle \in O \times \mathcal{P}(O) \), we obtain the set of attacks \( m(\langle o, C \rangle) = \{ \langle a, m(c') \rangle, \text{s.t. } c' \notin C \} \)

Using \( m^{-1} \) we can instead define the corresponding mapping from AFs to VSs:

- for each argument \( a \) we consider the corresponding option (candidate/voter) \( o = m^{-1}(a) \),
- for each argument \( a, b \in \mathcal{A}_{rgs} \), and the set \( B = \bigcup_{b \in \mathcal{A}_{rgs}} b \text{ s.t. } \langle a, b \rangle \in \mathcal{R} \), we consider the set of votes \( m^{-1}(a), m^{-1}(B') \), where \( m^{-1}(B') = \{ m^{-1}(b') \text{ s.t. } b' \notin B \} \)

Chosen a semantic \( s \) (i.e. chosen an argumentation function), the result of the election described by the composition \( m^{-1} \circ s \circ m \) as in Fig. 1 is a voting system. We will study which fairness criteria it satisfies.

![Fig. 1: The voting system \( vs = m^{-1} \circ s \circ m \).](image)

**Proposition 1.** The composition \( vs = m^{-1} \circ s \circ m : O \times V P \rightarrow \mathcal{P}(O) \) is a voting system.

**Theorem 2.** Given a semantic grounded \( s \), by the composition \( m^{-1} \circ s \circ m \) we obtain a voting method without ties that satisfies the IIA and monotonicity.

We can apply the mapping in Proposition 1 to the Example 1 of Section 2:
Example 2 (The mapping from block voting Example 1). The block voting example is transformed in the AF as in Figure 2. The elected candidates (using the grounded semantic) are the set \{A, D\}.

5 Conclusions and Future Works

Our proposal uses negative judgements (as attacks) instead of positive ones (preferences). The computation of (indirect) positive judgement given by explicitly negative judgment have been already used in Germany in 2005 to elect the German Bundestag [9]. We proved that the voting system constructed using grounded semantics (such as conflict free, admissible, stable,...) satisfies many fairness criteria but the majority criteria. Indeed there are several voting systems that do not satisfy this criterion, see [6]. Moreover we would like to consider the result of an election with a voting system how a specific semantics in AFs. Finally, the situations where \( c_{\text{and}} \neq V_{\text{ers}} \) as well as special restrictions on the voting mechanism (a candidate can vote only one candidate, or at least herself, or preference based votes) will be subject of further research.

References

A First Study of the Horn Fragment of the Modal Logic of Time Intervals

D. Bresolin\(^1\), E. Muñoz-Velasco\(^2\), and G. Sciavicco\(^3\)

\(^1\) Department of Computer Science and Engineering
University of Bologna (Italy)\((\text{davide.bresolin@unibo.it})\)
\(^2\) Department of Applied Mathematics
University of Malaga (Spain)\((\text{emilio@ctima.uma.es})\)
\(^3\) Department of Information, Engineering and Communications
University of Murcia (Spain)\((\text{guido@um.es})\)

Abstract. Interval temporal logics provide a natural framework for temporal reasoning about interval structures over linearly ordered domains, where intervals are taken as the primitive ontological entities. The most influential propositional interval-based logic is probably Halpern’s and Shoham Modal Logic of Time Intervals, a.k.a. HS. While most studies focused on the computational properties of the syntactic fragments that arise by considering only a subset of the set of modalities, the fragments that are obtained by weakening the propositional side of HS have received much less attention. Here, we approach this problem by considering the Horn fragment of HS and proving that the satisfiability problem remains undecidable, at least for discrete linear orders.

1 Introduction

Most temporal logics proposed in the literature assume a point-based model of time, and they have been successfully applied in a variety of fields. However, there are examples of relevant application domains, such as planning and temporal databases, where interesting problems are dealt with in an unsatisfactory way by point-based formalisms\([9, 10]\). Interval temporal logics provide a natural framework for temporal reasoning about interval structures over linearly (or partially) ordered domains. They take time intervals as the primitive ontological entities and define truth of formulas relative to time intervals, rather than time points; their modalities correspond to various relations between pairs of intervals. In particular, the well-known logic HS\([6]\) features a set of modalities that make it possible to express all Allen’s interval relations\([1]\). Unfortunately, in interval temporal logic undecidability is the rule and decidability the exception: HS is undecidable when interpreted on most meaningful classes of linearly ordered sets, and limiting the set of temporal modalities of the logic is not always sufficient to achieve decidability. For example, when formulas are interpreted over strongly discrete linear orders, only 44 decidable fragments exist\([3]\).

\(^\star\) The authors acknowledge the support from the Italian GNCS Project “Automata, games and temporal logics for verification and synthesis of safety-critical systems” (D. Bresolin), the Spanish Project TIN2012-39353-C04-01 (E. Muñoz-Velasco), and the Spanish fellowship program ‘Ramon y Cajal’ RYC-2011-07821 (G. Sciavicco).
In this context, it makes sense to study sub-propositional fragments of HS, such as the Horn fragment. Horn fragments of modal and temporal logics have been studied, for example, in [2, 4, 5, 7]. Being sub-propositional, the obvious question is whether or not the satisfiability problem of Horn HS remains undecidable, and, if so, on which classes of linearly ordered sets. The results presented in this preliminary study proves that, unfortunately, at least in the class of models built over \(\mathbb{Z}\) and other strongly discrete linear orders, this is the case. While discouraging, these results should be seen as the first attempt of studying sub-propositional fragments of interval temporal logics. Horn HS is still undecidable, but we might find out that the decidability frontier is different from the one for the syntactical fragments of full HS, and/or that some of the decidable fragments present a better computational behaviour.

2 The Logic HS and its Horn Fragment

Halpern and Shoham’s logic HS is a multi-modal logic with formulas built on a set \(\mathcal{AP}\) of proposition letters, the Boolean connectives \(\lor\) and \(\neg\), plus the six modalities to capture the existence of an interval in a particular Allen’s relation with the current one:

\[
\varphi ::= p \mid \neg \varphi \mid \varphi \lor \varphi \mid \varphi \land \varphi \mid \langle A \rangle \varphi \mid \langle A \rangle \varphi \mid \langle B \rangle \varphi \mid \langle B \rangle \varphi \mid \langle E \rangle \varphi \mid \langle E \rangle \varphi
\]

The other Boolean connectives, the box modalities, and the temporal modalities corresponding to other Allen’s relations (such as \(\langle D \rangle\), \(\langle L \rangle\), and \(\langle O \rangle\)) are definable in the language (e.g. \([A]\varphi \equiv \neg \langle A \rangle \neg \varphi\), \(\langle D \rangle \varphi = \langle E \rangle \langle B \rangle \varphi\), and \(\langle L \rangle \varphi = \langle A \rangle \langle A \rangle \varphi\)).

Let \(\mathbb{D} = \langle D, <\rangle\) be a discrete linearly ordered set. An interval over \(\mathbb{D}\) is an ordered pair \([x, y]\), where \(x, y \in D\) and \(x < y\) (strict semantics). An interval of the type \([x, x+1]\) is called unit. The semantics of HS is given in terms of interval models \(M = \langle \mathbb{I}(\mathbb{D}), V \rangle\), where \(\mathbb{I}(\mathbb{D})\) is the set of all intervals over \(\mathbb{D}\) and \(V : \mathcal{AP} \to 2^{\mathbb{I}(\mathbb{D})}\) is a valuation function that assigns to every \(p \in \mathcal{AP}\) the set of intervals \(V(p)\) over which \(p\) holds. The truth relation \(\models\) of a formula over a given interval \([x, y]\) in an interval model \(M\) is defined by structural induction on formulas (Boolean connectives are dealt with in the standard way):

- \(M, [x, y] \models p\) iff \([x, y] \in V(p)\);
- \(M, [x, y] \models \langle A \rangle \varphi\) iff there exists \(z > y\) such that \(M, [y, z] \models \varphi\);
- \(M, [x, y] \models \langle A \rangle \varphi\) iff there exists \(z < x\) such that \(M, [z, x] \models \varphi\);
- \(M, [x, y] \models \langle B \rangle \varphi\) iff there exists \(x < z < y\) such that \(M, [x, z] \models \varphi\);
- \(M, [x, y] \models \langle B \rangle \varphi\) iff there exists \(z > y\) such that \(M, [x, z] \models \varphi\);
- \(M, [x, y] \models \langle E \rangle \varphi\) iff there exists \(x < z < y\) such that \(M, [z, y] \models \varphi\);
- \(M, [x, y] \models \langle E \rangle \varphi\) iff there exists \(z < x\) such that \(M, [z, y] \models \varphi\).

A HS-formula \(\varphi\) is satisfiable if and only if there exists a model \(M\) and an interval \([x, y]\) such that \(M, [x, y] \models \varphi\). The satisfiability problem for HS is the problem of finding a model and an interval that satisfies a formula.

Following [7], we define the Horn fragment of HS. First, consider only formulas in negative normal form (nnf) (i.e., with only \(\land\) and \(\lor\) as Boolean connectives and such that \(\neg\) occurs only in front of propositions). A formula in nnf is called negative if and
only if every proposition is prefixed by negation; it is non-negative if it is not negative, and positive if its negation is a negative formula. An HS-formula \( \varphi \) is called a Horn HS-formula (or, simply, Horn) if and only if (i) \( \varphi \) is a proposition; (ii) \( \varphi \) is a negative formula; (iii) \( \varphi = [X] \psi \), \( \varphi = \langle X \rangle \psi \), or \( \varphi = \psi \land \rho \), where \( \psi \) and \( \rho \) are Horn and \( \langle X \rangle \) is any HS modality; (iv) \( \varphi = \psi \rightarrow \rho \), where \( \psi \) is positive and \( \rho \) is Horn; (v) \( \varphi \) is a disjunction of a negative formula and a Horn formula. Alternative definitions of modal Horn clauses can be found, for example, in [2, 4, 5]. However, these definitions are equivalent to Nguyen’s one [7], in the sense that every given set of Horn formulas (or clauses) in the latter can be translated into set of equi-satisfiable Horn clauses of any of the formers.

In the following we will make use of the universal modality \([U]\), that can be defined in Horn HS as a conjunction of boxes \([U] \varphi = [A][A][A] \varphi \land [A][A] \varphi \land [A][A][A] [A] \varphi \), and of the difference modality \([\not=]\), meaning for every interval, except the current one, that can be defined in a similar way.

### 3 Undecidability of Horn HS

In this section, we assume that Horn HS is interpreted over \( \mathbb{Z} \). Notice that this assumption can be immediately relaxed, to include Horn HS interpreted over \( \mathbb{N} \), over the class of all strongly discrete linear orders, and over the class of finite linear orders. Our construction, that closely follows both the original undecidability proof for full HS [6] and more recent undecidability proofs for fragments of HS [3], is based on a reduction from the halting problem of a deterministic Turing Machine on empty input [8].

A Turing Machine is defined as a tuple \( A = (Q, \Sigma, \Gamma, \delta, q_0, q_f) \), where \( Q \) is the set of states, \( q_0 \) (resp., \( q_f \)) is the initial (resp., final) state, \( \Sigma \) is the machine’s alphabet that does not contain \( \sqcup \) (blank), \( \Gamma = \Sigma \cup \{\sqcup\} \) is the tape alphabet, and \( \delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\} \) is the transition function (\( L, R \) represent the possible moves on the machine’s tape: left, right). Even under the assumption that \( \Sigma = \{0, 1\} \) and that the input is empty, the halting problem for a deterministic Turing Machine is undecidable [8].

In the following, we reduce the halting problem of a Turing Machine to the satisfiability problem for Horn HS over \( \mathbb{Z} \). The underlying idea is to represent the computation history of the machine using the propositional symbol \( * \) to separate successive configurations, the propositional symbols 0, 1, \( \sqcup \) to represent tape cells not under the machine’s head, and propositional symbols \( q_c \), with \( q \in Q \setminus \{q_f\} \) and \( c \in \{0, 1, \sqcup\} \), to represent the tape cell under the head and the current (non-final) state of the machine. We will use the propositional symbols \( Fr \) to encode the initial configuration, \( u \) to represent the units, \( Co \) to represent a generic configuration, and \( Cr \) to connect successive configurations. When the machine is in the final state \( q_f \) the computation immediately halts. For this reason we can discard the symbol under the head and use a unique propositional letter \( q_f \). We denote by \( L \) the set \( \{0, 1, \sqcup, *\} \cup \{q_c \mid q \in Q \setminus \{q_f\} \land c \in \{0, 1, \sqcup\}\} \cup \{q_f\} \). Any terminating run of the Turing Machine will make use of only a finite portion of the tape. Hence, we can assume w.l.o.g. that all configurations have the same length and that they are long enough to contain the relevant part of the tape.

\[
\langle A \rangle u \land [U] (u \rightarrow (\langle A \rangle u \land [B] \bot)) \quad \text{units’ structure}
\]
Intuitively, the above formulas guarantee that configurations (denoted by $Co$) are built of units, each one of them contains either $*$ or a tape/head element and that there is an infinite and unique sequence of configurations. The proposition $Fr$ sets the structure of the first configuration: a single $*$, followed by $q_0$ reading blank, and a number of blanks, followed by a $*$.

The relation between successive configurations is maintained by the proposition $Cr$ (corresponds), constrained by the following formulas. These are also used to guarantee that all configurations have the same length.

\[
\begin{align*}
[U](u \rightarrow ((A)Cr \land (\overline{A})Cr)) & \quad \text{each unit starts and ends a “Cr”} \\
[U](Cr \rightarrow ((B)\neg Cr \land [D]\neg Cr \land [E]\neg Cr)) & \quad \text{“Cr” structure} \\
[U](Co \rightarrow (\neg Cr \land [D]\neg Cr \land [D]\neg Cr \land [E]\neg Cr)) & \quad \text{“Cr/Co” relation}
\end{align*}
\]

It remains to ensure that the machine $A$ behaves as imposed by $\delta$. In the encoding of the transition function, we treat as special cases the situations in which (i) the head is at the last cell of the segment of the tape currently shown and the head must be moved to the right and, (ii) the head is at the first cell of the tape and the head must be moved to the left. In the following formulas, $c, c', c'' \in \{0, 1, \square\}$, $d \in \{0, 1, \square, *\}$, and $q, q' \in Q$ (by a little abuse of notation, we assume that all symbols $q_f^j$ are equal to $q_f$).

\[
\begin{align*}
\wedge_{d,\delta(q,c)=(q',c',R),c'}[U]((q^c \land (A)c'' \land (\overline{A})d)) & \rightarrow \quad \text{to the right, not the last cell} \\
[A](Cr \rightarrow (A)(c' \land (A)q_{c''} \land (\overline{A})d)) & \\
\wedge_{d,\delta(q,c)=(q',c',L),c'}[U]((q^c \land (A)c'' \land (\overline{A})d)) & \rightarrow \quad \text{to the right, last cell: forbidden} \\
[A](Cr \rightarrow (A)(c' \land (\overline{A})q_{c''} \land (A)d)) & \\
\wedge_{d,\delta(q,c)=(q',c',R),c'}[U]((q^c \land (\overline{A})c'' \land (A)d)) & \rightarrow \quad \text{to the left, first cell} \\
[A](Cr \rightarrow (A)(q_{c'} \land (\overline{A})d)) & \\
\wedge_{d,\delta(q,c)=(q',c',L),c'}[U]((q^c \land (\overline{A})c'' \land (A)d)) & \rightarrow \quad \text{to the left, first cell} \\
[A](Cr \rightarrow (A)(q_{c'} \land (\overline{A})d)) & \\
\wedge_{e,c',c''}[U]((c \land (\overline{A})c' \land (A)c'')) & \rightarrow [A](Cr \rightarrow (A)c))
\end{align*}
\]

Finally, we force cells located away from the head to remain unchanged.

\[
\wedge_{e,c',c''}[U]((c \land (\overline{A})c' \land (A)c'')) \rightarrow [A](Cr \rightarrow (A)c))
\]

The conjunction $\varphi_A$ of all formulas above encodes a computation of the Turing Machine.

**Theorem 1.** Let $A$ be a deterministic Turing Machine. Then, $A$ halts on an empty input if and only if the Horn HS-formula $\varphi_A \land \langle L \rangle q_f$ is satisfiable over $\mathbb{Z}$.

This proves that the satisfiability problem for Horn HS interpreted over $\mathbb{Z}$ is undecidable. The construction can be rephrased to deal with the cases where Horn HS is interpreted over $\mathbb{N}$, or in the class of all strongly discrete linear orders, or over finite linear orders.
4 Conclusions

Sub-propositional fragments, such as the Horn fragment, of classical and modal logics have been extensively studied. The generally high complexity of the (few) decidable interval temporal logics justifies the study of sub-propositional fragments of interval logics in search of expressive languages that present a better computational behaviour. In this paper we proved a first negative result in this sense, by showing that the well-known interval temporal logic $\text{HS}$ is still undecidable when its Horn fragment is considered, at least in the discrete case. Nevertheless, we believe that syntactical fragments of Horn $\text{HS}$ should be studied in the same way in which syntactical fragments of full $\text{HS}$ have been. In the long run, we plan to consider the Horn fragments of decidable interval logics like $\text{AA}$ and $\text{ABBL}$, to understand whether or not their satisfiability problem present a better computational behaviour.

References

Distributed Runtime Verification of JADE and Jason Multiagent Systems with Prolog*

Daniela Briola, Viviana Mascardi, and Davide Ancona
DIBRIS, Genoa University, Italy
daniela.briola,viviana.mascardi,davide.ancona@unige.it

1 Introduction

Verifying properties of interactions taking place inside open, complex, distributed, dynamic systems is of paramount importance for most applications and is mandatory for safety-critical ones. Verification can take place at design time (offline or static verification) or at runtime (online or dynamic). For runtime verification some layer between the monitor executing the verification engine and the system under monitoring must exist, so that actual interactions can be intercepted and the compliance of each one against the protocol can be checked. A common way to improve efficiency and fault tolerance of the runtime verification is to distribute it among many monitors. This requires that the protocol is projected onto subsets of participants.

If the system has been engineered as a multiagent system (MAS), which is a good option when openness, complexity, distribution, dynamics are characterizing features, then the choice of either JADE, http://jade.tilab.com/, or Jason http://jason.sourceforge.net/, as the platform for implementing it may be a very natural one. JADE, implemented in Java, is the state-of-the-art tool for MAS development and has been used for many real industrial applications. Jason, implemented in Java as well and based on a Prolog engine built from scratch by its developers, is one of the most widely used frameworks when the agents under development are designed according to the Belief, Desire, Intentions (BDI) architecture.

Due to the wide range of possible application fields of Jason and to the large amount of real use cases of JADE, being able to verify interactions taking place in MASs implemented in one of these two frameworks is a concrete step towards making MASs more reliable and enhancing their industrial and commercial usability. In this demo we show our contribution for the achievement of this goal.

We have in fact designed and implemented a framework for distributed runtime verification of MASs and ad hoc interfaces for monitoring JADE and Jason interactions. The framework consists of four layers: (1) a formalism for describing agent interaction protocols (AIPs) based on constrained global types [1] and their extension with attributes [7]; (2) a mechanism for projecting AIPs onto subsets of agents, obtaining a new protocol in the same formalism of constrained global types [2]; (3) a mechanism for verifying that interactions are compliant...

* Paper presented at the CILC 2014 Demo Session, based on published material [3, 4].
with the AIP [3]; and (4) a mechanism for intercepting messages involving the agents under monitoring, be them JADE or Jason ones, in a way as transparent as possible.

The strength of our framework, represented in Figure 1, is its high modularity and potential for code reuse: the first three layers are independent from the actual MAS under monitoring and have been implemented in Prolog. The “protocol representation” and “compliance verification” layers have been tested and improved over time, reaching an almost stable version now, whereas the projection layers works under the assumption that the protocol contains no attributes (namely, it is a “plain” constrained global type) and has not been tested extensively yet. The fourth layer depends on the MAS framework under monitoring: nothing prevents us from adding new agent frameworks at the bottom of our architecture by developing ad hoc mechanisms for message interception, still leaving the first three layers unchanged. By exploiting the components offered by our stacked framework it is possible to implement both monitors external to the MAS, implemented as completely separate processes that do not intervene in the observed system, and agents which are able to monitor the protocol executions and have the power to intervene when they detect a violation. Associating the “compliance verification” capability with an artifact (as in the first case) or with an agent (as in the second case) are two different design choices, each with pros and cons. We experimented both approaches, as discussed in Section 3 where monitoring is performed by a Java artifact that does not intervene in the MAS activity, and in Section 4 where the Jason agent in charge of the monitoring activity can prevent other agents from sending non compliant messages.

Whatever the choice, compliance verification should be an efficient process. Although efficiency issues are still to be explored, distributing the runtime verification by projecting onto subsets of agents could be a way to balance the load of the monitoring activity among more entities.

2 Background

Global types [6] are a behavioral type and process algebra approach to the problem of specifying and verifying multiparty interactions between distributed components. We took inspiration from global types to propose a formalism,
constrained global types, suitable for representing AIPs. Because of space constraints we cannot go into the details of the formalism which can be found in [7]. Since attribute global types are interpreted coinductively, it is possible to specify protocols that are not allowed to terminate like for example the SERVER protocol defined by the equation

\[ \text{SERVER} = (\text{receive} \_\text{request}, 0):(\text{serve} \_\text{request}, 0):\text{SERVER} \]

where \text{SERVER} is a logical variable which is unified with a recursive (or cyclic, or infinite) Prolog term consisting of a \text{receive} \_\text{request} producer interaction type, followed by a \text{serve} \_\text{request} producer interaction type (both requiring 0 consumers), followed by the term itself. This protocol models the (infinite) behavior of a server which is always ready to receive and serve requests; the only valid interaction trace is the infinite sequence \text{receive} \_\text{request} \text{ serve} \_\text{request} \text{ receive} \_\text{request} \text{ serve} \_\text{request} \ldots.

By means of attribute global types we were able to concisely represent protocols which are well known in the concurrent systems and MAS communities like the Alternating Bit Protocol (en.wikipedia.org/wiki/Alternating_bit_protocol) and the FIPA Iterated Contract Net Protocol (fipa.org/specs/fipa00030). FYPA (Find Your Path, Agent! [5]) is not as well known, but is a negotiation protocol for a real MAS used by Ansaldo STS, and is far more complex than the two others. We exploited our formalism to model FYPA as well [8].

Constrained global types can be easily expressed as a set of Prolog equations like the one defining SERVER. Attributes and constraints on their values are represented as additional Prolog facts. In order to allow agents to verify only a sub-protocol of the global interaction protocol, we designed a projection algorithm that takes a constrained global type and a set of agents \text{Ags} as input, and returns a constrained global type which contains only interactions involving agents in \text{Ags} as output [2]. Projection can be described as a function \( \Pi : \mathcal{CT} \times \mathcal{P}(\text{AGS}) \to \mathcal{CT} \) where \( \mathcal{CT} \) is the set of constrained global types and \( \text{AGS} \) is the set of agents. The intuition besides the algorithm is that interactions that do not involve agents in \text{Ags} are removed from the projected constrained global type.

Whatever the protocol to be monitored (global one or projection) and the framework (JADE or Jason), a monitor keeps track of the runtime evolution of the protocol by saving its current state (which is an attribute global type) and checking that each interaction taking place in the MAS is allowed by the current state (namely, can lead to a new state by means of the transition function which defines the semantics of attribute global types, implemented in Prolog). If the interaction is not allowed, an error is reported. The monitor also checks agents responsiveness by means of a time-out whose value can be set by the user: if the current state of the monitor corresponds to the empty protocol (that is, the protocol must terminate), then the monitor reports an error as soon as an interaction is detected (independently of the time-out); if the current state is not final (that is, the protocol is not allowed to terminate), then the monitor reports a warning as soon as the time-out expires, if no interaction is detected (and of
course an error is reported in case an invalid interaction is detected before the time-out).

3 Runtime Verification of JADE MASs

In order to verify the interactions taking place in a JADE MAS, we have designed a monitor meeting the following requirements for non intrusiveness and code reuse [4]: (1) the monitor must be able to supervise the execution of the MAS without interfering with it, (2) the monitor activity must require no changes to the agents’ code, (3) the formalism for representing the AIP must be attribute global types, and (4) the Prolog code already developed for implementing verification of interactions w.r.t. attribute global types and for protocol projection must be re-used as it is.

To meet requirements 1 and 2 we extended the JADE Sniffer agent which is able to sniff all the messages exchanged during the execution of the MAS in a non intrusive way: JADE is developed under the LGPL (Lesser General Public License) and the Java source code is available to the research community, so we were able to modify it to achieve our goals.

To meet requirements 3 and 4 we exploited the JLP library\(^1\) providing a bidirectional interface between Java and SWI Prolog. As all our previous work on attribute global types was implemented in Prolog, allowing our JADE Monitor to consult Prolog programs and to call Prolog predicates was the best way to ensure reusability.

The monitor reads a file containing the Prolog code implementing verification and projection, and a configuration file listing the agents to be monitored, and onto which the protocol projection will be performed. A log file is written as the monitoring goes on. The Prolog file contains definitions for three predicates:

- `initialize(LogFile, SniffedAgents)` which sets LogFile as the file where writing the outcome of the verification, and projects the global protocol which must be defined by the global-type/1 predicate, onto SniffedAgents.
- `remember(ParsedMsg)` which inserts the Prolog representation of the JADE sniffed message into a message list, where messages are ordered by time stamp (if they have a time stamp, which is not mandatory) or in order of arrival.
- `verify(CurrentTime)` which verifies the compliance of each message remembered in the message list and whose time stamp is lower than `CurrentTime` to the global protocol.

These predicates are called in different methods of the monitor code, implementing the wanted behavior.

With this approach no changes are required to the monitored agents and hence existing MASs can be monitored without accessing to their code, but the monitor detects a violation only after it took place and even in case of a protocol violation the MAS execution goes on.

\(^1\) [http://www.swi-prolog.org/packages/jpl/java_api/](http://www.swi-prolog.org/packages/jpl/java_api/)
4 Runtime Verification of Jason MASs

Since Jason agents can integrate Prolog facts and rules for defining their knowledge, the Jason monitor [3] can be generated in a trivial way by integrating directly into its code the Prolog code for protocol specification, monitoring and projection that the JADE monitor must instead read from the Prolog file. The way interactions are sniffed in Jason depends on some assumption on the agents’ code and requires some modifications to it: we assume that agents interact via asynchronous exchange of messages with tell performatives; their original code must be modified in the following way:

1. the .send built-in action for message delivery is replaced by !my_send and
2. two plans must be added for managing the interaction with the monitor.

The first plan is triggered by the !my_send internal goal; my_send has the same signature as the .send internal action, but, instead of sending a message with performative Perf and Content to Receiver, it sends a tell message to the monitor with content msg(Sender, Receiver, Perf, Content). When received, this message will be checked by the monitor against the attribute global type representing the protocol, as briefly explained in Section 2.

The second plan is triggered by the reception of the monitor’s message that allows the agent to actually send Content to Receiver. In reaction to the reception of such a message, the agent sends the corresponding message to the expected agent.

With this approach the code of the monitored agents must be conceived and implemented in a way that makes monitoring possible, but the monitor detects a violation prior than the actual message is sent and can stop the agent violating the protocol by not allowing it to send the “wrong” message.

References

Finding Commonalities in Linked Open Data

Simona Colucci¹, Silvia Giannini¹, Francesco M. Donini², and Eugenio Di Sciascio¹

¹ DEI, Politecnico di Bari, Bari, Italy
² DISUCOM, Università della Tuscia, Viterbo, Italy

Abstract. The availability of a data source as huge, open, accessible and machine-understandable as the Web of Data asks for new and sophisticated inferences to be implemented in order to deeply exploit such a rich informative content. Towards this direction, the paper proposes an approach for inferring clusters in collections of RDF resources on the basis of the features shared by their descriptions. The approach grounds on an algorithm for Common Subsumers computation proposed in a previous work of some of the authors. The clustering service introduced here returns not only different cluster proposals for a given collection, but also a description of the informative content shared by the RDF resources within the clusters, in terms of (generalized) RDF triples.

1 Introduction

The Web of Data [7], born as a research challenge supporting the Semantic Web [1] initiative, is nowadays a fact, as testified by the huge amount of data available in machine-understandable and inter-operable formats, like the Resource Description Framework³ (RDF). The Linked Open Data (LOD)⁴ initiative has in fact been joined by several organizations, that chose to publish their data following the RDF standard notation. Once such an open, continuously enriched and up to date data-source is at hand and accessible, a significantly rich informative content becomes available, opening the way to new challenges to be addressed through reasoning.

The approach proposed in this paper aims at finding commonalities in LOD by exploiting a specifically developed reasoning service, Common Subsumer (CS) of pairs of RDF resources [3], which copes with the difficulties arising from the attempts of reasoning over RDF [4]. As the service name may suggest, CS is defined in analogy with a specific DLs inference: Least Common Subsumer (LCS) [2]. Differently from LCS, CS computation gives up subsumption minimality and searches for knowledge pieces which may be inferred by both RDF input resources. In this paper we show how such information, although not subsumption-minimal, is still useful to deduce descriptions of clusters of RDF resources in

³ http://www.w3.org/RDF/
⁴ http://linkeddata.org/
a knowledge domain. In particular, we chose LOD by Chamber of Deputies of Italian Parliament as case study.

The paper is organized as follows: in the next section we describe the main features of the proposed approach, together with some details on its implementation. In Section 3, some preliminary results are shown, before closing the paper.

2 The Approach

The approach we propose here aims at automatically clustering a target collection of RDF resources according to a fully semantic-based classification. In particular, RDF descriptions are investigated to infer clusters of resources entailing the same sets of RDF triples, in order to provide a description of the informative content shared within each cluster.

The originality of the proposal lays in the choice of adopting deductive services to learn clusters description from examples represented in RDF. In fact, although clustering is a thoroughly investigated task in machine learning literature, approaches solving it usually adopt induction to identify clusters according to some—sometimes semantic-based—distance between elements in the same cluster ([6], [8]).

We propose to solve clustering through a deductive and fully semantic-based approach, which relies on the iteration of the following two steps: i) the CS of two randomly selected RDF resources (in the following referred as seed-resources) is computed; ii) the rest of the target collection is queried in order to find other items entailing the same CS. The sub-collection made up by the two initial resources and those returned by step ii) is one cluster of the collection. Therefore, it is subtracted from the initial target collection, and the two steps are iterated until there are no more resources to be clustered.

An anytime algorithm to compute a CS of pairs of RDF resources has been proposed in [3] by some of the authors. In order to ensure correctness and computability, the CS computation refers to a customized representation of RDF resources which we call r-graph: a portion of the Web of Data we consider relevant for the description of each input resource. In a nutshell, the algorithm for CS computation starts by computing the r-graphs corresponding to the input resources t and s, and returns their CS as a pair (x,T), made up by a blank node x (i.e., the CS of t and s itself) and a set of (generalized) RDF triples T, entailed by the r-graphs of both input resources.

Then, the set T of triples is used to model a SPARQL [5] query, which returns a subset P of the target collection R, such that the RDF description of each item in P entails all triples in T.

In order to exemplify our clustering approach, we adopt the LOD by Chamber of Deputies of Italian Parliament as use case. Such an informative source

---

5 http://dati.camera.it/data/en/
6 In Machine Learning vocabulary, what we do is called unsupervised learning
7 Due to space limits, the CS extraction algorithm [3] is only sketched through an example in the sequel.
is organized in about thirty different interlinked RDF datasets\(^8\) (last update on the 5th November, 2012), accessible through a public SPARQL endpoint\(^9\). Each dataset contains the metadata describing a resource (by properties \texttt{dc:date}, \texttt{dc:description}, \texttt{dc:title}, and \texttt{rdfs:label}), and the statements about possible relations between that resource and other domain-related or web ones. For the current experimental evaluation, we cluster only resources contained in the dataset \texttt{deputato.rdf}, even though their descriptions span multiple datasets.

In Figure 1, the reader may find two example r-graphs describing a pair of resources (\texttt{ocd:d3140\_10} and \texttt{ocd:d270\_10}) in our reference dataset. In the most general case, all triples of datasets of interest having the seed-resource as subject are considered relevant for the description of the resource itself. Here, we adopt a more restrictive strategy driven by specific knowledge of the domain, and discard as not relevant for the description of a resource \(r\) also triples \(\langle\langle r \ p \ o\rangle\rangle\) such that \(p \in \{\texttt{dc:date}, \texttt{dc:title}, \texttt{foaf:depiction}, \texttt{foaf:firstName}, \texttt{foaf:nick}, \texttt{foaf:surname}, \texttt{ocd:endDate}, \texttt{ocd:file}, \texttt{ocd:startDate}, \texttt{ods:modified}, \texttt{rdfs:comment}, \texttt{rdfs:label}, \texttt{terms:isReferencedBy}\}\).

![Fig. 1. Two possible r-graphs for RDF resources](image)

Figure 2 shows a CS of resources \texttt{ocd:d3140\_10} and \texttt{ocd:d270\_10} whose two possible r-graphs are those in Figure 1.

\(^8\) \url{http://data.camera.it/data/en/datasets/}

\(^9\) \url{http://dati.camera.it/sparql}
3 Results

In Table 1, we report a clustering proposal for all deputies of the 10th legislature, where resources ocd:d3140_10 and ocd:d270_10 of the above example\(^{10}\) were forcibly selected as seed pair for the extraction of the first cluster \((P_1)\). By looking at the first row, one can notice how no other resource in the collection shares the features of the CS originated by the first seed-pair, i.e., \(|P_1| = 2\). All subsequent clusters have been extracted with a random selection of seed’s URIs.

\(^{10}\) Notice that, although the resources are the same as in the above example, we here adopt a different criterion for selecting relevant triples.
Table 2. A clustering result adopting randomly selected initial seed pairs.

<table>
<thead>
<tr>
<th>P</th>
<th>Seed’s URIs</th>
<th>ocd:rif_mandatoCamera</th>
<th>ocd:membro</th>
<th>ocd:aderisce</th>
<th>foaf:gender</th>
<th>dc:description</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>([d19990_1, d20060_1])</td>
<td>.x1 .x2 .x3</td>
<td>“male”</td>
<td>“Laurea in giurisprudenza; avvocato.”@it</td>
<td>127</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>([d3140_1, d4290_1])</td>
<td>.x1 .x2 .x3</td>
<td>“female”</td>
<td>“Laurea in lettere; insegnante.”@it</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>([d12560_1, d13120_1])</td>
<td>.x1 .x2 .x3</td>
<td>“male”</td>
<td>.x4</td>
<td>431</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>([d26000_1, d10090_1])</td>
<td>.x1 .x2 .x3</td>
<td>“female”</td>
<td>.x5</td>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td>([d10800_1, d25610_1])</td>
<td>.x1 .x2 .x3</td>
<td>“male”</td>
<td>.x5</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P6</td>
<td>([d12140_1, d98920_1])</td>
<td>.x1 .x2 .x3</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2, instead, shows a clustering result obtained for the target collection \( R \) of 613 resources corresponding to deputies of the first legislature of the Italian Republic. It means that every pair of seed’s URIs \((t, s)\), randomly selected from \( R \), returns a CS described, at least, by the following triples: \(. x \ rdf : type ocd : deputato .\) and \(. x ocd : rif_leg ocd : repubblica_01 .\), where \(. x \) stands for the blank node associated to the CS of \( t \) and \( s \) (for improving readability, these triples are not reported in the table).

The six listed partitions have been obtained in 14.138 s. By looking at the first row as an example, one can notice how the algorithm aggregates all resources in \( R \) that (please follow the columns order): received an open mandate to the Chamber of Deputies; were members of a committee, joined a parliamentary group, are of male gender; worked as a lawyer, after obtaining a law degree.

4 Conclusion

This paper proposed a new and deductive strategy for clustering collections of RDF resources on the basis of the informative content shared by their descriptions expressed in form of generalized RDF triples. The clustering mechanism relies on the computation of the CS [3] of pairs of resources used as seed. In order for such a computation to be finite, we select a relevant portion of the Web of Data to describe the seed resources, according to a characteristic function to be determined on the basis of domain-dependent criteria.

The evaluated execution time of the whole clustering approach, together with the clustering results in terms of provided informative content, seem to support the effort spent in designing and implementing the clustering strategy.

Part of the future work will be devoted to the extension of CS definition and computation to other entailment regimes, to the investigation on (as general as possible) criteria for the selection of relevant triples, aimed at the combined optimization of both description expressiveness and computational complexity, and to a comparative experimental evaluation involving the definition of a metric for clusters quality assessment.
Acknowledgements

We acknowledge support of projects “A Knowledge based Holistic Integrated Research Approach” (KHIRA - PON 02_00563_3446857) and “Enhance Risk Management through Extended Sensors” (ERMES - PON01_03113/F3).

References

Abstract. In the temporal database literature, every fact stored in a database may be equipped with two temporal dimensions: the valid time, which describes the time when the fact is true in the modeled reality, and the transaction time, which describes the time when the fact is current in the database and can be retrieved. Temporal functional dependencies (TFDs) add valid time to classical functional dependencies (FDs) in order to express database integrity constraints over the flow of time. Currently, proposals dealing with TFDs adopt a point-based approach, where tuples hold at specific time points, to express integrity constraints such as “for each month, the salary of an employee depends only on his role”. To the best of our knowledge, there are no proposals dealing with interval-based temporal functional dependencies (ITFDs), where the associated valid time is represented by an interval and there is the need of representing both point-based and interval-based data dependencies. In this paper, we propose ITFDs based on Allen’s interval relations and discuss their expressive power with respect to other TFDs proposed in the literature: ITFDs allow us to express interval-based data dependencies, which cannot be expressed through the existing point-based TFDs. ITFDs allow one to express constraints such as “employees starting to work the same day with the same role get the same salary” or “employees with a given role working on a project cannot start to work with the same role on another project that will end before the first one”. Furthermore, we propose new algorithms based on B-trees to efficiently verify the satisfaction of ITFDs in a temporal database. These algorithms guarantee that, starting from a relation satisfying a set of ITFDs, the updated relation still satisfies the given ITFDs.

1 An example of interval-based constraints

Most health care institutions collect a large quantity of clinical information about patient and physician actions, such as therapies and surgeries, as well as about health care processes, such as admissions, discharges, and exam requests. All these pieces of information are temporal in nature and the associated temporal dimension needs to be carefully considered in order to be able to properly represent clinical data and to reason about them [2]. In this section, we briefly
introduce a real-world example taken from clinical medicine, namely that of patient therapies.

Suppose we have patients who undergo several different therapies: each therapy can be supervised by a physician, and consists of the administration of some drug to the patient. Information about patients and therapies is stored in a relation according to the schema $\text{PatTherapies}(\text{TherType}, \text{PatId}, \text{DrugCode}, \text{Qty}, \text{Phys}, \text{B}, \text{E})$, where $\text{TherType}$ identifies a type of pharmacological therapy, $\text{PatId}$ represents a patient ID, $\text{DrugCode}$ and $\text{Qty}$ the prescribed drug and its quantity, respectively, and $\text{Phys}$ the physician who made the prescription (and is responsible for the therapy). Finally, attributes $\text{B}$ and $\text{E}$ represent the beginning and end time points of the tuple valid interval, respectively: they represent the bounds of the interval specified by the physician for each therapy. An instance of $\text{PatTherapies}$ is provided in Fig. 1.

**Example 1.** A policy of the hospital may be described as follows:

Every patient may receive several therapies at the same time from different physicians, but overlapping therapies for the same patient must be prescribed by the same physician. In other words, if a patient during a therapy needs another therapy which lasts beyond the end of the current therapy, then this therapy must be prescribed by the same physician who prescribed the other one;
It is easy to see that in order to verify these policies through the acquired data, both the start points and the end points of every pair of tuples come into play.

2 Interval-based functional dependencies

Given a totally ordered set $\mathbb{O} = (O, \leq)$, an interval $I$ over $\mathbb{O}$ is a pair $I = [b, e]$ where $b, e \in O$ and $b \leq e$. For any interval $I = [b, e]$ over $\mathbb{O}$ let $\text{points}(I)$ denote the set of points in $O$ between $b$ and $e$: $\text{points}(I) = \{ p \mid p \in O \text{ and } b \leq p \leq e \}$. While the possible distinct relations between two points considering only the linear order are reduced to three (equality, successor, and predecessor), considering the order among the two endpoints of two intervals leads us to have thirteen possible relations. These relations are depicted in Fig. 2 according to the notation proposed by Allen in [1]. It is worth noting that every relation has its dual obtained by switching the position of the two intervals. Consider, for example, two intervals $I_1 = [b_1, e_1]$ and $I_2 = [b_2, e_2]$: we have that $I_1 \bar{D} I_2$ ($I_1 \text{ during } I_2$), if and only if $b_2 < b_1 < e_1 < e_2$. By reverting the arguments, we have that $I_2 \bar{D} I_1$ ($I_2 \text{ contains } I_1$), if and only if $b_2 < b_1 < e_1 < e_2$, which is equivalent to $I_1 \bar{D} I_2$.

More precisely, given two intervals $I = [b, e]$ and $I' = [b', e']$ we say that:

1. $I = I'$ iff $b = b'$ and $e = e'$;
2. $I M I'$ iff $e = b'$;
3. $I S I'$ iff $b = b'$ and $e < e'$;
4. $I F I'$ iff $b > b'$ and $e = e'$;
5. $I O I'$ iff $b < b'$ and $b' < e < e'$;
6. $I D I'$ iff $b' < b$ and $e < e'$;
7. $I B I'$ iff $e < b'$.

In discussing our new functional dependencies based on intervals within a relational framework, we use a simple temporal (relational) data model based on the concept of temporal relation. A temporal relation $r$ is a relation on a temporal relation schema $\mathcal{R}$ defined on attributes $U \cup \{B, E\}$, where $U$ represents a
set of atemporal attributes and $B, E$ are the temporal attributes describing the valid interval of a tuple. We assume that the domain of both attributes $B$ and $E$ is a totally ordered set $O$. Clearly, a tuple $t \in r$ satisfies $t[B] \leq t[E]$. We recall that, assuming the underlying domain for attributes $A_1$ and $A_2$ has a total order, atomic formulas for comparing tuples are either of the form $t[A_1] \theta t'[A_2]$ or of the form $t[A_1] \theta c$, with $\theta \in \{=, \neq, <, \leq, >, \geq\}$, $A_1, A_2$ being attribute names, $c$ a constant value and $t, t'$ tuples of relation $r$. To avoid ambiguities in the terminology employed, in the following we will use (temporal) instance for “(temporal) relation” and will let relation refer to Allen’s interval relations.

### 2.1 Interval-based temporal functional dependencies

Let us now consider the basic definition of an Interval-based Temporal Functional Dependency (ITFD). In the following, we will only deal with interval relations in the set $A = \{S, F, B, M, D, O, =\}$. Indeed, in this case it is not meaningful to distinguish between a relation and its dual, as it will be clear from the following definition of interval-based temporal functional dependency.

**Definition 1.** Let $X$ and $Y$ be sets of atemporal attributes of a temporal relation schema $R = R(U, B, E)$ and $\sim$ an Allen’s interval relation. An instance $r$ of $R$ satisfies an ITFD $X \rightarrow_{\sim} Y$ if for each pair of tuples $t_1$ and $t_2$ such that $[t_1[B], t_1[E]] \sim [t_2[B], t_2[E]]$ and $t_1[X] = t_2[X]$, it is also true that $t_1[Y] = t_2[Y]$.

Basically, ITFDs group tuples whose $B$ and $E$ attribute values satisfy the interval relation $\sim$. In the above definition, all the possible tuples having as valid interval either $[b, e]$ or $[b', e']$, where $[b, e] \sim [b', e']$ are considered together. If there exist two tuples having their valid intervals related through the considered relation $\sim$, respectively, and both tuples agree on (the tuple of) values of atemporal attributes $X$, then the ITFD imposes that both tuples must agree on (the tuple of) values of atemporal attributes $Y$.

As already mentioned, we focus only on (sub) set $A$ of Allen’s interval relations, without considering the dual ones. Indeed, dual relations are not needed for the specification and verification of ITFDs, because ITFDs are based on the equality of the considered (atemporal) values. Thus, each (ordered) pair of tuples satisfying an interval relation will satisfy also the dual one, where tuples will be considered in the pair with the opposite order. In other words, any ITFD with a given interval relation implies also the corresponding ITFD with the dual relation (and vice versa).

Let us now consider the first requirement expressed in Example 1 of Sect. 1: it can be rephrased as “overlapping drug administrations for a given patient must have the same physician”. This constraint can be expressed by the ITFD

\[ \text{PatId} \rightarrow_{\sim} \text{Phys}. \]

A time-oriented graphical account of tuples of relation PatTherapies is provided in the lower part of Fig. 1. As we may notice, the instance satisfies ITFD
PatId $\rightarrow_O$ Phys only for tuples related to the patient with PatId = 1. Dr. Cox added a therapy antipyretics, but the related valid interval is contained in the interval of therapy antiviral prescribed by Dr. Dorian. Tuples related to therapies of patient with PatId = 2 instead do not satisfy ITFD PatId $\rightarrow_O$ Phys, as both intervals of therapies prescribed by Dr. Reid overlap a therapy prescribed by another physician. This kind of property cannot be expressed with point-based TFDs.

Verifying the satisfaction of $X \sim Y$ may be considered in two different but intertwined ways: i) given an instance $r$ of $R$, check whether or not $r$ satisfies $X \sim Y$, ii) given an instance $r$ of $R$ satisfying $X \sim Y$ and a tuple $t$, verify whether $r \cup \{t\}$ still satisfies $X \sim Y$. We call the first problem checking ITFD satisfaction, while the second one is called incremental ITFD verification. It is not difficult to see that these two problems are closely related. In fact, checking ITFD satisfaction reduces to the incremental ITFD verification by adopting the algorithm developed for this problem and, starting from $i = 0$ with instance $r_0 = \emptyset$ with schema $R$, incrementally verifying whether $r_i \cup \{t_i\}$ with $t_i \in r \setminus r_i$ satisfies ITFD $X \sim Y$. If the update of $r_i$ with $t_i$ still verifies $X \sim Y$, then $r_{i+1} = r_i \cup \{t_i\}$, $i = i + 1$ and the algorithm is applied again. If $r$ satisfies $X \sim Y$, after $|r|$ iterations we can determine ITFD satisfaction. Some complexity improvements to this naive approach can be done as shown in Table 1.

<table>
<thead>
<tr>
<th>ITFD</th>
<th>tuple insertion</th>
<th>tuple deletion</th>
<th>ITFD satisfaction checking</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X \rightarrow_S Y$</td>
<td>$\mathcal{O}(</td>
<td>r</td>
<td>)$</td>
</tr>
<tr>
<td>$X \rightarrow_F Y$</td>
<td>$\mathcal{O}(</td>
<td>r</td>
<td>)$</td>
</tr>
<tr>
<td>$X \rightarrow_B Y$</td>
<td>$\mathcal{O}(\log(</td>
<td>r</td>
<td>))$</td>
</tr>
<tr>
<td>$X \rightarrow_M Y$</td>
<td>$\mathcal{O}(\log(</td>
<td>r</td>
<td>))$</td>
</tr>
<tr>
<td>$X \rightarrow_D Y$</td>
<td>$\mathcal{O}(\log(</td>
<td>r</td>
<td>))$</td>
</tr>
<tr>
<td>$X \rightarrow_Q Y$</td>
<td>$\mathcal{O}(\log(</td>
<td>r</td>
<td>))$</td>
</tr>
</tbody>
</table>

Table 1. The complexities for the tuple insertion, deletion, and ITFD satisfaction checking, by our proposed incremental verification algorithm of ITFDs

References

A computational model for MapReduce job flow

Tommaso Di Noia, Marina Mongiello, Eugenio Di Sciascio

Dipartimento di Ingegneria Elettrica e Dell’informazione
Politecnico di Bari
Via E. Orabona, 4 – 70125 BARI, Italy
{firstname.lastname}@poliba.it

Abstract. Massive quantities of data are today processed using parallel computing frameworks that parallelize computations on large distributed clusters consisting of many machines. Such frameworks are adopted in big data analytic tasks as recommender systems, social network analysis, legal investigation that involve iterative computations over large datasets. One of the most used framework is MapReduce, scalable and suitable for data-intensive processing with a parallel computation model characterized by sequential and parallel processing interleaving. Its open-source implementation – Hadoop – is adopted by many cloud infrastructures as Google, Yahoo, Amazon, Facebook.

In this paper we propose a formal approach to model the MapReduce framework using model checking and temporal logics to verify properties of reliability and load balancing of the MapReduce job flow.

1 Introduction and motivation

During the last decades the phenomenon of “Big Data” has steadily increased with the growth of the amounts of data generated in academic, industrial and social applications. Massive quantities of data are processed on large distributed clusters consisting of commodity machines. New technologies and storage mechanisms are required to manage the complexity for storage, analyzing and processing high volumes of data. Some of these technologies provide the use of computing as utility – cloud computing – and define new models of parallel and distributed computations. Parallel computing frameworks enable and manage data allocation in data centers that are the physical layer of cloud computing implementation and provide the hardware the cloud runs on. One of the most known framework is MapReduce. Developed at Google Research [1] it has been adopted by many industrial players due to its properties of scalability and suitability for data-intensive processing. The main feature of MapReduce with respect to other existing parallel computational model is the sequential and parallel computation interleaving. MapReduce computations are performed with the support of data storage Google File System (GFS). MapReduce and GFS are at the basis of an open-source implementation Hadoop\(^1\) adopted by many cloud infrastructures as Google, Yahoo, Amazon, Facebook.

\(^1\) http://hadoop.apache.org
In this paper we propose a formal approach to model the MapReduce framework using model checking and temporal logics to verify some relevant properties as reliability, load balancing, lack of deadlock of the MapReduce job flow. To the best of our knowledge only two works have combined MapReduce and model checking with a different aim from ours: in [2] MapReduce is adopted to compute distributed CTL algorithms and in [6] MapReduce is modeled using CSP formalism. The remaining of the paper is organized as follows. In Section 2 we recall basics of model checking and temporal logics, the formalism used to define and simulate our model. Section 3 provides a brief overview of the main features of MapReduce. Section 4 proposes our formal model of job flow in MapReduce computation while Section 5 proposes an analytical model in the Uppaal model checker language with properties to be checked. Conclusion and future works are drawn in the last section.

2 Model Checking and Temporal Logics

The logical language we use for the model checking task is the Computation Tree Logic (CTL), a propositional, branching, temporal logic [5].

The syntax of the formulae can be defined, using Backus-Naur form, as follows (where $p$ is an atomic proposition):

\[ \phi, \psi ::= p \mid \phi \land \psi \mid \phi \lor \psi \mid \neg \phi \mid \phi \rightarrow \psi \mid \phi \leftrightarrow \psi \mid EF\phi \mid EX\phi \mid EG\phi \mid E(\phi U\psi) \mid AG\phi \mid AF\phi \mid AX\phi \mid A(\phi U\psi). \]

An atomic proposition is the formula $true$ or a ground atom CTL formulae can also contain path quantifiers followed by temporal operators. The path quantifier $E$ specifies some path from the current state, while the path quantifier $A$ specifies all paths from the current state. The temporal operators are $X$, the next-state operator; $U$, the Until operator; $G$, the Globally operator; and $F$ the Future operator. The symbols $X$, $U$, $G$, $F$ cannot occur without being preceded by the quantifiers $E$ and $A$.

The semantics of the language is defined through a Kripke structure as the triple $\langle S, \rightarrow, L \rangle$ where $S$ is a collection of states, $\rightarrow$ is a binary relation on $S \times S$, stating that the system can move from state to state. Associated with each state $s$, the interpretation function $L$ provides the set of atomic propositions $L(s)$ that are true at that particular state [3]. The semantics of boolean connectives is as usual. The semantics for temporal connectives is as follows: $X\phi$ specifies that $\phi$ holds in the next state along the path; $\phi U\psi$ specifies that $\phi$ holds on every state along the path until $\psi$ is true. $G\phi$ specifies that $\phi$ holds on every state along the path. $F\phi$ specifies that there is at least one state along the path in which $\phi$ is true. The semantics of formulae is defined as follows: $EX\phi$: $\phi$ holds in some next state; $EF\phi$: a path exists such that $\phi$ holds in some Future state; $EG\phi$: a path exists such that $\phi$ holds Globally along the path; $E(\phi U\psi)$: a path exists such that $\phi$ Until $\psi$ holds on it; $AX\phi$: $\phi$ holds in every next state; $AF\phi$: for all paths there will be some Future state where $\phi$ holds; $AG\phi$: for all paths the property $\phi$ holds Globally; $A(\phi U\psi)$: All paths satisfy $\phi$ Until $\psi$. The model checking problem is the following: Given a model $M$, an initial state $s$ and a CTL formula $\phi$, check whether $M, s \models \phi$. $M \models \phi$ even the formula must be checked for every state of $M$.
3 MapReduce Overview and proposed model

MapReduce is a software framework for solving large-scale computing problems over large data-sets and data-intensive computing. It has grown to be the programming model for current distributed systems, i.e. cloud computing. It also forms the basis of the data-center software stack [4].

MapReduce framework was developed at Google Research as a parallel programming model with an associated implementation. The framework is highly scalable and location independent. It is used for the generation of data for Google’s production web search service, for sorting, for data-intensive applications, for optimizing parallel jobs performance in data-intensive clusters. The most relevant feature of MapReduce processing is that computation runs on a large cluster of commodity machines [1]; while the main feature with respect to other existing parallel computational models is the sequential and parallel computation interleaving.

The MapReduce model is made up of the Map and Reduce functions, which are borrowed from functional languages such as Lisp [1]. Users’ computations are written as Map and Reduce functions. The Map function processes a key/value pair to generate a set of intermediate key/value pairs. The Reduce function merges all intermediate values associated with the same intermediate key. Intermediate functions of Shuffle and Sorting are useful to split and sort the data chunks to be given in input to the Reduce function. We now define the computational model for MapReduce framework. We model jobs and tasks in MapReduce using the flow description as shown in Figure 1.

Definition 1 (MapReduce Graph (MRG)). A MapReduce Graph (MRG) is a Direct Acyclic Graph 
\[ G = (N, E), \]
where nodes \( N \) in the computation graph are the tasks of computation – \( N = M \cup S \cup SR \cup R \) (\( M = \text{map} \), \( S = \text{shuffle} \), \( SR = \text{sort} \), \( R = \text{reduce} \)) – and edges \( e \in E \) are such that:

1. \( E \subseteq (M \times S) \cup (S \times SR) \cup (SR \times R) \), i.e. “edges connect map with shuffle, shuffle with sort and sort with reduce tasks”;
2. \( e \in M \times S \) breaks input into tokens;
   \( e \in S \times SR \) sorts input tokens by type;
   \( e \in SR \times R \) gives sort tokens to reducer.
3. Reduce sends input data for cluster allocation to the file system

Definition 2 (MapReduce Task). A MapReduce task \( t \) is a token of computation such that \( t \in (M \cup S \cup SR \cup R) \).

Definition 3 (MapReduce Job). A MapReduce Job is the sequence \( t_1 \rightarrow t_2 \rightarrow \cdots \rightarrow t_n \) of MapReduce tasks where

\[ t_j = (M_i, \ldots, M_{i+p}) \rightarrow t_j^2 = S_k \rightarrow t_j^3 = SR_j \rightarrow t_j^4 = R_j \]

with \( M_i \in M, i = 1 \ldots n, S_j \in S, SR_j \in SR, R_j \in R, j = 1 \ldots m \).
In this Section we briefly describe the implemented model in the Uppaal\textsuperscript{2} model checker’s formalism. Uppaal model is described in XML data format of model checker description language and shown in the graphical interface of the tool as a graph model.

The analytical model is made up of three templates: job, mapper and Google File System (GFS). Figure 2 shows only the Uppaal graph model of the job described using the statechart notation. Main states of the job template are Map, Shuffle, Sort and Reduce as defined in the theoretical model in Section 3. Other states manage the flow of the job: Starvation models the state in which the task waits for unavailable resource, Middle_state manages exceptions, receiving_input and receiving_keyval manage data during job flow. Finally medium_level checks input data for the Shuffle state. Mapper template is made up of states: Prevision that checks the behavior of the mappers working for the given job. The Prevision state is followed by Error and out_of_order state in case of wrong behavior of the mapper, or Work state in case of correct behavior of the mapper. Finally the GFS template only manages the job execution.

To test the validity of the approach we simulated the model by instantiating a given number of jobs with relative mappers. We checked the two main properties of MapReduce framework, i.e. load balancing and fault tolerance.

**Load Balancing.** This property checks that the load of the Job $J$ will be distributed to all tasks of the Mapper $M_i$ with given map and reduce tasks. Hence when the job enters the state Map all the previson state of the mapper are verified, this means that the load is balanced between all the mappers.

$$EG(J.Map) \land AG(M_i.Prevision)$$

**Fault Tolerance.** If the $M_i$ map is out of service the job mapper schedules an alternative task to perform the missed function that belongs to remaining

\textsuperscript{2} http://www.uppaal.org/
In the simulation model, \textit{mapper\textsubscript{count}} is the counter of the total number of mappers and \textit{mapper\textsubscript{sched}} is the variable that counts the scheduled mappers. The assigned value, \textit{m} is the number of mappers.

$$EF(M.\text{Out\_of\_order}) \land AG(mapper\_count = m \land mapper\_sched = m)$$

From the simulation results we checked that the model ensures load balancing and fault tolerance.

5 Conclusion and future work

We proposed a formal model of MapReduce framework for data-intensive and computing-intensive environment. The model introduces the definition of MapReduce graph and MapReduce job and task. At this stage of the work we implemented and simulated the model with Uppaal model checker to verify basics properties of its computation as fault tolerance, load balancing and lack of deadlock. We are currently modeling other relevant features as scalability, data locality and extending the model with advanced job management activities such as job folding and job chaining.

We acknowledge support of project “A Knowledge based Holistic Integrated Research Approach” (KHIRA - PON 02 00563 3446857).
References

Hyper-extensionality and one-node elimination on membership graphs

E. Omodeo¹, C. Piazza², A. Policriti², and A. I. Tomescu³

¹ University of Trieste
² University of Udine
³ University of Helsinki

Abstract. A (hereditarily finite) set/hyperset $S$ can be completely depicted by a (finite pointed) graph $G_S$—dubbed its membership graph—in which every node represents an element of the transitive closure of $\{S\}$ and every arc represents a membership relation holding between its source and its target. In a membership graph different nodes must have different sets of successors and, more generally, if the graph is cyclic no bisimilar nodes are admitted. We call such graphs hyper-extensional. Therefore, the elimination of even a single node in a membership graph can cause different nodes to “collapse” (becoming representatives of the same set/hyperset) and the graph to lose hyper-extensionality and its original membership character.

In this note we discuss the following problem: given $S$ is it always possible to find a node $s$ in $G_S$ whose deletion does not cause any collapse?

Keywords: Hereditarily Finite Sets, Hypersets, Bisimulation, Membership Graphs.

Introduction

Two sets are equal if and only if they have the same elements. This principle—the so-called axiom of Extensionality—goes at the very heart of the notion of set, as it states that given $s$ and $s'$, the condition of them having the same elements is sufficient to guarantee that $s$ and $s'$ are the same thing. As a matter of fact, extensionality not only was among the postulates of the first axiomatisation of Set Theory—i.e. the Zermelo-Fraenkel axiomatic set theory ZF—but is also undisputedly present in any subsequent axiomatic presentation of sets.

Being able to establish equality by extensionality only, however, presupposes that membership is acyclic. In fact, admitting the possibility to have a cyclic membership relation, imagine two objects $a$ and $b$ satisfying the following simple set-theoretic equation $x = \{x\}$. In this case, in order to establish wether $a$ is equal to $b$ using extensionality, we must rely on our ability of establishing equality between their elements. That is equality between ... $a$ and $b$. Our argument (as the underlying membership relation) becomes cyclic!

Since the 1980s, the elegant notion of bisimilarity has been extensively used to sensibly extend the notion of set-theoretic equality to the case in which we drop the assumption that the membership relation must be acyclic. The
notion of bisimilarity was introduced (almost at the same time) in many different fields. Aczel, in particular, set up a graph-theoretic view on sets and hypersets, according to which the consequences of dropping acyclicity of $\in$ was rendered cleanly in its anti-foundation axiom (AFA [Acz88], see also [BM96]), stated in terms of bisimilarity.

In this note we study a simple-looking problem that can be stated on the graph-theoretic representation of sets and hypersets. The problem can be, informally, given as follows: given a set $S$ and its membership graph $G_S$—a graph representing the transitive closure of $S$—, does there always exist a node in $G_S$ (i.e. a set in the transitive closure of $S$) whose elimination from $G_S$ will cause no pair of nodes to become bisimilar? In other words, is it always possible to find a way to reduce a graph-theoretic representation of a hyperset by one element, without losing any inequality among the remaining hypersets in the transitive closure of $S$?

Notice that we pose and study the question in the hereditarily finite case. That is, not only we play with pure sets (i.e. sets whose only elements are themselves sets), but also on an entirely finite “chessboard”.

The question has an easy and positive answer for well-founded sets using the notion of rank. However, as the guidance for choosing which node to delete is exactly the feature we cannot count on when dealing with hypersets (that is the notion of rank) the case in which $\in$ can be cyclic becomes quickly more interesting. We present here a few partial and initial observations that, incidentally, suggest that probably the problem should be studied as a graph-theoretic one.

In the concluding remarks we briefly discuss a problem that, among others, brought us to get interested in the above-mentioned question.

1 Basics

Below we schematically recall some basic definitions. See [Jec78] and [Lev79] for detailed definitions. For a given well-founded set $x$ we say that $x$ is hereditarily finite if it is finite and all its elements are hereditarily finite as well. In formulae: $HF(x) \iff_{\text{Def}} ls_{\text{finite}}(x) \land \forall y \in x \ HF(y)$. Moreover, we define the rank and the transitive closure of $x$ as follows$^4$: $\text{rk}(x) =_{\text{Def}} \sup \{\text{rk}(y) + 1 : y \in x\}$, with $\text{rk}(\emptyset) = 0$, and $\text{trCl}(x) =_{\text{Def}} x \cup \bigcup \{\text{trCl}(y) : y \in x\}$.

If, as we do here, we do not assume $\in$ to be necessarily well-founded, a few words are in order to reasonably extend the notion of hereditarily finite set and of transitive closure. In fact, also the notion of rank can be redesigned for the non-well-founded arena$^5$. In order to state the anti-foundation axiom and capture more clearly the notion of hyperset, we need to specify the above-mentioned extension of the principle of extensionality.

$^4$ These definitions can be fully formally given by induction on $\in$, by exploiting any sensible notion of finiteness.

$^5$ Actually, this can be done in many different ways, but the real power of any such extension remains rather mysterious (see [PP04, DPP04]).
To introduce hereditarily finite hypersets we need the definition of *bisimulation relation*. This definition is first given for graphs—as follows—and then is used as an equality criterion to introduce the world of hypersets. This last step is done exploiting the fact that both sets and hypersets are naturally understood as *membership* graphs.

**Definition 1.** A *bisimulation* on \((V, E)\) is a relation \(\flat \subseteq V \times V\) that satisfies

1) to every child \(v_0\) of \(u_0\) there corresponds at least one child \(v_1\) of \(u_1\) such that \(v_0 \flat v_1\) holds, and

2) to every child \(v_1\) of \(u_1\) there corresponds at least one child \(v_0\) of \(u_0\) such that \(v_0 \flat v_1\) holds.

At this point we can define *bisimilarity* to be the relation \(\equiv (V, E)\) (or simply \(\equiv\)) defined between nodes \(u, v \in V\) as: \(u \equiv (V, E) v\) iff \(u \flat v\) holds for some bisimulation \(\flat\) on \((V, E)\). It plainly turns out that \(\equiv (V, E)\) is a bisimulation (actually, the largest of all bisimulations) on \((V, E)\); moreover, it is an equivalence relation over \(V\).

The following definitions (given following [Acz88]) establish the bridge between graphs and sets.

**Definition 2.** A *pointed graph* \(G = (G, v)\) is a graph \(G = (V, E)\) with a distinguished node \(v \in V\) (its point) from which every node in \(V\) is \(E\)-reachable.

**Definition 3.** Given a set \(S\), its *membership graph* \(G_S\) is the pointed graph \((G_S, S)\), where \(G_S = (\text{trCl}(\{S\}), E_S)\) with

\[E_S = \{\langle v, w \rangle : v \in \text{trCl}(\{S\}) \land w \in \text{trCl}(\{S\}) \land w \in v\}\]

With a slight abuse of terminology we will say that graph \(G\) (not pointed) is a membership graph if there exists a node \(s\) in the graph \(G\) such that \((G, s)\) is isomorphic to a membership graph. An acyclic membership graph corresponds to the transitive closure of a well-founded set. Below we give two simple results implying that bisimulation is, in fact, coherent with the extensionality principle.

**Proposition 1.** The membership graph of any hereditarily finite set has the identity relation as its only bisimulation.

Any finite, acyclic, pointed graph having identity as its only bisimulation is isomorphic to the membership graph of a hereditarily finite set.

On the basis of the above proposition, one can identify \(HF\) (i.e. the collection of \(x\)'s such that \(HF(x)\)) with the collection of those finite, acyclic, pointed graphs whose only bisimulation is the identity —which, in turn, is the collection of those finite acyclic pointed graphs in which no two different nodes have the same successor set. We can now proceed to define hypersets simply by dropping the acyclicity requirement and using bisimulation as equality criterion.

**Definition 4.** A *hyperset* is (the isomorphism class of) a pointed graph on which identity is the only bisimulation. Such an entity is said to be HEREDITARILY FINITE when it has finitely many nodes.
Recalling that the subgraph issuing from \( w \) in a graph \( G \) is the subgraph, pointed in \( w \), that consists of all nodes which are reachable from \( w \) in \( G \), we can readily introduce the membership relation between hypersets as follows.

**Definition 5.** Given two hypersets \( h \) and \( h' = (G, v) \), with \( G = (V, E) \) as usual, we say that \( h \in h' \) if \( h \) is (isomorphic to) the pointed subgraph of \( G \) issuing from a node \( w \) with \( \langle v, w \rangle \in E \).

The class of hereditarily finite hypersets includes the class of hereditarily finite sets. From now on we will identify any hypersets \( S \) (possibly well-founded) with its membership graph \( G_S \)—that is, with a representative of its isomorphism class. Moreover, we will say that a graph (not necessarily a membership graph) is hyper-extensional if its only bisimulation is the identity.

## 2 One-element elimination

Consider a hyperset \( S \) and recall that, by definition, \( G_S \) is hyper-extensional. For any given \( s \in \text{trCl}(\{S\}) \), we denote by \( G_S - s \) the graph obtained from \( G_S \) by eliminating \( s \) together with all the arcs incident to \( s \). Notice that it is possible that \( G_S - s \) is not a membership graph (e.g., the case in which \( s = S \)). As we said in the introduction, the question we want to discuss in this note is whether, given a hyperset \( S \), it is always possible to find \( s \in \text{trCl}(\{S\}) \) such that \( G_S - s \) is hyper-extensional. Clearly, if \( G_S \) is acyclic the question has a positive answer, as \( G_S - S \) is undoubtedly hyper-extensional. However, at least in the well-founded case, it is always possible to maintain (hyper-)extensionality even eliminating a node \( s \in \text{trCl}(S) \) in such a way that \( G_S - s \) remains a membership graph.

**Proposition 2.** Given a hereditarily finite set \( S \) there exists an \( s \in \text{trCl}(S) \) such that \( (G_S - s, S) \) is (isomorphic to) a membership graph.

**Proof.** (Sketch) We can determine \( s \) as follows: if there exist two elements of the same rank in the transitive closure of \( S \), let \( r \) be the maximum such rank and take \( s \) to be any element in the transitive closure of \( S \) of rank \( r \). Otherwise take \( s \) as the empty set.

The general case in which \( G_S \) is cyclic is more challenging. First of all, we observe that we can produce a scenario in which the only possible eliminable \( s \) is in fact the point \( S \).

**Example 1.** Consider the hyperset satisfying the following system of set-theoretic equations: \( S = \{T\}, T = \{U, S\}, U = \{T, \emptyset\} \). In the above case the only eliminable element in \( G_S \) is its point \( S \).

The above example marks a difference between the well-founded and the non well-founded case, as it tells us that the generalisation of Proposition 2 to the cyclic case does not hold. However, it leaves the question open as whether, possibly by permitting the elimination of the point, it is always possible to delete a node from \( G_S \) having the remaining graph hyper-extensional.
**Definition 6.** Let $G_{scc}^S$ be the graph having scc’s of $G_S$ as nodes, and an arc between $A$ and $B$ if and only if there exist an arc in $G_S$ having source in $A$ and target in $B$.

**Proposition 3.** For any membership graph $G_S$, the graph $G_{scc}^S$ is acyclic and has at most two sinks.

Even though—as we said—it is not easy to chose a notion of rank for non well-founded sets, let the rank of $A$ of $G_S$ to be the length of the longest simple path in $G_S$ from $A$. We do not know if, given a membership graph $G_S$, a node whose elimination does not disrupt hyper-extensionality always exists. However, if this is the case, one such node is not necessarily of maximal rank and it is not necessarily of maximal rank in the highest strongly connected component. See the following two examples.

**Example 2.** The following is an example of hyperset in which the element of maximum rank (defined as the element source of the longest simple path and indicated in parenthesis) cannot be eliminated without causing a collapse.

![Graph Example 2]

**Example 3.** The following is an example of hyperset in which the element of maximum rank in the highest strongly connected component cannot be eliminated without causing a collapse.

![Graph Example 3]

On the one hand, a reasonable point of view could be that a choice for an eliminable node should be strictly tied with a definition of some notion of rank compatible with cyclic structures. On the other hand, one could argue that on cyclic graphs an eliminable node must be characterised by two different features: a maximal rank—captured by the maximality of the strongly connected component where the node must be chosen—, and a different—unknown—feature, related with the cyclic character of the graph and guiding in the choice within the strongly connected component.
Concluding remarks

We consider that the problem presented here is simple and elegant enough to deserve a (computationally well characterized) answer without any further consideration. However, let us conclude by mentioning a context in which the question tackled here was raised, and for which a (positive) answer would be beneficial.

In [PT13] the problem of generating uniformly and at random a set with a given number of elements in its transitive closure was studied. The proposed solution was based on generating extensional acyclic digraphs with a given number of labeled vertices (since all of the $n!$ labelings of the vertices of an extensional acyclic digraph, or of a hyper-extensional digraph on $n$ vertices, lead to non-isomorphic labelled digraphs). The results in [PT13] are based on a Markov chain Monte Carlo-based algorithm, initially proposed for generating acyclic digraphs [MDBM01, MP04]. The key fact needed in order to show that the Markov chain converges to the uniform distribution were the irreducibility, aperiodicity, and symmetry of the chain. The idea exploited in the construction of the Markov chain was to show that a pair of elementary operations on graphs (implemented as basic transition rules of the Markov chain, akin to the elimination of a node) could be used to transform any graph $G$ into another graph $G'$ within the same family. Even though this problem was later solved in [RT13] by a deterministic algorithm based on a combinatorial decomposition (and a resulting counting recurrence), as mentioned above, we are far from having such a counterpart for hyper-extensional digraphs. However, a positive answer to the question posed in this note would allow one to extend the Markov chain Monte Carlo technique to the realm of hypersets, which would be the first result of its kind.

References


[PP04] Carla Piazza and Alberto Policriti, Ackermann encoding, bisimulations, and OBDDs, TPLP 4 (2004), no. 5-6, 695–718.