Proceedings of the
1st Workshop on Artificial Intelligence and Formal Verification, Logics, Automata and Synthesis (OVERLAY),
Rende, Italy, November 19–20, 2019

Verification with Answer Set Programming, Reasoning about Actions and Change, Constraints and Ontologies

Laura Giordano[1], Alberto Martelli[2], and Daniele Theseider Dupré[1]

[1]DISIT, Università del Piemonte Orientale, Alessandria, Italy
[2]Dipartimento di Informatica, Università di Torino, Italy
laura.giordano@uniupo.it,mrt@di.unito.it,dtd@uniupo.it

Abstract

In this extended abstract, we describe a line of research where logic-based knowledge representation and reasoning is used for both representing process and system models in knowledge-intensive domains and for performing formal verification on such models. In particular, we rely on Answer Set Programming (ASP). It allows, on the one hand, for reasoning about actions and change, for accommodating domain ontologies expressed in a low-complexity description logic as well as constraints on numerical variables (in the Constraint ASP extensions); on the other hand, Bounded Model Checking for linear time temporal logics can be encoded in ASP.

1 Introduction

Formal methods have extensively been applied in Computer Science for the specification and verification of systems, relying on different formal logics, e.g., standard propositional logic and modal temporal logics, with corresponding inference machinery, such as model checkers and SAT solvers, where transition systems are not necessarily explicitly expressed in a logic.

Within Artificial Intelligence, in the area of Knowledge Representation and Reasoning, logics have also been widely used for modeling knowledge about domains and systems and for reasoning about them. Among the many proposals, the following areas are relevant for the work described in this paper:

- **Nonmonotonic reasoning**, where default conclusions can be obtained based on incomplete information, thus departing from the semantics of classical logic. The field led to the development of **Answer Set Programming** (ASP) [6], a powerful framework for declarative problem solving which combines significant modeling capabilities with efficient solving, relying on inference techniques that include some of the ones used in SAT solvers. ASP modeling allows for specifying rules with logical variables, which, before actual *solving*, have to be instantiated, in a *grounding* phase, in principle to all possible terms built with constant and (if any) function symbols occurring in the model. In practice, *grounding* already performs some inference in order to limit the instantiation to a necessarily finite, and possibly small, set of terms.

- **Reasoning about actions and change** (RAC) [7], where a domain is described in terms of *fluents* (propositions whose value can change), possible actions, which have preconditions, direct effects in terms of fluents, and (in several approaches) static and dynamic causal laws which model dependencies between truth values of fluents or changes in such truth values. In most cases, reasoning is formalized as a form of nonmonotonic reasoning, in order to conclude that a fluent maintains its value in the absence of reasons for changing (*inertia*).

- **Description logics** [2]. Terminological knowledge has been identified as a form of knowledge which can be expressed in suitable fragments of first-order logic, *description logics* (DLs), as well as being useful in formalizing definitions of the terms used in several domains. While full first-order logic is undecidable, DLs offer a trade-off between expressiveness and computational complexity of reasoning, some of them enjoying low complexity while still being able to describe wide terminologies (e.g., SNOMED-CT which can be expressed in $\mathcal{EL}$ [1]). As a result, description logics have been chosen as the basis for the Semantic Web, and, in particular, the Web Ontology Language (OWL).

The work summarized in this paper relies on ASP for both representing the dynamics of a domain (with rich domain knowledge about fluent dependencies, also expressed by DL concept inclusions) and for reasoning about it. Such reasoning does not only include the usual forms of reasoning about actions and change (such as inferring which fluents hold after a sequence of actions, given an initial state) but it also allows for the Bounded Model Checking verification of formulae in Dynamic Linear Time Temporal Logic (DLTL) [17], an extension of Linear Time Temporal Logic (LTL) with regular expressions. Formulae in (D)LTL can also be used to model the domain in addition to usual statements in an action language. The domain model and the formulae to be verified are in any case mapped to ASP.

The following section briefly describes our work on Bounded Model Checking (BMC) in ASP about domains represented in an action language, enriched with DLTL constraints. We then sketch its application to Business Process model verification, and the extension of the approach to take into account terminological knowledge about a domain expressed in a low complexity Description Logic.

2 Reasoning about Action and Change and BMC in ASP

As mentioned in the introduction, nonmonotonic reasoning, and ASP in particular, is suitable for reasoning about action and change, and for defining a representation of a domain which is elaboration tolerant [21], i.e., which easily allows for modifications. This includes the formalization of *inertia*, which means that after an event, fluents (i.e. propositions in the state) maintain their value unless there is some reason to conclude they change. Such a reason may arise as a direct effect of the event, but also as a side effect (ramification), given that fluents are not independent. The term *action* is often used rather than "event", but other discrete events (not just actions performed by a human or artificial agent) can actually be modeled.

In our contribution [11] we introduce a temporal action language which includes formulae like the following ones, part of the domain description for typical examples about shooting a turkey with a gun:

$$\Box([shoot]\neg alive \leftarrow loaded) \tag{1}$$
$$\Box(frightened \leftarrow in\_sight, alive) \tag{2}$$
$$\neg loaded\, \mathcal{U}\, in\_sight \tag{3}$$

The first is an *action law* meaning that, in all states ($\Box$), if *loaded* holds (the gun is loaded), then, after *shoot* (i.e., if the action *shoot* is executed), $\neg alive$ holds (the turkey is not alive). The *causal law* (2) states that the turkey being in sight of the hunter causes it to be frightened (in the same state), if it is alive. (D)LTL constraints on the domain are also used, (3) above means that the gun is not loaded until the turkey is in sight. DLTL also allows for the *until* operator to be indexed with regular programs of propositional dynamic logic. For instance, the program

$$(\neg in\_sight?; wait)^*; in\_sight?; load; shoot \tag{1}$$

describes the behavior of the hunter who waits for a turkey until it appears and, when it is in sight, loads the gun and shoots. Actions $in\_sight?$ and $\neg in\_sight?$ are test actions (essentially as in dynamic logic).

For instance, the constraint

$$\langle(\neg in\_sight?; wait)^*; in\_sight?; load; shoot\rangle\top$$

can be included in the domain description, where $\langle\pi\rangle\alpha$ is defined as $\top\mathcal{U}^{\pi}\alpha$, and then the inclusion of a constraint $\langle\pi\rangle\top$ means that all the runs of the domain description which do not start with an execution of the program π will be filtered out. For instance, a scenario in which in the initial state the turkey is not in sight and the hunter loads the gun and shoots is not allowed.

The semantics of the language is defined in terms of a temporally extended notion of Answer Set, and the language is mapped to plain ASP relying on a representation of states as (a finite subset of) natural numbers, and predicates $holds(f, S)$, $occurs(a, S)$ to mean that fluent f is true in state S, and that action a is executed in state S. In this way, a standard Answer Set solver can be used to find answer sets corresponding to temporal answer sets of the domain description.

The evaluation of a (D)LTL formula α in a state S involves a predicate $sat(t\_alpha, S)$ meaning that the formula α represented by the term t_alpha holds in state S. The ASP constraint $\leftarrow$ *not* $sat(t\_alpha, 0)$, number 0 meaning the initial state, guarantees that α is satisfied in all answer sets, given that *not* $sat(t\_alpha, 0)$ is not allowed to be true.

The above framework can be used for usual reasoning tasks in RAC, but also for verification of (D)LTL formulae in a Bounded Model Checking [4] approach, as follows.

In order for checking validity of a temporal formula β, a counterexample for it, i.e., a (temporal) answer set satisfying $\neg\beta$, is searched for. The ASP constraint $\leftarrow$ *not* $sat(t\_alpha, 0)$ is added (as in case of formulae describing the domain), where t_alpha represents α which, in this case, is $\neg\beta$; then, only (temporal) answer sets satisfying $\neg\beta$ will be found.

In [11], the actual BMC technique is obtained as follows, building on the work in [16], where ASP is used for BMC in the verification of LTL formulas for systems modeled as Petri nets. The ASP encoding which is input to the solver includes, in addition to (the encoding of) the action domain model and the ASP constraint for the formula to be verified, ASP rules and constraints to impose that infinite temporal models are searched for, which can be finitely represented as finite sequences with a “k-loop”, given that search can be limited to such models [4]. The length of the finite sequence is used as a bound for the number of states in the ASP encoding and can be increased iteratively in the basic partial decision procedure of BMC; there are cases where completeness can be guaranteed [4].

In an alternative approach [13] to BMC in the same context, i.e., for domains represented in the temporal action language sketched before, and mapped to ASP, we provided a decision procedure for verification based on incremental answer set solving, where the completeness of the method is achieved exploiting a Büchi automaton construction. In this case, a counterexample is an accepting path of the product automaton combining the transition system, specified by the action theory, with the Büchi automaton for the negated property.

The number of ground rules in the ASP encoding is $O((|f|+|\phi|^3)\times k^2)$ for DLTL, and $O((|f|+|\phi|)\times k^2))$ for LTL, where $|f|$ is the number of fluents, $|\phi|$ is the size of the formula to be verified, and k is the step in incremental solving. Given a fixed k, verification, i.e., in this case, ensuring there is no answer set for the encoding (providing a counter example), is a problem in co-NP in the size of the domain description.

3 Reasoning on Business Processes and Constraints on integer variables

Several business process modeling paradigms have been proposed for supporting Business Process Management and automation [26], including the workflow-style languages YAWL [24] and BPMN [22], as well as the declarative framework Declare [25], whose semantics is given in terms of LTL.

We have used DLTL for modeling such processes [5] in the verification approach [11] described in the previous section. The notion of *commitment* from the multi-agent systems literature [19, 23] is used as follows (building on earlier work in [8] for modeling agent interaction and its verification). In a process, an activity (e.g., signing an order) may introduce (i.e., have as effect) a commitment $C(\alpha)$ to make true a formula α (e.g., that a contract has been sent). A causal rule discharges the commitment (i.e., makes $C(\alpha)$

false) if α is achieved. Formal verification could mean verifying $\Box(C(\alpha) \rightarrow \Diamond\alpha)$, but the model could be made more flexible in order to allow for activities (canceling the order, in the example) to discharge the commitment even if it is not fulfilled. In this case, $\Box(C(\alpha) \rightarrow \Diamond\neg C(\alpha))$ should be verified.

The idea is further elaborated in [12] in order to deal with different notions of obligations introduced in [15] including: *achievement obligations* where a condition must occur at least once before something else occurs (in particular, the fact that a deadline has been reached); *maintenance obligations* where a condition must hold in all states until something else occurs.

In the work described in detail in [9], we extend the approach in [11] in order to consider constraints and thus mapping the framework to Constraint Answer Set Programming, where an ASP solver is combined with a constraint solver: constraints in a given constraint language are considered as atoms by the ASP solver, and labeled as true or false during ASP solving; the constraint solver checks for satisfiability of the set of constraints resulting from the labeling, i.e., including C (resp., $\neg C$), if C is labeled *true* (resp., *false*).

As regards the application to Business Processes, in [9] we mainly concentrate on processes whose workflow is modeled with the basic constructs in YAWL and BPMN. Process activities correspond to actions in [11] and in the translation of the workflow to an action model, fluents are used to represent the enabling of actions. The process model can however be enriched with annotations specifying, as effects of actions, additional fluents in a domain knowledge which includes causal laws. XOR-splits in the workflow may be conditioned on such fluents, and fluents may occur in formulae to be verified. In addition, we consider process variables with a $[0..n]$ integer domain and constraints on such variables. Constraints may occur as effects of activities (e.g., in a process dealing with ordering goods, an activity sets the pn "piece number" variable for the order so that $0 \leq pn \leq 100000$), as conditions in XOR splits (e.g., a given branch is taken if $pn > 50000$) and business rules to be verified (e.g., for orders with $pn > 80000$, a solvency check is necessary before confirming the order). The BMC approach in [11] is used (slightly adapted, given that finite process executions which reach the end are considered), and experiments [9] show that, on the one hand, the cost of relying on a constraint solver is acceptable, and, on the other hand, the approach can deal with process models with 200 activities and runs of more than 100 activities, as well as processes with 10^{28} different runs.

4 Adding domain ontologies

In [10], we described how reasoning about action and change can be performed in ASP in case knowledge about the domain includes terminological knowledge expressed in the fragment $\mathcal{EL}^{\perp}$ of the description logic $\mathcal{EL}^{++}$ [1]. In this fragment, concepts can be constructed from class names and *nominals* such as $\{a\}$ (i.e., the concept of "being a") using intersection ($\sqcap$) of concepts and *existential restriction* $\exists r.C$ (the individuals which are in relation r with some member of the concept C).

Background knowledge about the domain is expressed as concept inclusions in $\mathcal{EL}^{\perp}$, such as "the ones who teach some university course are lecturers", $\exists \mathit{Teaches}.\mathit{UniversityCourse} \sqsubseteq \mathit{Lecturer}$.

In reasoning about actions and change, such inclusions can be regarded as *state constraints*, i.e., conditions that must hold in all states. In the field, it is well known that they may be related to causal laws, but they are not, in general, equivalent to them. In particular, if an action, as a direct effect, changes a fluent involved in a state constraint (which must hold before executing the action), it is required that the state constraint holds after the action as well, but there may be different ways to restore its truth, inferring side effects. Our approach extends to non-deterministic actions the approach proposed by Baader et al. [3]. In general, a subset of the causal rules corresponding to a state constraint is included. For example, the causal law

$$\textbf{caused } \mathit{Lecturer}(x) \textbf{ if } \mathit{Teaches}(x,y) \wedge \mathit{UniversityCourse}(y)$$

associated to the concept inclusion exemplified before, and suitably mapped to ASP, would allow to infer, as a side effect, that John is a lecturer if he is appointed the university course $CS101$. If, as a direct effect of some action, John ceases to be a lecturer, we would like to infer that he does no longer teach $CS101$, but presumably we do not consider the case that he still teaches $CS101$, which ceases to be a university course; then we would only like to include the first one of the following two causal rules:

$$\textbf{caused } -\mathit{Teaches}(x,y) \textbf{ if } -\mathit{Lecturer}(x) \wedge \mathit{UniversityCourse}(y)$$
$$\textbf{caused } -\mathit{UniversityCourse}(y) \textbf{ if } \mathit{Teaches}(x,y) \wedge -\mathit{Lecturer}(x)$$

In [10], we defined a semantics for action execution in such a context, based on the answer set semantics. As a result of the introduction of (some of) the causal rules corresponding to concept inclusions, and the limited expressive power of $\mathcal{EL}^{\perp}$ with respect to other Description Logics, reasoning can be performed in ASP only, with no need to exploit a DL reasoner. The encoding is polynomial in the size of the domain description; it follows that the temporal projection problem is in co-NP.

In [14][1], we describe how the work above can be used in the context of modeling Business Processes and reasoning about them, especially for verification.

In the context of Business Process Management, ontological knowledge can be exploited to describe terms used in the conditions on sequence flow and in the formulae to be verified. As [20] points out, a semantic layer is useful to abstract from the way some fact about the case at hand may be actually represented, or computed from stored data, in the process implementation. This is consistent with the idea of sharing terminological knowledge about a domain and reusing it in several applications (consider, e.g., the well-known SNOMED-CT medical terminology [18]).

While [14] presents a preliminary study, demonstrating the feasibility of the approach, including Bounded Model Checking verification, on a minimal set of features of BPMN, with the addition semantic annotations, current work includes more extensive pratical experimentation as well as the extension to other BPMN features, which would benefit from a deeper analysis of BPMN from an ontological perspective.

5 Conclusions

In this extended abstract we have summarized our research on the verification of processes and systems where a uniform approach is used for representing the system and performing formal verification on it. The approach is explicitly oriented to dealing with background knowledge about the domain represented as causal rules, possibly deriving from ontological knowledge. Answer Set semantics is used in all cases, and Answer Set solvers are used for performing reasoning, including Bounded Model Checking. The approach exploits the grounding capabilities of ASP systems in order to allow for representing domain knowledge using logical variables.

References

[1] F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope. In L. Kaelbling and A. Saffiotti, editors, *Proc. IJCAI 2005*, pages 364–369, Edinburgh, Scotland, UK, August 2005.

[2] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2007.

[3] F. Baader, M. Lippmann, and H. Liu. Using causal relationships to deal with the ramification problem in action formalisms based on description logics. In *LPAR*, pages 82–96, 2010.

[4] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in Computers*, 58:118–149, 2003.

[5] D. D'Aprile, L. Giordano, V. Gliozzi, A. Martelli, G. Pozzato, and D. Theseider Dupré. Verifying business process compliance by reasoning about actions. In *CLIMA 2010, LNCS 6245*, pages 99–116, 2010.

[6] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Answer Set Solving in Practice.* Morgan & Claypool Publishers, 2012.

[1]The work is also presented as a discussion paper at the 18th AI*IA Conference.

[7] M. Gelfond and V. Lifschitz. Action languages. *Electron. Trans. Artif. Intell.*, 2:193–210, 1998.

[8] L. Giordano, A. Martelli, and C. Schwind. Specifying and Verifying Interaction Protocols in a Temporal Action Logic. *Journal of Applied Logic (Special issue on Logic Based Agent Verification)*, 5:214–234, 2007.

[9] L. Giordano, A. Martelli, M. Spiotta, and D. Theseider Dupré. Business process verification with constraint temporal answer set programming. *Theory and Practice of Logic Programming*, 13:641–655, 2013.

[10] L. Giordano, A. Martelli, M. Spiotta, and D. Theseider Dupré. ASP for reasoning about actions with an EL-bot knowledge base. In *Proceedings of the 31st Italian Conference on Computational Logic*, pages 214–229, 2016.

[11] L. Giordano, A. Martelli, and D. Theseider Dupré. Reasoning about actions with temporal answer sets. *Theory and Practice of Logic Programming*, 13:201–225, 2013.

[12] L. Giordano, A. Martelli, and D. Theseider Dupré. Temporal deontic action logic for the verification of compliance to norms in ASP. In *Proc. ICAIL 2013*, 2013.

[13] L. Giordano, A. Martelli, and D. Theseider Dupré. Achieving completeness in the verification of action theories by bounded model checking in ASP. *J. Log. Comput.*, 25(6):1307–1330, 2015.

[14] L. Giordano and D. Theseider Dupré. Enriched modeling and reasoning on business processes with ontologies and answer set programming. In *Business Process Management Forum - BPM Forum 2018, Sydney*, pages 71–88, 2018.

[15] G. Governatori. Law, logic and business processes. In *Third International Workshop on Requirements Engineering and Law.* IEEE, 2010.

[16] K. Heljanko and I. Niemelä. Bounded LTL model checking with stable models. *Theory and Practice of Logic Programming*, 3(4-5):519–550, 2003.

[17] J. Henriksen and P. Thiagarajan. Dynamic linear time temporal logic. *Annals of Pure and Applied logic*, 96(1-3):187–207, 1999.

[18] International Health Terminology Standards Development Organization. SNOMED CT. http://www.ihtsdo.org/snomed-ct/.

[19] N. Jennings. Commitments and Conventions: the foundation of coordination in multi-agent systems. *The knowledge engineering review*, 8(3):233–250, 1993.

[20] L. T. Ly, S. Rinderle-Ma, K. Göser, and P. Dadam. On enabling integrated process compliance with semantic constraints in process management systems - requirements, challenges, solutions. *Information Systems Frontiers*, 14(2):195–219, 2012.

[21] J. McCarthy. Elaboration tolerance. In *Commonsense 1998. Revised version available at http://jmc.stanford.edu/articles/elaboration.html*, 1998.

[22] Object Management Group. Business Process Model and Notation. http://www.bpmn.org/.

[23] M. P. Singh. A social semantics for Agent Communication Languages. *Issues in Agent Communication, LNCS(LNAI) 1916*, pages 31–45, 2000.

[24] W. van der Aalst and A. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.

[25] W. M. P. van der Aalst, M. Pesic, and H. Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - R&D*, 23(2):99–113, 2009.

[26] M. Weske. *Business Process Management - Concepts, Languages, Architectures, Third Edition.* Springer, 2019.