Extending Classical Theorem Proving for the Semantic Web

Tanel Tammet
(Tallinn Technical University, tammet@staff.ttu.ee)

Abstract. We investigate the applicability of classical resolution-based theorem proving methods for the Semantic Web. We consider several well-known search strategies, propose a general schema for applying resolution provers and propose a new search strategy "chain resolution" tailored for large ontologies. Chain resolution is an extension of the standard resolution algorithm. The main idea of the extension is to treat binary clauses of the general form $A(x) \vee B(x)$ with a special chain resolution mechanism, which is different from standard resolution used otherwise. Chain resolution significantly reduces the size of the search space for problems containing a large number of simple implications, typically arising from taxonomies. Finally we present a compilation-based schema for practical application of resolution-based methods as inference engines for Semantic Web queries.

1 Introduction

Building efficient inference engines for large practical examples is one of the crucial tasks for the Semantic Web project as a whole (see an early paper [SemanticWeb01] and the project page [SemanticWebURL]). Since the current mainstream languages RDF and OWL are based on classical first order logic (or subsets of first order logic) it is particularly important to investigate specialised methods for the kinds of first order logic (FOL) inferencing tasks likely to arise in practical examples.

It is common knowledge that although RDF is a very simple subset of FOL, and Owl Lite is also severely restricted, already Owl Full is undecidable, hence in a certain sense equivalent to full first order logic (see [RDF], [OWL]). It is possible that as the field progresses, several new languages will appear. In particular, unless the use of the common equality predicate is tightly restricted, equality will lead to undecidable classes. Inevitably some parts of the knowledge on the web will be presented in decidable and some parts in undecidable fragments of FOL.

The nature of undecidability makes it impossible to obtain an "optimal" or even "close to optimal" algorithm. Inevitably a good inference system will need

to employ a variety of specialised algorithms and strategies, depending on the nature of the inference task at hand.

The majority of research in building inference engines for the Semantic Web is carried out in the context of description logics, which have also heavily influenced the development of Owl (see [Handbook03]). The inference systems for description logics are very well optimised for the specific decidable classes of FOL.

However, in the light of a likelihood of encountering undecidable classes of formulas among the information on the Web, we argue that it is important to also investigate methods suited for undecidable classes. Even in case of decidable classes the high complexity of the decision problem diminishes the likelihood that one or two "optimal" algorithms could be found.

Automated theorem proving in full FOL is a classical area of computer science. There is a significant amount of theory and a number of evolving, powerful provers (see [Casc]: an annual prover competition, [TPTP]: the largest currently existing collection of examples).

Similarly to the description logic community we argue that the standard methods of classical theorem proving in FOL are not well suited for Semantic Web tasks. However, the "standard methods" like resolution are not fixed, specific algorithms, but rather frameworks of algorithms which can be heavily modified, specialised with search strategies and extended by additional methods.

The particular importance of the classical resolution-based theorem proving methods stems from the research and experience collected for full, undecidable FOL.

The current paper is targeted at presenting relatively simple, yet important extensions to the standard resolution method, with the goal to make resolution-based methods a practically useful addition to the current alternatives.

Most of the current high-level theorem provers for full first order logic (FOL) are based on the resolution method. Although the basic resolution method as introduced by Robinson is not particularly efficient, it has proved to be a flexible framework for introducing efficient, specialised strategies. For example, relatively simple clause ordering strategies produce efficient decision algorithms for most of the well-known, classical decidable classes of FOL, see [Fermüller93], [Handbook01]. It is easy to devise strategies for top-down and bottom-up search directions, additional mechanisms for answering database-style queries etc.

We propose a simple "black box" system called "chain resolution" extending standard resolution in the context of answering queries in case large ontologies are present in the theory, as is common in the applications of semantic web and terminological reasoning.

An important feature of chain resolution is that it is easy to implement and easy to incorporate into existing theorem provers. The presence of a chain

resolution component should not seriously degrade the performance of a theorem prover for tasks where chain resolution is useless, while significantly improving performance for tasks where it can be employed.

In the last sections of the paper we present a general scheme for rule database compilation, employing both chain resolution and ordering strategies.

The presented scheme along with the chain resolution algorithm is currently being implemented as a layer on top of a high-perfomance resolution prover Gandalf, see [Tammet97], [Tammet98], [Tammet02].

2 Informal introduction to chain resolution

We assume familiarity with standard concepts of resolution, see [Handbook01]. We will use the following notions in our presentation:

Definition 1. *A* signed predicate symbol *is either a predicate symbol or a predicate symbol prefixed by a negation.*

Definition 2. *A* chain clause *is a clause of the form*

$$A(x_1, \ldots, x_n) \vee B(x_1, \ldots, x_n)$$

where $x_1, \ldots, x_n$ are different variables, A and B are signed predicate symbols.

Definition 3.
A clause C' is a propositional variation *of a clause C iff C' is derivable by a sequence of binary resolution steps from C and a set of chain clauses.*

Definition 4.
A search space *of a resolution prover is a set of clauses kept by the prover at some point during proof search.*

Definition 5.
A passive list *of a resolution prover is a subset of a search space containing exactly these clauses which have not been resolved upon yet.*

Throughout the paper we will only consider chain clauses of a simpler form $A(x) \vee B(x)$ where A and B are unary predicates. All the presented rules, algorithms and proofs obviously hold also for the general chain clauses as defined above.

We will bring a small toy example of a taxonomy containing only chain clauses: "a mammal is an animal", "a person is a mammal", "a person thinks", "a man is a person", "a woman is a person". The following clause set encodes this knowledge as implications in FOL: { $\neg mammal(x) \vee animal(x)$, $\neg person(x) \vee mammal(x)$, $\neg person(x) \vee thinks(x)$, $\neg man(x) \vee person(x)$,

$\neg woman(x) \vee person(x)$ }. Observe that the chain resolution method is applicable for general, unrestricted FOL, not just the case where we only have chain clauses present.

We note that the concept of the chain clause can in specific circumstances be extended to cases where the literals $A(x_1, \ldots, x_n) \vee B(x_1, \ldots, x_n)$ contain constant arguments in addition to the distinct variables. We will not consider this extension in the current paper.

The basic idea of chain resolution is simple. Instead of keeping chain clauses in the search space of the prover and handling them similarly to all the other clauses, the information content of the chain clauses is kept in a special data structure which we will call *the chain box* in the paper.

Chain clauses are never kept or used similarly to regular clauses. At each stage of the proof search each signed unary predicate symbol P is associated with a set S of signed unary predicate symbols $P'_1, P'_2, \ldots, P'_m$ so that the set S contains all these and exactly these signed unary predicate symbols P'_i such that an implication $\neg P(x) \vee P'_i(x)$ is derivable using only the chain clauses in the initial clause set plus all chain clauses derived so far during proof search.

The "black" chain box of chain resolution encodes implications of the form $A(x) \Rightarrow B(x)$ where A and B may be positive or negative signed predicates. The chain box is used during ordinary resolution, factorisation and subsumption checks to perform these operations modulo chain clauses as encoded in the box. For example, literals $A(t)$ and $B(t')$ are resolvable upon in case $A(t)$ and $\neg A(t')$ are unifiable and $\neg B$ is a member of the set S associated with the signed predicate symbol $\neg A$ inside the chain box. Due to the way the chain box is constructed, no search is necessary during the lookup of whether $\neg B$ is contained in a corresponding chain: this operation can be performed in constant, logarithmic or linear time, depending on the way the chain box is implemented.

A suggested way to implement the chain box is using a bit matrix of size $4 * n^2$ where n is a number of unary predicate symbols in the problem at hand. Even for a large n, say, 10000, the bit matrix will use up only a small part of a memory of an ordinary computer.

The main effect of chain resolution comes from the possibility to keep only one propositional variation of each clause in the search space. In order to achieve this the subsumption algorithm of the prover has to be modified to subsume modulo information in the chain box, analogously to the way the resolution is modified.

Since the main modification of an "ordinary" resolution prover for implementing chain resolution consists of modifying the equality check of predicate symbols during unification and matching operations of literals, it would be easy to modify most existing provers to take advantage of chain resolution.

3 Motivation for chain resolution

During experimental resolution-based proof searches for query answering in large ontologies it has become clear to us that:

- New chain clauses are produced during proof search.
- Some chain clauses are typically present in the proof, making a proof larger and hence harder to find.
- Chain clauses produce a large number of propositional variations of non-chain clauses, making the search space larger.

We will elaborate upon the last, critical item. Suppose we have a clause C of the form $A_1(t_1) \vee \ldots \vee A_n(t_n) \vee \Gamma$ in the search space. Suppose search space also contains m_i chain clauses of the form $\neg A_i(x) \vee A'_1(x), \ldots \neg A_i(x) \vee A'_{m_i}(x)$ for each i such that $1 \leq i \leq n$. Then the resolution method will derive $m_1 * m_2 * \ldots * m_n$ new clauses (propositional variations) from C using chain clauses alone.

Although the propositional variations of a clause are easy to derive, the fact that they significantly increase the search space will slow down the proof search. Hence we decided to develop a special "black box" implementation module we present in this paper. Chain resolution module allows the prover to keep only one propositional variation of a clause in the search space and allows the prover to never use chain clauses in ordinary resolution steps.

An important aspect of the semantic web context is that a crucial part of the prospective formalisations of knowledge consist of formalisations of ontologies. Taxonomies, ie hierarchies of concepts, typically form a large part of an ontology. Most prospective applications of the semantic web envision formalisations where the ontologies are not simple taxonomies, but rather more general, ordered uncyclic graphs.

A full ontology may also contain knowledge which is not representable by simple implications. Prospective applications and typical queries formalise knowledge which is also not representable by chain clauses. However, ontologies and simple implications are currently seen as a large, if not the largest part of prospective semantic web applications.

Chain resolution, although a very simple mechanism, makes efficient query answering possible also for the resolution-type methods.

4 Details of the chain resolution

We will present the chain resolution for a "general resolution algortihm", without bringing out specific details of different resolution algorithms, which are inessential for chain resolution.

4.1 Moving chain clauses to the chain box

Remember that the chain box contains a set of signed unary predicate symbols, each associated with a set of signed unary predicate symbols. Hence, the chain box is a set of pairs.

Definition 6.
A signed predicate symbol P is called a key *of a chain box in case P has a set of signed predicate symbols associated with it in the chain box. The associated set is called the* chain *of a key P, also denoted as chain(P). The pair of a key P and the chain set of P is called the* row of a chain box.

The chain resolution mechanism consists of two separate parts invoked during the proof search process:

- Moving chain clauses to the chain box. We will later show that moving chain clauses to the chain box may also produce new unit clauses which have to be added to the search space.
- Using the chain box in ordinary resolution, factorisation and subsumption operations.

The general algorithm of moving clauses to the chain box during proof search:

- Chain box initialisation. For each unit predicate symbol P in the initial clause two *chain box rows* are formed: one with they key P, one with the key $\neg P$. For each chain box key the initial chain set of the key will contain the key itself as a single element of the set.
- Before the proof search starts, all the chain clauses present in the initial clause set are moved to the chain box, one by one. Each chain clause moved to the chain box is removed from the initial clause set. In case a move to the chain box produces a set of new unit clauses S, these clauses are added to the initial clause set.
- During the proof search process each time the proof search produces a new chain clause, this clause is moved to the chain box and removed from search space. All the unit clauses produced by the moving operation are added to the passive list part of the search space, analogusly to clauses derived by ordinary resolution steps.

Observe that the chains in the chain box correspond to propositional implications from keys seen as propositional atoms to corresponding chain elements, also seen as propositional atoms. After moving a chain to the chain box we want all the implicit implications derivable from the chain box to be explicitly present as chains. A chain clause $A(x) \vee B(x)$ is seen both as a propositional implication $\neg A \Rightarrow B$ and an equivalent implication $\neg B \Rightarrow A$.

The algorithm we present for adding a chain clause to the chain box is one of the several possible algorithms for adding chain clauses. The main element of the algorithm adding a chain clause to the chain box is the following recursive procedure $addchain(k, p)$ which adds a signed predicate p to the chain of the key k:

- If $p \notin chain(k)$ then do:
  - $chain(k) := chain(k) \cup \{p\}$
  - for all $l \in chain(p)$ do $addchain(k, l)$
  - for each key $r \in chainbox$ where $k \in chain(r)$ do:
    - for all $m \in chain(k)$ do $addchain(r, m)$
- do $addchain(\neg p, \neg k)$

Regardless of the concrete details of adding a predicate to a chain, the algorithm must propagate all the new implicit implications (for example, a case where $\neg A$ is a member of some chain and the chain of B contains more than one element) to explicit implications by adding elements of chains to corresponding chains, until no more changes can be made.

The full algorithm for moving one chain clause $A(x) \vee B(x)$ to the chain box:

- Do $addchain(\neg A, B)$ (observe that this also does $addchain(\neg B, A)$).
- Construct a set S of unit clauses derivable from the chain box. A unit clause $p(x)$ is derivable from the chain box in case a chain c with a key $\neg p$ contains both r and $\neg r$ for some predicate symbol r.
- Add elements of S to the passive list in the search space of the prover.

A concrete implementation of chain resolution should optimize the creation of a set of S of unit clauses in the previous algorithm so that the same unit clauses are not repeatedly produced during several chain clause additions to the chain box.

Observe that chain box may become contradictory. As shown later, for the completeness properties of chain resolution it is not necessary to check whether the chain box is contradictory or not: derivation of unit clauses suffices for completeness.

4.2 Chain resolution, factorisation and subsumption

In this subsection we describe the modifications to the standard resolution, factorisation and subsumption operations. The easiest way to modify these operations is to modify literal unification and literal subsumption algorithms. Hence we will not bring formal rules for chain resolution, factorisation and subsumption. Instead we assume familiarity with standard rules and suggest each implementer to modify the concrete forms of these rules he/she is using.

In chain resolution the presented chain operations replace standard resolution, factorisation and subsumption operations.

The *chain resolution algorithm* for resolution allows to resolve upon two literals $A(t_1, \ldots, t_n)$ and $B(t'_1, \ldots, t'_n)$ iff $A(t_1, \ldots, t_n)$ and $A(t'_1, \ldots, t'_n)$ are unifiable using standard unification and either $B = \neg A$ or $B \in \mathit{chain}(\neg A)$. Observe that if $B \in \mathit{chain}(\neg A)$ then by the construction of the chain box also $A \in \mathit{chain}(\neg B)$. Chain resolution does not introduce any changes to the substitution computed by a successful unification operation.

The *chain factorisation* algorithm requires a more substantial modification of the standard factorisation algorithm. When factorising two literals $A(t_1, \ldots, t_n)$ and $B(t'_1, \ldots, t'_n)$ in a clause the predicate symbol in the resulting factorised literal depends on the implications encoded in the chain box. First, chain factorisation can be successful only if $A(t_1, \ldots, t_n)$ and $A(t'_1, \ldots, t'_n)$ are unifiable with standard unification. The resulting literal of the factorisation before substitution is applied is obtained in the following way:

- if $A = B$ then the resulting literal is $A(t_1, \ldots, t_n)$.
- if $A \in \mathit{chain}(B)$ then the resulting literal is $A(t_1, \ldots, t_n)$.
- if $B \in \mathit{chain}(A)$ then the resulting literal is $B(t_1, \ldots, t_n)$.

If both $A \in \mathit{chain}(B)$ and $B \in \mathit{chain}(A)$ hold, then the choice of the resulting literal between $A(t_1, \ldots, t_n)$ and $B(t_1, \ldots, t_n)$ does not matter and it is also not necessary to derive two different resulting literals. Chain factorisation does not introduce any changes to the substitution computed by a successful unification operation.

By the *chain subsumption* algorithm of literals a literal $A(t_1, \ldots, t_n)$ subsumes a literal $B(t'_1, \ldots, t'_n)$ iff $A(t_1, \ldots, t_n)$ subsumes $B(t'_1, \ldots, t'_n)$ using standard subsumption and either $B = A$ or $B \in \mathit{chain}(A)$. Chain subsumption does not introduce any changes to the substitution computed by a successful subsumption operation. Neither does it introduce any changes to the clause subsumption algorithm, except modifying an underlying literal subsumption algorithm.

Observe that while chain resolution and chain factorisation may give advantages for proof search efficiency, the main efficiency-boosting effect for chain resolution is obtained by replacing a standard subsumption algorithm with chain subsumption.

5 Correctness and completeness of chain resolution

Both the correctness and completeness proofs are relatively simple. Since the paper is application-oriented, we will not explicate the obvious details of the proofs.

Lemma 1. *Chain resolution is correct: if chain resolution derives a contradiction from a clause set S, then S is not satisfiable.*

Proof. The proof uses correctness of standard resolution as its base. We show that any clause derivable from S using chain resolution and factorisation rules is derivable using standard resolution and factorisation rules. The applications of the resolution and factorisation rules of chain resolution in D can be replaced by derivations using corresponding standard rules of resolution along with the chain clauses encoded in the chain box. All the chain clauses encoded in the chain box are either present in S or derivable from S using standard resolution.

Lemma 2. *Chain resolution is complete: if a clause set S is not satisfiable, then chain resolution will derive a contradiction from S.*

Proof. We will first show that chain resolution without a chain subsumption rule is complete. Proof is by induction on the derivation D of contradiction using standard resolution rules. We transform D to a derivation D' of contradiction using chain resolution rules.

- Indeed, every resolution step between a chain clause C in D and a non-chain clause N in D can only result with a modification N' of N where one of the signed predicate symbols in N is replaced. Every further use of N' in D for factorisation or resolution with another non-chain clause can be replaced by either a corresponding chain factorisation or chain resolution step, since C would be in the chain box at the moment of the rule application. Further uses of N' in D for resolution with other chain clauses are treated in the similar manner.
- A resolution step between two chain clauses in D results either with a unit clause or a chain clause. A unit clause U in D obtained from two chain clauses would be present in the search space of D, by the result of moving a chain clause to a chain box. A chain clause C' in D obtained from two chain clauses would be encoded in the chain box inside two chains, by the result of moving a chain clause to a chain box.
- Factorisation steps as well as resolution steps between non-chain clauses in D are unmodified.

Second, we will show that chain subsumption does not destroy completeness either. The proof is based on the completeness of ordinary subsumption. Indeed, if a clause C chain-subsumes a clause C' using a chain rule R encoded in the chain box, then any use of C' in D can be replaced by a use of either C or C along with R with the help of the chain factorisation or chain resolution rule. Let $C = (A(t) \vee \Gamma)$, $C' = (B(t') \vee \Gamma')$, $R = (\neg A(x) \vee B(x))$. Then any application of a standard rule to the literal $B(t')$ in C' can be replaced by an application of the chain rule to the literal $A(t)$ in C along with a chain clause R.

6 Chain resolution in combination with other strategies

In the following we will show completeness of chain resolution in combination with the following important strategies:

- Set of support strategy. We show that a naive combination of chain strategy with the set of a support strategy is incomplete, and present an alternative weak combination, which preserves completeness.
- A family of strategies for ordering literals in a clause, based on ordering the term structures in the literals.

6.1 Set of support strategy

The set of support (sos), see [Handbook01] is a classical, completeness-preserving resolution strategy, employed for focusing search to the query. As such it is very useful in cases where the clause set consists of a large set of facts and rules, known to be consistent, and a relatively small query. A typical semantic web query, similarly to typical datalog and prolog queries, is likely to have such a structure. The sos strategy can be seen as one of the ways to make a resolution search behave similarly to a tableaux search.

The idea of the sos strategy is following. An initial clause set S is split into two disjoint parts:

- R: a set of clauses, assumed to be consistent. Typically R consists of a set of known facts and rules.
- Q: another set of clauses, typically obtained from the query.

Newly derived clauses are added to the initially empty set Q'. *Sos condition*: one of the two clauses C_1 and C_2 in any resolution rule application is required to be either present in Q or Q'. In other words, derivations from R alone are prohibited.

Sos strategy is not compatible with most of the other resolution strategies.

By the *naive combination* of sos strategy with chain resolution we mean the following strategy:

- Resolution is restricted by the sos condition.
- Chain clauses are moved to chain box both from R, Q and Q'.
- It is always allowed to use a chain clause encoded in the chain box, regardless of whether the chain clause originates from R, Q or Q'.

It is easy to see that the naive combination is incomplete. Consider sets R=$\{\neg A(c), \neg B(c)\}$ and Q=$\{A(x) \vee B(x)\}$. The union of R and Q is obviously inconsistent. The sos strategy will derive a contradiction from these sets. However, if we move the chain clause $\{A(x) \vee B(x)\}$ to the chain box before we start

any resolution steps, the set Q will be empty and due to the sos condition we cannot derive any clauses at all.

We will define an alternative *weak combination* of sos strategy and chain resolution as follows:

- Resolution is restricted by the sos condition.
- Chain clauses are moved to the chain box from R, but not from Q or Q'.
- It is always allowed to use a chain clause encoded in the chain box.
- A clause in R is not allowed to chain subsume a clause in Q or Q'.
- A newly derived clause C may be chain subsumed by existing clauses in Q or Q'.

Lemma 3. *Chain resolution in weak combination with the sos strategy preserves completeness: if an empty clause can be derived from the clause sets R and Q using sos strategy, then it can be also derived using sos strategy weakly combined with the chain strategy.*

Proof. Observe that the previously presented proofs of the completeness of chain resolution hold, unmodified, in the situation where resolution is weakly combined with the sos strategy. Indeed, the sos condition is preserved during the transformation of an original derivation D to the derivation D'.

The completeness of the weak combination of sos and chain strategies is crucial for the compilation framework presented later in the paper.

6.2 Ordering strategies

By the *ordering strategies* we mean resolution strategies which:

- Define an ordering $\prec_o$ on literals in a clause.
- Ordering condition: prohibit resolution with a literal B in a clause C iff C contains a literal A such that $B \prec_o A$.

Ordering strategies form a powerful, well-studied family of resolution strategies. In particular, several completeness-preserving ordering strategies of resolution have been shown to always terminate on clause sets from several known decidable subclasses of first order logic, thus giving a resolution-based decision algorithm for these classes. Ordering strategies have been used to find new decidable classes. Some ordering strategies have been shown to decide different classes of description logics, including extensions of well-known description logics, see [HustadtSchmidt00], [Handbook01], [Fermüller93].

In this section we will consider a subfamily of ordering strategies, which we will call *term-based orderings*. An ordering $\prec_t$ is term-based iff it is nonsensitive to variable and predicate names and polarities, ie iff the following conditions hold:

– If $A \prec_t B$ then $A' \prec_t B'$ where A' and B' are obtained from A and B, respectively, by renaming some variables simultaneously in A and B.
– If $A \prec_t B$ then $A' \prec_t B'$ where A' and B' are obtained from A and B, respectively, by replacing predicate names (and/or) polarity in A and B.

Term-based orderings generate term-based ordering strategies for resolution.

By a combination of a term-based ordering strategy with chain resolution we mean a following strategy:

– Both ordinary and chain resolution are restricted by the ordering condition.
– Both the initial and generated chain clauses are moved to the chain box.
– Chain subsumption is allowed.

We can prove the completeness lemma similar to the lemmas proved previously.

Lemma 4. *Chain resolution combined with any completeness-preserving term-based ordering strategy is completeness-preserving: if a clause set S is inconsistent, an empty clause can be derived using the combination of the term-based ordering strategy and the chain strategy.*

Proof. Observe that the previously presented proofs of the completeness of chain resolution hold, unmodified, in the situation where resolution is weakly combined with the sos strategy. Since chain resolution rule applications only replace the predicates present in clauses, and the ordering is assumed to be insensitive to the predicates and their polarities, the ordering condition is preserved during the transformation of an original derivation D to the derivation D'.

In order to use the combined chain and ordering strategies as decision procedures we would have to show that the known termination properties of ordering strategies are preserved.

In this paper we will not tackle the termination properties of the chain strategy combination with ordering strategies. However, intuition indicates that the termination properties are likely preserved for most, if not all important term-based ordering strategies.

7 A resolution framework for the semantic web inference algorithms

We will now present a general compilation-based scheme for the practical usage of resolution-based inference engines for solving queries arising in the semantic web context. We are currently working on an actual implementation of the presented scheme on top of the resolution prover Gandalf.

The scheme we present is obviously only one of the ways to use the resolution-based inference engines in this context. Many open issues already known and continuously arising will have to be solved for the eventual industrial use of such systems.

Let us consider a likely situation where an inference engine will be used in our context. We will have a very large database of known facts and rules. A large part of the rules form a taxonomy, or rather, several taxonomies. A large part, if not all (depending on the task domain) of the other rules fall into a decidable fragment. However, some rules may be present (for example, rules involving equality, in case the use of equality is not tightly restricted) which do not fall into known decidable classes.

The system will regularly ask queries against such a database. The database will change over time and will be different for different task domains. However, it is likely that the number of queries asked over time is much higher than the number of times the rule database is changed during the same period.

Hence it will normally pay out to compile the database before queries are asked against it.

Hence, for the resolution-based inference systems we suggest the following scheme:

- *Compilation phase.*
  - *Analysis.* Analyse the rule base R to determine whether R or a large subset R_d of R falls into a known decidable class. Determine a suitable ordering-based strategy *os* for R or R_d.
  - *Terminating strategy run.* Run resolution with the strategy *os* combined with the chain strategy on R or R_d, for N seconds, where N is a predetermined limit. This will build both a chain box C_b and a set of new clauses R_1. In case inconsistency was found, stop, otherwise continue.
  - *First filtering.* If R was not in a decidable class, then the previous step probably did not terminate. Even if R was decidable, N may have been too small for the previous step to terminate in time. Hence, in case the strategy did not terminate, the set of new clauses R_1 is likely to be very large. It is sensible to employ heuristics to keep only some part R_f of the derived clauses R_1 for the future.
  - *Chain box building.* Regardless of whether the terminating strategy run terminated in time or not, it will be advantageous to increase the size of a chain box by performing another run of resolution on $R \cup R_f$ plus C_b, this time using a different strategy, optimised for fast derivation of chain clauses. After a predetermined time of M seconds the run will be stopped (or it may terminate by itself earlier).
  - *Final filtering and storage.* The set of clauses R_2 obtained after the previous run will be heuristically filtered, again storing a subset R_{f2} of R_2.

Finally, both the chain box constructed so far as well as the new clauses R_{f2} along with the (possibly) simplified initial clause set R will be stored as a *compilation* of the initial clause set R.

– *Query phase*. The main motivation for the compilation phase is a likely situation where one compilation can be used for a number of different queries and the recompilations are not too frequent.
In the query phase the clauses obtained from the query will be added to the compilation and several resolution runs will be conducted with different strategies. The obvious initial choice in case of a large inital R is the set of support resolution weakly combined with the chain strategy. For the set of support strategy the R part of clauses will be formed of the compilation along with the chain box built. The query clauses will form the Q part of clauses for the strategy.
Observe that the compilation probably contains a large chain box, thus the "weakness" of the combination does not prohibit efficient chain box usage.

References

[Casc] http://www.cs.miami.edu/~tptp/CASC/

[Fermüller93] Fermüller, C., Leitsch, A., Tammet, T., Zamov, N. Resolution methods for decision problems. Lecture Notes in Artificial Intelligence 679, Springer Verlag, 1993.

[Handbook01] Robinson, A.J., Voronkov, A., editors. Handbook of Automated Reasoning. MIT Press, 2001.

[Handbook03] Baader, F., Calvanese, V., McGuinness, V., Nardi, D., Patel-Schneider, P., editors. The Description Logic Handbook. Cambridge University Press, 2003.

[HustadtSchmidt00] Hustadt, U., Schmidt, R. A. Issues of decidability for description logics in the framework of resolution. In G. Salzer and R. Caferra, editors, Automated Deduction in Classical and Non-Classical Logics, pp. 192-206. LNAI 1761, Springer.

[OWL] OWL Web Ontology Language. Semantics and Abstract Syntax. W3C Working Draft 31 March 2003. W3C. Patel-Schneider, P.F, Hayes, P., Horrocks, I., eds. http://www.w3.org/TR/owl-semantics/

[RDF] RDF Semantics. W3C Working Draft 23 January 2003. W3C. Hayes, P. ed. http://www.w3.org/TR/rdf-mt/

[SemanticWeb01] Berners-Lee, T., Hendler, J., Lassila, O. The Semantic Web. Scientific American, May 2001.

[SemanticWebURL] Semantic web. http://www.w3.org/2001/sw/. W3C.

[TPTP] http://www.cs.miami.edu/~ tptp/

[Tammet97] Tammet, T. Gandalf. Journal of Automated Reasoning vol 18 No 2 (1997) 199–204.

[Tammet98] Tammet, T. Towards Efficient Subsumption. In CADE-15, pages 427-441, Lecture Notes in Computer Science vol. 1421, Springer Verlag, 1998.

[Tammet02] Tammet, T. Gandalf. http://www.ttu.ee/it/gandalf/, Tallinn Technical University.