Efficient Two-Phase Data Reasoning for Description Logics

Zsolt Zombori

Abstract Description Logics are used more and more frequently for knowledge representation, creating an increasing demand for efficient automated DL reasoning. However, the existing implementations are inefficient in the presence of large amounts of data. We present an algorithm to transform DL axioms to a set of function-free clauses of first-order logic which can be used for efficient, query oriented data reasoning. The described method has been implemented in a module of the DLog reasoner openly available on SourceForge to download.

Introduction

Description Logics (DL) consitute a family of languages designed for conveniently describing domain specific knowledge of various applications. The existing implementations for automated DL reasoning are mostly based on the so called tableau method which works just fine deducing new rules from existing ones, but it is rather slow when it comes to dealing with large amounts of data. In practice, however, the latter situation is becoming more and more typical.

We have developed the DLog system, an efficient DL data reasoner. This program can handle a data quantity that is too much to be loaded into main memory and hence can only be accessed through direct database queries. The reasoning task is broken into two parts: in the first phase only the rules of the knowledge base are considered and the second phase constitutes the data reasoning. The present paper deals with the first phase.

Section 1 gives a summary of description logics and first-order resolution, as well as a resolution based solution for DL theorem proving. Section 2 constitutes the core of this paper: it presents the first phase of DLog, i.e., how to transform the

Zsolt Zombori
Department of Computer Science and Information Theory, Budapest University of Technology and Economics, e-mail: zombori@cs.bme.hu

rules of a DL knowledge base into a function-free set of clauses which forms the basis of the subsequent efficient query driven data reasoning. Section 3 gives a brief overview to the DLog system which is described in detail in [5].

1 Background

This section gives a recollection of some notions necessary to understand the paper and gives references to relevant sources.

1.1 Description Logics

Description Logics (DLs) [3] is family of logic languages designed to be a convenient means of knowledge representation. They can be embedded into FOL, but – contrary to the latter – they are decidable which gives them a great practical applicability. A DL knowledge base consists of two parts: the TBox (terminology box) and the ABox (assertion box). The TBox contains rules that hold in a specific domain. The ABox stores knowledge about individuals. This paper is concerned with a language called SHIQ which is a widespread DL language, thanks to a good compromise between complexity and expressivity.

We are interested in the following reasoning task: for a given SHIQ knowledge base KB and a query expression Q, we would like to decide whether Q is a logical consequence of KB. If Q contains no variables we expect a yes/no answer. If variables appear in the query, we would like to obtain the complete list of constants that, when substituted for the variables, result in assertions that follow from KB.

1.2 Resolution

Resolution [7] is a complete method for proving first order theorems. Its two inference rules are summarised in Figure 1 where σ is the most general unifier of B and C ($\sigma = MGU(B,C)$). *Ordered resolution* [2] refines this technique by imposing an

$$\frac{A \vee B \qquad \neg C \vee D}{A\sigma \vee D\sigma} \qquad\qquad \frac{A \vee B \vee C}{A\sigma \vee C\sigma}$$

Fig. 1 Binary Resolution and Positive Factoring

order in which the literals of a clause can be resolved. This reduces the search space while preserving completeness. It is parametrised with an *admissible ordering* ($\succ$) on literals and a *selection function*. *Basic superposition* [1] is an extension of ordered resolution with explicit rules for handling equality. The rules are summarised

Hyperresolution	$\frac{(C_1 \vee A_1) \ldots (C_n \vee A_n) \quad (D \vee \neg B_1 \vee \cdots \vee \neg B_n)}{(C_1 \vee \cdots \vee C_n \vee D)\sigma}$
Positive factoring	$\frac{A \vee B \vee C}{A\sigma \vee C\sigma}$
Equality factoring	$\frac{C \vee s = t \vee s' = t'}{(C \vee t \neq t' \vee s' = t')\sigma}$
Reflexivity resolution	$\frac{C \vee s \neq t}{C\sigma}$
Superposition	$\frac{(C \vee s = t) \quad (D \vee E)}{(C \vee D \vee E[t]_p)\sigma}$

Fig. 2 Inference rules of Basic Superposition

in Figure 2, where $E|_p$ is a subexpression of E in position p, $E[t]_p$ is the expression obtained by replacing $E|_p$ in E with t, C and D denote clauses, A and B denote literals without equality and E is an arbitrary literal. The necessary conditions for the applicability of each rule are given in the following list:

Hyperresolution: *(i)* σ is the most general unifier such that $A_i\sigma = B_i\sigma$, *(ii)* each $A_i\sigma$ is maximal in $C_i\sigma$, and there is no selected literal in $(C_i \vee A_i)\sigma$, *(iii)* either every $\neg B_i$ is selected, or $n = 1$ and nothing is selected and $\neg B_1\sigma$ is maximal in $D\sigma$.
Positive factoring: *(i)* $\sigma = MGU(A,B)$, *(ii)* $A\sigma$ is maximal in $C\sigma$ and nothing is selected in $A\sigma \vee B\sigma \vee C\sigma$.
Equality factoring: *(i)* $\sigma = MGU(s,s')$, *(ii)* $t\sigma \not\succ s\sigma$, *(iii)* $t'\sigma \not\succ s'\sigma$, *(iv)* $(s = t)\sigma$ is maximal in $(C \vee s' = t')\sigma$ and nothing is selected in $(C \vee s = t \vee s' = t')\sigma$.
Reflexivity resolution: *(i)* $\sigma = MGU(s,t)$, *(ii)* in $(C \vee s \neq t)\sigma$ either $(s \neq t)\sigma$ is selected or nothing is selected and $(s \neq t)\sigma$ is maximal in $C\sigma$.
Superposition: *(i)* $\sigma = MGU(s, E|_p)$, *(ii)* $t\sigma \not\succ s\sigma$, *(iii)* if $E = {}'w = v'$ and $E|_p$ is in w then $v\sigma \not\succ w\sigma$ and $(s\sigma = t\sigma) \not\succ (w\sigma = v\sigma)$, *(iv)* $(s = t)\sigma$ is maximal in $C\sigma$ and nothing is selected in $(C \vee s = t)\sigma$, *(v)* in $(D \vee E)\sigma$ either $E\sigma$ is selected or nothing is selected and $E\sigma$ is maximal, *(vi)* $E|_p$ is not a variable position.

An important feature of basic superposition is that it remains complete even if we disallow superposition into variables or terms substituted for variables. Such positions are referred to as *variable positions* or *marked positions* and are surrounded with '[ ]'.

1.3 Resolution Based Reasoning for DL

In [6] a resolution based theorem proving algorithm for the SHIQ DL language is presented. The knowledge base, together with the query expression is transformed into a set of FOL clauses with a characteristic structure, called ALCHIQ clauses and are summarised in Figure 3, where:

- **P**(t) is a possibly empty disjunction $(\neg)P_1(t) \vee \cdots \vee (\neg)P_n(t)$ of unary literals;
- **P(f**(x)): is a possibly empty disjunction $\mathbf{P_1}(f_1(x)) \vee \cdots \vee \mathbf{P_n}(f_n(x))$;
- term t is not marked, [t] is marked and <t> may or may not be marked;
- $\# \in \{=, \neq\}$;

Fig. 3 ALCHIQ clauses

$$\neg R(x,y) \vee S(y,x) \tag{1}$$

$$\neg R(x,y) \vee S(x,y) \tag{2}$$

$$\mathbf{P}(x) \vee R(x, < f(x) >) \tag{3}$$

$$\mathbf{P}(x) \vee R([f(x)], x) \tag{4}$$

$$\mathbf{P_1}(x) \vee \mathbf{P_2}(< \mathbf{f}(x) >) \vee \bigvee (< f_i(x) > \# < f_j(x) >) \tag{5}$$

$$\mathbf{P_1}(x) \vee \mathbf{P_2}([g(x)]) \vee \mathbf{P_3}(< \mathbf{f}([g(x)]) >) \bigvee (< t_i > \# < t_j >) \tag{6}$$

where t_i and t_j are of the form $f([g(x)])$ or of the form x

$$\mathbf{P_1}(x) \vee \bigvee_{i=1}^{n} (\neg R(x, y_i)) \vee \bigvee_{i=1}^{n} \mathbf{P_2}(y_i) \vee \bigvee_{i,j=1}^{n \times n} (y_i = y_j) \tag{7}$$

$$\mathbf{R}(< a >, < b >) \vee \mathbf{P}(< t >) \vee \bigvee (< t_i > \# < t_j >) \tag{8}$$

where t, t_i and t_j are either a constant or a term $f_i([a])$

The reasoning task is reduced to deciding whether the obtained FOL clauses are satisfiable. This is answered using basic superposition extended with a method called *decomposition*. [6] shows that the set of ALCHIQ clauses is bounded and that any inference with premises taken from a subset N of ALCHIQ results in either *(i)* an ALCHIQ clause or *(ii)* a clause redundant in N[1] or *(iii)* a clause that can be decomposed to, i.e., substituted with two ALCHIQ clauses without affecting satisfiability. These results guarantee that the saturation of an ALCHIQ set terminates.

1.4 Separating TBox and ABox Reasoning

The drawback of the above resolution algorithm is that it can be painfully slow. Resolution with saturation is a bottom-up strategy and computes all logical consequences of the clause set, many of which are irrelevant to the current question. It would be nice to use some query oriented, top-down mechanism, however, such mechanisms are available only for more restrictive FOL languages, such as Horn Clauses. One can get around this problem by breaking the reasoning into two tasks:

[1] A redundant clause is a special case of other clauses in N and can be removed.

first perform a resolution based preprocessing to deduce whatever could not be deduced otherwise and then use a fast top-down reasoner.

Note that complex reasoning is required because of the rules (TBox) and that in a typical real life situation there is a small TBox and a large ABox. Furthermore, the rules in the TBox are likely to remain the same over time while the ABox data can change continuously. Hence we would like to bring forward all inferences involving the TBox only, perform them separately and then let the fast reasoner (whatever that will be) do the data related steps when a query arrives.

In the framework of basic superposition, when more than one inference steps are applicable, we are free to choose an order of execution, providing a means to achieve the desired separation. Elements from the ABox appear only in clauses of type (8). [6] gives two important results about the role of ABox axioms in the saturation process:

Theorem 1. *An inference from ALCHIQ clauses results in a conclusion of type* (8) *if and only if there is a premise of type* (8)*.*

Theorem 2. *A clause of type* (8) *cannot participate in an inference with a clause of type* (4) *or* (6)*.*

In light of Theorem 1, we can move forward ABox independent reasoning by first performing all inference steps involving only clauses of type (1) – (7). [6] calls this phase the saturation of the TBox. Afterwards, Theorem 2 allows us to eliminate clauses of type (4) and (6). This elimination is crucial because in the remaining clauses there can be no function symbol embedded into another. The importance of this result comes out in the second phase of the reasoning, because the available top down mechanisms are rather sensitive to the presence of function symbols.

By the end of the first phase DL reasoning has been reduced to deciding the satisfiability of FOL clauses of type (1) - (3), (5), (7) and (8), where every further inference involves at least one premise of type (8). For the second phase, [6] uses a datalog engine which requires function-free clauses. Therefore (unary) functional relations are transformed to new binary predicates and new constant names are added: for each constant a and each function f the new constant a_f is introduced to represent $f(a)$. Note that this transformation requires processing the whole ABox.

2 Towards Pure Two-Phase Reasoning

In this section we introduce modifications to the saturation of ALCHIQ clauses. We do this to be able to perform more inferences before accessing the ABox. This is not just a mere regrouping of tasks, we will see that the algorithm produces a crucially simpler input for the second phase with a huge impact on its performance efficiency and on the available data reasoning algorithms. The improvement is achieved by eliminating function symbols from the clauses derived from the TBox.

The initial SHIQ DL knowledge base was function-free. Then, after translating TBox axioms to FOL we eliminatee existential quantifiers using Skolemisation

which introduced new function symbols. The ABox remained function-free, hence everything that is to know about the functions is contained in the TBox. This means we should be able to perform all function-related reasoning before accessing the ABox.

2.1 The Modified Calculus

We modify basic superposition presented in 1.2 by altering the necessary conditions to apply each rule. The new conditions are given below, with the newly added conditions underlined:

HyperresolutionTBox: *(i)* σ is the most general unifier such that $A_i\sigma = B_i\sigma$, *(ii)* each $A_i\sigma$ is maximal in $C_i\sigma$, and either there is no selected literal in $(C_i \vee A_i)\sigma$ or A_i contains a function symbol, *(iii)* either every $\neg B_i$ is selected, or $n = 1$ and $\neg B_1\sigma$ is maximal in $D\sigma$, *(iv)* none of the premises contain constants.
HyperresolutionABox: *(i)* σ is the most general unifier such that $A_i\sigma = B_i\sigma$, *(ii)* each $A_i\sigma$ is maximal in $C_i\sigma$, and there is no selected literal in $(C_i \vee A_i)\sigma$, *(iii)* either every $\neg B_i$ is selected, or $n = 1$ and nothing is selected and $\neg B_1\sigma$ is maximal in $D\sigma$, *(iv)* each A_i is ground, *(v)* $D\sigma$ is function-free.
Positive factoring: *(i)* $\sigma = MGU(A,B)$, *(ii)* $A\sigma$ is maximal in $C\sigma$ and either nothing is selected in $A\sigma \vee B\sigma \vee C\sigma$ or A contains a function symbol.
Equality factoring: *(i)* $\sigma = MGU(s,s')$, *(ii)* $t\sigma \not\succ s\sigma$, *(iii)* $t'\sigma \not\succ s'\sigma$, *(iv)* $(s = t)\sigma$ is maximal in $(C \vee s' = t')\sigma$ and either nothing is selected in $C\sigma$ or $s = t \vee s' = t'$ contains a function symbol.
Reflexivity resolution: *(i)* $\sigma = MGU(s,t)$, *(ii)* in $(C \vee s \neq t)\sigma$ either $(s \neq t)\sigma$ is selected or $s \neq t$ contains a function symbol or nothing is selected and $(s \neq t)\sigma$ is maximal in $C\sigma$.
Superposition: *(i)* $\sigma = MGU(s, E|_p)$, *(ii)* $t\sigma \not\succ s\sigma$, *(iii)* if $E = {}'w = v'$ and $E|_p = w|_{p'}$ then $v\sigma \not\succ w\sigma$ and $(s\sigma = t\sigma) \not\succ (w\sigma = v\sigma)$, *(iv)* $(s = t)\sigma$ is maximal in $C\sigma$ and either nothing is selected in $(C \vee s = t)\sigma$ or $s = t$ contains a function symbol, *(v)* in $(D \vee E)\sigma$ either $E\sigma$ is selected or nothing is selected and $E\sigma$ is maximal, *(vi)* $E|_p$ is not a variable position.

Note that hyperresolution is broken into two rules (HyperrresolutionTBox and HyperresolutionABox) which differ only in the necessary conditions. In the following by *original calculus* we refer to the basic superposition presented in Section 1.2 and by *modified calculus* we mean the rules of basic superposition with the restrictions listed above. We will prove that the new calculus can be used to solve the reasoning task.

Proposition 1. *The modified calculus remains correct and complete.*

Proof. The inference rules of basic superposition are all valid even if we do not impose any restrictions on their applicability. Since in the new calculus only the conditions are altered, it remains correct.

The modifications that weaken the requirements to apply a rule only extend the deducible set of clauses, so they do not affect completeness.

In case of hyperresolution, let us first consider only the new condition *(iv)* and disregard condition *(v)* on HyperresolutionABox. The original hyperresolution step has a main premise of type (7) and of the side premises some are of type (3) – (4) and some of type (8). This can be broken into two by first resolving the main premise with all side premises of type (3) and (4) (HyperresolutionTBox) and then resolving the rest of selected literals with side premises of type (8) (HyperresolutionABox). A hyperresolution step in the original calculus can be replaced by two steps in the modified one, so completeness is preserved.

We now turn to condition *(v)* on HyperresolutionABox. Let us consider a refutation in the original calculus that uses a hyperresolution step. If all side premises are of type (3) and (4) then it can be substituted with a HyperresolutionTBox step. Similarly, if all side premises are of type (8), then we can change it to HyperresolutionABox, as clauses of type (7) are function-free, satisfying condition *(v)*. The only other option is that there are both some premises of type (3) and of type (8)[2]. The result of such step is a clause of the following type:

$$\mathbf{P_1}(x) \vee \bigvee \mathbf{P_2}(a_i) \vee \bigvee \mathbf{P_2}([f_i(x)]) \vee$$
$$\vee \bigvee (a_i = a_j) \vee \bigvee ([f_i(x)] = [f_j(x)]) \vee \bigvee ([f_i(x)] = a_j)$$

At some point each function symbol is eliminated from the clause (by the time we reach the empty clause everything gets eliminated). In the modified calculus we will be able to build an equivalent refutation by altering the order of the inference steps: we first apply HyperresolutionTBox which introduces all the function symbols, but none of the constants, then we bring forward the inference steps that eliminate function symbols and finally we apply HyperresolutionABox. The intermediary steps between HyperresolutionTBox and HyperresolutionABox are made possible by the weakening of the corresponding necessary conditions. Notice, that by the time HyperresolutionABox is applied, functions are eliminated so condition *(v)* is satisfied.

We conclude that for any proof tree in the original calculus we can construct a proof tree in the modified calculus, so the latter is complete. □

Proposition 2. *Saturation of a set of ALCHIQ clauses with the modified calculus terminates.*

Proof. (sketch) We build on the results in [6], that clauses of type (8) are initially of the form $C(a), R(a,b), \neg S(a,b), a = b$ or $a \neq b$, i.e., they do not contain any function symbols. We will also use the fact that in the original calculus any inference with premises taken from a subset N of ALCHIQ results in either *(i)* an ALCHIQ clause or *(ii)* a clause redundant in N or *(iii)* a clause that can be substituted with two ALCHIQ clauses via decomposition.

All modifications (apart from breaking hyperresolution into two) affect clauses having both function symbols and selected literals, in that we can resolve with the

[2] It is shown in [6] that clauses of type (8) and (4) participating in an inference result in a redundant clause so we need not consider this case.

literal containing the function symbol before eliminating all selected literals. Such a clause can only arise as a descendant of a HyperresolutionTBox step. After applying HyperresolutionTBox, we obtain a clause of the following form:

$$\mathbf{P_1}(x) \vee \bigvee(\neg R(x,y_i)) \vee \bigvee \mathbf{P_2}(y_i) \vee \bigvee \mathbf{P_2}([f_i(x)]) \vee \\ \vee \bigvee(y_i = y_j) \vee \bigvee([f_i(x)] = [f_j(x)]) \vee \bigvee([f_i(x)] = y_j) \tag{9}$$

In the following, it will be comfortable for us to consider a clause set that is somewhat broader than (9), in which function symbols can appear in inequalities as well. This set is:

$$\mathbf{P_1}(x) \vee \bigvee(\neg R(x,y_i)) \vee \bigvee \mathbf{P_2}(y_i) \vee \bigvee \mathbf{P_2}([f_i(x)]) \vee \\ \vee \bigvee(y_i = y_j) \vee \bigvee(< f_i(x) > \# < f_j(x) >) \vee \bigvee(< f_i(x) > \# \, y_j) \tag{10}$$

where $\# \in \{=, \neq\}$. Of course, every clause of type (9) is of type (10) as well.

Let us see what kind of inferences can involve clauses of type (10). First, it can be an superposition with a clause of type (3) or (5). In the case of (3) the conclusion is decomposed (in terms of [6]) into clauses of type (3) and (10), while in the case of (5) we obtain a clause of type (10). Second, we can resolve clauses of type (10) with clauses of type (10) or (5). The conclusion is of type (10). Finally, we can apply HyperresolutionABox with some side premises of the form $R(a,b_i)$, but notice that only if the literals with function symbols are missing. The result is of type (8). This means that during saturation, we will only produce clauses of type (1) – (8) and (10). It is easy to see that there can only be a limited number of clauses of type (10) over a finite signature[3]. Hence the modified calculus will only generate clauses from a finite set, so the saturation will terminate. □

2.2 Implementing Two-Phase Reasoning

We use the modified calculus to solve the reasoning task in two phases. Our separation differs from that of [6] in that function symbols are eliminated during the first phase, without any recourse to the ABox. The method is summarized in Algorithm 1, where steps (1) - (3) constitute the first phase of the reasoning and step (4) is the second phase, i.e., the data reasoning.

Proposition 3. *A function-free ground clause can only be resolved with function-free clauses. Furthermore, the conclusion is ground and function-free.*

Proof. It follows simply from the fact that a constant a cannot be unified with a term $f(x)$ and from condition *(v)* on HyperresolutionABox. □

We are now ready to state our main claim:

[3] We already know from [6] that the set of clauses of type (1) - (8) is finite.

Algorithm 1 SHIQ reasoning

1. We transform the SHIQ knowledge base to a set of clauses of types (1) - (8), where clauses of type (8) are function-free.
2. We saturate the TBox clauses (types (1) - (7)) with the modified calculus. The obtained clauses are of type (1) - (7) and (10).
3. We eliminate all clauses containing function symbols.
4. We add the ABox clauses (type (8)) and saturate the set.

Proposition 4. *Algorithm 1 is a correct, complete and finite DL theorem prover.*

Proof. We know from Proposition 2 that saturation with the modified calculus terminates. After saturating the TBox, every further inference will have at least one premise of type (8), because the conclusions inferred after this point are of type (8) (Proposition 3). From this follows, (using Proposition 3) that clauses with function symbols will not participate in any further steps, hence they can be removed. In light of this and taking into account that the modified calculus is correct and complete (Proposition 1), so is Algorithm 1. □

2.3 Benefits of Eliminating Functions

The following list gives some advantages of eliminating function symbols before accessing the ABox.

1. It is more **efficient**. Whatever ABox independent reasoning we perform after having accessed the data will have to be repeated for every possible substitution of variables.
2. It is **safer**. A top-down reasoner dealing with function symbols is very prone to fall into infinite loops. Special attention needs to be paid to ensure the reasoner doesn't generate goals with ever increasing number of function symbols.
3. ABox reasoning without functions is **qualitatively easier**. Some algorithms, such as those for datalog reasoning, are not available in the presence of function symbols. We have seen in Section 1.4 that [6] solves this problem by syntactically eliminating functions, but this requires scanning through the whole ABox, which might not be feasible when we have a lot of data.

3 The DLog System

The DLog system is a complete SHIQ DL reasoner which incorporates the results presented in this paper. As an input it takes a SHIQ knowledge base and the TBox is first transformed to a set of function-free clauses based on [6] and Section 2.1. The resulting clauses are next used to build a Prolog program. It is the execution of this

program – run with an adequate query – that performs the data reasoning. The transformation to Prolog uses the PTTP approach, a complete theorem prover technology for FOL [8]. The readers interested in the DLog system should consult [5]. The program is also available at `http://dlog-reasoner.sourceforge.net`.

We compared the performance of DLog with three description logic reasoning engines: RacerPro 1.9.0, Pellet 1.5.0 and the latest version of KAON2. KAON2 implements the methods described in [6] and hence it is in many ways similar to DLog. For a thorough performance evaluation see [5]. Here we only mention that the larger the ABox, the better DLog performed compared to its peers. Also, to our best knowledge, DLog is the only DL reasoner which doesn't need to scan through the whole ABox and load it to main memory, enabling it to reason over really large amounts of data stored in external databases.

Summary

This paper showed how to extend the results in [6] to transform a SHIQ TBox into a set of first-order clauses. The particularity of these clauses is that they have a rather simple structure, namely they are function-free. This opens the way for fast query oriented inference algorithms to perform data reasoning tasks originally formulated in DL. The DLog system illustrates how this can be achieved, though it should be noted that the transformation presented here doesn't use anything specific to the DLog and is available to other reasoning engines as well.

References

1. Bachmair, L., Ganzinger, H.: Strict basic superposition. Lecture Notes in Computer Science **1421**, 160–174 (1998). URL citeseer.ist.psu.edu/bachmair98strict.html
2. Bachmair, L., Ganzinger, H.: Resolution theorem proving. In: Handbook of Automated Reasoning, pp. 19–99 (2001). URL citeseer.ist.psu.edu/bachmair01resolution.html
3. Horrocks, I.: Reasoning with expressive description logics: Theory and practice. In: Proc. of the 18th Int. Conf. on Automated Deduction (CADE 2002), 2392, pp. 1–15. Springer (2002)
4. Horrocks, I., Kutz, O., Sattler, U.: The Even More Irresistible SROIQ. In: P. Doherty, J. Mylopoulos, C.A. Welty (eds.) KR, pp. 57–67. AAAI Press (2006). URL http://dblp.uni-trier.de/db/conf/kr/kr2006.html#HorrocksKS06
5. Lukácsy, G., Szeredi, P.: Efficient description logic reasoning in Prolog: the DLog system. Tech. rep., Budapest University of Technology and Economics (2008). URL http://sintagma.szit.bme.hu/lukacsy/publikaciok/dlog_tplp_submission.pdf. Submitted to Theory and Practice of Logic Programming
6. Motik, B.: Reasoning in Description Logics using Resolution and Deductive Databases. Ph.D. thesis, Univesität Karlsruhe (TH), Karlsruhe, Germany (2006)
7. Robinson, J.A.: A machine-oriented logic based on the resolution principle. J. ACM **12**(1), 23–41 (1965). DOI http://doi.acm.org/10.1145/321250.321253
8. Stickel, M.E.: A Prolog Technology Theorem Prover: A New Exposition and Implementation in Prolog. Theor. Comput. Sci. **104**(1), 109–128 (1992)