# Mona as a DL Reasoner

Eldar Karabaev and Carsten Lutz
Institute for Theoretical Computer Science
TU Dresden, Germany
{karabaev,lutz}@tcs.inf.tu-dresden.de

**Abstract**

We show how the Mona tool for reasoning in the monadic second order theories WS1S and WS2S can be used to obtain decision procedures for description logics. The performance of this approach is evaluated and compared to the dedicated DL reasoners FaCT and RACER.

## 1 Motivation

The weak monadic second order theories of one and two successors, commonly called WS1S and WS2S, are among the most powerful decidable logics known today [10]. This is witnessed by the fact that a large number of description logics, modal logics, and dynamic logics can be translated into them (see, e.g., [3]), and also by their impressive non-elementary complexity: there exists no positive integer $n$ such that satisfiability of WS1S or WS2S formulas can be decided in $n$-ExpTime [9]. Despite their powerful expressivity and immense computational complexity, with the Mona tool there exists an efficient implementation of both WS1S and WS2S [4]. As the Mona manual puts it, "efficient here means that the tool is fast enough to have been used in a variety of non-trivial settings". Indeed, an impressive list of successful applications can be found on the Mona homepage [1].

Since it is common knowledge that many DLs can be translated into WS2S, it is a natural idea to use Mona for DL reasoning. Thus, *the purpose of the current paper is to translate the basic description logic $\mathcal{ALC}$ into formulas digestible by Mona, and to evaluate Mona's performance for DL reasoning.* More precisely, we use a well-known translation of $\mathcal{ALC}$ into WS2S using a Rabin-style encoding of non-binary trees into binary trees, and exhibit a novel reduction of $\mathcal{ALC}$ into WS1S that is inspired by Pratt's "type elimination" technique for deciding the satisfiability of modal logic formulas. We then compare the performance of Mona with that of the dedicated DL reasoners FaCT and RACER [8, 5]. There are at least two reasons why such a comparison is interesting: first, it contrasts the performance of DL reasoners with the performance of more general reasoners, thus being in the line of [12] where the performance of the FaCT system is compared to that of the first order theorem prover Vampire. Second, Mona implements an automata-based decision procedure, while the two DL reasoners are tableau-based and thus a comparison may contribute to the understanding of the advantages and disadvantages of the two approaches.

The outcome of our investigation is as follows: if no TBoxes are involved, then Mona's performance is reasonable, though it cannot reach the performance of FaCT and RACER. In the presence of TBoxes, Mona's performance is extremely poor, rendering the Mona approach to DL reasoning virtually useless. However, we believe that using Mona for DL reasoning without TBoxes can be useful at least for prototyping purposes: WS1S and WS2S are powerful enough to accomodate many expressive description logics such as $\mathcal{SHIQ}$ or DLs involving transitive closure of roles, and implementing a translator is considerably less difficult than implementing an optimized, dedicated reasoner. Developing translations for more complex DLs, however, is outside the scope of this paper.

## 2 Translations to WS1S and WS2S

We assume familarity with the description logic $\mathcal{ALC}$, see, e.g., [2] for details. In TBoxes, we admit concept equations $C \doteq D$ with both $C$ and $D$ possibly complex. These TBoxes are interpreted according to the usual descriptive semantics. Due to space limitations, we cannot give a full description of the monadic weak second-order theories WS1S and WS2S. Intuitively, the syntax of our monadic second-order (MSO) language is obtained from the familiar first-order (FO) language without function symbols and constants (but with equality) by

1. restricting predicates to be unary;
2. adding second-order quantifiers "$\forall$" and "$\exists$" that can be used to quantify over unary predicates, which are in this context called second-order variables;
3. in the case of WS1S, adding an ordering predicate "$<$" and a successor function $s(\cdot)$;
4. in the case of WS2S, adding an ordering predicate "$<$" and two successor functions $s_\ell(\cdot)$ and $s_r(\cdot)$.

As for the semantics, formulas of WS1S are interpreted in the structure of one successor function, i.e. in one-side infinite words. The built-in ordering predicate "$<$" has the obvious interpretation on positions in such $\omega$-words, and the successor function can be used for going to the successive position. In the case of WS2S, formulas are interpreted in the structure of two successor functions, i.e. in infinite binary trees. The ordering predicate "$<$" describes the "offspring" relation in such trees, and the two successor functions $s_\ell(\cdot)$ and $s_r(\cdot)$ can be used for going to the left and right successor, respectively. For both WS1S and WS2S, the "W" stands for "weak" indicating that quantification is on *finite* sets rather than on arbitrary ones as in the closely related theories S1S and S2S.

### To WS1S and the Monadic Theory of Infinite Sets

We first present a translation of $\mathcal{ALC}$ concepts and TBoxes to WS1S, or rather to the MSO theory of infinite sets since we will not use WS1S's built-in ordering predicates and successor functions. The translation is inspired by the Pratt-style "type elimination" procedure for deciding the satisfiability of modal formuas [11]. Intuitively, type elimination is based on the following simple observation: if a domain element in an $\mathcal{ALC}$ interpretation satisfies a set of concepts $\Gamma$ containing an existential value restriction $\exists R.C$, then there must exist another domain element satisfying $C$ and all $D$ with $\forall R.D \in \Gamma$. This observation also constitutes the core of the WS1S reduction.

Let $C$ be an $\mathcal{ALC}$-concept and $\mathcal{T}$ a TBox. We use $\mathsf{sub}(C)$ to denote the set of subconcepts of $C$ and set

$$\mathsf{sub}(C,\mathcal{T}) := \mathsf{sub}(C) \cup \bigcup_{D \doteq E \in \mathcal{T}} \mathsf{sub}(D) \cup \mathsf{sub}(E).$$

Moreover, we use $\mathsf{cnam}(C,\mathcal{T})$ and $\mathsf{rnam}(C,\mathcal{T})$ to denote the sets of concept names and role names occurring in $C$ and $\mathcal{T}$, respectively. To translate $C$ and $\mathcal{T}$ into a corresponding second order formula, we introduce a unary FO-predicate $P_A$ for each $A \in \mathsf{cnam}(C,\mathcal{T})$, and a unary FO-predicate $Q_D$ for each $D \in \mathsf{sub}(C,\mathcal{T})$ of the form $\exists R.E$ or $\forall R.E$. Then, for each concept $D \in \mathsf{sub}(C,\mathcal{T})$ and each first-order variable $x$, we define a formula $D^\sharp(x)$ by replacing

1. concept names $A$ with $P_A(x)$;
2. $\sqcap$ with $\wedge$ and $\sqcup$ with $\vee$;
3. $\exists R.D$ with $Q_{\exists R.D}(x)$ and $\forall R.D$ with $Q_{\forall R.D}(x)$.

Concerning the last item, we only replace existential and universal value restrictions that are not contained inside another value restriction. For example, taking

$$C = A \sqcap \neg\forall R.(\forall S.(A \sqcap \neg B))$$

and the FO variable $x$, we obtain

$$C^\sharp(x) = P_A(x) \wedge \neg Q_{\forall R.(\forall S.(A \wedge \neg B))}(x).$$

Next, for each role name $R \in \mathsf{rnam}(C,\mathcal{T})$ we define a formula

$$\begin{aligned}\vartheta_R \;\; := \;\; \forall x.\exists W.\Big[ & \bigwedge_{\forall R.C \in \mathsf{sub}(X,\mathcal{T})} \Big(Q_{\forall R.C}(x) \rightarrow \big(\forall y.W(y) \rightarrow C^\sharp(y)\big)\Big) \\ \wedge & \bigwedge_{\exists R.C \in \mathsf{sub}(X,\mathcal{T})} \Big(Q_{\exists R.C}(x) \rightarrow \big(\exists y.W(y) \wedge C^\sharp(y)\big)\Big)\Big]\end{aligned}$$

where $x$ is a first order variable and $W$ is a second order variable (the same variables may be used for every $R \in \mathsf{rnam}(C,\mathcal{T})$). Finally, we can define the WS1S formula $\varphi^1_{C,\mathcal{T}}$, which is the translation of $C$ and $\mathcal{T}$:

$$\varphi^1_{C,\mathcal{T}} := \exists x.C^\sharp(x) \wedge \forall x.\big[\bigwedge_{D \doteq E \in \mathcal{T}} (D^\sharp(x) \leftrightarrow E^\sharp(x))\big] \wedge \bigwedge_{R \in \mathsf{rnam}(C,\mathcal{T})} \vartheta_R$$

Some remarks on this translation, which is called $\mathsf{T1}$ in the remainder of this paper, are in order. First, it is linear and may even result in an exponential compression of the input concept and TBox. Second, the formula $\varphi^1_{C,\mathcal{T}}$ does not refer to any successor functions and ordering predicates, and can hence be interpreted both in WS1S and in the MSO theory of infinite sets. And third, it is interesting to note that description logic roles are not explicitly represented on the second-order side. This is so because we have only a single relation available in WS1S: the successor function, which does not seem suitable for representing the in general non-functional role relationships. Thus we resort to a type elimination perspective as initially sketched: the existentially quantified variable $W$ in $\vartheta_R$ comprises those domain elements that are needed to satisfy all the existential value restrictions of $x$.

**Theorem 1.** *A concept $C$ is satisfiable w.r.t. a TBox $\mathcal{T}$ iff $\varphi^1_{C,\mathcal{T}}$ is satisfiable in the MSO-theory of infinite sets iff $\varphi^1_{C,\mathcal{T}}$ is WS1S-satisfiable.*

It is interesting to note that we can even eliminate the second-order quantifier in the previous translation by defining the formula $\vartheta_R$ in a different way. Here, the relation to type elimination is even more visible:

$$\vartheta'_R := \forall x. \bigwedge_{\forall R.C \in \mathsf{sub}(X,\mathcal{T})} \Big[ Q_{\exists R.C}(x) \rightarrow \Big( \exists y. \big( C^\sharp(y) \wedge \bigwedge_{\forall R.D \in \mathsf{sub}(X,\mathcal{T})} (Q_{\forall R.D}(x) \rightarrow D^\sharp(y)) \big) \Big) \Big]$$

With this modification, we could use our translation together with a first-order theorem prover rather than together with Mona—a path that we are not going to explore in the current paper. Note that the modified translation is quadratic rather than linear.

### To WS2S

The translation to WS2S is much more standard than our WS1S translation. Since we have two successor functions available in WS2S, we can now explicitly represent the relational structure on the second-order side: $\mathcal{ALC}$ is known to have the tree-model property, and an old trick of Rabin [10] can be used to transform $\mathcal{ALC}$'s tree models, which are not necessarily binary trees, into the binary tree structure of WS2S.

Let $C$ be an $\mathcal{ALC}$ concept and $\mathcal{T}$ a TBox. For the new translation, we fix an enumeration of the roles in $C$ and $\mathcal{T}$, and use $\#R$ to denote the position of a role $R$ in this enumeration. We introduce a unary FO-predicate $P_A$ for each $A \in \mathsf{cnam}(C, \mathcal{T})$, and another unary FO-predicate $M$. Then, we inductively define two translation functions $\cdot^x$ and $\cdot^y$, where $x$ and $y$ are first-order variables. Here is the $\cdot^x$ translation:

$$\begin{array}{rcll}
A^x & := & A(x) & \\
(\neg C)^x & := & \neg C^x & \\
(C \sqcap D)^x & := & C^x \wedge D^x & \\
(C \sqcup D)^x & := & C^x \vee D^x & \\
(\exists R.C)^x & := & \exists y.(y <_\ell s_r^n(x) \wedge C^y \wedge M(y)) & \text{where } n := \#R \\
(\forall R.C)^x & := & \forall y.((y <_\ell s_r^n(x) \wedge M(y)) \rightarrow C^y) & \text{where } n := \#R
\end{array}$$

In this translation, $s_r(\cdot)$ is the “right” successor function, and $s_r^n(\cdot)$ stands for going to the right successor $n$ times. Moreover, $<_\ell$ is the ordering over left successors only.

Since this is not available in Mona, we simulate $y <_\ell x$ by writing

$$\exists Q.\Big[x \in Q \wedge \forall z.\big(z \in Q \rightarrow (z = y \vee s_\ell(z) \in Q)\big)\Big].$$

Note that this formula only works since, in WS2S, quantification is over *finite* sets. The $\cdot^y$ translation is defined symmetrically to $\cdot^x$, details are omitted. Given a concept $C$ and a TBox $\mathcal{T}$, we now define their translation to WS2S as follows:

$$\varphi^2_{C,\mathcal{T}} := \exists x.(M(x) \wedge C^x) \wedge \bigwedge_{D \doteq E \in \mathcal{T}} \forall x.(D^x \leftrightarrow E^x).$$

The intuition behind this translation, which is called T2, is as follows: there is a one-to-one correspondence between domain elements of $\mathcal{ALC}$ interpretations and nodes in the WS2S tree structure that are in the extension of the predicate $M$. Let $x$ be a node in the WS2S tree that is in $M$. To find the $R$-successors of $x$ for a role name $R \in \mathsf{rnam}(C, \mathcal{T})$ with $\sharp R = n$, we start at $x$ and follow the right successor function exactly $n$ times to the node $y$. Then the $R$-successors of $x$ are $y$, its left successor, its left successor's left successor, and so forth. This also explains the use of the $M$ predicate: since we do not want to have infinitely many successors for each domain element and each role name, we mark the "existing ones" with $M$.

**Theorem 2.** *A concept $C$ is satisfiable w.r.t. a TBox $\mathcal{T}$ iff $\varphi^2_{C,\mathcal{T}}$ is WS2S-satisfiable.*

There are some interesting variations of this translation. For example, we can modify the translation $\cdot^x$ as follows (and $\cdot^y$ analogously):

$$\begin{array}{lcll} (\exists R.C)^x & := & M(s_r^n(x)) \wedge \exists y.(y <_\ell s_\ell(s_r^n(x)) \wedge C^y) & \text{where } n := \#R \\ (\forall R.C)^x & := & M(s_r^n(x)) \rightarrow \forall y.(y <_\ell s_\ell(s_r^n(x)) \rightarrow C^y) & \text{where } n := \#R \end{array}$$

Here, the intuition of the $M$ predicate is a different one since $M$ is only used to state whether a node $x$ has successors for a role name $R$ at all: this is the case if and only if $s_r^n(x)$ is in $M$, with $n = \sharp R$. This second variant of the S2S translation is denoted with T2b.

## 3 Evaluation

To evaluate the performance of Mona when used for deciding the satisfiability of $\mathcal{ALC}$ concepts without reference to TBoxes, we employed two different classes of concepts. Firstly, we tested our approach on the Tableaux'98 (henceforth T98) benchmark suite that is frequently used to evaluate DL reasoners, see, e.g., [8, 12]. The T98 suite consists of 18 sequences of concepts, each sequence comprised of 21 concepts with increasing difficulty. Since the T98 concepts are artificial in the sense that they have been designed with the only purpose of making reasoning difficult [6], we secondly used concepts that were extracted from the real world knowledge bases Galen and Platt—see [7] for more information on both of them.

All tests were perfomed five times: once for each of the three translations T1, T2, and T2b, and once using the well-known DL reasoners FaCT and RACER [8, 5]. The

| | T1 | T2 | T2b | RACER | FaCT |
|---|---|---|---|---|---|
| k_branch_n | 0 | 2 | 1 | 14 | 6 |
| k_branch_p | 0 | 2 | 1 | 20 | 8 |
| k_d4_n | 0 | 2 | 4 | 21 | 21 |
| k_d4_p | 0 | 3 | 6 | 21 | 21 |
| k_dum_n | 0 | 2 | 3 | 21 | 21 |
| k_dum_p | 0 | 6 | 7 | 21 | 21 |
| k_grz_n | 0 | 3 | 5 | 21 | 21 |
| k_grz_p | 0 | 4 | 5 | 21 | 21 |
| k_lin_n | 1 | 1 | 21 | 21 | 21 |
| k_lin_p | 1 | 7 | 10 | 21 | 21 |
| k_path_n | 0 | 3 | 16 | 21 | 8 |
| k_path_p | 0 | 4 | 17 | 21 | 10 |
| k_ph_n | 2 | 4 | 11 | 21 | 9 |
| k_ph_p | 2 | 4 | 11 | 9 | 8 |
| k_poly_n | 0 | 1 | 2 | 21 | 21 |
| k_poly_p | 0 | 1 | 1 | 21 | 21 |
| k_t4p_n | 0 | 0 | 6 | 21 | 21 |
| k_t4p_p | 0 | 1 | 10 | 21 | 21 |
| Total: | 6 | 50 | 137 | 358 | 301 |
| Galen-kris-1,% | 97.34 | 93.13 | 91.07 | 100.00 | 100.00 |
| Galen-kris-2,% | 99.54 | 98.66 | 97.83 | 100.00 | 100.00 |
| Platt,% | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 |

Figure 1: Experimental Results

results are summarized in Figure 1. In the table, the names k_*_* denote the concept sequences of T98. The entries are to be read as follows: the reasoners had 100 seconds to decide the satisfiability of each concept in a sequence. The number given in the table is then the number of the last concept that a reasoner was able to solve within this time. Thus, the entry "21" means that all concepts in a given sequence have been solved. For the "real world" concepts, for which the results can be found in the last three lines, we use a different scheme: there are again 100 seconds available for reasoning on each concept, but since these concepts are not ordered w.r.t. increasing difficulty, we simply give the percentage of concept that the reasoner was able to solve within the given time. Note that, in total, there were 1165 concepts for Galen-kris-1, 1936 concepts for Galen-kris-2, and 262 concepts for Platt

There are several interesting observations to be made in Figure 1. First, it is obvious that the dedicated DL reasoners RACER and FaCT outperform Mona in most cases, except for the translation T2b used on the sequences k_lin_n, k_path_p, k_path_n, k_ph_n, and k_ph_p—in the last case, Mona even beats FaCT *and* RACER. Second, on the T98 concepts the translation T2b performs much better than the other two translations, and T1 is worst solving only 6 concepts out of 378. Surprisingly, the situation is reversed for the real world concepts: here T1 outperforms both other translations and T2b is worst. And third, there is a surprising difference in the performance of the almost identical T2 and T2b translations (at least on the T98 concepts). Thus, Mona is apparently quite sensitive to small changes in the translation.

How should these results be judged? Let us start with saying that the better performance of FaCT and RACER is perhaps not too surprising: both Mona and the involved DL reasoners are highly optimized, but Mona is capable of dealing with a much more powerful logic. This is witnessed by the fact that the complexity of WS1S and WS2S is non-elementary [9], while FaCT and RACER implement ExpTime-complete logics. Still, we believe that the performance of our translations is reasonable. In particular, it should be taken into account that, compared to the implementation of a full-fledged DL reasoner, the implementation of our translations is a piece of cake. Since many different description logics can be translated to WS1S and WS2S, we believe that such translations can be useful at least for prototyping purposes.

To analyze why the three translations exhibit a different performance, we need to introduce some of Mona's implementation details. To decide the satisfiability of a formula, Mona constructs a finite automaton that accepts the empty language if and only if the input formula is unsatisfiable, and then performs an emptyness test on this automaton. The construction of the automaton, which works on $\omega$-words in the case of WS1S and on infinite trees in the WS2S case, involves a number of automata-theoretic operations: complementation for dealing with logical negation, union for disjunction, product for conjunction, and projection for quantifiers. Since Mona always works with *determinstic* automata, the most "dangerous" operation in this list is projection as it involves the determinization of a non-deterministic automaton. As is well-known, this may produce an exponential blowup in automaton size, in contrast to constant blowup produced by complementation and union, and quadratic blowup produced by the product. This is closely related to the fact that WS1S and WS2S are non-elementary: a separate projection cannot be avoided for each alternation of logical quantifiers, thus resulting in repeated exponential blowups.

Surprisingly, however, quantifier alternation is almost never the culprit of performance problems in our translation-based appoach to DL reasoning: the reason for non-termination is usually the more harmless looking product operation, i.e. the treatment of logical conjunction. To understand this, we must know some more details about Mona internals: the alphabet of the automaton constructed for a (sub)formula $\varphi$ is comprised of 0/1-strings whose length is identical to the number of free variables in $\varphi$—the intuition is that the domain element described by such a string is contained in the extension of exactly those free variables for which we find a "1" value in the string. Thus, the size of the alphabet is exponential in the length of the input formula. To cope with this, Mona stores the transition table of automata in a compressed way using binary decision diagrams (BDDs). Unfortunately, this compression only works well if the underlying formula enforces many interdependencies among the free variables.

Now consider our translation T1: each subformula $\vartheta_R$ is a huge conjunction inside a "$\forall\exists$" quantifier pattern. Moreover, each conjunct involves a large number of free variables: since first-order predicates are treated by Mona as free variables (which are implicitly existentially quantified on the outermost level), this includes the predicates $P_A$ introduced for concept names and the predicates $Q_C$ introduced for subconcepts $C$ of the form $\exists R.D$ and $\forall R.D$. Unfortunately, the interaction between these free variables turns out to be rather weak in general. Thus, the repeated product operations performed when processing the huge conjunction in $\vartheta_R$ produce an automaton whose BDD has an excessive number of nodes. The subsequent projection and determinization that is performed to treat the second operator in the "$\forall\exists$" pattern is not able to process this large automaton.

In contrast, the formulas produced by T2 and T2b reflect the structure of the original concept, and are thus not just a big conjunction. This explains why T2b is better on the T98 benchmark. But why, then, is T1 better on real world concepts? There seem to be two reasons: first, the concepts extracted from real world knowledge bases are much smaller than the T98 ones, which can be several megabytes in size. Thus, conjunctions produced by T1 are less gigantic in the real world case. Second, Mona can handle automata on $\omega$-words more efficiently than tree-automata, thus giving T1 an advantage over T2 and T2b.

We have also used our translation for deciding the satisfiability of concepts w.r.t. TBoxes, e.g. on the Platt knowledge base and on the two $\mathcal{ALC}$ versions of Galen. Unfortunately, the results were discouraging: Mona terminated only for very small TBoxes ($< 10$ concept equations) and was not able to classify any of the real world KBs. The reason for this is again due to the generation of BDDs with a massive number of nodes: in all our translations, TBoxes are translated into a huge conjunction inside a "$\forall$" quantifier. Hence, we can observe the same blowup pattern as with T1 on the T98 concepts. Even a preceeding "absorption" of concept equations as known from [7], and an elimination of "non-relevant" concept equations as proposed in [12] did not allow us to classify real world KBs.

## 4 Discussion

We have shown how the Mona tool can be exploited for reasoning about description logics. The outcome of our experiments suggests that, although Mona is outperformed by dedicated DL reasoners, her performance is sufficient at least for prototyping purposes. In this context, it should be noted that translations can be implemented rather quickly, and that Mona is expressive enough to capture a large class of description logics. For example, it should not be hard to come up with translations for more powerful DLs such as $\mathcal{SHIQ}$, and even to treat features that are very difficult for tableau reasoners, such as the transitive closure of roles.

It would be very interesting to determine a class of concepts on which Mona performs good, but the DL reasoners do not, and vice versa. A first idea is provided by the discussion in Section 3: Mona is good on disjunction (the corresponding operation "automata union" only yields a constant blowup in size), and bad on conjunction; for tableau algorithms, this is the other way round. Still, we found it difficult to come up with the desired class of concepts due to the intricate optimization techniques of FaCT and RACER. The two reasoners even behave quite differently: we have found two classes of formulas on which Mona performs good, but one of FaCT and RACER performs bad. A class of such formulas on which *both* FaCT and RACER perform bad, however, remains yet to be seen.

The translator and concepts used in the experiments can be downloaded from the internet, c.f. http://lat.inf.tu-dresden.de/∼clu/atom.tar.gz.

## References


[1] The Mona homepage. http://www.brics.dk/mona/.

[2] F. Baader, D. L. McGuiness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook: Theory, implementation and applications*. Cambridge University Press, 2003.

[3] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.

[4] J. Elgaard, N. Klarlund, and A. Moller. Mona 1.x: new techniques for ws1s and ws2s. In *Computer Aided Verification, CAV '98, Proceedings*, volume 1427 of *LNCS*. Springer Verlag, 1998.

[5] V. Haarslev and R. Möller. RACER system description. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Proceedings of the First International Joint Conference on Automated Reasoning (IJCAR'01)*, number 2083 in Lecture Notes in Artifical Intelligence, pages 701–705. Springer-Verlag, 2001.

[6] A. Heuerding and S. Schwendimann. A benchmark method for the propositional modal logics k, kt, s4. Technical Report IAM-96-015, University of Bern, Switzerland, 1996.

[7] I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. Phd thesis, University of Manchester, 1997.

[8] I. Horrocks. Using an expressive description logic: Fact or fiction? In *Proceedings of the Sixth International Conference on the Principles of Knowledge Representation and Reasoning (KR98)*, pages 636–647, 1998.

[9] A. Meyer. Weak monadic second order theory of successor is not elementary-recursive. In *LOGCOLLOQ: Logic Colloquium*. Lecture Notes in Mathematics, No. 453, Springer-Verlag, 1975.

[10] M.O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

[11] V. R. Pratt. Models of program logics. In *Proceedings of the Twentieth Annual Symposium on Foundations of Computer Science*, San Juan, Puerto Rico, 1979.

[12] D. Tsarkov and I. Horrock. Dl reasoner vs. first-order prover. In E. F. Diego Calvanese, Giuseppe De Giacomo, editor, *Proceedings of the International Workshop in Description Logics 2003 (DL2003)*, number 81 in CEUR-WS (http://ceur-ws.org/), pages 152–159, 2003.