A Practical Automata-Based Technique for Reasoning in Expressive Description Logics*

Diego Calvanese
KRDB Research Centre
Free Univ. of Bozen-Bolzano
Piazza Domenicani 3, Bolzano, Italy
calvanese@inf.unibz.it

Domenico Carbotta and **Magdalena Ortiz**
Institute of Information Systems
Vienna Univ. of Technology
Favoritenstraße 9-11, Vienna, Austria
ortiz@kr.tuwien.ac.at, domenico.carbotta@gmail.com

Abstract

In this work we describe the theoretical foundations and the implementation of a new automata-based technique for reasoning over expressive Description Logics that is worst-case optimal and lends itself to an efficient implementation. In order to show the feasibility of the approach, we have realized a working prototype of a reasoner based upon these techniques. An experimental evaluation of this prototype shows encouraging results.

1 Introduction

Description Logics (DLs) [Baader *et al.*, 2007] are a well-established family of logics designed for knowledge representation and reasoning. In DLs, the domain of interest is represented in terms of concepts and roles, denoting unary and binary predicates, respectively. Knowledge about the domain is represented in a knowledge base, which is constituted in general by a set of (inclusion) assertions between concepts and roles In expressive DLs, there is no limitation on the form of inclusion assertions, and it is well known that this makes reasoning EXPTIME-hard in general, though decidable and EXPTIME-complete in many significant cases [Baader *et al.*, 2007].

A reason for the robust decidability of DLs (shared with Modal Logics) is the fact that they enjoy the tree-model property [Vardi, 1997]. This property is either directly or indirectly at the basis of a variety of different techniques that have been proposed for reasoning in expressive DLs, the most prominent of which are tableaux-based calculi [Baader and Sattler, 2001; Motik *et al.*, 2009] resolution-based techniques [Hustadt *et al.*, 2008], and techniques based on automata on infinite trees [Vardi, 1998; Calvanese *et al.*, 2002]. Although not computationally optimal, tableau techniques provide the basis for most of the currently implemented state-of-the-art DL reasoning systems, such as FACT [Tsarkov and Horrocks, 2006], RACER [Baader *et al.*, 2007], PELLET [Sirin and Parsia, 2006], and HERMIT [Shearer *et al.*, 2008].

*This work was partially supported by the Austrian Science Fund (FWF) grant P20840, and by the EU under the ICT Collaborative Project ACSI (Artifact-Centric Service Interoperation), grant agreement n. FP7-257593.

The automata-based approach is based on translating a knowledge base (KB) whose satisfiability is to be checked into some variant of automata on infinite trees that accepts tree-shaped models of the KB, and checking such an automaton for non-emptiness. This approach is powerful and flexible. It is acknowledged that it provides a very robust basis for showing worst-case optimal complexity upper bounds, and has been applied for a wide range of expressive DLs and reasoning services (cf. [De Giacomo and Lenzerini, 1994; Tobies, 2001; Calvanese *et al.*, 2002; 2007] and their references). Interestingly, however, up to now such techniques have resisted implementation, since they essentially require to fully construct an exponentially large automaton, and therefore exhibit a best-case exponential behaviour. An exception is the inverse tableaux method reported in [Voronkov, 2001], which according to [Baader and Tobies, 2001] can be considered as an implementation of an automata-based algorithm for satisfiability in modal with global axioms, which essentially correspond to DL inclusion assertions.

We develop here theoretical results that represent a first significant step towards the development of practical automata-based techniques for reasoning in expressive DLs. Our core contribution is a technique for an *incremental* construction of the state space of a non-deterministic looping tree automaton, which accepts the same trees as an alternating looping tree automaton that directly encodes the KB. At each step of the incremental construction, part of the computation can be delegated to efficient off-the-shelf SAT solvers. The method is worst-case exponential in general, but it lends itself to an efficient implementation. In this paper we deal with TBoxes constituted by arbitrary inclusion assertions between concepts expressed in the propositionally complete DL $\mathcal{ALC}$, extended with functional roles. We have chosen a relatively simple DL in order to keep presentation and notation simple, however our results extend to more expressive DLs, e.g., to those containing inverse roles.

To the feasibility of our approach, we have realized a first working prototype of a reasoner that is based upon these techniques. Our prototype is not yet in the league of current state-of-the-art systems for expressive DLs, but shows already encouraging results in experimental evaluations.

For space reasons, full proofs are omitted here, but they can be found in [Carbotta, 2010], together with more detailed explanations and additional examples.

2 Preliminaries

We recall the syntax and semantics of the DL $\mathcal{ALCQ}^f$, which extends $\mathcal{ALC}$ with global functionality, and give some preliminaries on automata on infinite trees.

2.1 The Description Logic $\mathcal{ALCQ}^f$

Given two disjoint sets of *atomic concepts* and *atomic roles*, the syntax of $\mathcal{ALCQ}^f$ *concepts* is specified in Table 1, as usual. We use A, P, and C, to denote atomic concepts, atomic roles, and concepts, respectively. We use $\top$ as an abbreviation for $A \sqcup \neg A$, for some atomic concept A. The semantics is defined in terms of interpretations $\mathcal{I} = (\Delta_\mathcal{I}, \cdot^\mathcal{I})$. The interpretation function $\cdot^\mathcal{I}$ maps each atomic concept to a subset of $\Delta_\mathcal{I}$ and each atomic role to a binary relation over $\Delta_\mathcal{I}$, and is extended to all concepts as shown in Table 1. We use $\mathsf{sub}(C)$ to denote the set of all subconcepts of a concept C. By $\mathsf{nnf}(C)$ we denote the *negation normal form* (NNF) of C (i.e., $\neg$ is applied only to atomic concepts). In what follows we assume w.l.o.g. that all concepts are in NNF.

An *inclusion assertion* is an expression of the form $C \sqsubseteq D$, where C and D are concepts, and a *functionality assertion* is an expression of the form $(\mathsf{func}\ P)$, where P is a role. An interpretation $\mathcal{I}$ is a *model* of $C \sqsubseteq D$ if $C^\mathcal{I} \subseteq D^\mathcal{I}$, and it is a model of $(\mathsf{func}\ P)$ if every individual in $\Delta_\mathcal{I}$ has at most one P-*successor*, that is, for every $d \in \Delta_\mathcal{I}$ there is at most one $d' \in \Delta_\mathcal{I}$ such that $(d, d') \in P^\mathcal{I}$. A *TBox* $\mathcal{T}$ is a set of inclusion and functionality assertions; an interpretation $\mathcal{I}$ is a *model* of $\mathcal{T}$ (denoted $\mathcal{I} \models \mathcal{T}$) if it is a model of every assertion in $\mathcal{T}$.

A concept C is *satisfiable in a TBox* $\mathcal{T}$ if $\mathcal{T}$ has a model $\mathcal{I}$ such that $C^\mathcal{I} \neq \emptyset$. Other standard reasoning problems, such as satisfiability of a TBox, satisfiability of a concept, or logical implication of an inclusion assertion, can be reduced to satisfiability of a concept in a TBox in the usual way [Baader *et al.*, 2007].

2.2 Tree Automata

For $0 \leq i \leq k$, we use $[i : k]$ to denote $\{i, \ldots, k\}$. As usual, a k-*ary tree* N is a (possibly infinite) prefix-closed subset of $[1 : k]^*$. Its *root* is the empty word ε and the elements of N are called *nodes*. A node $v \cdot c \in N$ is a *child* of v, and v is the *parent* of $v \cdot c$. By convention, $v \cdot 0 = v$. A *leaf node* has no children. A *path* is a sequence of nodes from N where every node is the parent of the next one in the sequence, and a *branch* is a path starting from the root that either ends in a leaf node or is infinite. An *alphabet* is a finite set Σ, and a Σ-*labeled* k-*ary tree* is a tuple $T = (N, \tau)$, where N is a k-ary tree and $\tau : N \to \Sigma$ is a *labeling function*.

Syntax	Semantics
$\neg C$	$(\neg C)^\mathcal{I} = \Delta_\mathcal{I} \setminus C^\mathcal{I}$
$C_1 \sqcap C_2$	$(C_1 \sqcap C_2)^\mathcal{I} = {C_1}^\mathcal{I} \cap {C_2}^\mathcal{I}$
$C_1 \sqcup C_2$	$(C_1 \sqcup C_2)^\mathcal{I} = {C_1}^\mathcal{I} \cup {C_2}^\mathcal{I}$
$\forall P.C$	$(\forall P.C)^\mathcal{I} = \{d \in \Delta_\mathcal{I} \mid \forall d'.(d, d') \in P^\mathcal{I} \rightarrow d' \in C^\mathcal{I}\}$
$\exists P.C$	$(\exists P.C)^\mathcal{I} = \{d \in \Delta_\mathcal{I} \mid \exists d'.(d, d') \in P^\mathcal{I} \wedge d' \in C^\mathcal{I}\}$

Table 1: $\mathcal{ALCQ}^f$ syntax and semantics

We denote by $\mathsf{PBF}(X)$ the set of *positive boolean formulas* that can be built using $\wedge$ and $\vee$ from the symbols in $X \cup \{\mathsf{f}, \mathsf{t}\}$. For $\varphi \in \mathsf{PBF}(X)$ and $Y \subseteq X$, we say that Y *satisfies* φ, written $Y \models \varphi$, if setting to true the symbols in Y makes φ true in the usual sense.

An *alternating looping tree automaton (ATA)* is a tuple $\mathbf{A} = (\Sigma, Q, q_0, k, \delta)$, where Σ is the *input alphabet*, Q is the (finite) set of *states*, $q_0 \in Q$ is the *initial state*, $k \in \mathbb{N}$ is the *branching degree*, and $\delta : Q \times \Sigma \to \mathsf{PBF}([0 : k] \times Q)$ is the *transition function*. Given a Σ-labeled k-ary tree $T = (N, \tau)$, a *run* of $\mathbf{A}$ over T is a $N \times Q$-labeled tree $R = (N_R, \rho)$ such that: *(1)* $\rho(\varepsilon) = (\varepsilon, q_0)$; and *(2)* for every $r \in N_R$, if $\rho(r) = (v, q)$, there exists a set of index-state pairs $\{(i_1, q_1), \ldots, (i_n, q_n)\}$, for $n \geq 0$, that satisfies $\delta(q, \tau(v))$ and such that $r \cdot j \in N_R$ and $\rho(r \cdot j) = (v \cdot i_j, q_j)$, for every $j \in [1 : k]$. We denote by $\mathcal{L}(\mathbf{A})$ the *language accepted by* $\mathbf{A}$, i.e., the set of all Σ-labeled k-ary trees over which there exists a run of $\mathbf{A}$.

If the transition function of an ATA $\mathbf{A}$ does not include atoms whose first component is 0, and additionally for every pair $(q, \sigma) \in Q \times \Sigma$, no clause in the disjunctive normal form of $\delta(q, \sigma)$ contains two atoms with the same first component, we call $\mathbf{A}$ a *nondeterministic tree automaton (NTA)*.

3 Automata for $\mathcal{ALCQ}^f$

We first illustrate the classical approach to reasoning in expressive DLs based on alternating automata on infinite trees for concept satisfiability in $\mathcal{ALCQ}^f$ [Vardi, 1998; Calvanese *et al.*, 2002]. We then show how to adapt this approach so as to make it implementable.

Let $\mathcal{T}_f = \{C_1 \sqsubseteq D_1, \ldots, C_n \sqsubseteq D_n\}$ be an $\mathcal{ALCQ}^f$ TBox, and let $\mathcal{C}$ and $\mathcal{R}$ be respectively the sets of atomic concepts and atomic roles that appear in $\mathcal{T}_f$. Let $\mathcal{R}_f \subseteq \mathcal{R}$ be the set of *functional roles* of $\mathcal{T}_f$, i.e., those roles that appear in $\mathcal{T}_f$ in a functionality assertion, and let $\mathcal{T}$ be the $\mathcal{ALCQ}$ TBox obtained by removing from $\mathcal{T}_f$ all functionality assertions. Let C_0 be an $\mathcal{ALCQ}$ concept defined over $\mathcal{C}$ and $\mathcal{R}$.

Let $C_\mathcal{T} = \mathsf{nnf}(\bigsqcap_{C_i \sqsubseteq D_i \in \mathcal{T}} \neg C_i \sqcup D_i)$ be the *internalized* version of $\mathcal{T}$, let $\mathcal{E}$ be the set of all the existential subconcepts of $C_\mathcal{T}$ and C_0 that do not refer to a role from $\mathcal{R}_f$, and let $k = |\mathcal{E}| + |\mathcal{R}_f|$. Fix an arbitrary bijection idx from $\mathcal{E} \cup \mathcal{R}_f$ to $[1 : k]$. We make use of the following result, which follows directly from the standard tree-model property of $\mathcal{ALCQ}^f$.

Theorem 3.1 (Canonical witness). *If the concept C_0 is satisfiable in $\mathcal{T}_f$, there exists a* canonical witness $\mathcal{I} = (\Delta_\mathcal{I}, \cdot^\mathcal{I})$ *that satisfies the following properties:*

- $\Delta_\mathcal{I}$ *is a k-tree.*
- *For every role $P \in \mathcal{R}$ and every pair $(a, b) \in P^\mathcal{I}$, $b = a \cdot c$, for some $c \in [1 : k]$.*
- *$\mathcal{I}$ is a model of $\mathcal{T}$, and therefore $C_\mathcal{T}^\mathcal{I} = \Delta_\mathcal{I}$.*
- *$\varepsilon \in C_0^\mathcal{I}$.*
- *For every existential subconcept $\exists P.C \in \mathcal{E}$ and every node $v \in \Delta_\mathcal{I}$, let $v' = v \cdot \mathsf{idx}(\exists P.C)$:*
  - *if $v \in (\exists P.C)^\mathcal{I}$, then $v' \in \Delta_\mathcal{I}$, $(v, v') \in P^\mathcal{I}$ and $v' \in C^\mathcal{I}$;*
  - *otherwise, $v' \notin \Delta_\mathcal{I}$.*
- *For every functional role $P \in \mathcal{R}_f$ and every node $v \in (\exists P.\top)^\mathcal{I}$, let $v' = v \cdot \mathsf{idx}(\exists P.C)$. Then $v' \in \Delta_\mathcal{I}$ and $(v, v') \in P^\mathcal{I}$.*

By virtue of the previous theorem, we can decide whether C_0 is satisfiable in $\mathcal{T}_f$ by *(i)* constructing a tree automaton that recognizes a suitable encoding of the canonical witnesses, and *(ii)* checking whether it accepts a non-empty language.

In the following, let $\Sigma = 2^{\mathcal{C}} \cup \{\sigma_{\mathsf{ne}}\}$, with σ_{ne} a fresh symbol, intuitively labeling dummy nodes of the tree. Given a canonical witness $\mathcal{I} = (\Delta_{\mathcal{I}}, \cdot^{\mathcal{I}})$, we define its tree-encoding $\mathsf{treeEnc}(\mathcal{I}) = ([1:k]^*, \tau)$ as a Σ-labeled k-tree whose labeling τ is defined as follows:

- for every $v \in \Delta_{\mathcal{I}}$, let $\tau(v) = \{A \in \mathcal{C} \mid v \in A^{\mathcal{I}}\}$;
- for every $v \in [1:k]^* \setminus \Delta_{\mathcal{I}}$, let $\tau(v) = \sigma_{\mathsf{ne}}$.

Let $\mathsf{Aut}(\mathcal{T}_f, C_0) = (\Sigma, Q, q_0, k, \delta)$ be an ATA with $Q = \{q_0, q_{\mathsf{tbox}}, q_{\mathsf{ne}}\} \cup \mathsf{sub}(C_0) \cup \mathsf{sub}(C_{\mathcal{T}})$, where q_{tbox} is a state used to check that $\mathcal{T}$ is satisfied, and q_{ne} is a state used to check that the descendants of dummy nodes are also dummy. To define δ, consider the function $\mathsf{all} : \mathcal{R} \to 2^{[1:k]}$ defined as

$$\mathsf{all}(P) = \{i \in [1:k] \mid \mathsf{idx}^{-1}(i) = \exists P.C, \text{for some } C, \text{ or } \mathsf{idx}^{-1}(i) = P\}$$

Intuitively, $\mathsf{all}(P)$ is the set containing the indices of all the existential subconcepts that mention the role P. Then:

$$\begin{aligned}
\delta(q_0, c) &= (0, C_0) \wedge (0, q_{\mathsf{tbox}}) \\
\delta(q_{\mathsf{tbox}}, c) &= (0, C_{\mathcal{T}}) \wedge \textstyle\bigwedge_{1 \le i \le k}((i, q_{\mathsf{tbox}}) \vee (i, q_{\mathsf{ne}})) \\
\delta(q_{\mathsf{ne}}, \sigma_{\mathsf{ne}}) &= \textstyle\bigwedge_{1 \le i \le k}(i, q_{\mathsf{ne}}) \\
\delta(A, c) &= \mathsf{t}, \text{ if } A \in c \qquad \delta(A, c) = \mathsf{f}, \text{ if } A \notin c \\
\delta(\neg A, c) &= \mathsf{t}, \text{ if } A \notin c \qquad \delta(\neg A, c) = \mathsf{f}, \text{ if } A \in c \\
\delta(C \sqcap C', c) &= (0, C) \wedge (0, C') \\
\delta(C \sqcup C', c) &= (0, C) \vee (0, C') \\
\delta(\exists P.C, c) &= \begin{cases} (\mathsf{idx}(\exists P.C), C), & \text{if } P \text{ is not functional} \\ (\mathsf{idx}(P), C), & \text{if } P \text{ is functional} \end{cases} \\
\delta(\forall P.C, c) &= \textstyle\bigwedge_{i \in \mathsf{all}(P)}((i, q_{\mathsf{ne}}) \vee (i, C))
\end{aligned}$$

where $c \in 2^{\mathcal{C}}$, $A \in \mathcal{C}$, and $P \in \mathcal{R}$

Intuitively, a run of this automaton consists of two execution threads. One, starting from the state C_0, checks whether the root of the tree is a model of the given concept; the second one, starting from the state q_{tbox}, checks whether the current node is a model of $C_{\mathcal{T}}$ and propagates itself to all the "real" child nodes (those not denoted by the σ_{ne} symbol).

The following result is an immediate consequence of the classical results on the automata-theoretic approach to satisfiability on modal logics and DLs (see, e.g., [Vardi and Wilke, 2007]):

Theorem 3.2. *The automaton $\mathsf{Aut}(\mathcal{T}_f, C_0)$ accepts a tree iff it is the tree-encoding of a canonical witness of C_0 in $\mathcal{T}_f$.*

Hence we can reduce the problem of concept satisfiability to an emptiness check of an ATA.

Theorem 3.3. *An $\mathcal{ALC}$ concept C_0 is satisfiable in an $\mathcal{ALC}^f$ TBox $\mathcal{T}_f$ iff $\mathcal{L}(\mathsf{Aut}(\mathcal{T}_f, C_0)) \neq \emptyset$.*

3.1 Emptiness of an Automaton

We now address the problem of efficiently checking emptiness of an ATA. To this aim, rather than relying on emptiness testing procedures known from the literature, we devise a simple algorithm that allows for optimizations and that lies at the core of our practical decision procedure.

Definition 3.4. *We say that an ATA $\mathbf{A} = (\Sigma, Q, q_0, k, \delta)$ is* zero-layered *if there exists a total ordering $\prec$ on the states such that, for any two states q, q' and any symbol c, if $q \prec q'$ then the atom $(0, q)$ does not appear in $\delta(q', c)$. We say that $\mathbf{A}$ is a* zero-free looping tree automaton (ZFA) *if the transition function does not contain atoms whose first component is 0.*

Every zero-layered ATA $\mathbf{A} = (\Sigma, Q, q_0, k, \delta)$ can be transformed into a ZFA accepting the same language. To do so, we define by mutual recursion a transition function $\delta_{\mathsf{zf}} : Q \times \Sigma \to \mathsf{PBF}([1:k] \times Q)$, called the *zero-closure* of δ, and a substitution $\sigma_{q,c}$, for every state q and every symbol c:

$$\sigma_{q,c} = \{\big((0, q'),\ \delta_{\mathsf{zf}}(q', c)\big) \mid q \prec q'\}$$
$$\delta_{\mathsf{zf}}(q, c) = \delta(q, c)\sigma_{q,c}$$

Informally, the substitution $\sigma_{q,c}$ collapses all the zero-transitions involving states that follow q in the total ordering, by repeatedly replacing every atom of the form $(0, q')$ with the PBF that specifies the transition for q' and character c. We use δ_{zf} as the transition function of the desired ZFA.

Proposition 3.5. *Let $\mathbf{A} = (\Sigma, Q, q_0, k, \delta)$ be a zero-layered ATA, and $\mathbf{A}_{\mathsf{zf}} = (\Sigma, Q, q_0, k, \delta_{\mathsf{zf}})$. Then $\mathbf{A}_{\mathsf{zf}}$ is a ZFA and $\mathcal{L}(\mathbf{A}_{\mathsf{zf}}) = \mathcal{L}(\mathbf{A})$.*

Example 3.1. *Consider the automaton $\mathbf{A} = (\{0,1\}, \{q_0, q_1, q_a, q_b, q_h\}, q_0, 2, \delta)$, where δ is defined as follows:*

$$\begin{aligned}
\delta(q_0, 0) &= (1, q_1) \wedge (0, q_a) \wedge (1, q_h) \wedge (2, q_h) \\
\delta(q_1, 0) &= (1, q_0) \wedge (0, q_b) \\
\delta(q_1, 1) &= (1, q_0) \wedge (0, q_b) \\
\delta(q_a, 0) &= (2, q_1) \\
\delta(q_b, 1) &= (2, q_2) \\
\delta(q_h, 1) &= \mathsf{t}
\end{aligned}$$

and the missing entries are assumed to be f.

The automaton $\mathbf{A}$ is zero-layered: the ordering $q_0 \prec q_1 \prec q_a \prec q_b$ is one of the possible relations satisfying the requirement.

The value of δ_{zf} can be derived as follows:

$$\begin{aligned}
\delta_{\mathsf{zf}}(q_h, 1) &= \mathsf{t} \\
\delta_{\mathsf{zf}}(q_b, 1) &= \delta(q_b, 1) = (2, q_0) \\
\delta_{\mathsf{zf}}(q_a, 0) &= \delta(q_a, 0) = (2, q_1) \\
\delta_{\mathsf{zf}}(q_1, 0) &= \delta(q_1, 1)[(0, q_b)/(2, q_0)] = \\
&= (1, q_0) \wedge (2, q_0) \\
\delta_{\mathsf{zf}}(q_1, 1) &= \delta(q_1, 1)[(0, q_b)/(2, q_0)] = \\
&= (1, q_0) \wedge (2, q_0) \\
\delta_{\mathsf{zf}}(q_0, 0) &= \delta(q_0, 0)[(0, q_a)/(2, q_1)] = \\
&= (1, q_1) \wedge (2, q_1) \wedge (1, q_h) \wedge (2, q_h)
\end{aligned}$$

It is easy to check that the ZFA $\mathbf{A}' = (\{0,1\}, \{q_0, q_1, q_a, q_b\}, q_0, 2, \delta_{\mathsf{zf}})$ accepts the same language as $\mathbf{A}$.

The next step in our decision procedure transforms a ZFA into an NTA by applying a variant of the subset construction.

We let $\delta_{\mathsf{n}} : 2^Q \times \Sigma \to \mathsf{PBF}([1:k] \times 2^Q)$ be defined as:

$$\delta_{\mathsf{n}}(\omega, c) = \begin{cases} \mathsf{t}, & \text{if } \omega = \emptyset \\ \bigvee_{\substack{\Gamma_j \text{ such that} \\ \Gamma_j \models \bigwedge_{q \in \omega} \delta_{\mathsf{zf}}(q,c)}} \bigwedge_{(i,\omega_i^j) \in \mathsf{group}(\Gamma_j)} (i, \omega_i^j), & \text{if } \omega \neq \emptyset \end{cases}$$

$$\begin{aligned} \text{where } \mathsf{collect}(\Gamma, i) &= \{q \in Q \mid (i,q) \in \Gamma\}, \\ \mathsf{group}(\Gamma) &= \{(i, \mathsf{collect}(\Gamma, i)) \mid i \in [1:k]\}. \end{aligned}$$

Proposition 3.6. *Let $\mathbf{A}_{\mathsf{zf}} = (\Sigma, Q, q_0, k, \delta_{\mathsf{zf}})$ be a ZFA, and $\mathbf{A}_{\mathsf{n}} = (\Sigma, 2^Q, \{q_0\}, k, \delta_{\mathsf{n}})$. Then $\mathbf{A}_{\mathsf{n}}$ is an NTA and $\mathcal{L}(\mathbf{A}_{\mathsf{n}}) = \mathcal{L}(\mathbf{A}_{\mathsf{zf}})$.*

Intuitively, a run of $\mathbf{A}_{\mathsf{n}}$ "collapses" a run of $\mathbf{A}_{\mathsf{zf}}$: if $\mathbf{A}_{\mathsf{zf}}$ visits a node w of the input two times, in states q and q', $\mathbf{A}_{\mathsf{n}}$ will visit it once in the state $\{q, q'\}$. The transition function ensures that the labeling of the successors satisfies the transition condition for both q and q'.

Example 3.2. *Consider the automaton $\mathbf{A}'$ obtained in Example 3.1. We will calculate the value of $\delta_{\mathsf{n}}(\{q_1, q_h\}, 1)$ as follows.*

The models of $\delta_{\mathsf{zf}}(q_1, 1) \wedge \delta_{\mathsf{zf}}(q_h, 1)$ are all the supersets of the set $\{(1, q_0), (2, q_0)\}$. As a consequence, $\delta_{\mathsf{n}}(\{q_1, q_h\}, 1)$ has the following form:

$$\delta_{\mathsf{n}}(\{q_1, q_h\}, 1) = \bigvee_{\omega, \omega' \in 2^Q} (1, \{q_0\} \cup \omega) \wedge (2, \{q_0\} \cup \omega').$$

On the other hand, $\delta_{\mathsf{zf}}(q_h, 0)$ is unsatisfiable. As a consequence, $\delta_{\mathsf{zf}}(\{q_1, q_h\}, 0) = \mathsf{f}$.

By Theorem 3.3, satisfiability of a concept C_0 in an $\mathcal{ALC}^f$ TBox $\mathcal{T}_f$ can be reduced to checking non-emptiness of the ATA $\mathsf{Aut}(\mathcal{T}_f, C_0)$. We observe that such an ATA is zero-layered. To see this, we can consider the following ordering of its states: *(i)* q_0 precedes every other state; *(ii)* q_{tbox} precedes every other state, except for q_0; *(iii)* q_{ne} is preceded by every other state; *(iv)* for any two states $C, C' \in \mathsf{sub}(C_{\mathcal{T}}) \cup \mathsf{sub}(C_0)$, if C' is a subconcept of C then $C' \prec C$.

By virtue of this observation, and Propositions 3.5 and 3.6, we can concentrate on the emptiness problem for NTAs. Given an NTA $\mathbf{A}_{\mathsf{n}} = (\Sigma, \Omega, \omega_0, k, \delta_{\mathsf{n}})$, let $\mathsf{step} : 2^\Omega \to 2^\Omega$ be the function defined as:

$$\begin{aligned} \mathsf{step}(\Omega') = \{\omega \in \Omega' \mid\ & \exists \omega'_1, \ldots, \omega'_k \in \Omega', c \in \Sigma. \\ & \{(i, w'_i)\}_{1 \leq i \leq k} \models \delta_{\mathsf{n}}(\omega, c)\}. \end{aligned}$$

Intuitively, $\mathsf{step}(\Omega')$ contains all the states in Ω' that can be satisfied by a model containing only atoms from $[1:k] \times \Omega'$.

Let Ω_{fix} be the greatest fixed point of step, i.e., the biggest subset of Ω such that $\mathsf{step}(\Omega_{\mathsf{fix}}) = \Omega_{\mathsf{fix}}$. The function step is monotone decreasing, and by the Knaster-Tarski fixed point theorem, the greatest fixed point Ω_{fix} exists and can be calculated as $\Omega_{\mathsf{fix}} = \mathsf{step}^{|\Omega|}(\Omega)$.

The emptiness test for $\mathbf{A}_{\mathsf{n}}$ then reduced to computing Ω_{fix} and testing whether it contains ω_0. Indeed, it is possible to show that, for every state $\omega' \in \Omega_{\mathsf{fix}}$, one can build a tree such that $\mathbf{A}_{\mathsf{n}}$ accepts the tree when started in state ω'. By applying this to the initial state of $\mathbf{A}_{\mathsf{n}}$ we get the following result.

Theorem 3.7. *Let $\mathbf{A}_{\mathsf{n}} = (\Sigma, \Omega, \omega_0, k, \delta_{\mathsf{n}})$ be an NTA, and let Ω_{fix} be constructed as above. Then $\mathcal{L}(\mathbf{A}_{\mathsf{n}}) \neq \emptyset$ iff $\omega_0 \in \Omega_{\mathsf{fix}}$.*

Example 3.3. *Consider an NTA $\mathbf{A}_{\mathsf{n}} = (\{a, b\}, \{\omega_0, \omega_1, \omega_2, \omega_3, \omega_4\}, \omega_0, 2, \delta_{\mathsf{n}})$, where the function δ_{n} is defined as follows:*

$$\begin{aligned} \delta_{\mathsf{n}}(\omega_0, a) &= (1, \omega_1) \wedge (2, \omega_2)\ , \\ \delta_{\mathsf{n}}(\omega_1, a) &= (1, \omega_1) \wedge (2, \omega_1)\ , \\ \delta_{\mathsf{n}}(\omega_1, b) &= (1, \omega_3)\ , \\ \delta_{\mathsf{n}}(\omega_2, b) &= (1, \omega_2) \wedge (2, \omega_2)\ , \\ \delta_{\mathsf{n}}(\omega_3, a) &= (1, \omega_4)\ . \end{aligned}$$

and all the missing entries are equal to f.

The algorithm can proceed as follows:

$$\begin{aligned} \Omega_0 &= \{\omega_0, \omega_1, \omega_2, \omega_3, \omega_4\}\ , \\ \Omega_1 &= \mathsf{step}(\Omega_0) = \{\omega_0, \omega_1, \omega_2, \omega_3\}\ , \\ \Omega_2 &= \mathsf{step}(\Omega_1) = \{\omega_0, \omega_1, \omega_2\}\ , \\ \Omega_3 &= \mathsf{step}(\Omega_2) = \{\omega_0, \omega_1, \omega_2\}\ . \end{aligned}$$

We have therefore reached the greatest fixpoint. The state ω_0 belongs to Ω_{fix}: according to our procedure, the language recognized by the automaton is not empty.

Indeed, it is easy to check that $\mathbf{A}_{\mathsf{n}}$ accepts the tree $T = (\{1,2\}^*, \tau)$, where τ is defined as follows:

$$\begin{aligned} \tau(\varepsilon) &= a\ , \\ \tau(1 \cdot w') &= a \quad \textit{for every } w' \in \{1,2\}^*\ , \\ \tau(2 \cdot w') &= b \quad \textit{for every } w' \in \{1,2\}^*\ . \end{aligned}$$

3.2 Complexity Considerations

Given a $\mathcal{ALC}^f$ TBox $\mathcal{T}_f$ and a concept C_0, the ATA $\mathsf{Aut}(\mathcal{T}_f, C_0)$ is zero-layered and has a number of states that is linear in the combined size K of $\mathcal{T}_f$ and C_0. The zero-elimination procedure can be performed in polynomial time, and results in a ZFA having the same number of states as the original ATA. The transformation of this ZFA into an NTA results in exponential blowup of the state space.

Computing the value of $\mathsf{step}(\Omega')$ for a set $\Omega' \subseteq \Omega$ is polynomial in the size of Ω'. As a consequence, computing $\mathsf{step}^{|\Omega|}(\Omega)$ takes time polynomial in the size of Ω, and hence single exponential in K. We obtain therefore a procedure for deciding the non-emptiness of an ATA whose complexity is exponential in the number of states of the original ATA $\mathsf{Aut}(\mathcal{T}_f, C_0)$, and hence in K.

As a consequence, by using this automata-based procedure we can decide the satisfiability of C_0 in $\mathcal{T}_f$ in exponential time. This result matches the well known EXPTIME complexity of the reasoning problem [Baader *et al.*, 2007].

4 A Practical Decision Procedure

The algorithm described in the previous section has a major drawback: it requires the NTA to be fully constructed beforehand. This would lead to a quick exhaustion of the available memory: e.g., in order to decide any property of a small TBox containing 30 axioms, we would need to construct, store, and inspect an NTA that potentially has 2^{60} states. Since TBox reasoning in $\mathcal{ALC}$ is EXPTIME-hard [Baader *et al.*, 2007], this exponential blowup is unavoidable. However, we can try

to reduce the portion of the state space that we *actually* inspect in order to obtain an answer.

Given a zero-layered ATA $\mathbf{A} = (\Sigma, Q, q_0, k, \delta)$, let δ_{n} be the transition function and 2^Q the state space of the NTA obtained according to the procedure shown in Section 3.1. We formulate an iterative procedure that operates on subsets of 2^Q containing so-called *active* and *dead* sets of states. The main idea can be described as follows:

- The set Dead contains sets of states that are provably not in $\mathsf{step}^{2^{|Q|}}(2^Q)$;
- At each step, we will move from Act to Dead those sets of states that could be satisfied only by using sets of states that are already in Dead;
- Next, we will add enough sets of states to Act such that every state in (the previous value of) Act is satisfied by sets that only contain states in Act.

More precisely, our algorithm proceeds as follows.

1. Let $i = 0$, $\mathsf{Act}_0 = \{\{q_0\}\}$, and $\mathsf{Dead}_0 = \emptyset$.
2. Let $\mathsf{Succ}_i : \mathsf{Act}_i \to (2^Q)^k$ be a function that associates to each set of states $\omega \in \mathsf{Act}_i$ an (arbitrarily chosen) k-tuple of sets of states that satisfied the following conditions:
   - If $i > 0$ and $\mathsf{Succ}_{i-1}(\omega)$ does not contain elements from Dead_i, the k-tuple $\mathsf{Succ}_{i-1}(\omega)$;
   - otherwise, a k-tuple $\vec{y} \in (2^Q \setminus \mathsf{Dead}_i)^k$ such that $\mathsf{makeSet}(\vec{y}) \models \bigvee_{c\in\Sigma} \delta_{\mathsf{n}}(\omega, c)$, where $\mathsf{makeSet}((\omega_1, \ldots, \omega_k)) = \{(1, \omega_1), \ldots, (k, \omega_k)\}$;
   - If no such k-tuple exists, the special symbol $\bot$.
3. Let $\tilde{D}_i = \{\omega \in \mathsf{Act}_i \mid \mathsf{Succ}_i(\omega) = \bot\}$.
4. If $q_0 \in \tilde{D}_i$, return f.
5. Let $\tilde{A}_i = \{\omega' \mid \omega' \text{ appears in } \mathsf{Succ}_i(\omega),\ \omega \in \mathsf{Act}_i \setminus \tilde{D}_i\}$.
6. Let $\mathsf{Dead}_{i+1} = \mathsf{Dead}_i \cup \tilde{D}_i$.
7. Let $\mathsf{Act}_{i+1} = (\mathsf{Act}_i \setminus \tilde{D}_i) \cup \tilde{A}_i$.
8. If $\mathsf{Act}_{i+1} = \mathsf{Act}_i$, return t.
9. Increase i and go to Step 2.

There are $2^{|Q|}$ sets of states. In the worst case, they get added to the active set one at a time, and successively moved to the dead set one at a time. We can therefore claim:

Proposition 4.1 (Termination). *The above algorithm terminates after at most $2^{|Q|+1}$ iterations.*

At every step i of the decision procedure, the intermediate results Act_i and Dead_i obey the following invariants: *(i)* $\mathsf{Act}_i \cap \mathsf{Act}_{i+1} \subseteq \mathsf{step}(\mathsf{Act}_{i+1})$; *(ii)* $\mathsf{Dead}_i \cap \Omega_{\mathsf{fix}} = \emptyset$. Invariant (ii) makes it possible to terminate the algorithm as soon as $\{q_0\}$ is declared dead, since we know that the answer will be negative without the need to reach the fixpoint. Hence, we only reach the fixpoint if $\{q_0\}$ is not declared dead.

Suppose the procedure terminates after j steps, i.e., $\mathsf{Act}_j = \mathsf{Act}_{j+1}$. Invariant (i) can therefore be rewritten as $\mathsf{Act}_j \subseteq \mathsf{step}(\mathsf{Act}_j)$. By definition, $\mathsf{step}(\mathsf{Act}_j) \subseteq \mathsf{Act}_j$; as a consequence, $\mathsf{Act}_j = \mathsf{step}(\mathsf{Act}_j) = \mathsf{step}^{2^{|Q|}}(\mathsf{Act}_j)$. Since step is a monotone increasing function and $\mathsf{Act}_j \subseteq 2^Q$, it follows that $\mathsf{step}^{2^{|Q|}}(\mathsf{Act}_j) \subseteq \Omega_{\mathsf{fix}}$ and, therefore, $\mathsf{Act}_j \subseteq \Omega_{\mathsf{fix}}$. Hence:

Theorem 4.2 (Correctness). *Let $\mathbf{A} = (\Sigma, Q, q_0, k, \delta)$ be a zero-layered ATA. Then $\mathcal{L}(\mathbf{A}) \neq \emptyset$ iff the above algorithm returns* t.

4.1 Tuning the Transition Function

At each iteration of the previous algorithm, computing Succ_i might require a huge number of operations on formulas of the form $\bigvee_{c\in\Sigma} \delta_{\mathsf{n}}(\omega, c)$. We introduce now a different formulation of the transition function that makes it easier to compute Succ_i by delegating most of the work to a SAT solver.

Consider the ATA $\mathsf{Aut}(\mathcal{T}_f, C_0) = (\Sigma, Q, q_0, k, \delta)$, as defined in Section 3. We start by noticing that, for non-atomic concepts (existentials, universals, unions and disjunctions), the value of the transition function does not depend on the current symbol. For atomic and negated atomic concepts, the value of the transition function depends on the presence (resp., absence) of an atomic concept in the current symbol, rather than on the "whole" symbol.

Let $\mathsf{SPBF}([1:k] \times Q, \mathcal{C})$ be the language of the (semi-positive) boolean formulas built over the symbols from $[1:k] \times Q \cup \mathcal{C}$, with the additional constraints that only atoms from $\mathcal{C}$ are allowed to appear under the scope of a negation operator. We use these formulas to model the transition function in a more succinct form, by removing the redundancies described above. Let $\Delta : Q \to \mathsf{SPBF}([1:k] \times Q, \mathcal{C})$ be a function defined as follows:

$$
\begin{array}{rcl@{\qquad}rcl}
\Delta(q_0) & = & \Delta(q_{\mathsf{tbox}}) \wedge \Delta(C_0) & \Delta(q_{\mathsf{ne}}) & = & \mathsf{f} \\
\Delta(q_{\mathsf{tbox}}) & = & \Delta(C_{\mathcal{T}}) \wedge \bigwedge_{i\in[1:k]}((i, q_{\mathsf{tbox}}) \vee (i, q_{\mathsf{ne}})) & & & \\
\Delta(C \sqcap C') & = & \Delta(C) \wedge \Delta(C') & \Delta(A) & = & A \\
\Delta(C \sqcup C') & = & \Delta(C) \vee \Delta(C') & \Delta(\neg A) & = & \neg A \\
\Delta(\exists P.C) & = & \begin{cases} (\mathsf{idx}(\exists P.C), C), & \text{if } P \text{ is not functional} \\ (\mathsf{idx}(P), C), & \text{if } P \text{ is functional} \end{cases} & & & \\
\Delta(\forall P.C) & = & \bigwedge_{i\in\mathsf{all}(P)}((i, q_{\mathsf{ne}}) \vee (i, C)) & & &
\end{array}
$$

Let $\hat{\Delta} : 2^Q \to \mathsf{SPBF}([1:k] \times Q, \mathcal{C})$ be the function such that $\hat{\Delta}(\{q_{\mathsf{ne}}\}) = \mathsf{t}$, and $\hat{\Delta}(\omega) = \bigwedge_{q\in\omega} \Delta(q)$ for $\omega \neq \{q_{\mathsf{ne}}\}$. Let $\mathbf{A}_{\mathsf{n}} = (\Sigma, 2^Q, \{q_0\}, k, \delta_{\mathsf{n}})$ be the NTA derived from $\mathsf{Aut}(\mathcal{T}, C_0)$ by applying the zero-elimination procedure and the subset construction, as described in Section 3.1. For every symbol $c \in 2^{\mathcal{C}}$, let $\xi(c)$ be the substitution that maps every symbol $A \in c$ to t and every other symbol in $\mathcal{C} \setminus c$ to f.

Theorem 4.3. *Let δ_{zf} be the zero-closure of δ. For every state $q \in Q$ and every symbol $c \in 2^{\mathcal{C}}$, $\delta_{\mathsf{zf}}(q, c) = \Delta(q)\xi(c)$.*

We can observe that, for any set of states $\omega \in 2^Q$ and any symbol $c \in 2^{\mathcal{C}}$, a set $\Gamma' \subseteq [1:k] \times 2^Q$ models $\delta_{\mathsf{n}}(\omega, c)$ iff the set $\Gamma = \mathsf{group}^{-1}(\Gamma')$ models the formula $\Delta(\omega)\xi(c)$. A simple argument shows that, for any set of states $\omega \in 2^Q$, a set $\Gamma' \subseteq [1:k] \times 2^Q$ models the formula $\bigvee_{c\in\Sigma} \delta_{\mathsf{n}}(\omega, c)$ iff there exists a symbol c' such that the set $\{c'\} \cup \mathsf{group}^{-1}(\Gamma')$ models the formula $\hat{\Delta}(\omega)$.

This property of $\hat{\Delta}$ makes it possible to skip the explicit computation of the function δ_{n}. In order to calculate $\mathsf{Succ}_i(\omega)$, the reasoner can therefore proceed as follows:

1. Let $E = \emptyset$;
2. Find a set $Y \notin E$ such that $Y \models \hat{\Delta}(\omega)$;
3. If no such set exists, then $\mathsf{Succ}_i(\omega) = \bot$;
4. Let $\Omega_Y = \{\bigcup\{\omega \mid (j, \omega) \in Y\} \mid j \in [1:k]\}$;
5. If $\Omega_Y \cap \mathsf{Dead}_i = \emptyset$, then $\mathsf{Succ}_i(\omega) = Y$;
6. Otherwise, let $E = E \cup \{Y\}$ and return to step 2.

The second step can be delegated to a SAT solver — a software tool that, given a propositional formula, sequentially

produces all its models. While being worst-case exponential (they have to solve a NP-complete problem on a deterministic machine), SAT solvers are able to efficiently handle formulas containing thousands of variables.

Our formulas include a number of variables that is at most linear in the size of the inputs ($\mathcal{T}$, C_0), often much smaller. Hence, by using a SAT solver to handle this part of the problem, we can take advantage of decades of research and optimization and obtain a good level of performance.

5 Experimental Evaluation

The TREEHUG reasoner implements the decision procedure described in Section 4, with some simple optimizations.

The core of the reasoner is written in *Scala*, a language that integrates features of object-oriented and functional programming, and that interfaces seamlessly with Java. The work of calculating models for $\hat{\Delta}$ is delegated to the *MiniSAT* SAT solver [Eén and Sörensson, 2003]. MiniSAT is based on the DPLL algorithm [Davis *et al.*, 1962], with many enhancements to the handling of several commonly occurring patterns. The interface between these two components is a custom-built JNI library written in C++, that creates, manages and wraps MiniSAT instances. The resulting decoupling between the reasoner and the SAT solver makes it possible to switch to a different SAT library, and enables us to take advantage of low-level operations to speed up some tasks (e.g., feeding a large formula to the solver).

The system accepts TBoxes and concept expressions written in the Manchester syntax, the OWL 2 Functional Syntax and the KRSS syntax [1]. The values of $\hat{\Delta}$ are computed lazily (only when necessary), and directly in CNF. New instances of MiniSAT are spawned as necessary, fed with the clauses and polled for satisfying assignments, which get processed (as shown in the previous section) to calculate the successor sets Succ_i. The set of active states is monitored by a reference-counting garbage collection system, which gets rid of failed computation branches and decreases the amount of work done for each iteration.

5.1 Benchmarks

The reasoning algorithm proposed in this work has an unavoidably high worst-case complexity. The hope is, however, that the system performs well in realistic applications, in line with what holds for other DL reasoners [Baader *et al.*, 2007]. To verify this claim, however, we would have to test the performance of the system with a wide range of 'real-world' ontologies, similarly to the approach taken in [Gardiner *et al.*, 2006]. Within the limitations of the current prototype, though, this approach is not yet feasible. As a consequence we have opted for a set of synthetic benchmarks taken from the DL'98 System Comparison session, which were formulated to test systems which were in a similar stage of development as our prototype. We ran our tests on a 2.4 GHz Intel Core 2 Duo with 4 GB of RAM, running Mac OS X 10.6. As it is much faster than the one on which FaCT was tested in 1998, we have reduced the maximum running time for our prototype to 10 seconds instead of the original 100 seconds.

[1] http://www.w3.org/TR/owl2-overview/

Problem class	TREEHUG		FaCT 1.5	
	Unsat	Sat	Unsat	Sat
branch	11	11	6	4
d4	21	8	21	8
dum	21	21	21	21
grz	21	16	21	21
lin	6	21	21	21
path	21	21	8	6
ph	5	6	6	7
poly	21	21	21	21
t4p	21	21	21	21
Total	148	146	146	130

Table 2: Benchmark results from the T98-SAT track of the DL'98 System Comparison session. Each class comprises 21 satisfiable and 21 unsatisfiable problem instances. The numbers show how many problem instances were solved by our prototype and by FaCT 1.5, the version tested for DL'98.

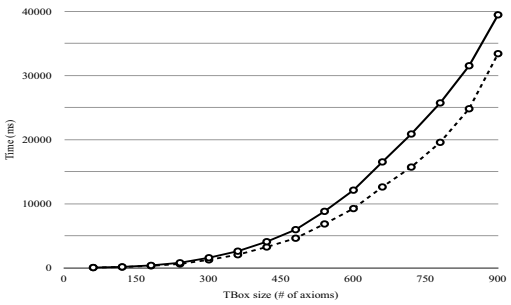

Figure 1: Synthetic benchmark results. The dashed line shows the unsatisfiable version.

The results show that the performance of our prototype is comparable to, and often better than, that of FaCT, whose development was arguably at a more advanced stage.

In addition to the DL'98 benchmarks, we have also developed a battery of synthetic benchmarks aimed at checking the scalability of TREEHUG. In these tests we used small "modules" of three axioms to build a TBox of an arbitrary size, and asked TREEHUG to check the satisfiability of a concept within a given TBox. Specifically, let $\mathcal{T}$ be a TBox containing assertions of the following form:

$$C_i \sqsubseteq (D_i \sqcap E_i) \sqcup \exists S.C_{i+1}$$
$$D_i \sqsubseteq \exists R.C_{i+1} \qquad E_i \sqsubseteq \forall R.\neg C_{i+1}$$

for every $i \in [1:k]$. Let $\mathcal{T}' = \mathcal{T} \cup \{C_{n+1} \sqsubseteq \bot\}$. We ask TREEHUG to check whether the concept C_1 is satisfiable in $\mathcal{T}$ (which is true) and in $\mathcal{T}'$ (which is false). The results we have obtained are shown in Figure 1.

6 Conclusions

In this work we have described a new automata-based technique for reasoning in expressive DLs. The resulting algorithm can decide the consistence of a concept in a TBox in exponential time, and is thus worst-case optimal. More remarkably, unlike previous automata based approaches, our algorithm seems well suited for implementation: we have

indeed developed a prototype reasoner, and its experimental evaluation reveals promising results.

Our approach can be extended to cover a wider range of DLs, beyond $\mathcal{ALC}^f$. In particular, it is possible to use this approach to handle any DL for which reasoning can be reduced to testing emptiness of looping tree automata: this includes languages up to $\mathcal{SHIQ}$, but excludes constructs such as regular roles. It does not seem feasible to adapt this algorithm for DLs with nominals, since they break the tree model property on which automata-theoretic approaches rely. On the other hand, we could extend our technique also to reason in the presence of ABoxes, e.g., by using an encoding of the ABox similar to the one presented in [Calvanese *et al.*, 2007]. The actual efficiency of the algorithm in this scenario remains to be evaluated.

Our prototype is still in the initial development phase, and its performance lags behind that of other established and highly optimized DL reasoners. Still, it gives a first positive answer to the long-standing open question of the practical applicability of techniques based on automata on infinite trees to reasoning in expressive DLs.

References

[Baader and Sattler, 2001] Franz Baader and Ulrike Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69(1):5–40, 2001.

[Baader and Tobies, 2001] Franz Baader and Stephan Tobies. The inverse method implements the automata approach for modal satisfiability. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, pages 92–106, 2001.

[Baader *et al.*, 2007] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2nd edition, 2007.

[Calvanese *et al.*, 2002] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. 2ATAs make DLs easy. In *Proc. of the 15th Int. Workshop on Description Logic (DL 2002)*, volume 53 of *CEUR,* `http://ceur-ws.org/`, pages 107–118, 2002.

[Calvanese *et al.*, 2007] Diego Calvanese, Thomas Eiter, and Magdalena Ortiz. Answering regular path queries in expressive description logics: An automata-theoretic approach. In *Proc. of the 22nd AAAI Conf. on Artificial Intelligence (AAAI 2007)*, pages 391–396, 2007.

[Carbotta, 2010] Domenico Carbotta. A practical automata-based technique for reasoning in expressive description logics. Master's thesis, Fakultät für Informatik, Technische Universität Wien, October 2010. Available at `http://www.emcl-study.eu/graduates.html`.

[Davis *et al.*, 1962] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.

[De Giacomo and Lenzerini, 1994] Giuseppe De Giacomo and Maurizio Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics. In *Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI'94)*, pages 205–212, 1994.

[Eén and Sörensson, 2003] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Proc. of the 6th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2003)*, volume 2919 of *LNCS*, pages 502–518. Springer, 2003.

[Gardiner *et al.*, 2006] Tom Gardiner, Ian Horrocks, and Dmitry Tsarkov. Automated benchmarking of description logic reasoners. In *Proc. of the 19th Int. Workshop on Description Logic (DL 2006)*, volume 189 of *CEUR,* `http://ceur-ws.org/`, 2006.

[Hustadt *et al.*, 2008] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Deciding expressive description logics in the framework of resolution. *Information and Computation*, 206(5):579–601, 2008.

[Motik *et al.*, 2009] Boris Motik, Rob Shearer, and Ian Horrocks. Hypertableau reasoning for description logics. *J. of Artificial Intelligence Research*, 36:165–228, 2009.

[Shearer *et al.*, 2008] Rob Shearer, Boris Motik, and Ian Horrocks. HermiT: A highly-efficient OWL reasoner. In *Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2008)*, volume 432 of *CEUR,* `http://ceur-ws.org/`, 2008.

[Sirin and Parsia, 2006] Evren Sirin and Bijan Parsia. Pellet system description. In *Proc. of the 19th Int. Workshop on Description Logic (DL 2006)*, volume 189 of *CEUR,* `http://ceur-ws.org/`, 2006.

[Tobies, 2001] Stephan Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, LuFG Theoretical Computer Science, RWTH-Aachen, Germany, 2001.

[Tsarkov and Horrocks, 2006] Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In *Proc. of the 3rd Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, pages 292–297, 2006.

[Vardi and Wilke, 2007] Moshe Vardi and Thomas Wilke. Automata: From logics to algorithms. In *Proc. of the Automata and Logic Workshop (WAL 2007)*, pages 645–753, December 2007.

[Vardi, 1997] Moshe Y. Vardi. Why is modal logic so robustly decidable. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 31, pages 149–184. American Mathematical Society, 1997.

[Vardi, 1998] Moshe Y. Vardi. Reasoning about the past with two-way automata. In *Proc. of the 25th Int. Coll. on Automata, Languages and Programming (ICALP'98)*, volume 1443 of *LNCS*, pages 628–641. Springer, 1998.

[Voronkov, 2001] Andrei Voronkov. How to optimize proof-search in modal logics: New methods of proving redundancy criteria for sequent calculi. *ACM Trans. on Computational Logic*, 2(2):182–215, 2001.