A Translation of Looping Alternating Automata into Description Logics

Jan Hladik* and Ulrike Sattler

Technische Universität Dresden, {hladik,sattler}@inf.tu-dresden.de

Abstract. We present a translation of (one-way and two-way) alternating automata into description logics, thus reducing the emptiness problem for alternating automata to satisfiability of the target description logic. The latter problem can then be decided using highly optimised, tableau-based description logic reasoners. The translation is a step towards the understanding of the relationship between automata- and tableau-based decision procedures for description and modal logics. Moreover, it yields some by-products: (i) a program deciding the emptiness problem for alternating automata and thus the satisfiability problem for logics with automata-based decision procedures; and (ii) tight complexity bounds for the target description logic.

1 Introduction

In the field of modal and description logics, *automata*- and *tableau*-based satisfiability algorithms are two widely used approaches with complementary advantages and disadvantages. An automata-based algorithm constructs, for a concept C (or a modal logic formula φ), an automaton $\mathcal{A}_C$ accepting all (abstractions of) models of C, see, e.g., [VW86, SE89, Var98, CGL99]. Thus satisfiability of C can be decided by testing the emptiness of the language accepted by $\mathcal{A}_C$. For a variety of logics, this is an elegant approach: if the translation uses well-known target automata for which the complexity of testing emptiness has already been established, one only needs to describe the translation and prove its correctness (plus possibly also define an appropriate abstraction of models). Moreover, especially when using *alternating* automata, the translation is rather straightforward. For many logics, this approach thus yields elegant ExpTime upper complexity bounds since either the translation is polynomial and the emptiness test is exponential or vice versa. However, implementations of automata-based satisfiability solvers for description logics can be said to be in their infancy, even if the first results are promising [PSV02].

A tableau-based algorithm tries to construct (an abstraction of) a model of an input concept C by breaking down C syntactically and thereby inducing constraints on this model, see, e.g., [HM92, BS01]. It either terminates with (an abstraction of) a model of C or with obvious inconsistencies. For a variety of logics, this approach is amenable to optimisations and behaves surprisingly

* The author is supported by the DFG, Project No. GR 1324/3-3.

well in practise, even for ExpTime-hard logics [Hor98, HM01]. However, natural tableau-based algorithms are non-deterministic and thus not optimal for ExpTime logics.

In short, the automata approach is well-suited to devise upper complexity bounds, whereas the tableau approach is well-suited for implementations. As a consequence, for many logics, in the absence of an approach enjoying the advantages of both, tableau- *and* automata-based algorithms were hand-crafted, which constitutes a possibly unnecessary overhead. In the absence of such a unifying approach, a translation of automata-based algorithms into tableau-based ones is highly desirable, thus reducing the overhead by mechanising the development of an implementable algorithm. As a first step towards this mechanisation, we present translations from looping one- and two-way alternating automata to description logics that are contained in $\mathcal{SHIQ}$ [HST99]. Thus, given an automata-based algorithm for a logic using alternating automata, we can transform it into a tableau-based one as follows: first, translate a concept C into an alternating automaton $\mathcal{A}_C$, then translate $\mathcal{A}_C$ into a description logic TBox $\mathcal{T}_C$, and decide satisfiability of the concept corresponding to $\mathcal{A}_C$ w.r.t. $\mathcal{T}_C$ using a tableau-based satisfiability solver available for $\mathcal{SHIQ}$ such as FaCT or RACER [Hor98, HM01]. This yields a satisfiability solver for a variety of logics for which only automata-based algorithms were known so far. We have implemented this translation for looping two-way alternating automata and report first results in Section 6.

In [KV98a], a translation of (one-way) weak alternating automata into the alternation-free μ-calculus is presented, which proves that both formalisms are of the same expressiveness and has some similarity to our translation in Section 4. However, as there is no system deciding satisfiability of μ-calculus formulae, this does not yield an implementation for weak alternating automata.

Summing up, besides a deeper understanding of the relationship between automata and tableaux, the translation presented in this paper yields (i) an implementation of the emptiness test for alternating automata and thus for the satisfiability of various (description) logics; (ii) an ExpTime-hardness result for the logic used in the translation; and (iii) a new method of generating “hard” problems for FaCT and RACER.

2 Description Logics and Tableau Algorithms

Description logics (DLs) are a family of knowledge representation formalisms designed for the representation of terminological knowledge and ontologies; for an introduction to DLs, see [BCM$^+$03]. They are closely related to modal logics [Sch91, GL94]; for example, the well-known DL $\mathcal{ALC}$ [SS91] is a notational variant of the multi modal logic $\mathbf{K}_n$.

Here, we use the rather inexpressive DL $\mathcal{ELU}_f$ together with expressive *TBoxes*, a DL-specific means of expressivity closely related to the universal modality in modal logics [Sch91]. The central entities of DLs are *concepts*, which can be viewed as formulae in one free variable.

Definition 1. Let $\mathsf{N_C}$ be a set of concept names and $\mathsf{N_F}$ a set of feature names. The set of $\mathcal{ELU}_f$ *concepts* over $\mathsf{N_C}$ and $\mathsf{N_F}$ is inductively defined as follows:

- $\top$, $\bot$, and each concept name $C \in \mathsf{N_C}$ is an $\mathcal{ELU}_f$-concept;
- if C and D are concepts, then $C \sqcup D$ and $C \sqcap D$ are concepts;
- if C is a concept and $f \in \mathsf{N_F}$ is a feature name, then $\exists f.C$ is a concept.

A *general concept inclusion axiom (GCI)* is of the form $C \sqsubseteq D$ (read "C is subsumed by D"), for concepts C and D. A *TBox* is a finite set of GCIs.

An *interpretation* $\mathcal{I}$ is a pair $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a set of individuals and $\cdot^{\mathcal{I}}$ is a function assigning, to every concept name C, a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$ and, to every feature f, a partial function $f^{\mathcal{I}} : \Delta^{\mathcal{I}} \to \Delta^{\mathcal{I}}$. We use $(d, e) \in f^{\mathcal{I}}$ for $f^{\mathcal{I}}(d) = e$. The function $\cdot^{\mathcal{I}}$ is inductively extended to complex concepts as follows:
$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$, $\bot^{\mathcal{I}} = \emptyset$, $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$, $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$,
$(\exists f.C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid \exists e : (d, e) \in f^{\mathcal{I}} \wedge e \in C^{\mathcal{I}}\}$.

An interpretation $\mathcal{I}$ *satisfies* a GCI $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$; $\mathcal{I}$ is a *model* of a TBox $\mathcal{T}$ if it satisfies all GCIs in $\mathcal{T}$; $\mathcal{I}$ is a *model* of a concept C if $C^{\mathcal{I}} \neq \emptyset$; $\mathcal{I}$ is a model of C *with respect to* $\mathcal{T}$ if $\mathcal{I}$ is a common model of C and $\mathcal{T}$; and a concept C is *satisfiable [w.r.t. $\mathcal{T}$]* if there is a model for C [and $\mathcal{T}$]. A concept C *is subsumed by* a concept D w.r.t. $\mathcal{T}$ (written $C \sqsubseteq_{\mathcal{T}} D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$.

$\mathcal{ELU}_f$ is restricted in several aspects: it does not provide negation, and it only provides existential value restrictions ($\exists f.C$), whereas standard DLs also provide universal restrictions ($\forall f.C$). Equally important, $\mathcal{ELU}_f$ only provides *features* (i.e., functional binary relations) whereas most DLs provide *roles* (i.e., arbitrary binary relations). However, $\mathcal{ELU}_f$ comes with GCIs, a very expressive means, as we will see soon.

In DLs with conjunction and negation $\neg$, subsumption can be linearly reduced to satisfiability: $C \sqsubseteq_{\mathcal{T}} D$ iff $C \sqcap \neg D$ is not satisfiable w.r.t. $\mathcal{T}$. For $\mathcal{ELU}_f$, in the absence of negation, this reduction is slightly more involved and requires a new concept symbol $\hat{D}$ to replace $\neg D$: $C \sqsubseteq_{\mathcal{T}} D$ iff $C \sqcap \hat{D}$ is unsatisfiable w.r.t. $\mathcal{T} \cup \{D \sqcap \hat{D} \sqsubseteq \bot\}$. Thus, we can use GCIs to express disjointness of D and $\hat{D}$ (as no individual can belong to both D and $\hat{D}$), and disjointness suffices to reduce subsumption to satisfiability. In the following, we will therefore concentrate on satisfiability problems.

For several expressive DLs, there exist efficient tableau-based implementations that decide satisfiability (and thus subsumption) of concepts w.r.t. a TBox [HM01, Hor98, BS01]. Intuitively, to decide the satisfiability of a concept C, a tableau algorithm starts with an instance x of C (here written $x : C$), and then recursively breaks down C syntactically, thus inferring new constraints on the model of C to be built. For example, if $y : D \sqcap E$ has already been inferred, it adds $y : D$ and $y : E$. For $y : \exists f.F$, it generates a new node, say z, and adds $(y, z) : f$ and $z : F$. Finally, it adds, for each GCI $C_i \sqsubseteq D_i$ in the TBox, the constraint $y : (\neg C_i \sqcup D_i)$ for each individual y of the model to be built.

Now, for logics with disjunctions, various tableau algorithms non-deterministically choose whether to add $y : D$ or $y : E$ for $y : D \sqcup E$. Getting rid of this non-determinism in a way that is more efficient than naive backtracking proves to be hard work for many logics [DM00]. This is one reason why for ExpTime logics, most tableau algorithms are not optimal. For example, the $\mathcal{SHIQ}$ tableau algorithm implemented in FaCT is 2-NExpTime instead of ExpTime [HST99]. Despite this sub-optimality, tableau algorithms allow for a set of well-known efficient optimisations, so that they perform much better in practise than their worst-case complexity suggests. FaCT [Hor98] and RACER [HM01] are examples of such efficient implementations.

Since we are talking about decision procedures, *termination* is an important issue. Even though tableau algorithms for many inexpressive DLs terminate "automatically", this is not the case for more expressive ones. For example, consider the algorithm sketched above on the input concept A and TBox $\{A \sqsubseteq \exists f.A\}$: it would create an infinite f-chain of instances of A. Thus, the tableau algorithm has to be stopped at a certain point; intuitively, at the point when the concepts remaining to be processed are just a repetition of concepts which were already processed. This mechanism, called *blocking*, often makes the correctness proof of the algorithm very complicated. Moreover, it can be difficult to choose an *efficient* blocking condition [HS02, Hla02].

In summary, tableau algorithms

- $\oplus$ are used in state-of-the-art implementations, and many well-understood optimisations are available,
- $\oplus$ have proven to perform much better for realistic concepts than their worst-case complexity suggests;
- $\ominus$ require special techniques to ensure termination (e.g. blocking) and get rid of non-determinism,
- $\ominus$ are often not worst-case optimal for deterministic complexity classes.

3 Alternating Automata

For many description and modal logics, the satisfiability of a concept C w.r.t. a TBox $\mathcal{T}$ can be decided by defining an automaton $\mathcal{A}$ which accepts exactly the (abstractions of) models of C and $\mathcal{T}$. Thus, the satisfiability problem is reduced to the emptiness problem of automata. Examples utilising automata can be found in [VW86, SE89, Var98, CGL99, LS01]. In most cases, abstractions of models are finite or infinite trees—depending on the logic. Thus the target automata are automata on finite or infinite trees. Moreover, we can use deterministic, non-deterministic, or, as a generalisation, *alternating* automata, where the latter class of automata allows for rather elegant translations of many logics. In many cases, the emptiness test for non-alternating automata is polynomial, whereas the translation yields an automaton of size exponential in the input concept (and TBox). In contrast, the translation into an alternating automaton usually yields an automaton of polynomial size (see, for example, [Var98, CGL99])—however, testing emptiness of alternating automata is ExpTime-complete [KV98b, Var98]. Thus, this approach yields worst-case optimal algorithms for ExpTime-complete logics. Before discussing the automata-based approach in more detail, we first define alternating automata on infinite trees.

Definition 2. Let K be a natural number. We define $[K] := \{1, \ldots, K\}$ and $[K]_0 := [K] \cup \{0\}$. A *K-ary infinite tree over an alphabet Σ* is a total mapping $\tau : [K]^* \to \Sigma$.

Here, the empty word ε denotes the root of the tree and, for $\ell \in [K]^*$ and $k \in [K]$, $\ell \cdot k$ denotes the k-th successor of ℓ; $\ell \cdot 0$ is defined as ℓ.

Alternating automata generalise nondeterministic automata by allowing not only several alternative successor states, i.e. a disjunction of alternatives, but also a conjunction or a combination of both. For example, the transition $\delta(a, q_1) = (1, q_3) \wedge ((1, q_2) \vee (3, q_1))$ is to be read as follows: if the automaton processes a node ℓ, is in state q_1, and reads the letter a, then it has to send one copy of the automaton in state q_3 to the first successor of ℓ and either another copy in state q_2 to the first successor of ℓ or a copy in state q_1 to the third one.

Definition 3. The set of *positive Boolean formulae* over a set V, $\mathcal{B}^+(V)$, consists of formulae built from $V \cup \{\mathit{true}, \mathit{false}\}$ using the binary operators $\wedge$ and $\vee$. A set $R \subseteq V$ *satisfies* a formula $\varphi \in \mathcal{B}^+(V)$ iff assigning true to all elements of R and false to all elements of $V \setminus R$ yields a formula that evaluates to true.

An *alternating automaton* $\mathcal{A}$ is a tuple (Q, Σ, q_0, δ), where $Q = \{q_0, \ldots, q_{\hat{q}}\}$ is a set of states, $\Sigma = \{\sigma_0, \ldots, \sigma_{\hat{\sigma}}\}$ is the input alphabet, q_0 is the initial state, and $\delta : Q \times \Sigma \to \mathcal{B}^+([K]_0 \times Q)$ is the transition relation.

The *width* of an automaton $w(\mathcal{A})$ is the number of literals that can appear on the right-hand side of a transition, i.e., $w(\mathcal{A}) := (\hat{q} + 1) \cdot (K + 1)$. A *run* ρ of $\mathcal{A}$ on a tree τ is a $w(\mathcal{A})$-ary infinite tree over $([K]^* \times Q) \cup \{\uparrow\}$ which satisfies the following conditions:

1. $\rho(\varepsilon) = (\varepsilon, q_0)$ and
2. for each node r with $\rho(r) = (t, q) \neq \uparrow$ and $\delta(q, \tau(t)) = \varphi$, there is a set $S = \{(t_1, q_1), \ldots, (t_n, q_n)\} \subseteq [K]_0 \times Q$ such that
   (a) S satisfies φ and,
   (b) for all $1 \leq i \leq n$, $\rho(r \cdot i) = (t \cdot t_i, q_i)$.

An automaton $\mathcal{A}$ *accepts* an input tree τ if there exists a run of $\mathcal{A}$ on τ. The *language accepted by $\mathcal{A}$*, $L(\mathcal{A})$, is the set of all trees accepted by $\mathcal{A}$.

Some remarks are in order: firstly, we have defined *looping* automata, i.e., there is no acceptance condition and each run is accepting. Secondly, a run labels each node r either with a pair (t, q) or with $\uparrow$, where the latter indicates that $\rho(r)$ is not important for the acceptance of the input tree.

Example 4. In Figure 1, we see part of a run ρ of an alternating automaton $\mathcal{A}$ on a tree τ. We only present those nodes relevant for the run, i.e., nodes r with $\rho(r) \neq \uparrow$. If the transition relation is $\delta(q_1, a) = ((0, q_4) \wedge (2, q_2)) \vee (3, q_3)$, $\delta(q_2, b) = (0, q_1) \wedge (3, q_4)$, and $\delta(q_4, a) = \delta(q_1, b) = \delta(q_4, c) = \mathit{true}$, all other nodes of ρ can be labelled with $\uparrow$ and all other nodes of τ can be labelled arbitrarily and τ is accepted.

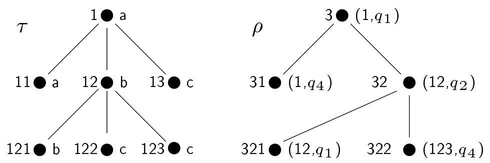

Fig. 1. Example of a tree and run.

Please observe that there is no one-to-one correspondence between the nodes of τ and ρ: both $\rho(3)$ and $\rho(31)$ refer to node 1, but none refers to node 13. Moreover, the ordering of successors is important in τ, but not in ρ: the definition of a run only requires the existence of certain successors.

A question we would like to answer next is why one would use such a seemingly complicated kind of automata. Firstly, standard abstraction techniques such as *unravelling* [Tho92] yield *infinite* tree abstractions of models. Using automata on infinite trees , we can freely work with these standard, infinite abstractions. This is a clear advantage for logics lacking the finite model property, or where it would be tedious to invent *finite* abstractions. In tableau algorithms, we had to work with *finite* representations of infinite abstractions to ensure termination, using blocking. In contrast, for the class of automata defined above, termination is not an issue since input trees and runs are, by definition, infinite structures.

Secondly, using non-deterministic automata, non-determinism due to disjunctions can be translated into non-deterministic transitions. For alternating automata, we can also translate "universal" quantification—e.g. due to conjunction—into the transition function. For example, when designing an alternating automaton for an $\mathcal{ELU}_f$-concept C with features from $f_1, \ldots, f_k$, one would use a state q_D for each sub-concept D of C. Nodes of input trees are labelled with sets of concept names and stand for individuals of a model. Examples of the transition function are $\delta(q_{D\sqcap E}, \sigma) = (0, q_D) \wedge (0, q_E)$, $\delta(q_{D\sqcup E}, \sigma) = (0, q_D) \vee (0, q_E)$, $\delta(q_{\exists f_j.E}, \sigma) = (j, q_E)$, and $\delta(q_X, \sigma) = \mathit{true}$ if $X \in \sigma$. Thus the description logics translates in a natural way into an automaton.

The main drawback of automata lies in the fact that their complexity is exponential not only in the worst case, but in every case: either the automaton $\mathcal{A}_\varphi$ is exponential in φ or, in the case of alternating automata, is polynomial but is translated into a non-deterministic automaton $\mathcal{A}'_\varphi$ of exponential size to decide its emptiness [KV98b, Var98]. Therefore, a naive implementation is doomed to failure and, to the best of our knowledge, only first yet promising steps towards implementing an automata-based satisfiability solver have been made [PSV02].

In summary, automata-based approaches to satisfiability

- ⊕ often allow for a very elegant and natural translation of a logic,
- ⊕ provide ExpTime upper complexity bounds and are thus optimal for ExpTime-hard logics,
- ⊕ handle infinite structures and non-determinism implicitly,
- ⊖ have only recently been implemented.

4 Translating Alternating Automata into $\mathcal{ELU}_f$

In this section, we describe how to translate an alternating automaton $\mathcal{A}$ into a TBox $\mathit{tr}(\mathcal{A})$ and a concept Q_0 such that $L(\mathcal{A})$ is non-empty iff Q_0 is satisfiable w.r.t. $\mathit{tr}(\mathcal{A})$. Intuitively, we translate the transition function δ into GCIs $\mathit{tr}(\mathcal{A})$ whose models correspond to runs of $\mathcal{A}$. To this purpose, we use a feature f_k for the k-th successor of a node in the input tree, i.e., for each $k \in [K]$.

Definition 5. Let $\mathcal{A} = (Q, \Sigma, q_0, \delta)$ be an alternating automaton with $Q = \{q_0, \ldots, q_{\hat{q}}\}$ and $\Sigma = \{\sigma_0, \ldots, \sigma_{\hat{\sigma}}\}$. The translation of $\mathcal{A}$ into an $\mathcal{ELU}_f$ TBox $\mathit{tr}(\mathcal{A})$ is defined as follows: for each $q_i \in Q$ we use a concept name Q_i, for each $\sigma_j \in \Sigma$, we use a concept name A_j, and set

$$\begin{array}{lll}
\mathit{tr}(\mathcal{A}) & := \{\mathsf{G}\top, \mathsf{G}\bot\} \cup \bigcup_{q \in Q, \sigma \in \Sigma} \mathit{tr}(\delta(q,\sigma)), \text{ where} \\
\mathsf{G}\top & := \quad \top \sqsubseteq A_1 \sqcup A_2 \sqcup \ldots \sqcup A_{\hat{\sigma}}, \\
\mathsf{G}\bot & := \quad \bigsqcup_{0 \le i < j \le \hat{\sigma}} (A_i \sqcap A_j) \sqsubseteq \bot, \\
\mathit{tr}(\delta(q,\sigma)) & := \quad \mathit{tr}(q) \sqcap \mathit{tr}(\sigma) \sqsubseteq \mathit{tr}(\varphi) \quad \text{if } \delta(q,\sigma) = \varphi,
\end{array}$$

and the translation of φ, q, and σ is defined as follows:

$$\begin{array}{llll}
\mathit{tr}(q_i) & := Q_i \quad \text{for } q_i \in Q, & \mathit{tr}(\sigma_i) & := A_i \quad \text{for } \sigma_i \in \Sigma, \\
\mathit{tr}(\alpha \wedge \beta) & := \mathit{tr}(\alpha) \sqcap \mathit{tr}(\beta), & \mathit{tr}(\alpha \vee \beta) & := \mathit{tr}(\alpha) \sqcup \mathit{tr}(\beta), \\
\mathit{tr}(\mathit{true}) & := \top, & \mathit{tr}(\mathit{false}) & := \bot, \\
\mathit{tr}(0,q) & := \mathit{tr}(q), & \mathit{tr}(k,q) & := \exists f_k.\mathit{tr}(q) \quad \text{for } k \neq 0.
\end{array}$$

We will see that tr ensures that each model $\mathcal{I}$ of $\mathit{tr}(\mathcal{A})$ corresponds to a run ρ on some tree τ. First, a node r in ρ is labelled with a node t in τ which, in turn, is labelled with exactly one $\sigma \in \Sigma$. Thus each r in ρ is associated with one $\sigma \in \Sigma$. To express this fact in $\mathit{tr}(\mathcal{A})$, we use the extra GCIs $\mathsf{G}\top$ and $\mathsf{G}\bot$: they guarantee that every individual of $\mathcal{I}$ is an instance of exactly one $\mathit{tr}(\sigma_i)$.[1]

Next, it will turn out to be useful to have the inverse tr^{-1} of tr, which is possible since tr is "almost" injective: the only ambiguity concerns q and $(0,q)$ since they are both mapped to Q by tr. However, this ambiguity can easily be resolved by agreeing to set $\mathit{tr}^{-1}(Q)$ to $(0,q)$ if Q appears on the right hand side of a GCI and to q otherwise.

Example 6. Figure 2 shows an example for the translation of a tree and run into an interpretation for a single node. The automaton is in state q_1 and reads node 1 which is labelled with a. For $\delta(a, q_1) = (1, q_1) \vee ((1, q_2) \wedge (0, q_4))$, the transition function yields the GCI $A \sqcap Q_1 \sqsubseteq \exists f_1.Q_1 \sqcup ((\exists f_1.Q_2) \sqcap Q_4)$. In our example, the transition function is satisfied via the second disjunct, and thus the individual 1 in the model $\mathcal{I}$ is an instance of both Q_1 and Q_4. We would like to point out that a model $\mathcal{I}$ of $\mathit{tr}(\mathcal{A})$ might have a structure different from

[1] Since each node is labelled with exactly one alphabet symbol, it is also possible to translate the alphabet symbols using a binary coding mechanism which requires only $\log_2 n$ concept names for n alphabet symbols. However, this would not reduce the number of GCIs in the TBox, which would still be exponential in n. Hence, we stick with the linear translation, which is also easier to read.

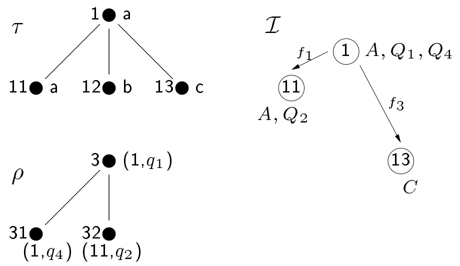

Fig. 2. Translation of a tree and run into a model.

– an input tree τ since nodes in $\mathcal{I}$ might have no f_i-successor for some i and $\mathcal{I}$ might not be a tree.
– a run ρ since different nodes of ρ that refer to the same node in τ are represented by the same individual in $\mathcal{I}$: intuitively, we label an individual $i \in \Delta^{\mathcal{I}}$ with the concept A_i of $\tau(i)$ and "collect" all Q_i concepts from nodes in ρ that refer to $\tau(i)$. Thus, some individuals are instances of several Q_i concepts, like 1, while others are instances of none, like 13.

Lemma 7. *The language accepted by an alternating automaton* $\mathcal{A} = (Q, \Sigma, q_0, \delta)$ *is non-empty iff* $\mathit{tr}(q_0)$ *its satisfiable w.r.t.* $\mathit{tr}(\mathcal{A})$.

Proof. We start with the "only-if"-direction. For this, we fix some notation: we use L to denote the set of concepts appearing in the right hand side of GCIs in $\mathit{tr}(\mathcal{A})$, $L := \{\exists f_i.Q_j \mid i \in [K], j \in [\hat{q}]\} \cup \{Q_j \mid j \in [\hat{q}]\}$, and we use $\mathcal{B}^+(L)$ for the set of positive Boolean concepts analogous to Section 3, with the symbols $\wedge, \vee, \mathit{true}$, and false replaced with $\sqcap, \sqcup, \top$, and $\bot$, respectively.

Let $\mathcal{T} := \mathit{tr}(\mathcal{A})$, $Q_0 = \mathit{tr}(q_0)$, and let $\tau \in L(\mathcal{A})$ with ρ a successful run of $\mathcal{A}$ on τ. We construct a model $\mathcal{I}$ of Q_0 w.r.t. $\mathcal{T}$ as follows:

$\Delta^{\mathcal{I}} := [K]^*$,
$f_k^{\mathcal{I}} := \{(\ell, \ell \cdot k) \mid \ell \in [K]^*\}$, for every $k \in [K]$,
$A_i^{\mathcal{I}} := \{t \mid \tau(t) = \sigma_i\}$, for every $\sigma_i \in \Sigma$,
$Q_i^{\mathcal{I}} := \{t \mid$ there is an r in ρ with $\rho(r) = (t, q_i)\}$, for every $q_i \in Q$,.

To prove that $\mathcal{I}$ is a model of Q_0 w.r.t. $\mathcal{T}$, we show that (i) $Q_0^{\mathcal{I}} \neq \emptyset$ and (ii) each individual t of $\mathcal{I}$ satisfies each GCI in $\mathcal{T}$.

Now (i) holds by definition of $\mathcal{I}$ since the $\rho(\varepsilon) = (\varepsilon, q_0)$ and thus $\varepsilon \in Q_0^{\mathcal{I}}$. For (ii), we distinguish three classes of GCIs in $\mathcal{T}$:

1. GCIs $\mathsf{G}\top$ and $\mathsf{G}\bot$,
2. GCIs of the form $Q \sqcap A \sqsubseteq \bot$ resulting from the translation of transitions $\delta(q, \sigma) = \mathit{false}$, and
3. GCIs $Q \sqcap A \sqsubseteq C$, for some concept $C \in \mathcal{B}^+(L) \setminus \{\bot\}$.

For the first class, $\mathcal{I}$ satisfies $\mathsf{G}\top$ and $\mathsf{G}\bot$ by definition since every node t in τ is labelled with exactly one letter σ.

For the remainder, consider a GCI $Q_i \sqcap A_j \sqsubseteq C$ in $\mathcal{T}$ with preimage $\delta(q_i, \sigma_j) = \varphi$ and some $t \in Q_i^{\mathcal{I}} \cap A_j^{\mathcal{I}}$. By definition of $\mathcal{I}$, there exists a node r with $\rho(r) = (t, q_i)$ and $\tau(t) = \sigma_j$. For the second class, if $\varphi = \mathit{false}$, then $C = \bot$, and the existence of t is a contradiction to ρ being a run.

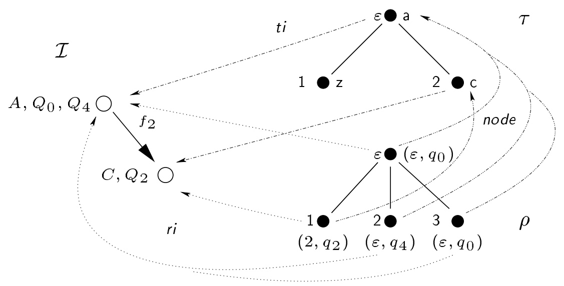

Fig. 3. Translation of a model into a tree and run.

For the third class, the definition of a run implies the existence of a set $S = \{i_1, \ldots, i_m\}$ and functions t, c, and s such that

– $\{(t(i), q_{c(i)}) \mid i \in S\}$ satisfies φ and
– there are successors $r \cdot s(i_1), \ldots, r \cdot s(i_s)$ of r which are labelled with the corresponding pairs, i.e. $\rho(r \cdot s(j)) = (t \cdot t(j), q_{c(j)})$ for all $j \in S$.

By construction of $\mathcal{I}$, there exist $f_{t(j)}$-successors u_j of t with $u_j \in Q^{\mathcal{I}}_{c(j)} \sqcap A^{\mathcal{I}}$ for $A := \mathit{tr}(\tau(t \cdot t(j)))$. This ensures that $t \in C^{\mathcal{I}}$, which concludes the proof of the "only-if"-direction.

For the "if"-direction, we will show how to construct a tree τ and a run ρ of $\mathcal{A}$ on τ from a model $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of Q_0 w.r.t. $\mathcal{T}$. We define the auxiliary functions

– ti and ri, which map nodes in τ and ρ, resp., to individuals of $\Delta^{\mathcal{I}}$,
– node and state, which map a node of ρ to the node and state component of its label, i.e. if $\rho(r) = (t, q)$, then $\mathit{node}(r) = t$ and $\mathit{state}(r) = q$, and
– $\mathit{letter} : \Delta^{\mathcal{I}} \to \{A_i \mid 1 \le i \le \hat{\sigma}\}$ and $\mathit{states} : \Delta^{\mathcal{I}} \to 2^{\{Q_i \mid 1 \le i \le \hat{q}\}}$, assigning, to each individual, the unique concept A_x and the set of Q_i concepts it is an instance of (letter is well-defined since $\mathcal{I}$ is a model of $\mathsf{G}\top$ and $\mathsf{G}\bot$).

Example 8. In Figure 3, we show a model $\mathcal{I}$ together with a tree τ, a run ρ, and some of the auxiliary functions. The tree node 1 is assumed to be related to a dummy individual d_u and its letter $z = \mathit{letter}(d_u)$, and only some nodes of ρ are presented. In the run ρ, the nodes ε, 2, and 3 are labelled with the same tree node ε since $\mathit{ri}(\varepsilon)$ is an instance of Q_0 and Q_4, and we might need nodes 2 and 3 for a run involving transitions $\delta(a, q_0) = (0, q_4) \wedge (0, q_0) \wedge \ldots$

Intuitively, τ is an an unravelling of $\mathcal{I}$, and ρ is an unravelling with "duplicate" successors. More precisely, τ and ρ are defined as follows. We begin the construction of τ with an individual $d_\varepsilon \in Q_0^{\mathcal{I}}$. Such an individual exists since $\mathcal{I}$ is a model of Q_0 w.r.t. $\mathcal{T}$. Moreover, we fix some "dummy" individual $d_u \in \Delta^{\mathcal{I}}$. We define $\tau(\varepsilon) := \mathit{tr}^{-1}(\mathit{letter}(d_\varepsilon))$, and $\mathit{ti}(\varepsilon) := d_\varepsilon$. Then, for each t such that $\mathit{ti}(t)$ is already defined and for each $k \in [K]$, we do the following:

– if $\mathit{ti}(t)$ has an f_k-successor d_k, define $\mathit{ti}(t \cdot k) := d_k$ and $\tau(t \cdot k) := \mathit{tr}^{-1}(\mathit{letter}(d_k))$ (this is well-defined since f_k is functional);
– otherwise, define $\mathit{ti}(t \cdot k) := d_u$ and $\tau(t \cdot k) := \mathit{tr}^{-1}(\mathit{letter}(d_u))$ for the "dummy" individual d_u.

Having defined τ, we now define a run ρ of $\mathcal{A}$ on τ as follows. Firstly, set $\rho(\varepsilon) := (\varepsilon, q_0)$ and $\mathit{ri}(\varepsilon) := d_\varepsilon$. Secondly, if $\mathit{ri}(r) = d$ is already fixed, we define $d_0 := d$ and d_i as the f_i-successor of d, if there exists one, and $d_i := d_u$ otherwise. Then we fix, for every d_i and every $q_{i_j} \in \mathit{states}(d_i)$, a different successor $r \cdot i_j$ of r with $\rho(r \cdot i_j) = (\mathit{node}(r) \cdot i, q_{i_j})$ and $\mathit{ri}(r \cdot i_j) = d_i$. Finally, if $\rho(r \cdot k)$ is not fixed through the previous step, we set $\rho(r') = \uparrow$ for each node r' in the sub-tree below $r \cdot k$ including $r \cdot k$. Thus, for every concept Q_i that d or a successor of d is an instance of, there is a successor of d in ρ labelled with q_i and the corresponding node in τ.

To prove that ρ is a run on τ, we first prove that the ri and ti functions are defined properly in the following sense:

Claim: For all nodes r in ρ, if $\rho(r) \neq \uparrow$, then $\mathit{ri}(r) = \mathit{ti}(\mathit{node}(r))$.

Proof of the claim. The proof is by induction on the depth of nodes in ρ. For the root node ε of ρ, the claim holds by definition. Now let r be a node of ρ with $\rho(r) \neq \uparrow$ for which the claim holds. Let $\mathit{ri}(r) = d$ and consider a successor $r \cdot k$ of r. If $\rho(r \cdot k) \neq \uparrow$, then there are i, j such that d has an f_i-successor $d_i \in Q_j^{\mathcal{I}}$, or $\mathit{ri}(r \cdot k) = d$, which means that $d \in Q_j^{\mathcal{I}}$ and $i = 0$. Then $\mathit{node}(r \cdot k) = \mathit{node}(r) \cdot i$ and $\mathit{ri}(r \cdot k) = d_i$ by definition of ρ. By induction, $\mathit{ti}(\mathit{node}(r)) = d$, and thus $\mathit{ti}(\mathit{node}(r) \cdot i) = d_i$ by definition of τ, which concludes the proof of the claim.

Now we can prove that ρ is a run on τ, see Definition 3. Property 1 of runs holds by definition of ρ. For Property 2, consider r with $\rho(r) = (t, q)$. Hence there is $d \in \Delta^{\mathcal{I}}$ with $d = \mathit{ri}(r)$ and $d \in \mathit{tr}(q)^{\mathcal{I}}$. Set $Q = \mathit{tr}(q)$. Moreover, by definition, for $\mathit{letter}(\mathit{ti}(t)) = A$, we have $\tau(t) = \mathit{tr}^{-1}(A)$. The claim yields $\mathit{ti}(t) = \mathit{ri}(r) = d$, which implies $A = \mathit{letter}(d)$ by construction. Summing up, we have $d \in A^{\mathcal{I}} \cap Q^{\mathcal{I}}$.

Since $\mathcal{I}$ is a model of $\mathit{tr}(\mathcal{A})$, $d \in C^{\mathcal{I}}$ for $A \sqcap Q \sqsubseteq C$ the translation of $\delta(\mathit{tr}^{-1}(A), q) = \varphi$. As d is an instance of C, there exists an $N = \{n_1, \ldots, n_\ell\} \subseteq L$ which "satisfies" C. For every $n_i, 1 \leq i \leq \ell$, we define a $p_i \in [K]_0 \times Q$ as follows:

- if $n_i = \exists f_k.Q$ for some f_k, Q, then d has an f_k-successor $d_k \in Q^{\mathcal{I}}$. By construction of τ, t has a k-successor $t \cdot k$, and r has a successor $r \cdot k'$ with $\rho(r') = (t \cdot k, \mathit{tr}^{-1}(Q))$. We set $p_i := (k, \mathit{tr}^{-1}(Q))$;
- if $n_i = Q$ for some Q, then $d \in Q^{\mathcal{I}}$. By construction, r has a successor $r \cdot j$ with $\rho(r \cdot j) = (t, q)$. We set $p_i := (0, q)$.

It can easily be seen that set $S := \{p_i \mid 1 \leq i \leq \ell\}$ satisfies $\varphi := \mathit{tr}^{-1}(C)$, and therefore Property 2(a) holds. Finally, Property 2(b) holds by construction of S, which concludes the proof of the "if"-direction. □

Lemma 7 has two consequences: firstly, the emptiness of a language given by an alternating automaton (and thus reasoning problems for various logics) can be decided by translating it into an $\mathcal{ELU}_f$-concept and TBox and then deciding their satisfiability using one of the existing DL systems, e.g. FaCT or RACER [Hor98, HM01]. Secondly, we have obtained tight complexity bounds for $\mathcal{ELU}_f$: In [SV01], satisfiability of hybrid μ-calculus formulae is reduced to emptiness of two-way alternating parity automata. It is easy to see that, discarding fixpoints, nominals, and inverse modalities, this yields a reduction from multi modal **K** extended with the universal modality to the emptiness problem of one-way alternating looping automata. This, together with the fact that **K** with the universal modality is known to be ExpTime-hard [Spa93], yields ExpTime-hardness of the emptiness problem for alternating looping automata. Now our translation being polynomial implies that satisfiability of $\mathcal{ELU}_f$-concepts w.r.t. TBoxes is ExpTime-hard. Finally, $\mathcal{ELU}_f$ is a fragment of deterministic propositional dynamic logic which is in ExpTime [BHP82] and allows for the internalisation of TBoxes (see, e.g., [CGL99]). Thus we have tight complexity bounds.

Corollary 9. *Satisfiability of $\mathcal{ELU}_f$-concepts w.r.t. general TBoxes is* ExpTime*-complete.*

5 Two-Way Alternating Automata

In this section, we extend the translation from one-way to two-way automata and thus can also test the emptiness of two-way automata using a DL reasoner. Since this extended translation involves *inverse* roles, we need a DL reasoner that can handle inverse roles such as FaCT or RACER.

Alternating automata from Definition 3 can be said to be *one-way* since the transition function tells the automaton to stay in the same node of the input tree or go to one of its successors. In *two-way* alternating automata, it can also tell to go to the predecessor.

Definition 10. For a natural number K, let $[K]_-$ be the set $\{-1, 0, \ldots, K\}$. For a word $w = v \cdot k \in [K]^+$ with $k \in [K]$, the concatenation $w \cdot (-1) = v$, and $\varepsilon \cdot (-1)$ is undefined.

A *two-way alternating automaton* $\mathcal{A} = (Q, \Sigma, q_0, \delta)$ is defined like a (one-way) alternating automaton, with the exception that δ is a function from $Q \times \Sigma$ to $\mathcal{B}^+([K]_- \times Q)$. A run and the language accepted by an automaton are defined accordingly, i.e., with $S \subseteq [K]_0 \times Q$ in Property 2 replaced with $S \subseteq [K]_- \times Q$.

For example, a transition $\delta(q_3, \sigma) = (2, q_4) \vee (-1, q_1)$ can be satisfied either by sending a copy of the automaton in state q_4 to the second successor of the current node or by sending one in state q_1 to its predecessor. Since $\varepsilon \cdot (-1)$ is undefined, it is impossible to go up from the root node.

To extend our translation to two-way automata, we use a logic which is more expressive than $\mathcal{ELU}_f$. To go up and down a tree, we use, additionally, inverse features which allow to go a relation "backwards". Moreover, to capture the notion of a run appropriately, we have to explicitly ensure uniqueness of predecessors since the inverse of a feature is not required to be functional. For example, the concept $\exists f_1^-.Q_1 \sqcap \exists f_1^-.Q_2$ can have an instance d with two f_1 predecessors which are labelled with different alphabet symbols, i.e., d's f_1-predecessor is not unique.

In the following, we describe a way to capture uniqueness of predecessors using value restrictions $\forall f_i^-.C$ to express that *all* f_i predecessors belong to C.[2]

[2] It is also possible to use role hierarchies instead of value restrictions: then, a "predecessor feature" f_{-1} is enforced to be interpreted as the union of the inverses of the features f_i.

The resulting logic is still a fragment of $\mathcal{SHIQ}$ and therefore it can be decided using the implementations mentioned above.

Definition 11. For a feature f, f^- is a called an *inverse feature*.

The set of $\mathcal{ALI}_f$ *concepts* is defined like the set of $\mathcal{ELU}_f$ concepts with the following addition: if f is a feature or an inverse feature and C is a concept, $\forall f.C$ and $\exists f.C$ are also concepts.

The interpretation function is extended with

$$\begin{aligned}(f^-)^\mathcal{I} &:= \{(d,e) \mid (e,d) \in f^\mathcal{I}\}\\ (\forall f.A)^\mathcal{I} &:= \{d \in \Delta^\mathcal{I} \mid \forall e : f^\mathcal{I}(d,e) \rightarrow e \in C^\mathcal{I}\}\end{aligned}$$

The translation tr' of a two-way automaton into $\mathcal{ALI}_f$ is defined like tr from Section 4 with the following addition:

$$\mathit{tr}'(-1,q) := \bigsqcap_{i\in[K]} \forall f_i^-.\mathit{tr}(q).$$

This enforces that the label of *each* predecessor contains $\mathit{tr}(q)$. Additionally, we have to ensure that there is one node which corresponds to the root node and therefore has no predecessors. Thus we reduce emptiness of $\mathcal{A}$ to the satisfiability of $\mathit{tr}'(\mathcal{A})$ and the concept $\mathit{tr}(q_0) \sqcap \bigsqcap_{i\in[K]} \forall f_i^-.\bot$.

Lemma 12. *The language accepted by a two-way alternating automaton* $\mathcal{A} = (Q, \Sigma, q_0, \delta)$ *is non-empty iff* $\mathit{tr}(q_0) \sqcap \bigsqcap_{i\in[K]} \forall f_i^-.\bot$ *is satisfiable w.r.t.* $\mathit{tr}'(\mathcal{A})$.

Proof. This proof is similar to the one for Lemma 7. For the only-if direction, we additionally have to show that, for an interpretation $\mathcal{I}$, an individual i satisfies $\bigsqcap_{i\in[K]} \forall f_i^-.Q$ if there is a node r in the run ρ which has a son s with $\mathit{state}(s) = q$, where s represents a transition to the father node t of $\mathit{node}(r)$. This is true since, by construction, there is only one f_i predecessor j of i, and j belongs to $Q^\mathcal{I}$ since it corresponds to t and therefore belongs to all Q_i relations for which there exists an r_i s.th. $\rho(r_i) = (t, q_i)$.

For the if-direction, consider a transition $(-1, q)$ which translates into $\bigsqcap_{i\in[K]} \forall f_i^-.Q$. In the construction of the run ρ, we introduce additional nodes: for nodes r in ρ and t in τ with $\mathit{node}(r) = t \neq \varepsilon$ and $t_{-1} := t\cdot(-1)$, we create, for every $q_i \in \mathit{states}(t_{-1})$, an additional successor r_i labelled with (t_{-1}, q_i). Then, the value restriction ensures that $q \in \mathit{states}(t_{-1})$. Thus one of the r_i is labelled with q and ρ is a run on τ. □

6 Implementation

To test the feasibility of our approach, we implemented the translation to $\mathcal{ALI}_f$ in Lisp. The results we are going to report are preliminary as we only tested it on few hand-crafted concepts and the translation routines themselves are not optimised.

The automata serving as input for our program result from the translation of concepts in the language $\mathcal{ALCIO}$, which stands for $\mathcal{ALC}$ with inverse roles and

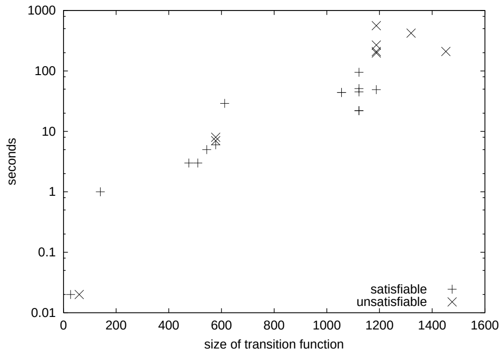

Fig. 4. Performance of satisfiability tests

nominals. *Nominals* are atomic concepts that are to be interpreted as singleton sets. Thus, while the concept $N = \exists R.(C \sqcap D) \sqcap \exists R.(C \sqcap \neg D)$ is obviously satisfiable in $\mathcal{ALC}$, it is unsatisfiable in $\mathcal{ALCIO}$ if C is a nominal. Satisfiability of $\mathcal{ALCIO}$ concept terms is ExpTime-complete [ABM99], and a translation from $\mathcal{ALCIO}$ into automata can be found in [SV01]. We use the system FaCT to reason about the resulting $\mathcal{ALI}_f$ TBox. Thus, the chain of translations is:

$$\mathcal{ALCIO} \xrightarrow[\substack{\text{linear in } \#Q \\ \text{exponential in } \#\Sigma}]{\text{[SV01]}} \text{Automaton} \xrightarrow[\substack{\text{polynomial in} \\ \#Q + \#\Sigma}]{\text{Sect. 5}} \mathcal{ALI}_f \rightsquigarrow \text{FaCT}$$

This enables us to reason about a language including nominals using a DL system which does not provide nominals.

In step one, the number of states of the automaton is linear, but the alphabet (and therefore also the transition function) is exponential. This makes our translation in step two, which is linear in the size of the transition function, exponential in the size of the input concept. Clearly, this sequence of translations is sub-optimal since it exponentially translates one ExpTime-complete logic into another one.

Thus, the rather unimpressive empirical results are not surprising: for example, the concept N mentioned above leads to an automaton with a transition function of size 1320 and could not be processed due to insufficient memory.[3] However, we use these concepts only as an example and our main focus is the behaviour of our algorithm in relation to the size of the input automaton's transition function. Figure 4 shows, on a logarithmic scale, the seconds it took to decide the satisfiability of a TBox in relation to the size of the transition function of the automaton, which is linear in the number of GCIs in the TBox (see Section 4). The calculation time increases almost exponential, which contrasts with the behaviour of FaCT on a TBox derived from a real-world knowledge base [Hor97]. Similarly, in [BCG01], it was observed that FaCT had severe difficulties classifying TBoxes resulting from the translation of comparably small

[3] The tests were performed on a Pentium 4 processor with 1.7 GHz and 512 MB of RAM running Allegro CL 6.2 on Linux.

UML diagrams. Together with our results, this indicates that TBoxes resulting from an automatic translation are significantly harder for current DL systems than "handcrafted" TBoxes. One reason for this behaviour are GCIs which cannot be absorbed [Hor99]. However, in our translation, all GCIs are absorbable [Hor97], which means that the reason for the bad performance is not yet fully understood.

7 Conclusion

We have presented a translation from one- and two-way alternating automata into description logics, which enables us to use available DL reasoners to decide the emptiness of the language accepted by an automaton. This yields satisfiability decision procedures for various logics for which automata-based algorithms are known. Our empirical results show that the computation time is indeed exponential in the size of the automaton's transition function. Thus, significant optimisations would need to be developed.We have also seen that even the inexpressive logic $\mathcal{ELU}_f$, when augmented with general TBoxes, becomes ExpTime-complete.

Concerning our ultimate goal to understand the relationship between automata- and tableau-based algorithms, we have achieved the following: let $\mathcal{L}$ be a logic with a decision procedure based on looping alternating automata and C an $\mathcal{L}$-concept with automaton $\mathcal{A}_C$. Then it can be easily seen that there is a one-to-one correspondence between the completion trees t constructed by the $\mathcal{SHIQ}$ tableau algorithm when started with $tr(\mathcal{A}_C)$ and $L(\mathcal{A}_C)$ in the following sense: if t corresponds to τ, then there is a one-to-one mapping between the "relevant" nodes of τ and the nodes of t such that their labelling (restricted to concept names) coincides (even though the logics $\mathcal{L}$ and the one used in $tr(\mathcal{A}_C)$ differ). Thus tableau- and automata-based algorithms indeed work on the same structures. Exploring this close relationship further is part of future work.

References

[ABM99] C. Areces, P. Blackburn, and M. Marx. A road-map on complexity for hybrid logics. In *Proc. of CSL'99*, vol. 1683 of *LNCS*, pages 307–321. Springer-Verlag, 1999.

[BCG01] D. Berardi, D. Calvanese, and G. de Giacomo. Reasoning on UML Class Diagrams using Description Logic Based Systems. In *Proc. of the KI'2001 Workshop on Applications of Description Logics*. CEUR (`http://ceur-ws.org/`), 2001.

[BCM+03] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[BHP82] M. Ben-Ari, J.Y. Halpern, and A. Pnueli. Deterministic propositional dynamic logic: finite models, complexity and completeness. *J. of Computer and System Science*, 25:402–417, 1982.

[BS01] F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69, 2001.

[CGL99] D. Calvanese, G. de Giacomo, and M. Lenzerini. Reasoning in expressive description logics with fixpoints based on automata on infinite trees. In *Proc. of IJCAI-99*. Morgan Kaufmann, 1999.

[DM00] F. M. Donini and F. Massacci. Exptime tableaux for $\mathcal{ALC}$. *Artificial Intelligence*, 124(1):87–138, 2000.

[GL94] G. de Giacomo and M. Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics (extended abstract). In *Proc. of AAAI-94*. AAAI Press, 1994.

[Hla02] J. Hladik. Implementation and optimisation of a tableau algorithm for the guarded fragment. *Proc. of Tableaux 2002*, vol. 2381 of *LNAI*. Springer-Verlag, 2002.

[HM92] J. Y. Halpern and Y. Moses. A guide to completeness and complexity for modal logic of knowledge and belief. *Artificial Intelligence*, 54:319–379, 1992.

[HM01] V. Haarslev and R. Möller. RACER system description. In *Proc. of IJCAR-01*, vol. 2083 of *LNAI*. Springer-Verlag, 2001.

[Hor97] I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, Univ. of Manchester, 1997.

[Hor98] I. Horrocks. Using an Expressive Description Logic: FaCT or Fiction? In *Proc. of KR-98*. Morgan Kaufmann, 1998.

[Hor99] I. Horrocks. FaCT and iFaCT. In *Proc. of DL'99*, 1999. CEUR (`http://ceur-ws.org/`).

[HS02] Ian Horrocks and Ulrike Sattler. Optimised reasoning for $\mathcal{SHIQ}$. In *Proc. of ECAI 2002*. IOS Press, 2002.

[HST99] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In *Proc. of LPAR'99*, vol. 1705 of *LNAI*. Springer-Verlag, 1999.

[KV98a] O. Kupferman and M. Y. Vardi. Freedom, weakness, and determinism: From linear-time to branching-time. In *Proc. of LICS'98*. IEEE Computer Society Press, 1998.

[KV98b] O. Kupferman and M. Y. Vardi. Weak alternating automata and tree automata emptiness. In *Proc. of STOC 98*. ACM, 1998.

[LS01] C. Lutz and U. Sattler. The complexity of reasoning with boolean modal logics. In *Advances in Modal Logics 3*. CSLI Publications, 2001.

[PSV02] G. Pan, U. Sattler, and M. Y. Vardi. BDD-based decision procedures for K. In *Proc. of CADE-18*, vol. 2392 of *LNAI*. Springer-Verlag, 2002.

[Sch91] K. Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of IJCAI-91*. Morgan Kaufmann, 1991.

[SE89] Robert S. Streett and E. Allen Emerson. An automata theoretic decision procedure for the propositional μ-calculus. *Information and Computation*, 81:249–264, 1989.

[Spa93] E. Spaan. *Complexity of Modal Logics*. PhD thesis, University of Amsterdam, 1993.

[SS91] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[SV01] U. Sattler and M. Y. Vardi. The hybrid μ-calculus. In *IJCAR-01*, vol. 2083 of *LNAI*. Springer-Verlag, 2001.

[Tho92] W. Thomas. Automata on infinite objects. In *Handbook of theoretical computer science*, volume B. Elsevier Science Publishers, 1992.

[Var98] M. Y. Vardi. Reasoning about the past with two-way automata. In *Proc. of ICALP'98*, vol. 1443 of *LNCS*, 1998. Springer-Verlag.

[VW86] M. Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *J. of Computer and System Science*, 32:183–221, 1986.