Reasoning about Nominals with FaCT and Racer

Jan Hladik*
Technische Universität Dresden

Abstract

We present a translation of looping alternating two-way automata into a comparably inexpressive description logic, which is contained in $\mathcal{SHIQ}$. This enables us to perform the emptiness test for a language accepted by such an automaton using the systems FaCT and Racer. We implemented our translation and performed a test using automata which accept models for $\mathcal{ALCIO}$ concepts, so that we can use $\mathcal{SHIQ}$ systems to reason about nominals. Our empirical results show, however, that the resulting knowledge bases are hard to process for both systems.

1 Introduction

Tableau- and automata-based algorithms are two mechanisms for testing satisfiability of a DL concept or TBox. The most significant advantage of automata algorithms (for examples, see e.g. [9, 7]) is elegance: the translation is often very intuitive, an ExpTime upper complexity bound is obtained automatically, non-determinism and infinite structures are handled implicitly. Their main drawback is their complexity, which is exponential not only in the worst case, but in every case. Tableau algorithms [1] on the other hand are well suited for implementation [5, 3] since several well-known optimizations have led to a good performance for many realistic applications. However, ensuring termination or obtaining an ExpTime upper complexity bound is a difficult task [2]. In the absence of an approach enjoying the advantages of both, for many logics tableau- *and* automata-based algorithms were hand-crafted, which constitutes a possibly unnecessary overhead. In [4], we present a way of translating looping two-way alternating automata into the relatively inexpressive description logic $\mathcal{ALI}_f$, which is contained in $\mathcal{SHIQ}$ [6]. This enables us to take a concept C in a language that is decidable by alternating automata, construct the corresponding automaton $\mathcal{A}_C$, then translate $\mathcal{A}_C$ into a description logic TBox $\mathcal{T}_C$, and use a $\mathcal{SHIQ}$ system on $\mathcal{T}_C$ to test satisfiability of the initial concept C. First results regarding the performance of FaCT [5] on a few hand-crafted concepts were presented in [4]. Here, we generated a larger set of test concepts and tested their satisfiability with both FaCT and Racer [3].

2 Preliminaries

In this section, we describe the looping alternating two-way automata, which we translate, and the DL $\mathcal{ALI}_f$, which we translate into. This class of automata is very useful

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

for description logics: to decide the satisfiability problem for a DL concept C, one defines an automaton $\mathcal{A}_C$ which accepts all (abstractions of) models of C. Then, one can decide satisfiability of C by performing the emptiness test for the language accepted by $\mathcal{A}_C$. First, we define the data structure these automata operate on.

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

Intuitively, 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. The terms $\ell \cdot 0$ and $\ell \cdot (-1)$ are used within the transition function of two-way automata: $\ell \cdot 0$ is defined as ℓ and, for an $\ell = w \cdot v$ with $w \in [K]^*$ and $v \in [K]$, $\ell \cdot (-1)$ is defined as w (i.e. the father node of ℓ); $\varepsilon \cdot (-1)$ is undefined. The use of 0 and -1 allows the automaton, after processing a tree node n, not only to continue with n's sons, but also to stay in n or return to n's father. Moreover, the Boolean connectives $\wedge$ and $\vee$ can be used in the transition function. Thus, for example, the transition $\delta(a, q_1) = (1, q_3) \wedge ((-1, q_2) \vee (3, q_1))$ is to be interpreted as follows: if the automaton processes a node ℓ, is in state q_1, and reads the letter a, then it sends one copy of the automaton in state q_3 to the first successor of ℓ and either another copy in state q_2 to ℓ's predecessor or a copy in state q_1 to the third successor. Formally, the transition function is defined using positive Boolean formulae which are described in the following.

Definition 2 (PBF, Alternating Automaton, Run) The set of *positive Boolean formulae* over a set V, $\mathcal{B}^+(V)$, consists of all 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., $w(\mathcal{A}) := (\hat{q} + 1) \cdot (K + 2)$. 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 S satisfies φ and, 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 $\mathcal{A}$.

The target language for our translation, $\mathcal{ALI}_f$, is comparably inexpressive: it contains features (and their inverses), but neither non-functional roles nor the advanced constructors present in state-of-the-art DLs, e.g. qualified number restrictions.

Definition 3 (Attributive language with inverses and features, $\mathcal{ALI}_f$) Let $\mathsf{N_C}$ be a set of concept names and $\mathsf{N_F}$ a set of feature names. The set of $\mathcal{ALI}_f$ *concepts* and *features* over $\mathsf{N_C}$ and $\mathsf{N_F}$ are inductively defined as follows:

- if f is a feature name, then f^- is an *inverse feature*;
  f is a *feature* if it is a feature name or an inverse feature;
- $\top$, $\bot$, and each concept name $C \in \mathsf{N_C}$ is an $\mathcal{ALI}_f$-concept;
  if C and D are concepts, then $C \sqcup D$ and $C \sqcap D$ are concepts;
  if C is a concept and f is a feature, then $\exists f.C$ and $\forall f.C$ are concepts.

GCIs, TBoxes, interpretations and models are defined as usual.

3 The Translation

In this section, we describe how to translate an alternating automaton $\mathcal{A}$ into a TBox $\mathit{tr}(\mathcal{A})$ and a concept C such that $L(\mathcal{A})$ is non-empty iff C is satisfiable w.r.t. $\mathit{tr}(\mathcal{A})$. Intuitively, we translate the transition function δ into a set of GCIs $\mathit{tr}(\mathcal{A})$ whose models correspond to runs of $\mathcal{A}$. To this purpose, we use concept names to represent the automaton's states and alphabet symbols and, for each $k \in [K]$, a feature name f_k to represent the "k-th successor" relation for a node in the input tree.

Definition 4 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 $\mathcal{ALI}_f$ 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

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

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

$$
\begin{array}{lcllcl}
\mathit{tr}(q_i) & := & Q_i \quad \text{for } q_i \in Q, & \mathit{tr}(\sigma_i) & := & A_i \quad \text{for } \sigma_i \in \Sigma, \\
\mathit{tr}(\alpha \wedge \beta) & := & \mathit{tr}(\alpha) \sqcap \mathit{tr}(\beta), & \mathit{tr}(\alpha \vee \beta) & := & \mathit{tr}(\alpha) \sqcup \mathit{tr}(\beta), \\
\mathit{tr}(\mathit{true}) & := & \top, & \mathit{tr}(\mathit{false}) & := & \bot, \\
\mathit{tr}(0,q) & := & \mathit{tr}(q), & \mathit{tr}(k,q) & := & \exists f_k.\mathit{tr}(q) \quad \text{for } k > 0, \\
\mathit{tr}(-1,q) & := & \prod\limits_{i \in [K]} \forall f_i^-.\mathit{tr}(q). & & &
\end{array}
$$

Lemma 5 The language accepted by an alternating automaton $\mathcal{A} = (Q, \Sigma, q_0, \delta)$ is non-empty iff the concept $C = \mathit{tr}(q_0) \sqcap \prod_{i \in [K]} \forall f_i^-.\bot$ its satisfiable w.r.t. $\mathit{tr}(\mathcal{A})$.

We have shown in [4] that tr ensures that each model $\mathcal{I}$ of $\mathit{tr}(\mathcal{A})$ corresponds to a run ρ on some tree τ. Intuitively, a node r in the domain of ρ, $\mathsf{dom}(\rho)$, is labelled with a node t in $\mathsf{dom}(\tau)$ which, in turn, is labelled with exactly one $\sigma \in \Sigma$. Thus each r in $\mathsf{dom}(\rho)$ 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 A_i. The translation of transitions $\mathit{tr}(-1,q)$ going to the predecessor node ensures that the label of *each* predecessor contains $\mathit{tr}(q)$. Additionally, we have to enforce that there is one node which corresponds to the root node and therefore has no predecessors. Thus we reduce emptiness of $\mathcal{A}$ to unsatisfiability of the concept $C = \mathit{tr}(q_0) \sqcap \prod_{i \in [K]} \forall f_i^-.\bot$ w.r.t. the TBox $\mathcal{A}$, i.e. there exists an instance of the concept corresponding to the initial state which is not a successor of any other individual.

4 The Test Concepts

The automata we used for testing result from a decision procedure for the DL $\mathcal{ALCIO}$, i.e. $\mathcal{ALC}$ with inverse roles and nominals. In [8], an algorithm is described which uses two-way looping automata to decide satisfiability of formulae in the hybrid μ-calculus, a modal logic which corresponds to $\mathcal{ALCIO}$ extended with fixpoints. The size of the automaton's transition function is exponential in the size of the input concept C. Since our translation in Section 3 is linear, the size of the TBox $\mathcal{T}_C$ is also exponential in C. Table 1 shows the concept patterns we used for our test. Here, the expression $(\forall R)^i$ means $\underbrace{\forall R \ldots \forall R}_{i \text{ times}}$. For every pattern, we used the concepts for $i \in \{0, \ldots, 5\}$.

The idea behind the structure of these patterns is the following: we test the influence of the mere existence of a nominal using concept patterns which share the same structure, but one of which uses a nominal symbol and the other one uses a concept symbol (...-c and ...-n/nc); we use the special features of nominals (root-nc); and we exploit the interaction between universal and existential restriction (all-...) as well as the interaction between a role and its inverse (all-inv-...).

Name	Satisfiable Concept	Unsatisfiable Concept
ex-c	$(\exists R)^i.A$	$(\exists R)^i.(A \sqcap \neg A)$
ex-n	$(\exists R)^i.N$	$(\exists R)^i.(N \sqcap \neg N)$
ex-nc	$(\exists R)^i.(A \sqcap N)$	$(\exists R)^i.(A \sqcap \neg A \sqcap N)$
all-c	$(\exists R)^i.A \sqcap (\forall R)^i.B$	$(\exists R)^i.A \sqcap (\forall R)^i.\neg A$
all-nc	$(\exists R)^i.A \sqcap (\forall R)^i.N$	$(\exists R)^i.A \sqcap (\forall R)^i.(N \sqcap \neg A)$
all-inv-c	$B \sqcap (\exists R)^i.(\forall R^-)^i.A$	$\neg A \sqcap (\exists R)^i.(\forall R^-)^i.A$
all-inv-nc	$N \sqcap (\exists R)^i.(\forall R^-)^i.A$	$(N \sqcap A) \sqcap (\exists R)^i.(\forall R^-)^i.(\neg A)$
root-nc	$N \sqcap A \sqcap (\exists R)^i.(N \sqcap B)$	$N \sqcap A \sqcap (\exists R)^i.(N \sqcap \neg A)$

Table 1: Test concepts

5 Results

We tested our concepts with FaCT version 2.31.7 and Racer version 1.6.7 on the following system: hardware: Pentium-IV 1.7GHz, 512MB RAM, 1.5GB swap-space; software: Linux, Allegro Common Lisp 6.2 (FaCT) or 6.1 (Racer). Table 2 shows for every concept pattern the maximum i for which the concept could be tested within the time limit of 1000 seconds. The adjacent column shows the reason why the test of the next harder concept failed: "T" stands for timeout, "M" for insufficient memory. The total in the bottom rows also includes the concepts for $i = 0$ and is therefore (by 8) higher than the sum of the above rows.

Comparing the "ex-c" and "ex-n" concepts, it is obvious that the overhead introduced by nominals is significant. The same holds for every ...-c concept in comparison with the corresponding ...-nc concept. Moreover, unsatisfiable concepts are significantly harder to process than their satisfiable counterparts. From the "root-nc" concepts, only the trivial one for $i = 0$ could be decided. Comparing the performance of the two systems, one can see that FaCT solves some more concepts than Racer.

Next, we examine if the calculation time does indeed increase exponentially in the size of the input automaton or if this behaviour can be prevented by the optimisations

of the tableau algorithms. To this end, Figure 3 displays, for some selected formula patterns, the calculation times in relation to the size of the input automaton's transition function on a logarithmic scale. Although there are only few measuring points per pattern, the nearly linear graphs suggest that the calculation time increases nearly exponentially in the size of the transition function/TBox.

	FaCT				Racer			
Concept	sat		unsat		sat		unsat	
ex-c	5		2	T	4	T	1	T
ex-n	3	M	1	T	1	T	0	T
ex-nc	2	M	0	T	0	M	0	T
all-c	3	M	2	T	3	T	1	T
all-nc	0	M	0	T	0	M	0	M
all-inv-c	3	T	2	T	1	M	1	M
all-inv-nc	0	M	1	T	0	M	0	M
root-nc	0	M	0	T	0	M	0	M
Total	24		16		17		11	
	40				28			

Table 2: Number of successful tests

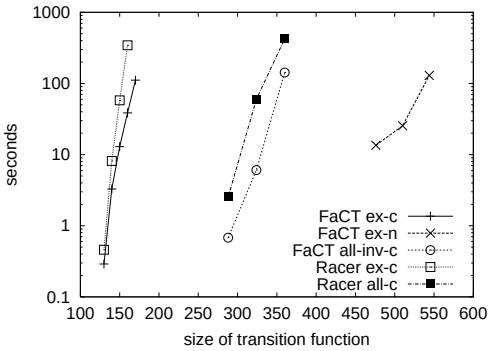

Figure 3: Calculation times

6 Conclusion

We presented a translation of looping alternating automata into a description logic, which enables us to use existing DL systems to perform the emptiness test of the language accepted by an automaton. In order to test its efficiency, we implemented the translation procedure, created a set of test automata and evaluated the performance of FaCT and Racer on their translation. Our results indicate that the time needed for the satisfiability test of the corresponding TBox is exponential in the size of the input automaton's transition function.

References

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

[2] F. M. Donini and F. Massacci. EXPTIME tableaux for ALC. *Artificial Intelligence*, 124(1):87–138, 2000.

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

[4] J. Hladik and U. Sattler. A translation of looping alternating automata to description logics. In *Proc. of the 19th Conference on Automated Deduction (CADE-19)*, Lecture Notes in Artificial Intelligence. Springer Verlag, 2003.

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

[6] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proc. of LPAR'99*, volume 1705 of *LNAI*, pages 161–180. Springer-Verlag, 1999.

[7] C. Lutz and U. Sattler. The complexity of reasoning with boolean modal logics. In F. Wolter, H. Wansing, M. de Rijke, and M. Zakharyaschev, editors, *Advances in Modal Logics 3*. CSLI Publications, Stanford, 2001.

[8] U. Sattler and M. Y. Vardi. The hybrid μ-calculus. In *IJCAR-01*, volume 2083 of *LNAI*, pages 76–91. Springer-Verlag, 2001.

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