Description Logics: a Nice Family of Logics — Automata-Based Decision Procedures —

Uli Sattler¹  Thomas Schneider²

¹School of Computer Science, University of Manchester, UK
²Department of Computer Science, University of Bremen, Germany

ESSLLI, 8 August 2012
Plan for today

**Yesterday**, we looked at tableau-based decision procedures:

- based on the simple idea of model construction
- yield the finite model property and the tree model property
- often require hard termination proofs
- often don’t yield tight upper complexity bounds

**Today**, we want to explore automata-based decision procedures:

- elegant and simple
- don’t require termination proofs
- yield tight EXPTIME upper bounds
- are difficult to implement

**Thanks** to Carsten Lutz for most of the material on these slides.
Plan for today

1. Automata basics
2. An $\text{EXPTIME}$ upper bound for $\mathcal{ALC}$
3. Extensions
4. Final remarks
And now . . .

1. Automata basics

2. An \textsc{ExpTime} upper bound for $\mathcal{ALC}$

3. Extensions

4. Final remarks
Types of automata:

- Finite automata (DFA/NFA): work on finite words
- $\omega$-automata: work on infinite words
- Automata on finite trees
- Automata on infinite trees
Trees

**Infinite $k$-ary tree:**

- Nodes $\in \{1, \ldots, k\}^*$:
  - $\varepsilon, 0, \ldots, k, 00, \ldots, kk, \ldots$
- $\varepsilon$ denotes the root
- node $n$ has successors $n_1, \ldots, n_k$ (ordered!)
- e.g., node 12 is the 2$^{\text{nd}}$-left succ. of the 1$^{\text{st}}$-left succ. of the root

**$k$-ary $M$-tree $T$:**

- nodes labelled with elements from $M$
- e.g.: $T(12) = a$

Q: $T(22) = ?$
Idea for deciding satisfiability w.r.t. TBoxes:

1. Choose a DL that has the tree model property
   (infinite trees are ok)
2. For concept $C_0$ and TBox $\mathcal{T}$, define automaton $A(C_0, \mathcal{T})$
   that accepts precisely the tree models of $C_0$ and $\mathcal{T}$
3. Check whether the language recognised by $A(C_0, \mathcal{T})$ is empty
   (If you don’t have tree model property: try some tricks)

Establish $\text{ExpTime}$ upper bound:

- Size of $A(C_0, \mathcal{T})$ is usually exponential in the size of $C_0$ and $\mathcal{T}$
- Emptiness can be decided in deterministic polynomial time
Looping tree automata

LTAs are tuples $\mathcal{A} = (S, M, I, \Delta)$ where:

- $S$ is a finite set of states
- $M$ is an alphabet
- $I \subseteq Q$ is a set of initial states
  - i.e., every run (= computation) of $\mathcal{A}$ starts in a state from $I$
- $\Delta \subseteq S \times M \times S^k$ is a transition relation
  - i.e., $\Delta$ consists of tuples $(s_0, a, s_1, \ldots, s_k)$, meaning:
    - “if $\mathcal{A}$ is in state $s_0$ and reads $a$ in the current node’s label, $\mathcal{A}$ next visits the $k$ successor nodes in states $s_1, \ldots, s_k$, resp.”
  - non-deterministic choices:
    - several tuples starting with the same $(s_0, a)$ are allowed

Language recognised by $\mathcal{A}$: a set of $k$-ary $M$-trees
Example automaton and its runs

**Example:** LTA $\mathcal{A}$ on alphabet $\{a, b\}$

$$S = \{s_a, t\}$$  \hspace{1cm} $\Delta = \{(s_a, a, s_a, t),$  

$$M = \{a, b\}$$  \hspace{1cm} $(s_a, a, t, s_a),$  

$$l = \{s_a\}$$  \hspace{1cm} $(t, a, t, t),$  

$$= \{(t, b, t, t)\}$$

Recognised language: all trees with infinite $a$-path starting at root
Definition of a run

Example: LTA on alphabet \{a, b\}

\[ S = \{s_a, t\} \quad \Delta = \{(s_a, a, s_a, t), (s_a, a, t, s_a), (t, a, t, t), (t, b, t, t)\} \]

Definition: a run \( r \) of \( A \) on \( T \)
assigns to each node in \( T \) a state from \( S \) such that

- \( T \)'s root is labelled with a state from \( I \)
- \( (r(n), T(n), r(n1), \ldots, r(nk)) \) \( \in \Delta \)

for all nodes \( n \in \{1, \ldots, k\}^* \)

Recognised language: \( L(A) = \{ T \mid \text{there is a run of } A \text{ on } T \} \)
And now . . .

1. Automata basics

2. An $\text{EXPTIME}$ upper bound for $\mathcal{ALC}$

3. Extensions

4. Final remarks
Roadmap

**Goal:** prove that $\mathcal{ALC}$-satisfiability w.r.t. TBoxes is in $\exp\text{TIme}$

2 steps:

1. Represent tree interpretations as **Hintikka trees**
   - Tree models have *labelled* edges (roles), automata trees don’t
   - Convenient to label nodes with *complex* concepts

2. Define automaton that accepts exactly those Hintikka trees that represent models for the input concept + TBox
   - This reduces sat. w.r.t. TBoxes to emptiness of the automaton
Hintikka sets

... are used as node labels in Hintikka trees (\(\sim\) constitute set \(M\))

**Intuitively**, a HS contains relevant concepts satisfied by some domain element

**Definition:** Let \(C_0, T\) be in NNF; \(\text{sub}(C_0, T) = \text{sub}(T \cup \{a: C_0\})\)

(i.e., \(\text{sub}(C_0, T)\) consists of all subconcepts of \(C\), in \(T\), and of \(\neg C \sqcup D\) for each \(C \sqsubseteq D \in T\))

A **Hintikka set** for \(C_0\) and \(T\) is a subset \(\mathcal{H} \subseteq \text{sub}(C_0, T)\) such that:

(H1) If \(C \sqcap D \in \mathcal{H}\), then \(C \in \mathcal{H}\) and \(D \in \mathcal{H}\).

(H2) If \(C \sqcup D \in \mathcal{H}\), then \(C \in \mathcal{H}\) or \(D \in \mathcal{H}\).

(H3) For all \(C \in \text{sub}(C_0, T)\), \(\mathcal{H}\) does not contain \(C\) and \(\neg C\) at the same time.

(H4) If \(C \sqsubseteq D \in T\), then \(\neg C \sqcup D \in \mathcal{H}\).

\(\mathcal{H}(C_0, T)\): set of all Hintikka sets for \(C_0\) and \(T\)
Excursion: Hintikka sets vs. 1-types

A Hintikka set

- contains **relevant** concepts satisfied by some domain element
- does not need to have “full knowledge” about that element
- in particular, can be empty

A 1-type (aka type) has stronger requirements:

- contains **all** concepts satisfied by some domain element
- thus has “full knowledge” about that domain element
- is a subset $t \subseteq \text{sub}(C_0, T)$ such that:
  
  (T1) $C \sqcap D \in t$ iff $C \in t$ and $D \in t$.
  
  (T2) $C \sqcup D \in t$ iff $C \in t$ or $D \in t$.
  
  (T3) For all $C \in \text{sub}(C_0, T)$, $C \in t$ iff $\neg C \notin t$.
  
  (T4) If $C \sqsubseteq D \in T$, then $\neg C \sqcup D \in t$. 

Uli Sattler, Thomas Schneider
Hintikka trees

- Let $k$ be the number of successors a domain element can be forced to have:

$$k = \# \{ D \in \text{sub}(C_0, \mathcal{T}) \mid D \text{ is of the form } \exists R.C \}$$

- Hintikka sets will be $k$-ary $\mathcal{H}(C_0, \mathcal{T})$-trees

How can we deal with the non-labelled edges?

- Intuitively, there is one potential successor for each $\exists R.C$

$\hookrightarrow$ The connecting role for each successor is already fixed!

- Enumerate all concepts $\exists R.C$ using $E_1, \ldots, E_k$

- If $E_i = \exists R.C$ is . . .
  - in node $n$’s label, then the role between $n$ and $n_i$ is $R$
  - not in $n$’s label, then the connection btn. $n, n_i$ is a “dummy”
Example

Let \( k = 2 \) \hspace{1cm} E_1 = \exists R.C \hspace{1cm} E_2 = \exists R.D \hspace{1cm} E_3 = \exists S.D

d = \text{dummy}
**Hintikka Trees II**

**Next step:** describe relationship between

- the Hintikka set of each node $n$ and
- the Hintikka sets of $n$’s successors

**Definition:**

A $(k+1)$-tuple of Hintikka sets $\mathcal{H}, \mathcal{H}_1, \ldots, \mathcal{H}_k$ is **matching** if, for every $i = 1, \ldots, k$ with $E_i = \exists R.C \in \mathcal{H}$:

(M1) $C \in \mathcal{H}_i$ (for satisfying $E_i$, it suffices to consider $i$-th successor)

(M2) if $\forall R.D \in \mathcal{H}$, then $D \in \mathcal{H}_i$
Definition

A Hintikka tree for $C_0$ and $T$ is a $k$-ary $\mathcal{H}(C_0, T)$-tree such that:

\begin{enumerate}[(T1)]
  \item $C_0 \in T(\varepsilon)$ – i.e., $C_0$ is in the root’s label
  \item For every node $n$,
    \begin{itemize}
      \item the tuple $\left(T(n), T(n1), \ldots, T(nk)\right)$ is matching.
    \end{itemize}
\end{enumerate}

Lemma

$C_0$ is satisfiable w.r.t. $T$ iff there is a Hintikka tree for $C_0$ and $T$. 
Basic idea:

- Use Hintikka sets as states and define $\Delta$ such that
  $$s_0 = \ell \text{ in all tuples } (s_0, \ell, s_1, \ldots, s_k) \in \Delta$$
  Recall: $\Delta \subseteq S \times M \times S^k$

  $\leadsto$ If there is an accepting run, it will be identical to the tree

- Use initial states to ensure that $C_0 \in T(\varepsilon)$

- Check matching via transition relation, e.g.,
  whenever $(s_0, \ell, s_1, \ldots, s_k) \in \Delta$ and $E_i = \exists R.C \in s_0$, then:
  (M1) $C \in s_i$
  (M2) if $\forall R.D \in s_0$, then $D \in s_i$
Constructing automata II

Automaton for $C_0$ and $T$:
$$A(C_0, T) = (S, M, I, \Delta), \text{ where}$$

$$S = \mathcal{H}(C_0, T)$$
$$M = \mathcal{H}(C_0, T)$$
$$I = \{s \in S \mid C_0 \in s\}$$

and $(s_0, \ell, s_1, \ldots, s_k) \in \Delta$ iff
- $s_0 = \ell$ and
- the tuple $(s_0, s_1, \ldots, s_k)$ is matching

Lemma
$$T \in L(A(C_0, T)) \text{ iff } T \text{ is a Hintikka tree for } C_0 \text{ and } T.$$
Results

**Size of $\mathcal{A}(C_0, T)$:** Let $|C_0, T| = |C_0| + |T|$. 
Number of Hintikka sets exponential in $|C_0, T|$

$\Rightarrow |Q|, |I|, |M|$ exponential in $|C_0, T|$

$\Rightarrow |\Delta|$ exponential in $|C_0, T|$ since $|\Delta| = |M| \cdot |S|^{k+1}$

$\Rightarrow$ Size of $\mathcal{A}(C_0, T)$ exponential in $|C_0, T|$

**Decision procedure for $\mathcal{ALC}$-concept satisfiability w.r.t. TBoxes:**

1. Given $C_0, T$, construct $\mathcal{A}(C_0, T)$ – in time exp. in $|C_0, T|$
2. Test emptiness of $\mathcal{A}(C_0, T)$ – in time polynomial in $|\mathcal{A}(C_0, T)|$

**Theorem**

$\mathcal{ALC}$-concept satisfiability w.r.t. TBoxes is in $\text{ExpTime}$.

Complexity bound is optimal: $\mathcal{ALC}$ with TBoxes is $\text{ExpTime}$-hard.
Emptiness problem of looping automata

Determine in $|S|$ rounds the set of blocking states $B \subseteq S$:

- **Initialisation:**
  
  Set $B_0 \leftarrow \{ s \in S \mid \text{there is no } (s, a, s_1, \ldots, s_k) \in \Delta \}$

- **Round $i$:**

  Set $B_i \leftarrow B_{i-1} \cup \{ s \in S \mid \text{for all } (s, a, s_1, \ldots, s_k) \in \Delta \\
  \quad \text{there is } 1 \leq i \leq k \text{ with } s_i \in B_{i-1} \}$

- Set $B = B_{|S|}$

**Lemma**

$L(A) = \emptyset$ iff $I \subseteq B$.

Computation of $B$ is clearly in polynomial time.
And now . . .

1. Automata basics
2. An \textsc{ExpTime} upper bound for $\mathcal{ALC}$
3. Extensions
4. Final remarks
Transfer to the other standard reasoning problems

The procedure shown can be applied to decide . . .

**TBox Consistency.** These are equivalent:

- $\mathcal{T}$ is consistent
- some fresh\(^1\) $C_0$ is satisfiable w.r.t. $\mathcal{T}$

**Consistency of ontologies.** Transform $(\mathcal{T}, \mathcal{A})$ into $(\mathcal{T}', \mathcal{A}')$, where

- $\mathcal{A}'$ consists of a single concept assertion $a : C_0$
- but $\mathcal{T}'$ is in $ALCIF_{\text{reg}}$

Then test satisfiability of $(C_0, \mathcal{T}')$ with the decision procedure extended to $ALCIF_{\text{reg}}$

**Other reasoning problems:** as shown on Tuesday

---

\(^1\)i.e., $C_0$ or $r$ doesn’t occur in $\mathcal{T}$
Extension to \textit{ALCI}

\textbf{Recall:} \textit{ALCI} = \textit{ALC} + inverse roles: $\exists R^.C$ and $\forall R^.C$

\textbf{Question:} what do we need to change in the

- definition of a Hintikka set?
- definition of a Hintikka tree?
- construction of the automaton?
- elsewhere?

\textbf{Answer:} only

- the matching condition for Hintikka trees
- and its “encoding” in the automaton’s transition function

From now on, $R$ denotes a role or its inverse.
Adapting Hintikka Trees to ALCI

**Remember:** they describe relationship between

- the Hintikka set of each node $n$ and
- the Hintikka sets of $n$’s successors

![Diagram of Hintikka sets with existential assertion](image)

**Definition:**

A $(k+1)$-tuple of Hintikka sets $\mathcal{H}, \mathcal{H}_1, \ldots, \mathcal{H}_k$ is **matching** if, for every $i = 1, \ldots, k$ with $E_i = \exists R.C \in \mathcal{H}$:

(M1) $C \in \mathcal{H}_i$ (for satisfying $E_i$, it suffices to consider $i$-th successor)

(M2) if $\forall R.D \in \mathcal{H}$, then $D \in \mathcal{H}_i$

(M3) if $\forall \text{Inv}(R).D \in \mathcal{H}_i$, then $D \in \mathcal{H}$

$\text{Inv}(P) = P^-, \text{Inv}(P^-) = P$
Adapting the automata construction to $\mathcal{ALCI}$

Remember – basic idea:

- Use Hintikka sets as states and define $\Delta$ such that
  \[ s_0 = \ell \quad \text{in all tuples} \quad (s_0, \ell, s_1, \ldots, s_k) \in \Delta \]
  Recall: $\Delta \subseteq S \times M \times S^k$

  $\leadsto$ If there is an accepting run, it will be identical to the tree

- Use initial states to ensure that $C_0 \in T(\varepsilon)$

- Check **matching** via transition relation, e.g.,
  whenever $(s_0, \ell, s_1, \ldots, s_k) \in \Delta$ and $E_i = \exists R. C \in s_0$, then:

  (M1) $C \in s_i$

  (M2) if $\forall R. D \in s_0$, then $D \in s_i$

  (M3) if $\forall \text{Inv}(R). D \in s_i$, then $D \in s_0$
And now . . .

1 Automata basics

2 An EXP\textsc{TIME} upper bound for $\textit{ALC}$

3 Extensions

4 Final remarks
What we haven’t covered

- More expressive DLs $\leadsto$ more complex automata models
  - Büchi tree automata for eventualities (trans. closure of roles)
  - and variants thereof

- Alternative approach to \textsc{ExpTime}-decision procedures: \textbf{alternating automata}
  - States are formulas, not sets of formulas
  - Size of automaton is polynomial in $|C_0, T|$}
  - Emptiness check is in \textsc{ExpTime}

$\leadsto$ avoid the problem of constructing an exp. large automaton
Automata versus tableaux: complexity

Tableau algorithms

- usually don’t yield tight upper bounds (e.g., EXPSpace for ALC)
  - are usually not worst-case optimal
- but can be optimised in many ways
  - are efficient in many cases

Automata-based algorithms

- often yield tight upper bounds (e.g., EXPTIME for ALC)
  - are often worst-case optimal
- rely on the construction of an exponential-size automaton
  - are exponential in the best and average case too
  - leave less room for optimisations
Automata versus tableaux: summary

**Tableau algorithms**

- based on a simple idea (model construction)
- amenable to optimisation techniques
- basis for state-of-the-art DL reasoners
- bad for proving deterministic upper time bounds
- termination proofs can become very hard

**Automata-based algorithms**

- elegant and simple
- well-suited for proving $\text{EXP} \text{TIME}$ upper bounds
- no termination proofs
- no optimised implementations exist (?)

That’s all for today. Thanks!