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

Uli Sattler[1] *Thomas Schneider*[2]

[1]School of Computer Science, University of Manchester, UK

[2]Department 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 ExpTime upper bound for $\mathcal{ALC}$
3. Extensions
4. Final remarks

And now . . .

Automata

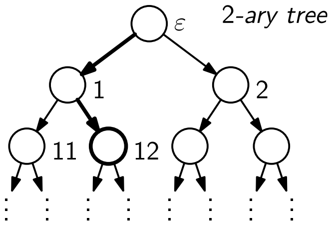
Types of automata:

- Finite automata (DFA/NFA): work on finite words
- ω-automata: work on infinite words
- Automata on finite trees
- **Automata on infinite trees**

Trees

Infinite k-ary tree:

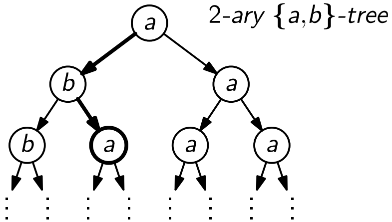
- Nodes $\in \{1, \ldots, k\}^*$:
  $\varepsilon, 0, \ldots, k, 00, \ldots, kk, \ldots$
- ε denotes the root
- node n has successors
  $n1, \ldots, nk$ (ordered!)
- e.g., node 12 is the
  2nd-left succ. of the 1st-left succ. of the root



k-ary M-tree T:

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

Q: $T(22) = ?$



Automata and DLs

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 $\mathcal{A}(C_0, \mathcal{T})$ that accepts precisely the tree models of C_0 and $\mathcal{T}$
3. Check whether the language recognised by $\mathcal{A}(C_0, \mathcal{T})$ is empty

(If you don't have tree model property: try some tricks)

Establish ExpTime upper bound:

- Size of $\mathcal{A}(C_0, \mathcal{T})$ 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**

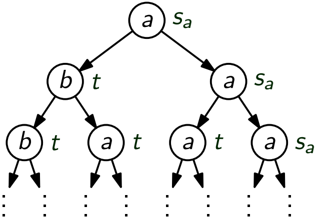
  i.e., every run (=computation) of $\mathcal{A}$ starts in a state from I
- $\Delta \subseteq S \times M \times S^k$ is a **transition relation**
  - i.e., Δ consists of tuples $(s_0, a, s_1, \dots, s_k)$, meaning:
    "if $\mathcal{A}$ is in state s_0 and reads a in the current node's label, $\mathcal{A}$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

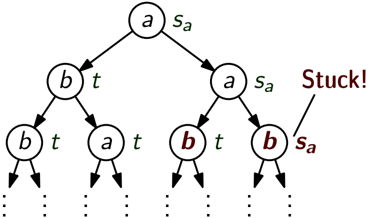
Language recognised by $\mathcal{A}$: a set of k-ary M-trees

Example automaton and its runs

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

$$S = \{s_a, t\} \qquad \Delta = \{\ (s_a, a, s_a, t),$$
$$M = \{a, b\} \qquad (s_a, a, t, s_a),$$
$$I = \{s_a\} \qquad (t, a, t, t),$$
$$(t, b, t, t)\ \}$$





Recognised language: all trees with infinite a-path starting at root

Definition of a run

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

$$S = \{s_a, t\} \qquad \Delta = \{ (s_a, a, s_a, t),$$
$$M = \{a, b\} \qquad (s_a, a, t, s_a),$$
$$I = \{s_a\} \qquad (t, a, t, t),$$
$$(t, b, t, t) \}$$

Definition: a **run** r of $\mathcal{A}$ on T
assigns to each node in T a state from S such that

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

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

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

And now . . .

Roadmap

Goal: prove that $\mathcal{ALC}$-satisfiability w.r.t. TBoxes is in EXPTIME

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 ($\rightsquigarrow$ constitute set M)

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

Definition: Let $C_0, \mathcal{T}$ be in NNF; $\mathsf{sub}(C_0, \mathcal{T}) = \mathsf{sub}(\mathcal{T} \cup \{a : C_0\})$
(i.e., $\mathsf{sub}(C_0, \mathcal{T})$ consists of all subconcepts of C, in $\mathcal{T}$,
and of $\dot{\neg} C \sqcup D$ for each $C \sqsubseteq D \in \mathcal{T}$)

A **Hintikka set** for C_0 and $\mathcal{T}$ is a subset $\mathcal{H} \subseteq \mathsf{sub}(C_0, \mathcal{T})$ 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 $C \in \mathsf{sub}(C_0, \mathcal{T})$,
  $\mathcal{H}$ does not contain C and $\dot{\neg} C$ at the same time.
- **(H4)** If $C \sqsubseteq D \in \mathcal{T}$, then $\dot{\neg} C \sqcup D \in \mathcal{H}$.

$\mathfrak{H}(C_0, \mathcal{T})$: set of all Hintikka sets for C_0 and $\mathcal{T}$

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 $t \subseteq \text{sub}(C_0, \mathcal{T})$ 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 $C \in \text{sub}(C_0, T)$, $\quad C \in t$ **iff** $\dot{\neg} C \notin t$.
  - **(T4)** If $C \sqsubseteq D \in \mathcal{T}$, then $\dot{\neg} C \sqcup D \in t$.

Hintikka trees

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

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

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

How can we deal with the non-labelled edges?

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

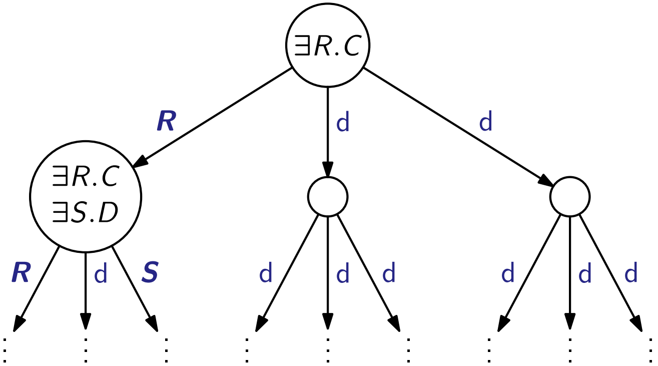
$\rightsquigarrow$ The **connecting role** for each successor is already fixed!

- 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 ni is R
  - *not* in n's label, then the connection btn. n, ni is a "dummy"

Example

Let $k = 2$ $\quad E_1 = \exists R.C \quad E_2 = \exists R.D \quad E_3 = \exists S.D$

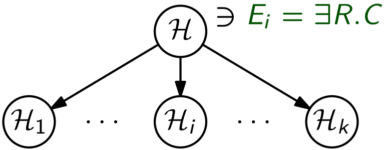
d = dummy



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 $D \in \mathcal{H}_i$

Hintikka Trees III

Definition

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

(T1) $C_0 \in T(\varepsilon)$ – i.e., C_0 is in the root's label

(T2) For every node n,
the tuple $\big(T(n), T(n1), \ldots, T(nk)\big)$ is matching.

Lemma

C_0 is satisfiable w.r.t. $\mathcal{T}$ iff there is a Hintikka tree for C_0 and $\mathcal{T}$.

Constructing automata I

Basic idea:

- Use Hintikka sets as states and define Δ such that

$$s_0 = \ell \quad \text{in all tuples} \quad (s_0, \ell, s_1, \ldots, s_k) \in \Delta$$

Recall: $\Delta \subseteq S \times M \times S^k$

$\rightsquigarrow$ 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 $\mathcal{T}$:

$\mathcal{A}(C_0, \mathcal{T}) = (S, M, I, \Delta)$, where

$$\begin{aligned} S &= \mathfrak{H}(C_0, \mathcal{T}) \\ M &= \mathfrak{H}(C_0, \mathcal{T}) \\ I &= \{s \in S \mid C_0 \in s\} \end{aligned}$$

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

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

Results

Size of $\mathcal{A}(C_0, \mathcal{T})$: Let $|C_0, \mathcal{T}| = |C_0| + |\mathcal{T}|$.

Number of Hintikka sets exponential in $|C_0, \mathcal{T}|$

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

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

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

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

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

Theorem

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

Complexity bound is optimal: $\mathcal{ALC}$ with TBoxes is 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:

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

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

Lemma

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

Computation of B is clearly in polynomial time.

And now . . .

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 $\mathcal{ALCIF}_{\text{reg}}$

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

Other reasoning problems: as shown on Tuesday

[1] i.e., C_0 or r doesn't occur in $\mathcal{T}$

Extension to $\mathcal{ALCI}$

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

Question: what do we need to change in the

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

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 $\mathcal{ALCI}$

Remember: they describe relationship between

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

$$
\begin{array}{c}
\mathcal{H} \ni E_i = \exists R.C \\
\swarrow \quad \downarrow \quad \searrow \\
\mathcal{H}_1 \quad \cdots \quad \mathcal{H}_i \quad \cdots \quad \mathcal{H}_k
\end{array}
$$

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

$\mathsf{Inv}(P) = P^-$, $\mathsf{Inv}(P^-) = P$

Adapting the automata construction to $\mathcal{ALCI}$

Remember – basic idea:

- Use Hintikka sets as states and define Δ 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 $\forall \mathsf{Inv}(R).D \in s_i$, then $D \in s_0$

And now . . .

What we haven't covered

- More expressive DLs $\rightsquigarrow$ more complex automata models
  - Büchi tree automata for eventualities (trans. closure of roles)
  - and variants thereof
- Alternative approach to ExpTime-decision procedures:
  alternating automata
  - States are formulas, not sets of formulas
  - Size of automaton is polynomial in $|C_0, \mathcal{T}|$
  - Emptiness check is in ExpTime

$\rightsquigarrow$ 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 $\mathcal{ALC}$)

  $\rightsquigarrow$ are usually not worst-case optimal
- but can be optimised in many ways

  $\rightsquigarrow$ are efficient in many cases

Automata-based algorithms

- often yield tight upper bounds (e.g., ExpTime for $\mathcal{ALC}$)

  $\rightsquigarrow$ are often worst-case optimal
- rely on the construction of an exponential-size automaton

  $\rightsquigarrow$ are exponential in the best and average case too

  $\rightsquigarrow$ 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 ExpTime upper bounds
- ⊕ no termination proofs
- ⊖ no optimised implementations exist (?)

That's all for today. Thanks!