Description Logics

Franz Baader
Theoretical Computer Science
TU Dresden
Germany

1. Motivation and introduction to Description Logics
2. Tableau-based reasoning procedures
3. Automata-based reasoning procedures
4. Complexity of reasoning in Description Logics
5. Reasoning in inexpressive Description Logics
Reasoning procedures

requirements

1. The procedure should be a decision procedure for the problem.

2. The procedure should be as efficient as possible:
   preferably optimal w.r.t. the (worst-case) complexity of the problem

3. The procedure should be practical:
   easy to implement and optimize, and behave well in applications

The tableau-based reasoning procedure for $ALC$

- satisfies the first requirement, as shown in the previous lecture.

- Highly-optimized implementations in systems like FaCT and RACER demonstrate that it satisfies the third requirement.

- It does not satisfy the second requirement in the presence of GCI's.
Tableau-based procedures

- the consistency problem for $\mathcal{ALC}$ with GCIs is ExpTime-complete, but it is very hard to design a tableau-based algorithm that is better than NExpTime:
  - exponentially long chains of role successors may be generated before blocking occurs
  - to each individual in the chain, non-deterministic rules may be applied

- termination requires blocking:
  - proof of termination and soundness becomes more complicated
  - for more expressive DLs (e.g., with number restrictions and inverse roles) one needs sophisticated blocking conditions
Automata-based procedures

For simplicity, we restrict the attention to satisfiability, i.e., consistency of an ABox of the form \( \{C_0(a_0)\} \) w.r.t. a general TBox \( \mathcal{T} \).

- Show that \( C_0 \) is satisfiable w.r.t. \( \mathcal{T} \) iff \( \mathcal{T} \) and \( \{C_0(a_0)\} \) have a tree-shaped model with root \( a_0 \).
- Translate \( C_0, \mathcal{T} \) into a tree automaton \( A_{C_0,\mathcal{T}} \) that accepts exactly the tree-shaped models of \( \mathcal{T} \) and \( \{C_0(a_0)\} \).
- Test \( A_{C_0,\mathcal{T}} \) for emptiness: is there a tree accepted by \( A_{C_0,\mathcal{T}} \)?
Automata-based procedures advantages and disadvantages

+ separation between DL-dependent part (translation) from DL-independent part (emptiness test)

+ termination is not an issue if we use automata working on infinite trees

+ well-suited for showing ExpTime upper-bounds:
  translation is exponential, emptiness test polynomial

  — usually also best-case exponential:
    translation required before emptiness test can be applied

  — no optimized implementations available
Infinite trees

We consider infinite trees with a fixed out-degree \( k \), whose nodes are labeled with elements from a finite alphabet \( \Sigma \):

Example: \( k = 2 \) and \( \Sigma = \{a, b\} \)

This tree is described by the mapping \( t : \{0, 1\}^* \rightarrow \Sigma \) with

\[
\begin{cases}
  b & \text{if } u \text{ starts with 0} \\
  a & \text{otherwise}
\end{cases}
\]

\( k \)-ary tree over \( \Sigma \):

\( t : \{0, \ldots, k - 1\}^* \rightarrow \Sigma \)
Automata on infinite trees

informal description

The automaton labels nodes of the tree with states.

\[ Q = \{ q_0, q_1, q_2 \} \]
\[ I = \{ q_0 \} \]
\[ (q_0, a) \rightarrow (q_1, q_2) \]
\[ (q_0, a) \rightarrow (q_2, q_1) \]
\[ (q_1, b) \rightarrow (q_1, q_1) \]
\[ (q_2, a) \rightarrow (q_2, q_2) \]

The root is labeled with an initial state.

The labeling of the other nodes must be compatible with the transition relation.

The transition relation may be non-deterministic.
Automata on infinite trees

A looping automaton working on $k$-ary trees is of the form $\mathcal{A} = (Q, \Sigma, I, \Delta)$ where

- $Q$ is a finite set of states, and $I \subseteq Q$ the set of initial states;
- $\Sigma$ is a finite alphabet;
- $\Delta \subseteq Q \times \Sigma \times Q^k$ is the transition relation.

A run of this automaton on a $k$-ary tree
$t : \{0, \ldots, k-1\}^* \rightarrow \Sigma$ is a $k$-ary tree
$r : \{0, \ldots, k-1\}^* \rightarrow Q$ such that

- $(r(u), t(u)) \rightarrow (r(u0), \ldots, r(u(k-1))) \in \Delta$.

The run is called initial if
- $r(\epsilon) \in I$.

Looping automaton: no additional condition based on accepting states
Accepted tree language

The tree language accepted by the looping automaton $\mathcal{A}$ is

$$L(\mathcal{A}) := \{ t \mid \text{there is an initial run of } \mathcal{A} \text{ on the } k\text{-ary tree } t\}$$

Consider the following binary tree language over $\Sigma = \{a, b\}$:

$$L := \{ t \mid a \text{ never occurs below a } b \text{ in } t\}$$

$\mathcal{A} = (Q, \Sigma, I, \Delta)$ with

- $Q := \{q_a, q_b\}$;
- $I := \{q_a, q_b\}$;
- $\Delta := \{(q_b, b) \rightarrow (q_b, q_b)\} \cup \{(q_a, a) \rightarrow (q, q') \mid q, q' \in Q\}$
Accepted tree language

The tree language accepted by the looping automaton $A$ is

$$L(A) := \{ t \mid \text{there is a run of } A \text{ on the } k\text{-ary tree } t \}$$

Consider the following binary tree language over $\Sigma = \{a, b\}$:

$$L := \{ t \mid a \text{ never occurs below a } b \text{ in } t \}$$

$A = (Q, \Sigma, I, \Delta)$ with

- $Q := \{q_a, q_b\}$;
- $I := \{q_a, q_b\}$;
- $\Delta := \{(q_b, b) \rightarrow (q_b, q_b)\} \cup \{(q_a, a) \rightarrow (q, q') \mid q, q' \in Q\}$
The emptiness problem for looping tree automata

Given: a looping tree automaton $A$
Question: is $L(A) = \emptyset$?

Top-down approach:
- label root with an initial state;
- apply transition relation to label successor nodes.

Problem:
- termination requires blocking if states are repeated on a path;
- if the automaton is non-deterministic, then we must consider all possible initial states and transitions.

NP
The emptiness test

Bottom-up approach

- Compute all **bad** states, i.e., states that cannot occur in a run.
- \( L(\mathcal{A}) = \emptyset \) iff all initial states are bad.

\[
\begin{align*}
\text{Bad}_0(\mathcal{A}) & := \emptyset \\
\text{Bad}_1(\mathcal{A}) & := \{ q \mid \text{there is no transition } (q, \cdot) \rightarrow (\cdots) \} \\
i & := 1 \\
\text{while } \text{Bad}_i(\mathcal{A}) \neq \text{Bad}_{i-1}(\mathcal{A}) \text{ do} \\
\quad \text{Bad}_{i+1}(\mathcal{A}) & := \text{Bad}_i(\mathcal{A}) \cup \{ q \mid \text{for all transitions } (q, \cdot) \rightarrow (q_1, \ldots, q_k) \\
& \quad \text{there is } j \text{ with } q_j \in \text{Bad}_i(\mathcal{A}) \} \\
i & := i + 1 \\
\text{od} \\
\text{Answer "empty" iff } I \subseteq \text{Bad}_i(\mathcal{A})
\end{align*}
\]
The emptiness test

The algorithm decides the emptiness problem in polynomial time:

- the while-loop always terminates after at most $|Q|$ iterations:
  \[ \text{Bad}_0(\mathcal{A}) \subseteq \text{Bad}_1(\mathcal{A}) \subseteq \text{Bad}_2(\mathcal{A}) \subseteq \ldots \subseteq \text{Bad}_k(\mathcal{A}) = \text{Bad}_{k+1}(\mathcal{A}) \]
  for some $k \leq |Q|$;

- every single iteration of the loop can be done in polynomial time;

- if $q \in \text{Bad}_i(\mathcal{A})$ for some $i \geq 0$ then $q$ cannot occur in a run of $\mathcal{A}$;

- if $q \not\in \text{Bad}_k(\mathcal{A})$ then there is a run containing $q$ as label of the root; for some tree

- if $i \in I \setminus \text{Bad}_k(\mathcal{A})$ then there is an initial run.
Tree model property of $\mathcal{ALC}$.

Interpretations can be viewed as graphs:
- nodes are the elements of $\Delta^I$;
- interpretation of roles yields edges;
- interpretation of concepts yields node labels.

Starting with a given node, the graph can be unraveled into a tree without "changing membership" in concepts.
Tree model property of $\mathcal{ALC}$.

$\mathcal{T}$ general $\mathcal{ALC}$-TBox, $C$ $\mathcal{ALC}$-concept:

$C$ is satisfiable w.r.t. $\mathcal{T}$ iff

there is a tree model of $\mathcal{T}$ whose root belongs to $C$.
Subdescriptions of $\mathcal{ALC}$-concept descriptions

- $C \in N_C$: $\text{Sub}(A) := \{A\}$ for $A \in N_C$;
- $C = C_1 \cap C_2$ or $C = C_1 \cup C_2$: $\text{Sub}(C) := \{C\} \cup \text{Sub}(C_1) \cup \text{Sub}(C_2)$;
- $C = \neg D$ or $C = \exists r.D$ or $C = \forall r.D$: $\text{Sub}(C) := \{C\} \cup \text{Sub}(D)$.

$\text{Sub}(A \cap \exists r.(A \cup B)) = \{A \cap \exists r.(A \cup B), A, \exists r.(A \cup B), A \cup B, B\}$

$\text{Sub}(\mathcal{T}) := \bigcup_{\mathcal{C} \subseteq D \in \mathcal{T}} \text{Sub}(C) \cup \text{Sub}(D)$

- the cardinality of $\text{Sub}(C)$ is bounded by the size of $C$;
- the size of the elements of $\text{Sub}(C)$ is bounded by the size of $C$;
- cardinality and size of $\text{Sub}(\mathcal{T})$ is polynomial in the size of $\mathcal{T}$. 
Extension of tree models to trees labeled with subdescriptions

Let $\mathcal{T}$ be a general TBox, $C_0$ a concept description, and $\mathcal{I}$ a tree model of $\mathcal{T}$ whose root belongs to $C_0$.

Extend node labels to subdescriptions from $S := \text{Sub}(\mathcal{T}) \cup \text{Sub}(C_0)$:

$$\ell(d) := \{C \in S \mid d \in C^\mathcal{I}\}.$$ 

```
I
{A}
{B}
{A}
{B}
{A}

a_0 \in A^\mathcal{I}
\ell(a_0) = \{A, A \sqcup B, \exists r.B, \exists s.A\}

b_0
{A} r
{A} r
s
s

a_1 \quad b_1\quad c_1
\ell(b_0) = \{B, A \sqcup B, \exists r.A, \exists s.A\}
\ell(c_0) = \{A, A \sqcup B\}

r
\ell(a_1) = \{A, \exists r.B\}

s
\ell(b_1) = \{B, \exists r.A\}

s
\ell(c_1) = \{A, A \sqcup B\}

Sub(\mathcal{T}) \cup \text{Sub}(A) = \{A, \exists r.B, B, \exists r.A, A \sqcup B, \exists s.A\}
```
Tree automaton

main idea

Given $\mathcal{T}$ and $C_0$, construct a looping automaton that accepts the extended tree models of $\mathcal{T}$ whose root label contains $C_0$.

Problem: mismatch between the underlying kinds of trees

1. Edge labels: extended tree models have roles as edge labels, automata work on trees without edge labels

Solution: add role names to node label of successors

$$\{r, A, A \sqcup B, \exists r. B, \exists s. A\}$$

$$\{r, B, A \sqcup B, \exists r. A, \exists s. A\}$$

$$\{s, A, A \sqcup B\}$$
Tree automaton

Problem: mismatch between the underlying kinds of trees

2. Varying arity: extended tree models have no fixed number of successors, automata work on trees with fixed arity \( k \)

Solution: take as \( k \) the number of all existential restrictions in \( S \)

\[ S = \{ A, \exists r. B, B, \exists r. A, A \sqcup B, \exists s. A \} \rightarrow k = 3 \]

- a given tree model can be modified into one where nodes have at most \( k \) successors
- for missing successors we can generated dummies

\[ \{ r, B, A \sqcup B, \exists r. A, \exists s. A \} \quad \{ s, A, A \sqcup B \} \]
Preliminaries

required to define the trees that our automata are supposed to accept

Let $\mathcal{T}$ be a general TBox and $C_0$ a concept description.

Normalization 1:
Without loss of generality we assume that the GCIs in $\mathcal{T}$ are of the form $\top \subseteq D$:

$$C \subseteq D \text{ can be replaced by } \top \subseteq \neg C \cup D$$

Normalization 2:
Without loss of generality we assume that $C_0$ and all concept descriptions in $\mathcal{T}$ are in negation normal form (NNF).

We define

$$S := \text{Sub}(\mathcal{T}) \cup \text{Sub}(C_0)$$

$$k := \text{card}(\{C \in S \mid C \text{ is an existential restriction}\})$$
Hintikka trees

the trees that our automata are supposed to accept

The node labels of these trees are Hintikka sets.

A set \( L \subseteq S \cup N_R \) is called Hintikka set if \( L = \emptyset \) or

- \( L \) contains exactly one role name occurring in \( S \);

- if \( \top \subseteq D \in \mathcal{T} \) then \( D \in L \);

- if \( C \cap D \in L \) then \( \{ C, D \} \subseteq L \);

- if \( C \cup D \in L \) then \( \{ C, D \} \cap L \neq \emptyset \);

- \( \{ A, \neg A \} \not\subseteq L \) for all concept names \( A \).

\[ \mathcal{H} \]

set of all Hintikka sets
Hintikka trees are the trees that our automata are supposed to accept.

The $k$-ary tree $h : \{0, \ldots, k - 1\}^* \rightarrow \mathcal{H}$ is a Hintikka tree for $\mathcal{T}$ and $C_0$ if

- $C_0 \in h(\varepsilon)$;
- For all nodes $u$, the tuple $(h(u), h(u0), \ldots, h(u(k - 1)))$ satisfies the following Hintikka successor conditions:
  - if $h(u) = \emptyset$ then $h(u_i) = \emptyset$ for all $i \in \{0, \ldots, k - 1\}$;
  - if $\exists r. C \in h(u)$ then there is an $i$ with $\{C, r\} \subseteq h(u_i)$;
  - if $\forall r. C \in h(u)$ and $r \in h(u_i)$ then $C \in h(u_i)$.

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

$A_{C_0,\mathcal{T}} := (Q, \Sigma, I, \Delta)$ where

- $Q := \Sigma := \mathcal{H}$; states and node labels are Hintikka sets
- $I := \{L \in Q \mid C_0 \in L\}$; initial states contain $C_0$
- $\Delta := \{(q, \sigma, q_0, \ldots, q_{k-1}) \in Q \times \Sigma \times Q^k \mid$
  \hspace{1cm} $q = \sigma$ and $(q, q_0, \ldots, q_{k-1})$ satisfies the Hintikka successor condition\}
  run identical to tree

The $k$-ary tree $h : \{0, \ldots, k - 1\}^* \to \mathcal{H}$ is accepted by $A_{C_0,\mathcal{T}}$

iff

it is a Hintikka tree for $\mathcal{T}$ and $C_0$
Main result

Satisfiability of $\mathcal{ALC}$-concept descriptions w.r.t. general $\mathcal{ALC}$-TBoxes can be decided in exponential time.

1. $C_0$ is satisfiable w.r.t. $\mathcal{T}$ iff there is a Hintikka tree for $\mathcal{T}$ and $C_0$
   iff $L(\mathcal{A}_{C_0,\mathcal{T}}) \neq \emptyset$

2. The size of $\mathcal{A}_{C_0,\mathcal{T}}$ is exponential in the size of $C_0$ and $\mathcal{T}$.

3. The emptiness test is polynomial in the size of $\mathcal{A}_{C_0,\mathcal{T}}$.

Note:

this bound is worst-case optimal since one can show
ExpTime hardness of the problem