Declarative Knowledge Processing
Lecture 4: Tableau Algorithms for $\mathcal{ALC}$

Magdalena Ortiz

Knowledge Base Systems Group
Institute of Information Systems

ortiz@kr.tuwien.ac.at

WS 2012/2013
Outline

1. Introduction

2. Tableau Algorithms for $ALC$
   2.1 Tableau Algorithm for concept satisfiability
   2.2 Tableau Algorithm for KB satisfiability

3. Summary
Tableau Algorithms for DLs

Tableau-based techniques try to decide the satisfiability of a formula or theory by using rules to construct (a representation of) a model.

- They have been used in FOL and MLs for many years.
- For DLs, they have been extensively explored since the late 1990s.
- They are considered well suited for implementation.
- In fact, many of the most successful DL reasoners implement tableau techniques or variations of them. E.g.: RACER, FaCT++, Pellet, Hermit, etc.

Note: We will talk more about reasoners and their reasoning techniques later.
Recommended Reading
for this lecture and the next


A Tableau Algorithm for $\mathcal{ALC}$ Concepts - overview

We will describe an algorithm that decides concept satisfiability. For an input $\mathcal{ALC}$ concept $C_0$, it tries to build a graph representation of a model $\mathcal{I}$ for $C_0$:

- It works with labeled, tree-shaped graphs $G$
  - the nodes are labeled with concepts, and
  - the arcs are labeled with roles.

- At any moment, the algorithm stores a set $\mathcal{G}$ of labeled graphs $G$

- It starts with one graph containing just one node labeled $C_0$

- It uses tableau rules corresponding to the constructors, to infer a new set $\mathcal{G}_{i+1}$ of graphs from the current set $\mathcal{G}_i$

- Intuitively, each new graph makes explicit some constraint resulting from $\mathcal{K}$ that was still implicit in the previous step
A Tableau Algorithm for $\mathcal{ALC}$ Concepts - overview (cont’d)

- Each rule may:
  - add new nodes to a graph $G$, or
  - add new labels to the existing nodes of a graph $G$.

- The rules are non-deterministic
  - Some rules may be applied in more than one way, resulting in different possible graphs

- If a graph contains a a clash, i.e. an explicit contradiction, it is dropped and not expanded further

- Otherwise it continues until some $G$ is complete, i.e. no more rules are applicable

- A complete and clash-free $G$ represents a model $\mathcal{I}$ of $C_0$
Negation Normal Form

- For simplicity, we assume input concepts in Negation Normal Form (NNF) \( \sim \) negation occurs only in front of concept names; no \( \top, \bot \).

- The NNF of \( C \) is obtained by ‘pushing negation inside’ with the following equivalences:

\[
\begin{align*}
\neg(\forall R. C) & \equiv \exists R. \neg C \\
\neg(C \sqcup D) & \equiv \neg C \sqcap \neg D \\
\neg\neg C & \equiv C
\end{align*}
\]

- Translation process terminates in linear time

- \( C \) and \( \text{nnf}(C) \) are equivalent, i.e., \( C^\mathcal{I} = \text{nnf}(C)^\mathcal{I} \) in every interpretation \( \mathcal{I} \)
Completion Graphs

Assume a given concept $C_0$ in NNF.

We denote by $\text{sub}(C_0)$ the set that contains all the subconcepts of $C$ and their negations (in NNF)\(^1\)

---

**Completion graph**

A completion graph for $C_0$ is a labeled graph $\langle V, E, L \rangle$, where

- $V$ is a finite set of nodes, and $E \subseteq V \times V$ the set of edges,
- $L$ is a labeling function that maps:
  - each node $v \in V$ to a set of concepts $L(v) \subseteq \text{sub}(C_0)$, and
  - each edge $(v, v')$ to a role $L(v, v')$.

---

\(^1\)Negations are not important now, but will be later
Initial Completion Graph, Clash

Initial Completion graph

The initial completion graph $G_0$ for $C_0$ is the graph that contains only one node $v_0$, no edges, and has $L(v_0) = \{C_0\}$.

A clash is an explicit contradiction in a completion graph

Clash, Clash Free Completion graph

A completion graph $G = \langle V, E, L \rangle$ contains a clash if for some $v \in V$ and some concept $C$, we have $\{C, \neg C\} \subseteq L(v)$.

A completion graph is called clash-free if it contains no clash.
Complete completion graph

The idea of the algorithm is to start from the initial completion graph, and to apply expansion rules until some clash free-graph is reached to which no more rules are applicable.

Complete Completion graph

A completion graph $G$ is complete if no expansion rule can be applied to it

We will see the expansion rules next.
Expansion Rules for $\mathcal{ALC}$ Concepts

<table>
<thead>
<tr>
<th>Expansion Rules for $\mathcal{ALC}$ concept satisfiability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>$\sqcap$-rule:</strong> if $C_1 \sqcap C_2 \in \mathcal{L}(v)$, and ${C_1, C_2} \not\subseteq \mathcal{L}(v)$ then $\mathcal{L}(v) := \mathcal{L}(v) \cup {C_1, C_2}$</td>
</tr>
<tr>
<td><strong>$\sqcup$-rule:</strong> if $C_1 \sqcup C_2 \in \mathcal{L}(v)$, and ${C_1, C_2} \cap \mathcal{L}(v) = \emptyset$ then $\mathcal{L}(v) := \mathcal{L}(v) \cup {D}$ for some $D \in {C_1, C_2}$</td>
</tr>
<tr>
<td><strong>$\exists$-rule:</strong> if $\exists R.C \in \mathcal{L}(v)$, there is no $w$ such that $\mathcal{L}(v, w) = R$ and $C \in \mathcal{L}(w)$ then create new node $w$ and an arc $(v, w)$, and set $\mathcal{L}(v, w) := R$ and $\mathcal{L}(w) := {C}$</td>
</tr>
<tr>
<td><strong>$\forall$-rule:</strong> if $\forall R.C \in \mathcal{L}(v)$, there is some $w$ such that $\mathcal{L}(v, w) = R$ and $C \not\in \mathcal{L}(w)$ then $\mathcal{L}(w) := \mathcal{L}(w) \cup {C}$</td>
</tr>
</tbody>
</table>
The Tableau Procedure for $\mathcal{ALC}$ satisfiability

1. Let $G_0 = \{G_0\}$ be the set that contains only the initial completion graph for $C_0$.

2. For $i \geq 0$, obtain the set $G_{i+1}$ of all clash-free graphs that can be obtained by applying an expansion rule to some $G \in G_i$.

3. If for some $i \geq 0$ there is a complete $G \in G_i$, then the algorithm answers yes.
   If for some $i \geq 0$ we have $G_i = \emptyset$, then the algorithm answers no.

Now we show that this yields a sound and complete algorithm for deciding concept satisfiability.

**Theorem**

The procedure terminates, and it answers yes iff $C_0$ is satisfiable.
Termination

Lemma

The tableau algorithm terminates

(Informal) proof sketch:

- Each completion graph $G$ is a finite tree:
  - its depth is linearly bounded by $|C_0|$ (in fact, by the quantifier depth)
  - its breadth is linearly bounded by $|C_0|$ (in fact, by the number of existentials)

- All concepts added to the labels are subconcepts of $C_0$, and all roles added to the arc labels occur in $C_0$. Hence the labels are finite.

- The graphs grow ‘monotonically’: no deleting and regenerating of nodes or labels

- Every completion graph $G$ obtained from $G_0$ will eventually be expanded into some $G'$ that either (a) contains a clash, or (b) is complete. Hence, the algorithm will eventually answer yes or no.
Completion graphs as Model Representations

A complete and clash-free completion graph can be seen as a representation of an interpretation

**Definition (Interpretation induced by a completion graph)**

Let $G$ be a completion graph. We define the interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ as follows:

- The domain $\Delta^{\mathcal{I}}$ are the nodes in $G$
- The interpretation function is given by the labels:
  - For each concept name $A$, $A^{\mathcal{I}} = \{v \mid A \in \mathcal{L}(v)\}$
  - For each role $R$, $R^{\mathcal{I}} = \{(v, w) \mid R = \mathcal{L}(v, w)\}$. 
Completion graphs as Model Representations (cont’d)

Then we can prove the following

**Lemma**

*Let $G$ be complete and clash-free completion graph. Then, for every $v$ and for every $ALC$ concept $C$

\[ C \in \mathcal{L}(v) \text{ implies } v \in C^\mathcal{I}_G. \]

**Proof.** By induction on the concept $C$ (exercise!).
## Soundness

With this lemma, it is very easy to show that the algorithm is sound:

### Lemma (L1)

Let \( G \) be a complete and clash-free completion graph for \( C_0 \) constructed by the algorithm. Then \( \mathcal{I}_G \models C_0 \).

### Proof.

By construction of \( G \), we know \( C_0 \in \mathcal{L}(v_0) \). Hence by the lemma above we have \( v_0 \in C_0 \mathcal{I}_G \), and thus \( \mathcal{I}_G \models C_0 \) as desired.

### Corollary (Soundness)

If the algorithm builds a complete and clash-free completion graph for \( C_0 \), then \( C_0 \) is satisfiable.
Simulating models in completion graphs

**Definition**

We say that an interpretation $\mathcal{I}$ is *simulated* by a completion graph $G = \langle V, E, \mathcal{L} \rangle$ is there exists a mapping $\pi : V \to \Delta^\mathcal{I}$ such that

- $C \in \mathcal{L}(v)$ implies $\pi(v) \in C^\mathcal{I}$
- $R \in \mathcal{L}(v, v')$ implies $(\pi(v), \pi(v')) \in R^\mathcal{I}$

**Note:** A completion graph that simulates an interpretation is always clash-free
Completeness

Lemma (L2)

If $\mathcal{I} \models C_0$, then there exists some $i \geq 0$ and some complete and clash-free $G \in G_i$ such that $G$ simulates $\mathcal{I}$.

Proof. [(sketch)] Roughly, we show that:

1. $G_0$ simulates $\mathcal{I}$

2. If some $G \in G_i$ simulates $\mathcal{I}$ and $G$ is not complete, then there is some $G' \in G_{i+1}$ that also simulates $\mathcal{I}$

Informal intuition: $\mathcal{I}$ shows us how to apply the expansion rules in such a way that the simulation is preserved

The claim follows from this and the fact that rule application eventually leads to a complete graph.

Corollary (Completeness)

If $C_0$ is satisfiable, then the algorithm builds a complete and clash-free $G$ for $C_0$. 

Tree shaped interpretation

It is not hard to see that the completion graphs generated by the algorithm, and the interpretations they induce, have a very specific shape:

**Definition**

An interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ is **tree-shaped** if the graph $\langle V, E \rangle$ with $V = \Delta^\mathcal{I}$ and $E = \{(d, d') \mid (d, d') \in R^\mathcal{I} \text{ for some role } R\}$ is a tree.

A simple inspection of the expansion rules reveals that each $\mathcal{I}_G$ induced from a constructed completion graph is tree-shaped.

Formally, we can show:

**Lemma (L3)**

If $\mathcal{I}_G$ is an interpretation induced by a completion graph $G$ obtained with the algorithm above, then it is **tree shaped**.
Tree model property

We have seen that:

- If $C_0$ has a model, then there is a complete and clash-free completion graph for $C_0$ (which simulates that model) Lemma L3

- If there is a complete and clash-free completion graph for $C_0$, then there is a tree-shaped model of $C_0$ (induced by that graph) Lemmas L1, L2

Hence, putting this together, we get that if $C_0$ has a model, then it has a tree shaped model

**Theorem**

*Every satisfiable ALC concept has a tree shaped model*
Tree model property (cont’d)

This is a very important and useful property

- We only need to look at tree shaped structures when reasoning about $\mathcal{ALC}$ concepts
- Trees are computationally ‘friendly’
- We can apply techniques for trees to obtain algorithms and complexity bounds

However, this property also exposes a limitation in the expressive power of $\mathcal{ALC}$ concepts

- Intuitively, they can not distinguish (non)tree-shapedness
- They can not describe, for example, structures with cycles

Note: The tree shaped property shows another way in which DLs are a decidable (semantic) fragment of FOL
Complexity

Lemma

_Satisfiability of ALC concepts can be decided in PSpace_

The tableau algorithm can be adapted to a non-deterministic version that uses only polynomial space. This gives an NPspace upper bound, and by Savitch’s Theorem, NPspace = PSpace.

Roughly, we only store one completion graph at any given time. For this completion graph:

- Each branch of the tree needs only polynomial space:
  - its depth is linear in $|C_0|$
  - the label of each node is also linear in $|C_0|$
- Branches are independent from each other
- We can construct the tree depth first, and re-use space from already constructed branches
A Tableau Algorithm for $\mathcal{ALC}$ KBs

We extend our algorithm to decide $KB$ satisfiability.

The algorithm is essentially the same: we start from the initial completion graph, and to apply expansion rules until some complete and clash-free graph is reached.

But there are a few differences:

- The initial graph is more complex: it is a representation of the ABox
- Arc labels in completion graphs are sets of roles instead of just one role (to allow pairs of individuals to be connected by multiple roles)
- The labeled graphs we obtained will not be trees, but forests
- The expansion rules will need slight extension/adaptation
Initial Completion Graph

Assume a given $\mathcal{ALC}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$.

**Initial Completion graph**

The initial completion graph $G_0 = \langle V, E, \mathcal{L} \rangle$ for $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ is defined as follows:

- $V$ it contains one node $\hat{a}$ for each individual $a$ occurring in $\mathcal{A}$
- each $\hat{a}$ has the label $\mathcal{L}(\hat{a}) = \{A \mid A(a) \in \mathcal{A}\}$
- there is an arc $(\hat{a}, \hat{b})$ with $R$ in its label iff $R(a, b) \in \mathcal{A}$
Ensuring the Satisfaction of the TBox

To ensure that the TBox $\mathcal{T}$ is satisfied we build a concept

$$C_\mathcal{T} = \bigcap_{C \sqsubseteq D \in \mathcal{T}} \neg C \sqcup D.$$  

and add a new expansion rule to make sure that $C_\mathcal{T}$ is satisfied everywhere:

$\mathcal{T}$-rule: if $C_\mathcal{T} \not\in \mathcal{L}(v)$, then $\mathcal{L}(v) := \mathcal{L}(v) \cup \{C_\mathcal{T}\}$

Lemma (L4)

An interpretation $\mathcal{I}$ is a model of $\mathcal{T}$ iff $C^\mathcal{I}_\mathcal{T} = \Delta^\mathcal{I}$

But the new rule causes a problem:

What happens, for example, if $C \sqsubseteq \exists R.C$ is in $\mathcal{T}$?
Blocking

The naive extension of the algorithm does not terminate!

**Idea:** to regain termination, avoid generating new successors for nodes that look exactly like some ancestor (cycle-detection).

**Definition (Blocking)**

Let $G = \langle V, E, \mathcal{L} \rangle$ be a completion graph, and let $v' \in V$ be reachable from $v \in V$.

- We say that $v'$ is **directly blocked** by $v$ if
  - $\mathcal{L}(v) = \mathcal{L}(v')$, and
  - there is no directly blocked $v''$ such that $v'$ is reachable from $v''$
- A node is **blocked** if it is directly blocked or it is reachable from a directly blocked node.

We restrict the application of the $\exists$-rule to nodes which are **not blocked**
# Expansion Rules for $\mathcal{ALC}$ KBs

<table>
<thead>
<tr>
<th>Expansion Rules for $\mathcal{ALC}$ KB satisfiability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>$\sqcap$-rule:</strong> if $C_1 \sqcap C_2 \in \mathcal{L}(v)$, and ${C_1, C_2} \notin \mathcal{L}(v)$</td>
</tr>
<tr>
<td>then $\mathcal{L}(v) := \mathcal{L}(v) \cup {C_1, C_2}$</td>
</tr>
<tr>
<td><strong>$\square$-rule:</strong> if $C_1 \sqcup C_2 \in \mathcal{L}(v)$, and ${C_1, C_2} \cap \mathcal{L}(v) = \emptyset$</td>
</tr>
<tr>
<td>then $\mathcal{L}(v) := \mathcal{L}(v) \cup {D}$ for some $D \in {C_1, C_2}$</td>
</tr>
<tr>
<td><strong>$\exists$-rule:</strong> if $\exists R.C \in \mathcal{L}(v)$, $v$ is not blocked,</td>
</tr>
<tr>
<td>and there is no $w$ such that $R \in \mathcal{L}(v,w)$ and $C \in \mathcal{L}(w)$</td>
</tr>
<tr>
<td>then create new node $w$ and an arc $(v,w)$,</td>
</tr>
<tr>
<td>and set $\mathcal{L}(v,w) := {R}$ and $\mathcal{L}(w) := {C}$</td>
</tr>
<tr>
<td><strong>$\forall$-rule:</strong> if $\forall R.C \in \mathcal{L}(v)$, there is some $w$ such that $R \in \mathcal{L}(v,w)$ and $C \notin \mathcal{L}(w)$</td>
</tr>
<tr>
<td>then $\mathcal{L}(w) := \mathcal{L}(w) \cup {C}$</td>
</tr>
<tr>
<td><strong>$T$-rule:</strong> if $C_T \notin \mathcal{L}(v)$,</td>
</tr>
<tr>
<td>then $\mathcal{L}(v) := \mathcal{L}(v) \cup {C_T}$</td>
</tr>
</tbody>
</table>
The Tableau Procedure for $\mathcal{K}$

The rest of the algorithm is exactly as for $\mathcal{ALC}$ concepts.

This extended tableau algorithm is a decision procedure for KB satisfiability.

**Theorem**

For an $\mathcal{ALC}$ KB $\mathcal{K}$, the procedure terminates, and it answers yes iff $\mathcal{K}$ is satisfiable.
Termination

Lemma

The tableau algorithm terminates

- Each graph $G$ is a forest:
  - it has one root node for each individual in $\mathcal{A}$
  - blocking ensures that the depth of each branch is finite (bounded by an exponential in $|\mathcal{K}|$)
  - Its breadth is still linearly bounded by $|\mathcal{K}|$ (in fact, by the number of existentials in $\mathcal{T}$)

Hence the generated graphs are always finite

- And as before:
  - All concepts added to the labels occur in $\mathcal{A}$ or in $C_\mathcal{T}$
  - $G$ is constructed ‘monotonically’: no deleting and regenerating of nodes or labels
Soundness

Similarly as before, a complete and clash-free $G$ induces a model $\mathcal{I}_G$ of $\mathcal{K}$, but the induced model is slightly different

**Definition (Interpretation induced by a completion graph)**

Let $G$ be a completion graph for $\mathcal{K}$. We define the interpretation $\mathcal{I} = (\Delta I, \cdot I)$ as follows:

- The domain $\Delta I$ are the nodes in $G$ that are not blocked
- Individuals are interpreted as the corresponding initial node $a^I = \hat{a}$
- The rest of interpretation function is given by the labels, but taking blocked nodes into account:
  - For each concept name $A$, $A^I = \{v \mid A \in \mathcal{L}(v)\}$
  - For each role $R$,
    $$R^I = \{(v, w) \mid R \in \mathcal{L}(v, w)\} \cup \{(v, w') \mid R \in \mathcal{L}(v, w) \text{ and } w' \text{ blocks } w\}.$$
Soundness (cont’d)

Lemma

Let $G$ be a complete and clash-free completion graph for $\mathcal{K}$ constructed by the algorithm. Then $\mathcal{I}_G \models \mathcal{K}$.

Proof.

1. As we did before, we show that if $G$ is complete and clash-free, then for every $v$ and for every concept $C$, $C \in \mathcal{L}(v)$ implies $v \in C^{\mathcal{I}_G}$.

2. By construction of the initial graph, we know that:
   - $C \in \mathcal{L}(\hat{a})$ for each $C(a) \in \mathcal{A}$, and
   - $R \in \mathcal{L}(\hat{a}, \hat{b})$ for each $R(a, b) \in \mathcal{A}$.

   So, by construction of $\mathcal{I}_G$ and item 1, $\mathcal{I}_G \models \mathcal{A}$

3. Since $C_T \in \mathcal{L}(v)$ for every $v$, we have $C_T^{\mathcal{I}_G} = \Delta^{\mathcal{I}_G}$ and then (by Lemma L4) $\mathcal{I}_G \models T$.

4. $\mathcal{I}_G \models \mathcal{A}$ and $\mathcal{I}_G \models T$ imply $\mathcal{I}_G \models K$. 

\[\square\]
Soundness (cont’d)

Corollary (Soundness)

*If the algorithm builds a complete and clash-free completion graph for $\mathcal{K}$, then $\mathcal{K}$ is satisfiable.*
Completeness

Lemma

If $\mathcal{K}$ is satisfiable, then the algorithm builds a complete and clash-free $G$ for $\mathcal{K}$.

Proof.[sketch] As before, we show that every model of $\mathcal{K}$ is simulated by a complete and clash free completion graph that is constructed by the algorithm. The notion of simulation $\pi$ is very similar, but it only needs to map the non-blocked nodes, and additionally we require $\pi(\hat{a}) = a^I$ for the initial nodes.
Forest model property

- Completion graphs for a knowledge base are not necessarily tree-shaped: arbitrary relations between individuals

- However, a completion graph is composed of a set of trees rooted at the (possibly interconnected) objects representing the individuals

**Definition**

An interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ is forest-shaped if the graph $\langle V, E \rangle$ with $V = \Delta^\mathcal{I}$ and

$$E = \{ (d, d') \mid (d, d') \in R^\mathcal{I} \text{ for some role } R \text{ and } d, d' \not\in \{ a^\mathcal{I} \mid a \in N_I \} \}$$

is a set of (disconnected) trees.
Forest model property (cont’d)

The model $I_G$ is not forest-shaped in general (the blocked nodes create cycles), but it can be shown that the following property holds:

**Theorem (Forest model property)**

*Every satisfiable ALC KB has a forest-shaped model*

Unlike the case of ALC concepts, trees may now be infinite!

This property is practically as good (and as restrictive as) as the tree-model property of ALC concepts

**Note:** Many DLs have similar tree/forest model properties, but in some cases we need to adapt slightly the definition of tree/forest shaped models
Complexity

The complexity of the tableau algorithm is not optimal:

- The forest can be very big:
  - branches in the forest can have exponential depth
  - the whole forest can be double exponentially large
- The overall algorithm needs non-deterministic exponential time (in $2\text{NExpTime}$)
- With some adaptations and modified blocking strategies, one can make forests be of at most single exponential size
- This could give us a non-deterministic exponential upper bound (in $\text{NExpTime}$)
- Reasoning in $\mathcal{ALC}$ is ‘only’ ExpTime-hard
- To obtain worst-case optimal decision procedures we need different techniques
Some of the Main Ideas of Today

- We became familiar with DL tableau algorithms

- We presented tableau algorithms for reasoning in $\mathcal{ALC}$ about concepts only and about KBs (and saw that the latter is more involved than the former)

- We learned how to prove soundness and completeness of tableau algorithms

- We discussed the tree/forest model property of $\mathcal{ALC}$

- We saw that tableau algorithms do not always give optimal complexity bounds

In the next lecture, we will see how tableau algorithms can be extended to other DLs, and discuss some optimizations