Gerd Gröner, Matthias Thimm
{groener,thimm}@uni-koblenz.de

Institute for Web Science and Technologies (WeST)
University of Koblenz-Landau

July 9, 2013
Recap

- Ontologies: definition and applications
- Structure of an ontology
- The description logic $\mathcal{ALC}$:
  - Syntax: signature, concepts, relations, axioms
  - Semantic: interpretations, models
  - Inference: entailment
  - $\mathcal{ALC}$ and first-order logics
Recap

- Ontologies: definition and applications
- Structure of an ontology
- The description logic $\mathcal{ALC}$:
  - Syntax: signature, concepts, relations, axioms
  - Semantic: interpretations, models
  - Inference: entailment
  - $\mathcal{ALC}$ and first-order logics

Home assignment:
Consider $\mathcal{K}_2 = (\mathcal{T}_2, \mathcal{A}_2)$ given via

$$\mathcal{T}_2 = \{ \quad B \sqsubseteq D \quad \}$$
$$\mathcal{A}_2 = \{ \quad d : B, (c, d) : S \quad \}$$

Is the entailment $\mathcal{K}_2 \models c : \forall S . D$ valid? (Home assignment)
Formal syntax and semantics provide the basis for understanding description logics
Formal syntax and semantics provide the basis for understanding description logics.

Implementing the semantics of e.g. $\mathcal{ALC}$ is intractable for obtaining a proof procedure.
- Generate all interpretations
- Check whether an interpretation satisfies a potential conclusions
Overview

- Formal syntax and semantics provide the basis for understanding description logics
- Implementing the semantics of e.g. $\mathcal{ALC}$ is intractable for obtaining a proof procedure
  - Generate all interpretations
  - Check whether an interpretation satisfies a potential conclusions
- Today we have a look at a very simple proof procedure for deciding consistency: the *Tableau Algorithm*
Overview

- Formal syntax and semantics provide the basis for understanding description logics
- Implementing the semantics of e.g. ALC is intractable for obtaining a proof procedure
  - Generate all interpretations
  - Check whether an interpretation satisfies a potential conclusions
- Today we have a look at a very simple proof procedure for deciding consistency: the Tableau Algorithm
- We also take another look at ontology languages in general and applications

Gerd Gröner, Matthias Thimm
Outline

1 Reasoning with Description Logics
2 Ontology languages revisited
3 Tools
4 Summary and Exercises
Outline

1. Reasoning with Description Logics
2. Ontology languages revisited
3. Tools
4. Summary and Exercises
In description logics one usually distinguishes the following inference tasks:

- **Subsumption problem**: Given concepts \( C_1, C_2 \) does \( \mathcal{K} \) entail \( C_1 \sqsubseteq C_2 \)?
- **Instance checking problem**: Given concept \( C \) and individual \( t \) does \( \mathcal{K} \) entail \( t : C \)?
- **Relation checking problem**: Given relation \( R \) and individuals \( t, t' \) does \( \mathcal{K} \) entail \( (t, t') : R \)?
- **Consistency problem**: Is there \( \mathcal{I} \) with \( \mathcal{I} \models \mathcal{K} \)?
The subsumption and the consistency problem are closely related:

- $C_2$ subsumes $C_1$ if $C_1$ and $\neg C_2$ are inconsistent:

$$
\mathcal{K} \models C_1 \sqsubseteq C_2 \quad \text{if and only if} \quad \neg \exists \mathcal{I} : \mathcal{I} \models \mathcal{K} \cup \{ t : (C_1 \cap \neg C_2) \}
$$
The subsumption and the consistency problem are closely related:

- $C_2$ subsumes $C_1$ if $C_1$ and $\neg C_2$ are inconsistent:

$$\mathcal{K} \models C_1 \sqsubseteq C_2 \quad \text{if and only if} \quad \neg \exists \mathcal{I} : \mathcal{I} \models \mathcal{K} \cup \{ t : (C_1 \cap \neg C_2) \}$$

→ it suffices to investigate algorithms for checking consistency.
The Tableau Algorithm

Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base and consider the question

$$\exists \mathcal{I} : \mathcal{I} \models \mathcal{K} ?$$
The Tableau Algorithm

Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base and consider the question

$$\exists \mathcal{I} : \mathcal{I} \models \mathcal{K} ?$$

The tableau algorithm tries to construct a model $\mathcal{I}$ of $\mathcal{K}$:

- If this is successful, $\mathcal{K}$ is consistent
- otherwise it is inconsistent
The Tableau Algorithm

Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base and consider the question

$$\exists \mathcal{I} : \mathcal{I} \models \mathcal{K} ?$$

The tableau algorithm tries to construct a model $\mathcal{I}$ of $\mathcal{K}$:

- If this is successful, $\mathcal{K}$ is consistent
- otherwise it is inconsistent

It works on a set $\mathcal{S}$ of ABoxes and iteratively expands on it:

- $\mathcal{S}$ is initialized with the singleton $\mathcal{S} = \{ \mathcal{A} \}$
The Tableau Algorithm

Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base and consider the question

$$\exists \mathcal{I} : \mathcal{I} \models \mathcal{K} ?$$

The *tableau algorithm* tries to construct a model $\mathcal{I}$ of $\mathcal{K}$:

- If this is successful, $\mathcal{K}$ is consistent
- Otherwise it is inconsistent

It works on a set $\mathcal{S}$ of ABoxes and iteratively expands on it:

- $\mathcal{S}$ is initialized with the singleton $\mathcal{S} = \{ \mathcal{A} \}$
- Apply different rules on the elements of $\mathcal{S}$ depending on the axioms in $\mathcal{T}$
Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base and consider the question

$$\exists \mathcal{I} : \mathcal{I} \models \mathcal{K} ?$$

The *tableau algorithm* tries to construct a model $\mathcal{I}$ of $\mathcal{K}$:

- If this is successful, $\mathcal{K}$ is consistent
- otherwise it is inconsistent

It works on a set $\mathcal{S}$ of ABoxes and iteratively expands on it:

- $\mathcal{S}$ is initialized with the singleton $\mathcal{S} = \{ \mathcal{A} \}$
- Apply different rules on the elements of $\mathcal{S}$ depending on the axioms in $\mathcal{T}$
- If no more rules are applicable, $\mathcal{K}$ is consistent if there is an ABox $\mathcal{A}'$ in $\mathcal{S}$ that is consistent (contains no axioms $t : C$, $t : \neg C$)
In order to apply the tableau algorithm we have to assume that $\mathcal{K}$ is in \textit{negation normal form}:

- $\neg(C_1 \sqcup C_2) \rightarrow \neg C_1 \sqcap \neg C_2$
- $\neg(C_1 \sqcap C_2) \rightarrow \neg C_1 \sqcup \neg C_2$
- $\neg\exists R. C \rightarrow \forall R. (\neg C)$
- $\neg\forall R. C \rightarrow \exists R. (\neg C)$

From now on, we assume that every concept appearing in an axiom in $\mathcal{K}$ is in \textit{negation normal form}. For example $\neg(C_1 \sqcap C_2) \sqsubseteq \neg\exists R. (\neg C)$. 
In order to apply the tableau algorithm we have to assume that $\mathcal{K}$ is in negation normal form:

- $\neg(C_1 \sqcup C_2) \rightarrow \neg C_1 \sqcap \neg C_2$
- $\neg(C_1 \sqcap C_2) \rightarrow \neg C_1 \sqcup \neg C_2$
- $\neg \exists R. C \rightarrow \forall R. (\neg C)$
- $\neg \forall R. C \rightarrow \exists R. (\neg C)$

From now on, we assume that every concept appearing in an axiom in $\mathcal{K}$ is in negation normal form.
In order to apply the tableau algorithm we have to assume that $\mathcal{K}$ is in *negation normal form*:

- $\neg(C_1 \sqcup C_2) \rightarrow \neg C_1 \sqcap \neg C_2$
- $\neg(C_1 \sqcap C_2) \rightarrow \neg C_1 \sqcup \neg C_2$
- $\neg\exists R. C \rightarrow \forall R. (\neg C)$
- $\neg\forall R. C \rightarrow \exists R. (\neg C)$

From now on, we assume that every concept appearing in an axiom in $\mathcal{K}$ is in negation normal form. For example

$\neg(C_1 \sqcap C_2) \sqsubseteq \neg\exists R. \neg(C_3 \sqcup C_4) \rightarrow \neg C_1 \sqcup \neg C_2 \sqsubseteq \forall R. (C_3 \sqcup C_4)$
Let $S$ be a set of ABoxes (initialized with $S = \{A\}$).
Let $S$ be a set of ABoxes (initialized with $S = \{ \mathcal{A} \}$).

Let $\mathcal{A}' \in S$.
Let $S$ be a set of ABoxes (initialized with $S = \{A\}$).

Let $A' \in S$.

- $\cap$-rule: if $t : C_1 \cap C_2 \in A'$ and $\{t : C_1, t : C_2\} \not\subseteq A'$ then remove $A'$ from $S$ and add $A' \cup \{t : C_1, t : C_2\}$ to $S$. 
Let $S$ be a set of ABoxes (initialized with $S = \{A\}$).

Let $A' \in S$.

- $\sqcap$-rule: if $t : C_1 \sqcap C_2 \in A'$ and $\{t : C_1, t : C_2\} \not\subseteq A'$ then remove $A'$ from $S$ and add $A' \cup \{t : C_1, t : C_2\}$ to $S$.

- $\sqcup$-rule: if $t : C_1 \sqcup C_2 \in A'$ and $\{t : C_1, t : C_2\} \cap A' = \emptyset$ then remove $A'$ from $S$ and add both $A' \cup \{t : C_1\}$ and $A' \cup \{t : C_2\}$ to $S$. 

Gerd Gröner, Matthias Thimm

Semantic Web
Let $S$ be a set of ABoxes (initialized with $S = \{A\}$).

Let $A' \in S$.

- **⊓-rule**: if $t : C_1 \sqcap C_2 \in A'$ and $\{t : C_1, t : C_2\} \not\subseteq A'$ then remove $A'$ from $S$ and add $A' \cup \{t : C_1, t : C_2\}$ to $S$.

- **⊔-rule**: if $t : C_1 \sqcup C_2 \in A'$ and $\{t : C_1, t : C_2\} \cap A' = \emptyset$ then remove $A'$ from $S$ and add both $A' \cup \{t : C_1\}$ and $A' \cup \{t : C_2\}$ to $S$.

- **∃-rule**: if $t : \exists R.C \in A'$ and there is no $t'$ with $\{(t, t') : R, t' : C\} \subseteq A'$ then remove $A'$ from $S$, create a new individual $t''$, and add $A' \cup \{(t, t'') : R, t'' : C\}$ to $S$. 

Gerd Gröner, Matthias Thimm

Semantic Web
Let $S$ be a set of ABoxes (initialized with $S = \{A\}$).
Let $S$ be a set of ABoxes (initialized with $S = \{A\}$).
Let $A' \in S$. 
Let $S$ be a set of ABoxes (initialized with $S = \{A\}$).

Let $A' \in S$.

▶ ∀-rule: if \[
\{ t : \forall R.C, (t, t') : R \} \subseteq A' \text{ and } \{ t' : C \} \not\in A' \]
then remove $A'$ from $S$ and add $A' \cup \{ t' : C \}$ to $S$. 

▶ ⊑-rule: if $C_1 \sqsubseteq C_2 \in T$ and $t : (\neg C_1 \sqcup C_2) \not\in A'$ for $t$ appearing in $A'$ then remove $A'$ from $S$ and add $A' \cup \{ t : (\neg C_1 \sqcup C_2) \}$ to $S$. 
Let $S$ be a set of ABoxes (initialized with $S = \{A\}$).

Let $A' \in S$.

- $\forall$-rule: if $\{t : \forall R.C, (t, t') : R\} \subseteq A'$ and $\{t' : C\} \notin A'$ then remove $A'$ from $S$ and add $A' \cup \{t' : C\}$ to $S$.

- $\sqsubseteq$-rule: if $C_1 \sqsubseteq C_2 \in \mathcal{T}$ and $t : (\neg C_1 \sqcup C_2) \notin A'$ for $t$ appearing in $A'$ then remove $A'$ from $S$ and add $A' \cup \{t : (\neg C_1 \sqcup C_2)\}$ to $S$. 
Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and $\mathcal{A}$ be consistent.
Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and $\mathcal{A}$ be consistent.

The tableau algorithm:

1. Set $\mathcal{S} = \{\mathcal{A}\}$
2. Is some rule applicable?
   - yes: goto 3
   - no: $\mathcal{K}$ is consistent; exit
3. Apply the rule to $\mathcal{S}$
4. Remove all $\mathcal{A}' \in \mathcal{S}$ with $t : C, t : \neg C \in \mathcal{A}'$ (for some $t, C$)
5. $\mathcal{S} = \emptyset$?
   - yes: $\mathcal{K}$ is inconsistent; exit
   - no: goto 2
Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be given via

\[
\mathcal{T} = \{ \quad A \sqsubseteq \exists R.C \quad \}
\]
\[
\mathcal{A} = \{ \quad a : A, b : D, (a, b) : R \quad \}
\]
Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be given via

$$\mathcal{T} = \{ A \subseteq \exists R.C \}$$
$$\mathcal{A} = \{ a : A, b : D, (a, b) : R \}$$

Observe:

- $\mathcal{K}$ is in negation normal form
- $\mathcal{A}$ is consistent
Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be given via

$$\mathcal{T} = \{ A \sqsubseteq \exists R . C \}$$
$$\mathcal{A} = \{ a : A, b : D, (a, b) : R \}$$

Observe:

- $\mathcal{K}$ is in negation normal form
- $\mathcal{A}$ is consistent

Initialize $S = \{ \{ a : A, b : D, (a, b) : R \} \}$. 
\[ T = \{ A \subseteq \exists R.C \} \]
\[ A = \{ a : A, b : D, (a, b) : R \} \]
\[ S = \{ \{ a : A, b : D, (a, b) : R \} \} \]
The Tableau Algorithm - Example cont’d

\[ T = \{ \ A \sqsubseteq \exists R \cdot C \ \} \]
\[ \mathcal{A} = \{ \ a : A, b : D, (a, b) : R \ \} \]
\[ S = \{ \{a : A, b : D, (a, b) : R\}\} \]

\sqsubseteq\text{-rule: if } C_1 \sqsubseteq C_2 \in T \text{ and } t : (\neg C_1 \sqcup C_2) \notin \mathcal{A}' \text{ for } t \text{ appearing in } \mathcal{A}' \text{ then remove } \mathcal{A}' \text{ from } S \text{ and add } \mathcal{A}' \cup \{ t : (\neg C_1 \sqcup C_2) \} \text{ to } S. \]
The Tableau Algorithm - Example cont’d

\[ T = \{ \; A \sqsubseteq \exists R.C \; \} \]
\[ \mathcal{A} = \{ \; a : A, b : D, (a, b) : R \; \} \]
\[ S = \{\{a : A, b : D, (a, b) : R\}\} \]

\sqsubseteq\text{-rule: if } C_1 \sqsubseteq C_2 \in T \text{ and } t : (\neg C_1 \sqcup C_2) \notin \mathcal{A}' \text{ for } t \text{ appearing in } \mathcal{A}' \text{ then remove } \mathcal{A}' \text{ from } S \text{ and add } \mathcal{A}' \cup \{t : (\neg C_1 \sqcup C_2)\} \text{ to } S.\]

\[ S = \{\{a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C\}\} \]

WeST
People and Knowledge Networks
\( \mathcal{T} = \{ \ A \sqsubseteq \exists R.C \ \} \)

\( \mathcal{A} = \{ \ a : A, b : D, (a, b) : R \ \} \)

\( S = \{ \{a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C\} \} \)
The Tableau Algorithm - Example cont’d

\[ \mathcal{T} = \{ \ A \sqsubseteq \exists R.C \ \} \]
\[ \mathcal{A} = \{ \ a : A, b : D, (a, b) : R \ \} \]
\[ \mathcal{S} = \{ \{a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C\}\} \]

⊔-rule: if \( t : C_1 \sqcup C_2 \in \mathcal{A}' \) and \( \{t : C_1, t : C_2\} \cap \mathcal{A}' = \emptyset \) then remove \( \mathcal{A}' \) from \( \mathcal{S} \) and add both \( \mathcal{A}' \cup \{t : C_1\} \) and \( \mathcal{A}' \cup \{t : C_2\} \) to \( \mathcal{S} \).
The Tableau Algorithm - Example cont’d

\[ T = \{ A \sqsubseteq \exists R.C \} \]
\[ A = \{ a : A, b : D, (a, b) : R \} \]
\[ S = \{ \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C \} \} \]

\[ \Box \text{-rule: if } t : C_1 \sqcup C_2 \in A' \text{ and } \{ t : C_1, t : C_2 \} \cap A' = \emptyset \text{ then} \]
remove \( A' \) from \( S \) and add both \( A' \cup \{ t : C_1 \} \) and \( A' \cup \{ t : C_2 \} \) to \( S \).

\[ S = \{ \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \neg A \}, \]
\[ \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \exists R.C \} \} \]
The Tableau Algorithm - Example cont’d

\[ T = \{ \quad A \sqsubseteq \exists R.C \quad \} \]
\[ A = \{ \quad a : A, b : D, (a, b) : R \quad \} \]
\[ S = \{ \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C \} \} \]

\[ \sqcap\text{-rule: if } t : C_1 \sqcup C_2 \in \mathcal{A}' \text{ and } \{ t : C_1, t : C_2 \} \cap \mathcal{A}' = \emptyset \text{ then remove } \mathcal{A}' \text{ from } S \text{ and add both } \mathcal{A}' \cup \{ t : C_1 \} \text{ and } \mathcal{A}' \cup \{ t : C_2 \} \text{ to } S. \]

\[ S = \{ \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \neg A \}, \leftarrow \]
\[ \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \exists R.C \} \} \]
The Tableau Algorithm - Example cont’d

\[ \mathcal{T} = \{ \, A \sqsubseteq \exists R.C \, \} \]
\[ \mathcal{A} = \{ \, a : A, b : D, (a, b) : R \, \} \]
\[ S = \{ \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C \} \} \]

\(
\bigcup\)-rule: if \( t : C_1 \sqcup C_2 \in \mathcal{A}' \) and \( \{ t : C_1, t : C_2 \} \cap \mathcal{A}' = \emptyset \) then remove \( \mathcal{A}' \) from \( S \) and add both \( \mathcal{A}' \cup \{ t : C_1 \} \) and \( \mathcal{A}' \cup \{ t : C_2 \} \) to \( S \).

\[ S = \{ \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \neg A \}, \]
\[ \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \exists R.C \} \} \]
The Tableau Algorithm - Example cont’d

\[ T = \{ \ A \sqsubset \exists R. C \ \} \]

\[ A = \{ \ a : A, b : D, (a, b) : R \ \} \]

\[ S = \{ \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R. C, a : \exists R. C \} \} \]
The Tableau Algorithm - Example cont’d

\[ T = \{ A \sqsubseteq \exists R.C \} \]
\[ A = \{ a : A, b : D, (a, b) : R \} \]
\[ S = \{ \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \exists R.C \} \} \]

\(\exists\)-rule: if \( t : \exists R.C \in A' \) and there is no \( t' \) with \( \{(t, t') : R, t' : C\} \subseteq A' \) then remove \( A' \) from \( S \), create a new individual \( t'' \), and add \( A' \cup \{(t, t'') : R, t'' : C\} \) to \( S \).

\[ S = \{ \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \exists R.C, (a, t'') : R, t'' : C \} \} \]
$T = \{ \ A \sqsubseteq \exists R.C \ \}$

$A = \{ a : A, b : D, (a, b) : R \ \}$

$S = \{ \{a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \exists R.C, \ (a, t^{'''}) : R, t^{'''} : C\} \}$
\[ \mathcal{T} = \{ \quad A \sqsubseteq \exists R.C \quad \} \]
\[ \mathcal{A} = \{ \quad a : A, b : D, (a, b) : R \quad \} \]
\[ S = \{ \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \exists R.C, \\
(a, t'') : R, t'' : C \} \} \]

\(\sqsubseteq\)-rule: if \(C_1 \sqsubseteq C_2 \in \mathcal{T}\) and \(t : (\neg C_1 \sqcup C_2) \notin \mathcal{A}'\) for \(t\) appearing in \(\mathcal{A}'\) then remove \(\mathcal{A}'\) from \(S\) and add \(\mathcal{A}' \cup \{ t : (\neg C_1 \sqcup C_2) \}\) to \(S\).
The Tableau Algorithm - Example cont’d

\[ T = \{ \; A \sqsubseteq \exists R.C \; \} \]
\[ A = \{ \; a : A, b : D, (a, b) : R \; \} \]
\[ S = \{ \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \exists R.C, \\
(a, t'') : R, t'' : C \} \} \]

\[ \sqsubseteq \text{-rule: if } C_1 \sqsubseteq C_2 \in T \text{ and } t : (\neg C_1 \sqcup C_2) \notin A' \text{ for } t \text{ appearing in } A' \text{ then remove } A' \text{ from } S \text{ and add } A' \cup \{ t : (\neg C_1 \sqcup C_2) \} \text{ to } S. \]

\[ S = \{ \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \exists R.C, (a, t'') : R, t'' : C, \\
b : \neg A \sqcup \exists R.C \} \} \]
The Tableau Algorithm - Example cont’d

\[ T = \{ A \sqsubseteq \exists R.C \} \]
\[ A = \{ a : A, b : D, (a, b) : R \} \]
\[ S = \{ \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \exists R.C, (a, t'') : R, t'' : C, b : \neg A \sqcup \exists R.C \} \} \]
The Tableau Algorithm - Example cont’d

\[ T = \{ \ A \sqsubseteq \exists R.C \ \} \]
\[ A = \{ \ a : A, b : D, (a, b) : R \ \} \]
\[ S = \{ \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \exists R.C, (a, t'') : R, t'' : C, b : \neg A \sqcup \exists R.C \} \} \]

\[ \sqcup\text{-rule: if } t : C_1 \sqcup C_2 \in A' \text{ and } \{ t : C_1, t : C_2 \} \cap A' = \emptyset \text{ then remove } A' \text{ from } S \text{ and add both } A' \cup \{ t : C_1 \} \text{ and } A' \cup \{ t : C_2 \} \text{ to } S. \]
The Tableau Algorithm - Example cont’d

\[ T = \{ \ A \sqsubseteq \exists R.C \ \} \]

\[ A = \{ \ a : A, b : D, (a, b) : R \ \} \]

\[ S = \{ \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \exists R.C, (a, t'') : R, t'' : C, \]

\[ b : \neg A \sqcup \exists R.C \} \}\]

\[ \sqcup \text{-rule: if } t : C_1 \sqcup C_2 \in A' \text{ and } \{ t : C_1, t : C_2 \} \cap A' = \emptyset \text{ then remove } A' \text{ from } S \text{ and add both } A' \cup \{ t : C_1 \} \text{ and } A' \cup \{ t : C_2 \} \text{ to } S. \]

\[ S = \{ \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \exists R.C, (a, t'') : R, t'' : C, \]

\[ b : \neg A \sqcup \exists R.C, b : \neg A \}, \]

\[ \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \exists R.C, (a, t'') : R, t'' : C, \]

\[ b : \neg A \sqcup \exists R.C, b : \exists R.C \}\} \]
\[
T = \{ A \sqsubseteq \exists R.C \} \\
A = \{ a : A, b : D, (a, b) : R \} \\
S = \{ \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \exists R.C, (a, t''') : R, t''' : C, \\
       b : \neg A \sqcup \exists R.C, b : \neg A \}, \\
    \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \exists R.C, (a, t'') : R, t'' : C, \\
       b : \neg A \sqcup \exists R.C, b : \exists R.C \} \}
\]
The Tableau Algorithm - Example cont’d

\[\mathcal{T} = \{ \ A \sqsubseteq \exists R.C \} \]
\[\mathcal{A} = \{ \ a : A, b : D, (a, b) : R \} \]
\[\mathcal{S} = \{ \{a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \exists R.C, (a, t'') : R, t'' : C, b : \neg A \sqcup \exists R.C, b : \exists R.C\}, \\
\{a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \exists R.C, (a, t'') : R, t'' : C, b : \neg A \sqcup \exists R.C, b : \exists R.C\}\} \]

\(\exists\)-rule: if \(t : \exists R.C \in \mathcal{A}'\) and there is no \(t'\) with \(\{(t, t') : R, t' : C\} \subseteq \mathcal{A}'\) then remove \(\mathcal{A}'\) from \(\mathcal{S}\), create a new individual \(t''\), and add \(\mathcal{A}' \cup \{(t, t'') : R, t'' : C\}\) to \(\mathcal{S}\).
The Tableau Algorithm - Example cont’d

\[ T = \{ A \sqsubseteq \exists R.C \} \]
\[ A = \{ a : A, b : D, (a, b) : R \} \]
\[ S = \{ \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \exists R.C, (a, t'') : R, t'' : C, \\
   b : \neg A \sqcup \exists R.C, b : \neg A \}, \\
   \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \exists R.C, (a, t'') : R, t'' : C, \\
   b : \neg A \sqcup \exists R.C, b : \exists R.C \} \} \]

∃-rule: if \( t : \exists R.C \in A' \) and there is no \( t' \) with \( \{(t, t') : R, t' : C\} \subset A' \) then remove \( A' \) from \( S \), create a new individual \( t'' \), and add \( A' \cup \{(t, t'') : R, t'' : C\} \) to \( S \).

\[ S = \{ \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \exists R.C, (a, t'') : R, t'' : C, \\
   b : \neg A \sqcup \exists R.C, b : \neg A \}, \\
   \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \exists R.C, (a, t'') : R, t'' : C, \\
   b : \neg A \sqcup \exists R.C, b : \exists R.C \} \} \]
\[ T = \{ \quad A \sqsubseteq \exists R.C \quad \} \]
\[ A = \{ \quad a : A, b : D, (a, b) : R \quad \} \]
\[ S = \{\{a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \exists R.C, (a, t'') : R, t'' : C, \\
\quad b : \neg A \sqcup \exists R.C, b : \neg A\}, \]
\[ \{a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \exists R.C, (a, t'') : R, t'' : C, \\
\quad b : \neg A \sqcup \exists R.C, b : \exists R.C, (b, t''') : R, t''' : C\}\} \]
\( \mathcal{T} = \{ \quad A \sqsubseteq \exists R.C \quad \} \)

\( \mathcal{A} = \{ \quad a : A, b : D, (a, b) : R \quad \} \)

\( \mathcal{S} = \{ \{a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \exists R.C, (a, t'') : R, t'' : C, \\
    b : \neg A \sqcup \exists R.C, b : \neg A\}, \\
    \{a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \exists R.C, (a, t'') : R, t'' : C, \\
    b : \neg A \sqcup \exists R.C, b : \exists R.C, (b, t'''') : R, t''' : C\}\} \)

Observations:

- No more rules applicable
The Tableau Algorithm - Example cont’d

\[ T = \{ A \sqsubseteq \exists R. C \} \]
\[ A = \{ a : A, b : D, (a, b) : R \} \]
\[ S = \{ \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R. C, a : \exists R. C, (a, t'') : R, t'' : C, b : \neg A \sqcup \exists R. C, b : \exists A \}, \]
\[ \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R. C, a : \exists R. C, (a, t'') : R, t'' : C, b : \neg A \sqcup \exists R. C, b : \exists R. C, (b, t'''') : R, t''' : C \} \}

Observations:

- No more rules applicable
- All ABoxes in \( S \) are consistent
The Tableau Algorithm - Example cont’d

\[ T = \{ A \sqsubseteq \exists R.C \} \]
\[ \mathcal{A} = \{ a : A, b : D, (a, b) : R \} \]
\[ \mathcal{S} = \{ \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \exists R.C, (a, t'') : R, t'' : C, \\
  b : \neg A \sqcup \exists R.C, b : \neg A \}, \\
  \{ a : A, b : D, (a, b) : R, a : \neg A \sqcup \exists R.C, a : \exists R.C, (a, t'') : R, t'' : C, \\
  b : \neg A \sqcup \exists R.C, b : \exists R.C, (b, t'''') : R, t''' : C \} \} \]

Observations:

- No more rules applicable
- All ABoxes in \( \mathcal{S} \) are consistent

\( \Rightarrow \) The knowledge base \( \mathcal{K} \) is consistent.
The Tableau Algorithm - Observations

- If a rule has been applied to an ABox in $S$ it will never be applied again (with the same parameters)
The Tableau Algorithm - Observations

- If a rule has been applied to an ABox in $S$ it will never be applied again (with the same parameters)
- Only the $\sqcap$-rule adds a new ABox to $S$
If a rule has been applied to an ABox in $S$ it will never be applied again (with the same parameters)

Only the $\sqcup$-rule adds a new ABox to $S$

If $\mathcal{A}'$ replaces $\mathcal{A}$ then $\mathcal{A} \subseteq \mathcal{A}'$
The Tableau Algorithm - Observations

- If a rule has been applied to an ABox in $S$ it will never be applied again (with the same parameters)
- Only the $\sqcup$-rule adds a new ABox to $S$
- If $A'$ replaces $A$ then $A \subseteq A'$

Theorem

- If the tableau algorithm terminates with $S = \emptyset$ then $K$ is inconsistent
- If the tableau algorithm terminates with $S \neq \emptyset$ then $K$ is consistent
The Tableau Algorithm - Observations

- If a rule has been applied to an ABox in $S$ it will never be applied again (with the same parameters)
- Only the $\sqcap$-rule adds a new ABox to $S$
- If $A'$ replaces $A$ then $A \subseteq A'$

Theorem

- If the tableau algorithm terminates with $S = \emptyset$ then $K$ is inconsistent
- If the tableau algorithm terminates with $S \neq \emptyset$ then $K$ is consistent

What about termination?
What happens when applying the tableau algorithm to

\[ T = \{ A \sqsubseteq \exists R.A \} \]
\[ A = \{ a : A \} \]

→ Infinite application of the $\sqsubseteq$- and $\sqcup$- and $\exists$-rules.

To ensure termination we introduce the notion of block. If $\triangleright t$ is an individual created by application of a rule and $\triangleright$ there is an individual $t'$ with

1. $\{ C | t : C \in A \} \subseteq \{ C | t' : C \in A \}$
2. $t'$ has been created before $t$

then $t$ is blocked (by $t'$).
What happens when applying the tableau algorithm to

\[ T = \{ \quad A \subseteq \exists R.A \quad \} \]

\[ A = \{ \quad a : A \quad \} \quad ? \]

\[ \rightarrow \text{Infinite application of the } \subseteq \text{- and } \sqcup \text{- and } \exists \text{-rules.} \]
What happens when applying the tableau algorithm to

\[ T = \{ \ A \subseteq \exists R.A \ \} \]
\[ A = \{ \ a : A \ \} \]

\[ \rightarrow \text{Infinite application of the } \subseteq \text{- and } \sqcup \text{- and } \exists \text{-rules.} \]

To ensure termination we introduce the notion of block. If

- \( t \) is an individual created by application of a rule and
- there is an individual \( t' \) with
  1. \( \{ C \mid t : C \in A \} \subseteq \{ C \mid t' : C \in A \} \) and
  2. \( t' \) has been created before \( t \)

then \( t \) is blocked (by \( t' \)).
The Tableau Algorithm - Blocking Rules cont’d

▶ \sqcap\text{-rule}: if \( t : C_1 \sqcap C_2 \in A' \), \( t \) is not blocked, and \( \{ t : C_1, t : C_2 \} \not\subseteq A' \) then remove \( A' \) from \( S \) and add \( A' \cup \{ t : C_1, t : C_2 \} \) to \( S \).

▶ \sqcup\text{-rule}: if \( t : C_1 \sqcup C_2 \in A' \), \( t \) is not blocked, and \( \{ t : C_1, t : C_2 \} \cap A' = \emptyset \) then remove \( A' \) from \( S \) and add both \( A' \cup \{ t : C_1 \} \) and \( A' \cup \{ t : C_2 \} \) to \( S \).

▶ \exists\text{-rule}: if \( t : \exists R.C \in A' \), \( t \) is not blocked, and there is no \( t' \) with \( \{(t, t') : R, t' : C\} \subseteq A' \) then remove \( A' \) from \( S \), create a new individual \( t'' \), and add \( A' \cup \{(t, t'') : R, t'' : C\} \) to \( S \).

▶ \forall\text{-rule}: if \( \{ t : \forall R.C, (t, t') : R \} \subseteq A' \), \( t \) is not blocked, and \( \{ t' : C \} \not\subseteq A' \) then remove \( A' \) from \( S \) and add \( A' \cup \{ t' : C \} \) to \( S \).

▶ \sqsubseteq\text{-rule}: if \( C_1 \sqsubseteq C_2 \in T \), \( t \) is not blocked, and \( t : (\neg C_1 \sqcup C_2) \not\in A' \) for \( t \) appearing in \( A' \) then remove \( A' \) from \( S \) and add \( A' \cup \{ t : (\neg C_1 \sqcup C_2) \} \) to \( S \).
What happens now to

$$\mathcal{T} = \{ \ A \sqsubseteq \exists R.A \ \}$$

$$\mathcal{A} = \{ \ a : A \ \}$$

?
What happens now to

\[ T = \{ \ A \sqsubseteq \exists R.A \ \} \]
\[ A = \{ \ a : A \ \} \ ]

Theorem

- When the tableau algorithm with blocking terminates with \( S = \emptyset \) then \( \mathcal{K} \) is inconsistent
What happens now to

\[ T = \{ \; A \sqsubseteq \exists R.A \; \} \]
\[ A = \{ \; a : A \; \} \quad \text{?} \]

**Theorem**

- **When the tableau algorithm with blocking terminates with**
  \( S = \emptyset \) **then** \( K \) **is inconsistent**

- **When the tableau algorithm with blocking terminates with**
  \( S \neq \emptyset \) **then** \( K \) **is consistent**
The Tableau Algorithm - Blocking Rules cont’d

What happens now to

\[ T = \{ A \sqsubseteq \exists R.A \} \]
\[ A = \{ a : A \} \]

Theorem

- When the tableau algorithm with blocking terminates with \( S = \emptyset \) then \( K \) is inconsistent
- When the tableau algorithm with blocking terminates with \( S \neq \emptyset \) then \( K \) is consistent
- The tableau algorithm with blocking always terminates on \( ALC \).
What happens now to

\[ T = \{ A \sqsubseteq \exists R.A \} \]

\[ \mathcal{A} = \{ a : A \} \]

**Theorem**

- *When the tableau algorithm with blocking terminates with* \( S = \emptyset \) *then* \( \mathcal{K} \) *is inconsistent*

- *When the tableau algorithm with blocking terminates with* \( S \neq \emptyset \) *then* \( \mathcal{K} \) *is consistent*

- *The tableau algorithm with blocking always terminates on* \( \mathcal{ALC} \).

- *The tableau algorithm with blocking runs in* \( \text{EXPSPACE} \) * (worst case).*
Outline

1. Reasoning with Description Logics
2. Ontology languages revisited
3. Tools
4. Summary and Exercises
Ontology languages revisited

- $\mathcal{ALC}$ is just one example of a description logic
Ontology languages revisited

- \textit{ALC} is just one example of a description logic
- Over the years a lot of different description logics have been proposed that differ in
  - complexity
  - expressiveness
 Ontology languages revisited

- $\mathcal{ALC}$ is just one example of a description logic
- Over the years a lot of different description logics have been proposed that differ in
  - complexity
  - expressiveness
- The search for the right description logic is ongoing
Ontology languages revisited

- $\mathcal{ALC}$ is just one example of a description logic
- Over the years a lot of different description logics have been proposed that differ in
  - complexity
  - expressiveness
- The search for the right description logic is ongoing
- There is always the issue of balancing between complexity and expressiveness
Ontology languages revisited

The base description logic is $\mathcal{ALC}$ (Attributive Language with Complements).
Ontology languages revisited

The base description logic is \textit{ALC} (Attributive Language with Complements).

Further \textit{features} include

\begin{itemize}
  \item[$\mathcal{N}$:] unqualified number restrictions: \( \geq 3 \ hasChild \)
\end{itemize}
Ontology languages revisited

The base description logic is $\mathcal{ALC}$ (Attributive Language with Complements).

Further features include

- $\mathcal{N}$: unqualified number restrictions: ($\geq 3 \ hasChild$)
- Qualified number restrictions: ($\geq 2 \ hasChild.Female$)
Ontology languages revisited

The base description logic is $\mathcal{ALC}$ (Attributive Language with Complements).

Further features include:

- $\mathcal{N}$: unqualified number restrictions: $(\geq 3 \text{ hasChild})$
- Qualified number restrictions: $(\geq 2 \text{ hasChild.Female})$
- One-of (nominals): $\{t_1, \ldots, t_n\}$
Ontology languages revisited

The base description logic is $\mathcal{ALC}$ (Attributive Language with Complements).

Further features include

- $\mathcal{N}$: unqualified number restrictions: ($\geq 3 \text{ hasChild}$)
- Qualified number restrictions: ($\geq 2 \text{ hasChild.Female}$)
- One-of (nominals): $\{t_1, \ldots, t_n\}$
- $\mathcal{F}$unctionality: ($\leq \text{ hasFather}$)
Ontology languages revisited

The base description logic is \textit{ALC} (Attributive Language with Complements).

Further \textit{features} include

\begin{itemize}
  \item[$\mathcal{N}$:] unqualified number restrictions: ($\geq 3 \ \text{hasChild}$)
  \item Qualified number restrictions: ($\geq 2 \ \text{hasChild}.\text{Female}$)
  \item One-of (nominals): $\{t_1, \ldots, t_n\}$
  \item $\mathcal{F}$unctionality: ($\leq \ \text{hasFather}$)
  \item Role operators:
    \begin{itemize}
      \item $\mathcal{I}$: role inverse: $\text{hasChild}^{-} \equiv \text{hasParent}$
    \end{itemize}
\end{itemize}
Ontology languages revisited

The base description logic is $\mathcal{ALC}$ (Attributive Language with Complements).

Further features include

- $\mathcal{N}$: unqualified number restrictions: $(\geq 3 \text{ hasChild})$
- Qualified number restrictions: $(\geq 2 \text{ hasChild.Female})$
- $\mathcal{O}$ne-of (nominals): \{t_1, \ldots, t_n\}
- $\mathcal{F}$unctionality: $(\leq \text{ hasFather})$
- Role operators:
  - $\mathcal{I}$: role inverse: $\text{hasChild}^{-} \equiv \text{hasParent}$
  - $\mathcal{S}$: Transitive roles $tr(R)$ ($tr(\text{hasParent}) \equiv \text{hasAncestor}$)
Ontology languages revisited

The base description logic is \( \mathcal{ALC} \) (Attributive Language with Complements).

Further features include

- \( \mathcal{N} \): unqualified number restrictions: \( \geq 3 \, \text{hasChild} \)
- Qualified number restrictions: \( \geq 2 \, \text{hasChild}.\text{Female} \)
- One-of (nominals): \( \{ t_1, \ldots, t_n \} \)
- Functionality: \( \leq \, \text{hasFather} \)
- Role operators:
  - \( \mathcal{I} \): role inverse: \( \text{hasChild}^{-} \equiv \text{hasParent} \)
  - \( \mathcal{S} \): Transitive roles \( \text{tr}(R) \) (\( \text{tr}(\text{hasParent}) \equiv \text{hasAncestor} \))
  - \( \mathcal{H} \): role hierarchies: \( R \circ R' \subseteq R'' \)
    (\( \text{hasParent} \circ \text{hasParent} \equiv \text{hasGrandparent} \))
  - \( \ldots \)
Other description logic types can be described by their names:

- **ALCQIO**: ALC with qualified number restrictions, inverse roles, and nominals.
Other description logic types can be described by their names:

- **ALCQIO**: ALC with qualified number restrictions, inverse roles, and nominals.
- **SHOIN**: ALC with transitive roles, role hierarchies, role inverse, nominals, unqualified number restrictions (this is the same as OWL-DL)
Other description logic types can be described by their names:

- **ALCQIO**: ALC with qualified number restrictions, inverse roles, and nominals.
- **SHOIN**: ALC with transitive roles, role hierarchies, role inverse, nominals, unqualified number restrictions (this is the same as OWL-DL)

Some description languages with further restrictions:

- **\(\mathcal{EL}\)**: Only \(C_1 \sqcap C_2\) and \(\exists R . \top\) allowed
Ontology languages revisited

- Other description logic types can be described by their names:
  - \textit{ALCQIO}: \textit{ALC} with qualified number restrictions, inverse roles, and nominals.
  - \textit{SHOIN}: \textit{ALC} with transitive roles, role hierarchies, role inverse, nominals, unqualified number restrictions (this is the same as OWL-DL)
  - Some description languages with further restrictions:
    - \textit{EL}: Only $C_1 \sqcap C_2$ and $\exists R. \top$ allowed
    - \textit{EL}++: \textit{EL} with nominals and some additional role operators
1. Reasoning with Description Logics
2. Ontology languages revisited
3. Tools
4. Summary and Exercises
In contrast to many other KR languages (propositional logic, default logic, ...) description logics have been developed out of the need to apply them.
In contrast to many other KR languages (propositional logic, default logic, ...) description logics have been developed out of the need to apply them.

Therefore, a lot of tools are around that enable ontology engineering using description logics.
Tools: Protégé

- Free open-source ontology editor
- Webpage: http://protege.stanford.edu
Tools: RacerPro

- Commercial description logic reasoner
- Webpage: http://www.franz.com/agraph/racer
Further Tools

HermiT

- Free open-source description logic reasoner (OWL)
- Webpage: http://hermit-reasoner.com
Further Tools

HermiT

- Free open-source description logic reasoner (OWL)
- Webpage: http://hermit-reasoner.com

FacT++

- Free open-source description logic reasoner
- Webpage: http://owl.man.ac.uk/factplusplus/
Further Tools

HermiT
- Free open-source description logic reasoner (OWL)
- Webpage: http://hermit-reasoner.com

FacT++
- Free open-source description logic reasoner
- Webpage: http://owl.man.ac.uk/factplusplus/

OWL API
- Official open source JAVA interfaces for programming DL applications
Outline

1. Reasoning with Description Logics
2. Ontology languages revisited
3. Tools
4. Summary and Exercises
The tableau algorithm for $\mathcal{ALC}$:
- checks consistency of a knowledge base
- sound and complete
- terminates always when using blocks

Ontology languages revisited
- Nomenclature of description logics
- expressivity vs. complexity

Tools for working with description logics
Pointers to further reading

- The Description Logic Complexity Navigator
  http://www.cs.man.ac.uk/~ezolin/dl/


Exercises

▶ Apply the tableau algorithm to check whether the knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ is consistent:

$$\mathcal{T} = \{ A \sqsubseteq C, B \sqsubseteq \neg C \}$$

$$\mathcal{A} = \{ a : A, a : B \}$$

▶ Download Prot´eg´e (http://protege.stanford.edu) and play around with it (Home assignment)

▶ Apply the tableau algorithm with blocking to check whether the knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ is consistent:

$$\mathcal{T} = \{ A \sqsubseteq \exists R. B, B \sqsubseteq A \sqcap \forall S. C \}$$

$$\mathcal{A} = \{ a : A, a : B \}$$

(Home assignment)
Exercises

▶ Apply the tableau algorithm to check whether the knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ is consistent:

$\mathcal{T} = \{ A \sqsubseteq C, B \sqsubseteq \neg C \}$
$\mathcal{A} = \{ a : A, a : B \}$

▶ Download Protégé (http://protege.stanford.edu) and play around with it (Home assignment)
Exercises

▶ Apply the tableau algorithm to check whether the knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ is consistent:

\[
\mathcal{T} = \{ \quad A \subseteq C, B \subseteq \neg C \quad \}
\]
\[
\mathcal{A} = \{ \quad a : A, a : B \quad \}
\]

▶ Download Protégé (http://protege.stanford.edu) and play around with it (Home assignment)

▶ Apply the tableau algorithm with blocking to check whether the knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ is consistent:

\[
\mathcal{T} = \{ \quad A \subseteq \exists R.B, \quad B \subseteq A \cap \forall S.C \quad \}
\]
\[
\mathcal{A} = \{ \quad a : A \quad \}
\]

(Home assignment)