Description Logic 2: Reasoning

Leif Harald Karlsen

Autumn 2016
Last time

- As we saw last time, description logics allows us to model knowledge in a natural way.
- Today we will see why we make the restrictions, and what makes exactly these restrictions important.
Contents

Assumptions

Reasoning problems

Tableau algorithm for $\text{ALCN}$

Soundness, Completeness and Termination

Removing the assumptions

Complexity
Contents

Assumptions

Reasoning problems

Tableau algorithm for $ALCN$

Soundness, Completeness and Termination

Removing the assumptions

Complexity
Assumptions

Before we introduce the tableau algorithm for DLs, we will first make a few assumptions, which we will eliminate towards the end of the talk.
Assumptions

Before we introduce the tableau algorithm for DLs, we will first make a few assumptions, which we will eliminate towards the end of the talk.

First, we will make the following assumptions:

– We only allow equivalence axioms, $A \equiv D$, where $A$ is atomic and $D$ is not atomic. Each atomic concept should only occur once on a left-hand side.
Assumptions

Before we introduce the tableau algorithm for DLs, we will first make a few assumptions, which we will eliminate towards the end of the talk.

First, we will make the following assumptions:

– We only allow equivalence axioms, $A \equiv D$, where $A$ is atomic and $D$ is not atomic. Each atomic concept should only occur once on a left-hand side.
– We only allow acyclic TBoxes, so e.g. no $A \equiv \exists R.D$, where $D$ is defined in terms of $A$. 
Assumptions

Before we introduce the tableau algorithm for DLs, we will first make a few assumptions, which we will eliminate towards the end of the talk.

First, we will make the following assumptions:

– We only allow equivalence axioms, $A \equiv D$, where $A$ is atomic and $D$ is not atomic. Each atomic concept should only occur once on a left-hand side.

– We only allow acyclic TBoxes, so e.g. no $A \equiv \exists R.D$, where $D$ is defined in terms of $A$.

– We only allow ABox axioms on the form $A(c)$ for atomic concepts $A$ (and $R(a, b)$ as usual). (Not really a restriction, as $D(c)$ for complex $D$ can be expressed as $A_D \equiv D$ and $A_D(c)$ for some fresh concept name $A_D$)
Example ontology in $\mathcal{ALCN}$

TBox:

\[
\begin{align*}
Animal & \equiv \leq 2 \text{hasParent} \sqcap \geq 2 \text{hasParent} \\
Donkey & \equiv Animal \sqcap \text{Stubborn} \\
\text{Horse} & \equiv Animal \sqcap \neg \text{Stubborn} \\
\text{Mule} & \equiv Animal \sqcap \exists \text{hasParent}.\text{Horse} \sqcap \exists \text{hasParent}.\text{Donkey}
\end{align*}
\]
Example ontology in $\mathcal{ALCN}$

TBox:

$$\begin{align*}
    \text{Animal} & \equiv \leq 2 \text{hasParent} \land \geq 2 \text{hasParent} \\
    \text{Donkey} & \equiv \text{Animal} \land \text{Stubborn} \\
    \text{Horse} & \equiv \text{Animal} \land \neg \text{Stubborn} \\
    \text{Mule} & \equiv \text{Animal} \land \exists \text{hasParent}.\text{Horse} \land \exists \text{hasParent}.\text{Donkey}
\end{align*}$$

ABox:

$$\begin{align*}
    \text{Horse}(\text{mary}) & \quad \text{Mule}(\text{peter}) & \quad \text{Horse}(\text{hannah}) \\
    \text{hasParent}(\text{peter}, \text{mary}) & \quad \text{hasParent}(\text{peter}, \text{carl}) \\
    \text{hasParent}(\text{sven}, \text{hannah}) & \quad \text{hasParent}(\text{sven}, \text{carl})
\end{align*}$$
Contents

Assumptions

Reasoning problems

Tableau algorithm for $ALCN$

Soundness, Completeness and Termination

Removing the assumptions

Complexity
Problems

For concepts $C$ and $D$:

- **Satisfiability** of concepts ($C$ is satisfiable w.r.t $\mathcal{T}$ if there exist a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^\mathcal{I}$ is nonempty).
Problems

For concepts $C$ and $D$:

- **Satisfiability** of concepts ($C$ is satisfiable w.r.t $\mathcal{T}$ if there exist a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^\mathcal{I}$ is nonempty).

- **Subsumption** of concepts ($C$ is subsumed by $D$ w.r.t $\mathcal{T}$, written $\mathcal{T} \models C \sqsubseteq D$, if $C^\mathcal{I} \subseteq D^\mathcal{I}$ for every model $\mathcal{I}$ of $\mathcal{T}$).
Problems

For concepts $C$ and $D$:

- **Satisfiability** of concepts ($C$ is satisfiable w.r.t $\mathcal{T}$ if there exist a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^\mathcal{I}$ is nonempty).

- **Subsumption** of concepts ($C$ is subsumed by $D$ w.r.t $\mathcal{T}$, written $\mathcal{T} \models C \sqsubseteq D$, if $C^\mathcal{I} \subseteq D^\mathcal{I}$ for every model $\mathcal{I}$ of $\mathcal{T}$).

- **Equivalence** of concepts ($C$ is equivalent to $D$ w.r.t $\mathcal{T}$, written $\mathcal{T} \models C \equiv D$, if $C^\mathcal{I} \equiv D^\mathcal{I}$ for every model $\mathcal{I}$ of $\mathcal{T}$).
Problems

For concepts $C$ and $D$:

- **Satisfiability** of concepts ($C$ is satisfiable w.r.t $\mathcal{T}$ if there exist a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^\mathcal{I}$ is nonempty).

- **Subsumption** of concepts ($C$ is subsumed by $D$ w.r.t $\mathcal{T}$, written $\mathcal{T} \models C \sqsubseteq D$, if $C^\mathcal{I} \subseteq D^\mathcal{I}$ for every model $\mathcal{I}$ of $\mathcal{T}$).

- **Equivalence** of concepts ($C$ is equivalent to $D$ w.r.t $\mathcal{T}$, written $\mathcal{T} \models C \equiv D$, if $C^\mathcal{I} \equiv D^\mathcal{I}$ for every model $\mathcal{I}$ of $\mathcal{T}$).

- **Disjointness** of concepts ($C$ is disjoint from $D$ w.r.t $\mathcal{T}$, if $C^\mathcal{I} \cap D^\mathcal{I} = \emptyset$ for every model $\mathcal{I}$ of $\mathcal{T}$).
Problems

For concepts $C$ and $D$:

- **Satisfiability** of concepts ($C$ is satisfiable w.r.t $\mathcal{T}$ if there exist a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^\mathcal{I}$ is nonempty).

- **Subsumption** of concepts ($C$ is subsumed by $D$ w.r.t $\mathcal{T}$, written $\mathcal{T} \models C \subseteq D$, if $C^\mathcal{I} \subseteq D^\mathcal{I}$ for every model $\mathcal{I}$ of $\mathcal{T}$).

- **Equivalence** of concepts ($C$ is equivalent to $D$ w.r.t $\mathcal{T}$, written $\mathcal{T} \models C \equiv D$, if $C^\mathcal{I} \equiv D^\mathcal{I}$ for every model $\mathcal{I}$ of $\mathcal{T}$).

- **Disjointness** of concepts ($C$ is disjoint from $D$ w.r.t $\mathcal{T}$, if $C^\mathcal{I} \cap D^\mathcal{I} = \emptyset$ for every model $\mathcal{I}$ of $\mathcal{T}$).

For knowledge bases $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$:

- **Instance checking** (whether for some concept $C$ and individual $a$, $\mathcal{K} \models C(a)$).
Problems

For concepts $C$ and $D$:

- **Satisfiability** of concepts ($C$ is satisfiable w.r.t $T$ if there exist a model $I$ of $T$ such that $C^I$ is nonempty).

- **Subsumption** of concepts ($C$ is subsumed by $D$ w.r.t $T$, written $T \models C \subseteq D$, if $C^I \subseteq D^I$ for every model $I$ of $T$).

- **Equivalence** of concepts ($C$ is equivalent to $D$ w.r.t $T$, written $T \models C \equiv D$, if $C^I \equiv D^I$ for every model $I$ of $T$).

- **Disjointness** of concepts ($C$ is disjoint from $D$ w.r.t $T$, if $C^I \cap D^I = \emptyset$ for every model $I$ of $T$).

For knowledge bases $\mathcal{K} = \langle A, T \rangle$:

- **Instance checking** (whether for some concept $C$ and individual $a$, $\mathcal{K} \models C(a)$).
- **ABox consistency** (whether $A$ is consistent w.r.t. $T$).
Reductions

Theorem
For concepts $C$ and $D$, we have

(i) $C$ is subsumed by $D \iff C \sqcap \neg D$ is unsatisfiable;

(ii) $C$ and $D$ are equivalent $\iff$ both $(C \sqcap \neg D)$ and $(\neg C \sqcap D)$ are unsatisfiable;

(iii) $C$ and $D$ are disjoint $\iff C \sqcup D$ is unsatisfiable.
Theorem

For concepts $C$ and $D$, we have

(i) $C$ is subsumed by $D$ $\iff$ $C \sqcap \neg D$ is unsatisfiable;

(ii) $C$ and $D$ are equivalent $\iff$ both $(C \sqcap \neg D)$ and $(\neg C \sqcap D)$ are unsatisfiable;
Theorem

For concepts $C$ and $D$, we have

(i) $C$ is subsumed by $D \iff C \sqcap \neg D$ is unsatisfiable;
(ii) $C$ and $D$ are equivalent $\iff$ both $(C \sqcap \neg D)$ and $(\neg C \sqcap D)$ are unsatisfiable;
(iii) $C$ and $D$ are disjoint $\iff C \sqcap D$ is unsatisfiable.
Removing the TBox

For reasoning in KBs with acyclic TBoxes $\mathcal{T}$, we can in fact remove the TBox by extending the ABox. This is done as follows:

- First expanding $\mathcal{T}$ by replacing every definition $A \equiv D$ in $\mathcal{T}$, with $A \equiv D'$, where $D'$ is obtained by recursively replacing every name concept $C$ in $D$ with $C'$ if $C \equiv C'$ is in $\mathcal{T}$. We call the extended TBox $\mathcal{T}'$. E.g.:
  
  \[
  \text{Donkey} \equiv \text{Animal} \sqcap \text{Stubborn} \rightsquigarrow \text{Donkey} \equiv \leq 2 \text{hasParent} \sqcap \geq 2 \text{hasParent} \sqcap \text{Stubborn}
  \]

- Then we replace every assertion $A(a)$ with $D'(a)$ in $A$, if $A \equiv D'$ is in $\mathcal{T}'$. E.g.:
  
  \[
  \text{Donkey}(peter) \rightarrow (\leq 2 \text{hasParent} \sqcap \geq 2 \text{hasParent} \sqcap \text{Stubborn})(peter)
  \]
Removing the TBox

For reasoning in KBs with acyclic TBoxes $\mathcal{T}$, we can in fact remove the TBox by extending the ABox. This is done as follows:

– First expanding $\mathcal{T}$ by replacing every definition $A \equiv D$ in $\mathcal{T}$, with $A \equiv D'$, where $D'$ is obtained by recursively replacing every name concept $C$ in $D$ with $C'$ if $C \equiv C'$ is in $\mathcal{T}$. We call the extended TBox $\mathcal{T}'$. E.g.:

$$\text{Donkey} \equiv \text{Animal} \sqcap \text{Stubborn} \Rightarrow \text{Donkey} \equiv \leq 2 \text{hasParent} \sqcap \geq 2 \text{hasParent} \sqcap \text{Stubborn}$$
Removing the TBox

For reasoning in KBs with acyclic TBoxes $\mathcal{T}$, we can in fact remove the TBox by extending the ABox. This is done as follows:

- First expanding $\mathcal{T}$ by replacing every definition $A \equiv D$ in $\mathcal{T}$, with $A \equiv D'$, where $D'$ is obtained by recursively replacing every name concept $C$ in $D$ with $C'$ if $C \equiv C'$ is in $\mathcal{T}$. We call the extended TBox $\mathcal{T}'$. E.g.:

$$\text{Donkey} \equiv \text{Animal} \sqcap \text{Stubborn} \rightsquigarrow$$

$$\text{Donkey} \equiv \leq 2 \text{hasParent} \sqcap \geq 2 \text{hasParent} \sqcap \text{Stubborn}$$
Removing the TBox

For reasoning in KBs with acyclic TBoxes $\mathcal{T}$, we can in fact remove the TBox by extending the ABox. This is done as follows:

- First expanding $\mathcal{T}$ by replacing every definition $A \equiv D$ in $\mathcal{T}$, with $A \equiv D'$, where $D'$ is obtained by recursively replacing every name concept $C$ in $D$ with $C'$ if $C \equiv C'$ is in $\mathcal{T}$. We call the extended TBox $\mathcal{T}'$. E.g.:

$$\text{Donkey} \equiv \text{Animal} \sqcap \text{Stubborn} \rightsquigarrow$$
$$\text{Donkey} \equiv \leq 2 \text{hasParent} \sqcap \geq 2 \text{hasParent} \sqcap \text{Stubborn}$$

- Then we replace every assertion $A(a)$ in $\mathcal{A}$, if $A \equiv D'$ is in $\mathcal{T}'$. E.g.:

$$\text{Donkey}(\text{peter}) \rightsquigarrow$$
$$(\leq 2 \text{hasParent} \sqcap \geq 2 \text{hasParent} \sqcap \text{Stubborn})(\text{peter})$$
Theorem
Assume $C \equiv D$ is replaced by $C \equiv D'$ in an expansion of $\langle A, T \rangle$ to $\langle A', \emptyset \rangle$. Then:

(i) $C$ is satisfiable w.r.t. $T \iff \{D'(x)\}$ is consistent.
Important results

Theorem
Assume $C \equiv D$ is replaced by $C \equiv D'$ in an expansion of $\langle A, T \rangle$ to $\langle A', \emptyset \rangle$. Then:

(i) $C$ is satisfiable w.r.t. $T$ $\iff$ $\{D'(x)\}$ is consistent.

(ii) $A$ is consistent w.r.t. $T$ $\iff$ $A'$ is consistent.
Theorem
Assume $C \equiv D$ is replaced by $C \equiv D'$ in an expansion of $\langle A, T \rangle$ to $\langle A', \emptyset \rangle$. Then:

(i) $C$ is satisfiable w.r.t. $T$ $\iff$ $\{D'(x)\}$ is consistent.

(ii) $A$ is consistent w.r.t. $T$ $\iff$ $A'$ is consistent.

(iii) $\langle A, T \rangle \models C(a) \iff A' \cup \{\neg D'(a)\}$ is inconsistent.
Negated Normal Form

For our reasoning algorithm, we need our concepts in Negated Normal Form (NFF):

\[
\neg (C \sqcap D) \iff \neg C \sqcup \neg D \\
\neg (C \sqcup D) \iff \neg C \sqcap \neg D \\
\neg \exists R. \mathcal{C} \iff \forall R. \neg \mathcal{C} \\
\neg \forall R. \mathcal{C} \iff \exists R. \neg \mathcal{C} \\
\neg \leq n R \iff \geq (n + 1) R \\
\neg \geq n R \iff \leq (n - 1) R
\]
Contents

Assumptions

Reasoning problems

Tableau algorithm for $ALCN$

Soundness, Completeness and Termination

Removing the assumptions

Complexity
Tableau algorithm for $\mathcal{ALC}$

(i) Start with a set $S_0 = \{A_0\}$ (on NNF).
Tableau algorithm for $\mathcal{ALC}$

(i) Start with a set $S_0 = \{A_0\}$ (on NNF).

(ii) While a rule is applicable to an element $A \in S_i$:

   - Apply rule to $A$ resulting in $S_A = \{A_1, A_2, \ldots, A_n\}$. 
Tableau algorithm for \( \mathcal{ALC} \)

(i) Start with a set \( S_0 = \{A_0\} \) (on NNF).

(ii) While a rule is applicable to an element \( A \in S_i \):

- Apply rule to \( A \) resulting in \( S_A = \{A_1, A_2, \ldots, A_n\} \).
- Set \( S_{i+1} = (S_i \setminus \{A\}) \cup S_A \).
Tableau algorithm for $\mathcal{ALC}$

(i) Start with a set $S_0 = \{A_0\}$ (on NNF).

(ii) While a rule is applicable to an element $A \in S_i$:

- Apply rule to $A$ resulting in $S_A = \{A_1, A_2, \ldots, A_n\}$.
- Set $S_{i+1} = (S_i \setminus \{A\}) \cup S_A$.

$\Box$-rule

**Condition:** $A$ contains $(C_1 \Box C_2)(x)$, but not both $C_1(x)$ and $C_2(x)$.

**Action:** $A' = A \cup \{C_1(x), C_2(x)\}$. 

$\Eu$-rule
Tableau algorithm for $\mathcal{ALC}$

(i) Start with a set $S_0 = \{A_0\}$ (on NNF).
(ii) While a rule is applicable to an element $A \in S_i$:

- Apply rule to $A$ resulting in $S_A = \{A_1, A_2, \ldots, A_n\}$.
- Set $S_{i+1} = (S_i \setminus \{A\}) \cup S_A$.

$\square$-rule

**Condition:** $A$ contains $(C_1 \cap C_2)(x)$, but not both $C_1(x)$ and $C_2(x)$.

**Action:** $A' = A \cup \{C_1(x), C_2(x)\}$.

$\Box$-rule

**Condition:** $A$ contains $(C_1 \sqcup C_2)(x)$, but neither $C_1(x)$ nor $C_2(x)$.

**Action:** $A' = A \cup \{C_1(x)\}$, $A'' = A \cup \{C_2(x)\}$. 
Tableau algorithm for $\mathcal{ALC}$

(i) Start with a set $S_0 = \{A_0\}$ (on NNF).

(ii) While a rule is applicable to an element $A \in S_i$:
- Apply rule to $A$ resulting in $S_A = \{A_1, A_2, \ldots, A_n\}$.
- Set $S_{i+1} = (S_i \setminus \{A\}) \cup S_A$.

\(\Box\)-rule  
**Condition:** $A$ contains $(C_1 \cap C_2)(x)$, but not both $C_1(x)$ and $C_2(x)$.
**Action:** $A' = A \cup \{C_1(x), C_2(x)\}$.

\(\square\)-rule  
**Condition:** $A$ contains $(C_1 \sqcup C_2)(x)$, but neither $C_1(x)$ nor $C_2(x)$.
**Action:** $A' = A \cup \{C_1(x)\}$, $A'' = A \cup \{C_2(x)\}$.

\(\exists\)-rule  
**Condition:** $A$ contains $(\exists R.C)(x)$, but there is no $z$ such that $C(z)$ and $R(x, z)$ in $A$.
**Action:** $A' = A \cup \{C(y), R(x, y)\}$, $y$ fresh.
Tableau algorithm for $\mathcal{ALC}$

(i) Start with a set $S_0 = \{A_0\}$ (on NNF).

(ii) While a rule is applicable to an element $A \in S_i$:
- Apply rule to $A$ resulting in $S_A = \{A_1, A_2, \ldots, A_n\}$.
- Set $S_{i+1} = (S_i \setminus \{A\}) \cup S_A$.

$\Box$-rule  
**Condition:** $A$ contains $(C_1 \sqcap C_2)(x)$, but not both $C_1(x)$ and $C_2(x)$.

**Action:** $A' = A \cup \{C_1(x), C_2(x)\}$.

$\sqcup$-rule  
**Condition:** $A$ contains $(C_1 \sqcup C_2)(x)$, but neither $C_1(x)$ nor $C_2(x)$.

**Action:** $A' = A \cup \{C_1(x)\}$, $A'' = A \cup \{C_2(x)\}$.

$\exists$-rule  
**Condition:** $A$ contains $(\exists R.C)(x)$, but there is no $z$ such that $C(z)$ and $R(x, z)$ in $A$.

**Action:** $A' = A \cup \{C(y), R(x, y)\}$, $y$ fresh.

$\forall$-rule  
**Condition:** $A$ contains $(\forall R.C)(x)$ and $R(x, y)$, but not $C(y)$.

**Action:** $A' = A \cup \{C(y)\}$.
Extension with $\mathcal{N}$

$\geq$-rule

**Condition:** $\mathcal{A}$ contains $(\geq n R)(x)$, but there are no $z_1, \ldots, z_n$ such that $R(x, z_i) \ (1 \leq i \leq n)$ and $z_i \neq z_j \ (1 \leq i < j \leq n)$ are in $\mathcal{A}$.

**Action:** $\mathcal{A}' = \mathcal{A} \cup \{R(x, y_i) \mid 1 \leq i \leq n\} \cup \{y_i \neq y_j \mid (1 \leq i < j \leq n)\}$, distinct and fresh $y_1, \ldots, y_n$. 
Extension with $\mathcal{N}$

$\geq$-rule  
**Condition:** $\mathcal{A}$ contains $(\geq n R)(x)$, but there are no $z_1, \ldots, z_n$ such that $R(x, z_i) \ (1 \leq i \leq n)$ and $z_i \neq z_j \ (1 \leq i < j \leq n)$ are in $\mathcal{A}$.

**Action:** $\mathcal{A}' = \mathcal{A} \cup \{R(x, y_i) \mid 1 \leq i \leq n\} \cup \{y_i \neq y_j \mid (1 \leq i < j \leq n)\}$, distinct and fresh $y_1, \ldots, y_n$.

$\leq$-rule  
**Condition:** $\mathcal{A}$ contains $(\leq n R)(x)$ and $R(x, y_1), \ldots, R(x, y_{n+1})$ for distinct names $y_1, \ldots, y_{n+1}$, but not $y_i \neq y_j$ for some $i \neq j$.

**Action:** $\mathcal{A}_{i,j} = \mathcal{A}[y_i/y_j]$, for each pair $y_i, y_j$ such that $i > j$ and $y_i \neq y_j$ is not in $\mathcal{A}$.
We want to check whether the concept

$$\forall R. (\neg C \sqcup D) \sqcap \exists R. (C \sqcap D)$$

is satisfiable. (We write $A_n^k$ for the $k$-th set in $A_n$)
Example derivation(1)

We want to check whether the concept

$$\forall R.(\neg C \sqcup D) \sqcap \exists R.(C \sqcap D)$$

is satisfiable. (We write $A_n^k$ for the $k$-th set in $A_n$)

$$A_0 = \{(\forall R.(\neg C \sqcup D) \sqcap \exists R.(C \sqcap D))(x_0)\}$$
Example derivation (1)

We want to check whether the concept

$$\forall R. (\neg C \sqcup D) \sqcap \exists R. (C \sqcap D)$$

is satisfiable. (We write $\mathcal{A}_n^k$ for the $k$-th set in $\mathcal{A}_n$)

- $\mathcal{A}_0 = \{((\forall R. (\neg C \sqcup D) \sqcap \exists R. (C \sqcap D))(x_0)\}$
- $\mathcal{A}_1 = \mathcal{A}_0 \cup \{((\forall R. (\neg C \sqcup D))(x_0), (\exists R. (C \sqcap D))(x_0)\}$ by $\sqcap$-rule
We want to check whether the concept

$$\forall R. (\neg C \sqcup D) \sqcap \exists R. (C \sqcap D)$$

is satisfiable. (We write $A_n^k$ for the $k$-th set in $A_n$)

\[
\begin{align*}
A_0 &= \{ ((\forall R. (\neg C \sqcup D) \sqcap \exists R. (C \sqcap D))(x_0)) \} \\
A_1 &= A_0 \cup \{ ((\forall R. (\neg C \sqcup D))(x_0), (\exists R. (C \sqcap D))(x_0)) \} \quad \text{by } \sqcap \text{-rule} \\
A_2 &= A_1 \cup \{ R(x_0, y), (C \sqcap D)(y) \} \quad \text{by } \exists \text{-rule}
\end{align*}
\]
We want to check whether the concept

$$\forall R. (\neg C \sqcup D) \sqcap \exists R. (C \sqcap D)$$

is satisfiable. (We write $A_n^k$ for the $k$-th set in $A_n$)

\begin{align*}
A_0 &= \{((\forall R. (\neg C \sqcup D) \sqcap \exists R. (C \sqcap D))(x_0))\} \\
A_1 &= \{A_0^1 \cup \{((\forall R. (\neg C \sqcup D))(x_0), (\exists R. (C \sqcap D))(x_0))\}\} \quad \text{by } \sqcap \text{-rule} \\
A_2 &= \{A_1^1 \cup \{R(x_0, y), (C \sqcap D)(y)\}\} \quad \text{by } \exists \text{-rule} \\
A_3 &= \{A_2^1 \cup \{(\neg C \sqcup D)(y)\}\} \quad \text{by } \forall \text{-rule}
\end{align*}
Example derivation(1)

We want to check whether the concept

$$\forall R. (\neg C \sqcup D) \sqcap \exists R. (C \sqcap D)$$

is satisfiable. (We write $A_n^k$ for the $k$-th set in $A_n$)

\begin{align*}
A_0 &= \{((\forall R. (\neg C \sqcup D) \sqcap \exists R. (C \sqcap D))(x_0)\} \\
A_1 &= \{A_0^1 \cup \{((\forall R. (\neg C \sqcup D))(x_0), (\exists R. (C \sqcap D))(x_0)\}\} \text{ by } \sqcap\text{-rule} \\
A_2 &= \{A_1^1 \cup \{R(x_0, y), (C \sqcap D)(y)\}\} \text{ by } \exists\text{-rule} \\
A_3 &= \{A_2^1 \cup \{\neg C \sqcap D)(y)\}\} \text{ by } \forall\text{-rule} \\
A_4 &= \{A_3^1 \cup \{(\neg C)(y)\}, \quad A_3^1 \cup \{D(y)\}\} \text{ by } \sqcup\text{-rule}
\end{align*}
We want to check whether the concept
\[ \forall R. (\neg C \sqcup D) \sqcap \exists R. (C \sqcap D) \]
is satisfiable. (We write $A^k_n$ for the $k$-th set in $A_n$)

\[
\begin{align*}
A_0 &= \{ ((\forall R. (\neg C \sqcup D) \sqcap \exists R. (C \sqcap D))(x_0)) \} \\
A_1 &= \{ A_0^1 \cup \{ (\forall R. (\neg C \sqcup D))(x_0), (\exists R. (C \sqcap D))(x_0) \} \} \text{ by } \sqcap \text{-rule} \\
A_2 &= \{ A_1^1 \cup \{ R(x_0, y), (C \sqcap D)(y) \} \} \text{ by } \exists \text{-rule} \\
A_3 &= \{ A_2^1 \cup \{ (\neg C \sqcup D)(y) \} \} \text{ by } \forall \text{-rule} \\
A_4 &= \{ A_3^1 \cup \{ (\neg C)(y) \}, \quad A_3^1 \cup \{ D(y) \} \} \text{ by } \sqcup \text{-rule} \\
A_5 &= \{ A_4^1 \cup \{ C(y), D(y) \}, \quad A_4^2 \} \text{ by } \sqcap \text{-rule}
\end{align*}
\]
Example derivation(1)

We want to check whether the concept

\[ \forall R. (\neg C \sqcup D) \cap \exists R. (C \cap D) \]

is satisfiable. (We write \( A^n_k \) for the \( k \)-th set in \( A^n \))

\[
\begin{align*}
A_0 &= \{((\forall R. (\neg C \sqcup D) \cap \exists R. (C \cap D))(x_0))\} \\
A_1 &= \{A_0^1 \cup \{(\forall R. (\neg C \sqcup D))(x_0), (\exists R. (C \cap D))(x_0)\}\} \quad \text{by } \cap\text{-rule} \\
A_2 &= \{A_1^1 \cup \{R(x_0,y), (C \cap D)(y)\}\} \quad \text{by } \exists\text{-rule} \\
A_3 &= \{A_2^1 \cup \{\neg (C \sqcup D)(y)\}\} \quad \text{by } \forall\text{-rule} \\
A_4 &= \{A_3^1 \cup \{\neg C)(y)\}, A_3^1 \cup \{D(y)\}\} \quad \text{by } \sqcup\text{-rule} \\
A_5 &= \{A_4^1 \cup \{C(y), D(y)\}, A_4^2\} \quad \text{by } \cap\text{-rule} \\
A_6 &= \{A_5^1, A_5^2 \cup \{C(y), D(y)\}\} \quad \text{by } \cap\text{-rule}
\end{align*}
\]

Now no more rules apply.
We want to check whether the concept

$$\leq 1 \ R \sqcap \geq 2 \ R$$

is satisfiable.
Example derivation (2)

We want to check whether the concept

\[ \leq 1 \ R \sqcap \geq 2 \ R \]

is satisfiable.

\[ \mathcal{A}_0 = \{ ((\leq 1 \ R \sqcap \geq 2 \ R)(x_0)) \} \]
We want to check whether the concept

\[ \leq 1 \ R \sqcap \geq 2 \ R \]

is satisfiable.

\[ \mathcal{A}_0 = \{(\leq 1 \ R \sqcap \geq 2 \ R)(x_0)\} \]

\[ \mathcal{A}_1 = \mathcal{A}_0 \cup \{(\leq 1 \ R)(x_0), (\geq 2 \ R)(x_0)\} \quad \text{by } \sqcap\text{-rule} \]
We want to check whether the concept

\[
\leq 1 \ R \sqcap \geq 2 \ R
\]

is satisfiable.

\[\begin{align*}
\mathcal{A}_0 &= \{ ((\leq 1 \ R \sqcap \geq 2 \ R)(x_0)) \} \\
\mathcal{A}_1 &= \{ \mathcal{A}_0^1 \cup \{ (\leq 1 \ R)(x_0), (\geq 2 \ R)(x_0) \} \} \quad \text{by \ \sqcap\text{-rule}} \\
\mathcal{A}_2 &= \{ \mathcal{A}_1^1 \cup \{ R(x_0, y_1), R(x_0, y_2), y_1 \neq y_2 \} \} \quad \text{by \ \geq\text{-rule}}
\end{align*}\]
Example derivation(2)

We want to check whether the concept

\[ \leq 1 \ R \sqcap \geq 2 \ R \]

is satisfiable.

\[ A_0 = \{ ((\leq 1 \ R \sqcap \geq 2 \ R)(x_0)) \} \]
\[ A_1 = \{ A_0^1 \cup \{(\leq 1 \ R)(x_0), (\geq 2 \ R)(x_0)\} \} \quad \text{by } \sqcap\text{-rule} \]
\[ A_2 = \{ A_1^1 \cup \{R(x_0, y_1), R(x_0, y_2), y_1 \neq y_2\} \} \quad \text{by } \geq\text{-rule} \]
Example derivation (3)

We want to check whether the concept

\[ \leq 1 \ R \cap \exists R. \ C \cap \exists R. \ D \]

is satisfiable.
Example derivation (3)

We want to check whether the concept

\[ \leq 1 \ R \cap \exists R. C \cap \exists R. D \]

is satisfiable.

\[ \mathcal{A}_0 = \{ (\leq 1 \ R \cap \exists R. C \cap \exists R. D)(x_0) \} \]
We want to check whether the concept

\[ \leq 1 \, R \cap \exists R. C \cap \exists R. D \]

is satisfiable.

\[ A_0 = \{((\leq 1 \, R \cap \exists R. C \cap \exists R. D)(x_0))\} \]

\[ A_1 = A_0 \cup \{((\leq 1 \, R)(x_0), (\exists R. C)(x_0), (\exists R. D)(x_0))\} \quad \text{by } \cap \text{-rule} \]
Example derivation (3)

We want to check whether the concept

\[ \leq 1 R \cap \exists R.C \cap \exists R.D \]

is satisfiable.

\[ \mathcal{A}_0 = \{ ((\leq 1 R \cap \exists R.C \cap \exists R.D)(x_0)) \} \]
\[ \mathcal{A}_1 = \mathcal{A}_0 \cup \{ (\leq 1 R)(x_0), (\exists R.C)(x_0), (\exists R.D)(x_0) \} \] by \( \cap \)-rule
\[ \mathcal{A}_2 = \mathcal{A}_1 \cup \{ R(x_0, y_1), C(y_1) \} \] by \( \exists \)-rule
We want to check whether the concept

\[ \leq 1 \ R \cap \exists R. C \cap \exists R. D \]

is satisfiable.

\[ \mathcal{A}_0 = \{ \{(\leq 1 \ R \cap \exists R. C \cap \exists R. D)(x_0)\}\} \]

\[ \mathcal{A}_1 = \{ \mathcal{A}_0^1 \cup \{(\leq 1 \ R)(x_0), (\exists R. C)(x_0), (\exists R. D)(x_0)\}\} \quad \text{by } \cap\text{-rule} \]

\[ \mathcal{A}_2 = \{ \mathcal{A}_1^1 \cup \{R(x_0, y_1), C(y_1)\}\} \quad \text{by } \exists\text{-rule} \]

\[ \mathcal{A}_3 = \{ \mathcal{A}_2^1 \cup \{R(x_0, y_2), D(y_2)\}\} \quad \text{by } \exists\text{-rule} \]
Example derivation (3)

We want to check whether the concept

\[ \leq 1 \ R \cap \exists R. C \cap \exists R. D \]

is satisfiable.

\[ A_0 = \{(\leq 1 \ R \cap \exists R. C \cap \exists R. D)(x_0)\} \]

\[ A_1 = \{A_0^1 \cup \{(\leq 1 \ R)(x_0), (\exists R. C)(x_0), (\exists R. D)(x_0)\}\} \quad \text{by } \cap\text{-rule} \]

\[ A_2 = \{A_1^1 \cup \{R(x_0, y_1), C(y_1)\}\} \quad \text{by } \exists\text{-rule} \]

\[ A_3 = \{A_2^1 \cup \{R(x_0, y_2), D(y_2)\}\} \quad \text{by } \exists\text{-rule} \]

\[ A_4 = \{A_3^1[y_2/y_1]\} \quad \text{by } \leq\text{-rule} \]
Example derivation (3)

We want to check whether the concept

\[ \leq 1 \ R \cap \exists R.\ C \cap \exists R.\ D \]

is satisfiable.

\[ A_0 = \{ ((\leq 1 \ R \cap \exists R.\ C \cap \exists R.\ D) (x_0)) \} \]

\[ A_1 = A_0 \cup \{ (\leq 1 \ R) (x_0), (\exists R.\ C) (x_0), (\exists R.\ D) (x_0) \} \text{ by } \sqcap \text{-rule} \]

\[ A_2 = A_1 \cup \{ R(x_0, y_1), C(y_1) \} \text{ by } \exists \text{-rule} \]

\[ A_3 = A_2 \cup \{ R(x_0, y_2), D(y_2) \} \text{ by } \exists \text{-rule} \]

\[ A_4 = A_3[y_2/y_1] \text{ by } \leq \text{-rule} \]

\[ = A_2 \cup \{ R(x_0, y_1), D(y_1) \} \]
Contents

Assumptions

Reasoning problems

Tableau algorithm for $ALCN$

Soundness, Completeness and Termination

Removing the assumptions

Complexity
Definitions

Definition

(i) The algorithm terminates on one ABox $\mathcal{A}$ when no rules are applicable. Such an ABox is then called *complete*.

(ii) A *clash* is an obvious contradiction, that is, $\mathcal{A}$ contains a clash if either:

- $\bot(x) \in \mathcal{A}$, or
- $\{C(x), \neg C(x)\} \subseteq \mathcal{A}$, or
- $\{(\leq n R)(x)\} \cup \{R(x, y_i) \mid 1 \leq i \leq n + 1\} \cup \{y_i \neq y_j \mid (1 \leq i < j \leq n + 1)\} \subseteq \mathcal{A}$. 
Let $\hat{S}$ be the set of complete ABoxes resulting from applying the tableau algorithm to $\{A\}$.

**Theorem (Soundness)**

If $A$ has a model, then at least one of the ABoxes of $\hat{S}$ has a model.

**Proof.** Done by induction on the proofs, showing that each rule preserves consistency.

**Theorem (Completeness)**

If at least one of the ABoxes, $\hat{A}$ of $\hat{S}$ is clash free, then at $A$ has a model.

**Proof.** Done by constructing a model for $A$ from $\hat{A}$.
Soundness and completeness

Let $\hat{S}$ be the set of complete ABoxes resulting from applying the tableau algorithm to $\{A\}$.

Theorem (Soundness)

*If $A$ has a model, then at least one of the ABoxes of $\hat{S}$ has a model.*

Proof.

Done by induction on the proofs, showing that each rule preserves consistency.  

Theorem (Completeness)

If at least one of the ABoxes, $\hat{A}$ of $\hat{S}$ is clash free, then at $A$ has a model.

Proof.

Done by constructing a model for $A$ from $\hat{A}$.
Soundness and completeness

Let $\hat{\mathcal{S}}$ be the set of complete ABoxes resulting from applying the tableau algorithm to $\{\mathcal{A}\}$.

Theorem (Soundness)
If $\mathcal{A}$ has a model, then at least one of the ABoxes of $\hat{\mathcal{S}}$ has a model.

Proof.
Done by induction on the proofs, showing that each rule preserves consistency.

Theorem (Completeness)
If at least one of the ABoxes, $\hat{\mathcal{A}}$ of $\hat{\mathcal{S}}$ is clash free, then at $\mathcal{A}$ has a model.

Proof.
Done by constructing a model for $\mathcal{A}$ from $\hat{\mathcal{A}}$. 
Assumptions

− We only allow equivalence axioms, $A \equiv D$, where $A$ is atomic and $D$ is not atomic. Each atomic concept should only occur once on a left-hand side.

− We only allow acyclic TBoxes, so e.g. no $A \equiv \exists R.D$, where $D$ is defined in terms of $A$.

− We only allow ABox axioms on the form $A(c)$ for atomic concepts $A$ (and $R(a, b)$ as usual). (Not really a restriction, as $D(c)$ for complex $D$ can be expressed as $A_D \equiv D$ and $A_D(c)$ for some fresh concept name $A_D$)
Assumptions

- We only allow equivalence axioms, $A \equiv D$, where $A$ is atomic and $D$ is not atomic. Each atomic concept should only occur once on a left-hand side.

- We only allow acyclic TBoxes, so e.g. no $A \equiv \exists R.D$, where $D$ is defined in terms of $A$.

- We only allow ABox axioms on the form $A(c)$ for atomic concepts $A$ (and $R(a, b)$ as usual). (Not really a restriction, as $D(c)$ for complex $D$ can be expressed as $A_D \equiv D$ and $A_D(c)$ for some fresh concept name $A_D$)

- **New:** We only allow ABoxes on the form $\{C(x_0)\}$ as input to the tableau algorithm.
Termination on single concept

Theorem (Termination)

If $C_0$ is an $\mathcal{ALCN}$-concept, then the tableau algorithm terminates on $\{\{C_0(x_0)\}\}$, that is, there cannot be an infinite sequence of rule applications

$$\{\{C_0(x_0)\}\} \rightarrow S_1 \rightarrow S_2 \rightarrow \ldots$$
To prove termination, we do the following:

- We first define a function $f$ mapping each state $S$ in the proof (each set of ABoxes) to a set $Q$ for which there is a strict well-ordering $<$. 
Proof-sketch of termination

To prove termination, we do the following:

– We first define a function $f$ mapping each state $S$ in the proof (each set of ABoxes) to a set $Q$ for which there is a strict well-ordering $<$. 

– Then, we prove that if $S'$ is the result of applying a rule to a state $S$, then $f(S') < f(S)$. 


Proof-sketch of termination

To prove termination, we do the following:

– We first define a function $f$ mapping each state $S$ in the proof (each set of ABoxes) to a set $Q$ for which there is a strict well-ordering $<$.  
– Then, we prove that if $S'$ is the result of applying a rule to a state $S$, then $f(S') < f(S)$.  
– The result then follows from the fact that any strictly decreasing sequence in a well-ordered set is finite.
Lemma

Let $A$ be an ABox contained in $S_i$ for some $i \geq 1$.

- For every individual $x \neq x_0$ occurring in $A$, there is a unique sequence $R_1, \ldots, R_l$ ($l \geq 1$) of role names and a unique sequence $x_1, \ldots, x_{l-1}$ of individual names such that $\{R_1(x_0, x_1), R_2(x_1, x_2), \ldots, R_l(x_{l-1}, x)\} \subseteq A$. In this case, we say that $x$ occurs on level $l$ in $A$. 
Proof-sketch of termination

Lemma

Let $A$ be an ABox contained in $S_i$ for some $i \geq 1$.

- For every individual $x \neq x_0$ occurring in $A$, there is a unique sequence $R_1, \ldots, R_l$ ($l \geq 1$) of role names and a unique sequence $x_1, \ldots, x_{l-1}$ of individual names such that $\{R_1(x_0, x_1), R_2(x_1, x_2), \ldots, R_l(x_{l-1}, x)\} \subseteq A$. In this case, we say that $x$ occurs on level $l$ in $A$.

- If $C(x) \in A$ for an individual $x$ on level $l$, then the maximal role depth of $C$ (i.e. the maximal nesting of constructors involving roles) is bounded by the maximal role depth of $C_0$ minus $l$. Consequently, the level of any individual in $A$ is bounded by the maximal role depth of $C_0$. 
Proof-sketch of termination

Lemma (Cont.)

– If $C(x) \in A$ then $C$ is a subdescription of $C_0$. Consequently, the number of different concept assertions on $x$ is bounded by the size of $C_0$. 
Lemma (Cont.)

– If $C(x) \in \mathcal{A}$ then $C$ is a subdescription of $C_0$. Consequently, the number of different concept assertions on $x$ is bounded by the size of $C_0$.

– The number of different role successors of $x$ in $\mathcal{A}$ (i.e. individuals $y$ such that $R(x, y) \in \mathcal{A}$ for a role name $R$) is bounded by the sum of the numbers occurring in the at-least restrictions in $C_0$ plus the number of different existential restrictions in $C_0$. 
Finite tree model property

The facts stated in this lemma imply the following:

– The canonical interpretation constructed by the tableaux algorithm has the shape of a finite tree;
Finite tree model property

The facts stated in this lemma imply the following:

- The canonical interpretation constructed by the tableaux algorithm has the shape of a finite tree;
- the depth of the tree is linearly bounded by the size of $C_0$;
Finite tree model property

The facts stated in this lemma imply the following:

- The canonical interpretation constructed by the tableaux algorithm has the shape of a finite tree;
- the depth of the tree is linearly bounded by the size of $C_0$;
- the branching factor of the tree is bounded by the sum of the numbers occurring in the at-least restrictions plus the number of different existential restrictions in $C_0$. 

This means that $ALCN$ enjoys the finite tree model property, that is, any satisfiable concept $C_0$ is satisfiable in a finite interpretation that has the shape of a tree whose root belongs to $C_0$. 
The facts stated in this lemma imply the following:

- The canonical interpretation constructed by the tableaux algorithm has the shape of a finite tree;
- the depth of the tree is linearly bounded by the size of $C_0$;
- the branching factor of the tree is bounded by the sum of the numbers occuring in the at-least restrictions pluss the number of different existential restrictions in $C_0$.
- This means that $\mathcal{ALCN}$ enjoys the finite tree model property, that is, any satisfiable concept $C_0$ is satisfiable in a finite interpretation that has the shape of a tree whose root belongs to $C_0$. 
Contents

Assumptions

Reasoning problems

Tableau algorithm for \( \mathcal{ALCN} \)

Soundness, Completeness and Termination

Removing the assumptions

Complexity
Extension to ABox problems

What happens if we apply the algorithm to a general extended $\mathcal{ALCN}$-ABox?
Extension to ABox problems

What happens if we apply the algorithm to a general extended $\mathcal{ALCN}$-ABox? It might not terminate:

$$\mathcal{A}_0 = \{R(a,a), (\leq 1 R)(a), (\forall R.\exists R.A)(a)\}$$
What happens if we apply the algorithm to a general extended $\mathcal{ALCN}$-ABox? It might not terminate:

\[
\begin{align*}
\mathcal{A}_0 & = \{ R(a, a), (\leq 1 R)(a), (\forall R. \exists R. A)(a) \} \\
\mathcal{A}_1 & = \mathcal{A}_0 \cup \{ (\exists R. A)(a) \}
\end{align*}
\]
Extension to ABox problems

What happens if we apply the algorithm to a general extended $\mathcal{ALCN}$-ABox? It might not terminate:

$$
\mathcal{A}_0 = \{R(a, a), (\leq 1 R)(a), (\forall R. \exists R.A)(a)\}
$$

$$
\mathcal{A}_1 = \mathcal{A}_0 \cup \{ (\exists R.A)(a) \}
$$

$$
\mathcal{A}_2 = \mathcal{A}_1 \cup \{R(a, x_0), A(x_0)\}
$$

However, if we restrict the $\exists$-rule and the $\geq$-rule to only be applicable when no other rules are, then we can guarantee termination also for general $\mathcal{ALCN}$-ABoxes.
Extension to ABox problems

What happens if we apply the algorithm to a general extended $\mathcal{ALCN}^\prime$-ABox? It might not terminate:

$\mathcal{A}_0 = \{ R(a, a), (\leq 1 R)(a), (\forall R. \exists R.A)(a) \}$

$\mathcal{A}_1 = \mathcal{A}_0 \cup \{ (\exists R.A)(a) \}$

$\mathcal{A}_2 = \mathcal{A}_1 \cup \{ R(a, x_0), A(x_0) \}$

$\mathcal{A}_3 = \mathcal{A}_2 \cup \{ (\exists R.A)(x_0) \}$

However, if we restrict the $\exists$-rule and the $\geq$-rule to only be applicable when no other rules are, then we can guarantee termination also for general $\mathcal{ALCN}^\prime$-ABoxes.
Extension to ABox problems

What happens if we apply the algorithm to a general extended $\mathcal{ALCN}$-ABox? It might not terminate:

\[
\begin{align*}
A_0 &= \{ R(a, a), (\leq 1 R)(a), (\forall R. \exists R.A)(a) \} \\
A_1 &= A_0 \cup \{ (\exists R.A)(a) \} \\
A_2 &= A_1 \cup \{ R(a, x_0), A(x_0) \} \\
A_3 &= A_2 \cup \{ (\exists R.A)(x_0) \} \\
A_4 &= A_3 \cup \{ R(x_0, x_1), A(x_1) \}
\end{align*}
\]

However, if we restrict the $\exists$-rule and the $\geq$-rule to only be applicable when no other rules are, then we can guarantee termination also for general $\mathcal{ALCN}$-ABoxes.
Extension to ABox problems

What happens if we apply the algorithm to a general extended $\textit{ALCN}$-ABox? It might not terminate:

\begin{align*}
\mathcal{A}_0 &= \{ R(a, a), (\leq 1 R)(a), (\forall R. \exists R. A)(a) \} \\
\mathcal{A}_1 &= \mathcal{A}_0 \cup \{(\exists R. A)(a)\} \\
\mathcal{A}_2 &= \mathcal{A}_1 \cup \{ R(a, x_0), A(x_0) \} \\
\mathcal{A}_3 &= \mathcal{A}_2 \cup \{(\exists R. A)(x_0)\} \\
\mathcal{A}_4 &= \mathcal{A}_3 \cup \{ R(x_0, x_1), A(x_1) \} \\
\mathcal{A}_4 &= \mathcal{A}_1 \cup \{ A(a), R(a, x_1), A(x_1) \}
\end{align*}

However, if we restrict the $\exists$-rule and the $\geq$-rule to only be applicable when no other rules are, then we can guarantee termination also for general $\textit{ALCN}$-ABoxes.
Extension to ABox problems

What happens if we apply the algorithm to a general extended $\mathcal{ALCN}$-ABox? It might not terminate:

\begin{align*}
\mathcal{A}_0 &= \{R(a, a), (\leq 1 R)(a), (\forall R. \exists R.A)(a)\} \\
\mathcal{A}_1 &= \mathcal{A}_0 \cup \{(\exists R.A)(a)\} \\
\mathcal{A}_2 &= \mathcal{A}_1 \cup \{R(a, x_0), A(x_0)\} \\
\mathcal{A}_3 &= \mathcal{A}_2 \cup \{(\exists R.A)(x_0)\} \\
\mathcal{A}_4 &= \mathcal{A}_3 \cup \{R(x_0, x_1), A(x_1)\} \\
\mathcal{A}_4 &= \mathcal{A}_1 \cup \{A(a), R(a, x_1), A(x_1)\} \\
\vdots \\
\mathcal{A}_i &= \mathcal{A}_1 \cup \{A(a), R(a, x_j), A(x_j)\}
\end{align*}
Extension to ABox problems

What happens if we apply the algorithm to a general extended $\mathcal{ALCN}$-ABox? It might not terminate:

$$\mathcal{A}_0 = \{R(a, a), (\leq 1 R)(a), (\forall R. \exists R. A)(a)\}$$

$$\mathcal{A}_1 = \mathcal{A}_0 \cup \{(\exists R. A)(a)\}$$

$$\mathcal{A}_2 = \mathcal{A}_1 \cup \{R(a, x_0), A(x_0)\}$$

$$\mathcal{A}_3 = \mathcal{A}_2 \cup \{(\exists R. A)(x_0)\}$$

$$\mathcal{A}_4 = \mathcal{A}_3 \cup \{R(x_0, x_1), A(x_1)\}$$

$$\mathcal{A}_4 = \mathcal{A}_1 \cup \{A(a), R(a, x_1), A(x_1)\}$$

$$\vdots$$

$$\mathcal{A}_i = \mathcal{A}_1 \cup \{A(a), R(a, x_j), A(x_j)\}$$

However, if we restrict the $\exists$-rule and the $\geq$-rule to only be applicable when no other rules are, then we can guarantee termination also for general $\mathcal{ALCN}$-ABoxes.
Other extensions

Atomic Inclusions

- If we allow (acyclic) Aboxes with inclusions of the form $A \sqsubseteq C$ where $A$ is a base name, then we can just make a fresh concept $A_{\text{new}}$ and extend the inclusion to a definition, by replacing it with $A \equiv C \sqcap A_{\text{new}}$. 

E.g. $\text{Donkey} \sqsubseteq \text{Animal} \sqcap \text{Stubborn} \Downarrow \text{Donkey} \equiv \text{Animal} \sqcap \text{Stubborn} \sqcap \text{Donkey}_{\text{new}}$
If we allow (acyclic) Aboxes with inclusions of the form $A \sqsubseteq C$ where $A$ is a base name, then we can just make a fresh concept $A_{new}$ and extend the inclusion to a definition, by replacing it with $A \equiv C \cap A_{new}$.

E.g.

$$Donkey \sqsubseteq Animal \cap Stubborn$$

$$\downarrow$$

$$Donkey \equiv Animal \cap Stubborn \cap Donkey_{new}$$
Other extensions

General Inclusions

– If we allow TBoxes with general inclusions of the form $C \sqsubseteq D$ for complex concepts $C$ and $D$, then it is enough to only handle the inclusion

$$\top \sqsubseteq (\neg C_1 \sqcup D_1) \cap (\neg C_2 \sqcup D_2) \cap \cdots \cap (\neg C_n \sqcup D_n)$$

where $\{C_i \sqsubseteq D_i | 1 \leq i \leq n\}$ are all the inclusions in the TBox.

– Thus, for any individual $x$ in the ABox, we can just add $(\neg C_1 \sqcup D_1) \cap (\neg C_2 \sqcup D_2) \cap \cdots \cap (\neg C_n \sqcup D_n)(x)$ whenever they are introduced.

– However, we now lose termination, for instance for the knowledge base with ABox $\{\top(x_0)\}$ and TBox $\{\top \sqsubseteq \exists R.\top\}$. This can be fixed with blocking.
Other extensions

General Inclusions

– If we allow TBoxes with general inclusions of the form $C \sqsubseteq D$ for complex concepts $C$ and $D$, then it is enough to only handle the inclusion

$$\top \sqsubseteq (\neg C_1 \sqcup D_1) \cap (\neg C_2 \sqcup D_2) \cap \cdots \cap (\neg C_n \sqcup D_n)$$

where $\{C_i \sqsubseteq D_i | 1 \leq i \leq n\}$ are all the inclusions in the TBox.

– Thus, for any individual $x$ in the ABox, we can just add

$$((\neg C_1 \sqcup D_1) \cap (\neg C_2 \sqcup D_2) \cap \cdots \cap (\neg C_n \sqcup D_n))(x)$$

whenever they are introduced.
Other extensions

General Inclusions

- If we allow TBoxes with general inclusions of the form $C \sqsubseteq D$ for complex concepts $C$ and $D$, then it is enough to only handle the inclusion

$$\top \sqsubseteq (\neg C_1 \sqcup D_1) \sqcap (\neg C_2 \sqcup D_2) \sqcap \cdots \sqcap (\neg C_n \sqcup D_n)$$

where $\{C_i \sqsubseteq D_i | 1 \leq i \leq n\}$ are all the inclusions in the TBox.

- Thus, for any individual $x$ in the ABox, we can just add

$$( (\neg C_1 \sqcup D_1) \sqcap (\neg C_2 \sqcup D_2) \sqcap \cdots \sqcap (\neg C_n \sqcup D_n))(x)$$

whenever they are introduced.

- However, we now lose termination, for instance for the knowledge base with ABox $\{\top(x_0)\}$ and TBox $\{\top \sqsubseteq \exists R. \top\}$. 
Other extensions

General Inclusions

- If we allow TBoxes with general inclusions of the form $C \sqsubseteq D$ for complex concepts $C$ and $D$, then it is enough to only handle the inclusion

$$
\top \sqsubseteq (\neg C_1 \sqcup D_1) \sqcap (\neg C_2 \sqcup D_2) \sqcap \cdots \sqcap (\neg C_n \sqcup D_n)
$$

where $\{C_i \sqsubseteq D_i | 1 \leq i \leq n\}$ are all the inclusions in the TBox.

- Thus, for any individual $x$ in the ABox, we can just add

$$
((\neg C_1 \sqcup D_1) \sqcap (\neg C_2 \sqcup D_2) \sqcap \cdots \sqcap (\neg C_n \sqcup D_n))(x)
$$

whenever they are introduced.

- However, we now lose termination, for instance for the knowledge base with ABox $\{\top(x_0)\}$ and TBox $\{\top \sqsubseteq \exists R. \top\}$.

- This can be fixed with blocking.
Other extensions

Inclusions: Blocking

Definition
We will say that a variable $y$ is an ancestor of a variable $x$ if there exists some $R$ where either $R(y, x)$, or there exists some variable $z$ where $z$ is an ancestor of $x$ and $R(y, z)$.
Other extensions

Inclusions: Blocking

Definition
We will say that a variable $y$ is an *ancestor* of a variable $x$ if there exists some $R$ where either $R(y, x)$, or there exists some variable $z$ where $z$ is an ancestor of $x$ and $R(y, z)$.

Definition
We say that an application of a $\exists$-rule or a $\geq$-rule to a variable $x$ is *directly blocked* by a variable $y$ if

$$\{D | D(x) \in \mathcal{A}\} \subseteq \{D' | D'(y) \in \mathcal{A}\}$$

and $y$ is an *ancestor* of $x$. 

Other extensions

Inclusions: Blocking

Definition
We will say that a variable $y$ is an ancestor of a variable $x$ if there exists some $R$ where either $R(y, x)$, or there exists some variable $z$ where $z$ is an ancestor of $x$ and $R(y, z)$.

Definition
We say that an application of a $\exists$-rule or a $\geq$-rule to a variable $x$ is directly blocked by a variable $y$ if

\[
\{D | D(x) \in A\} \subseteq \{D' | D'(y) \in A\}
\]

and $y$ is an ancestor of $x$.

Definition
We say that an application of a $\exists$-rule or a $\geq$-rule to a variable $x$ is blocked if it is directly blocked, or if an ancestor of $x$ is directly blocked.
Contents

Assumptions

Reasoning problems

Tableau algorithm for $ALCN$

Soundness, Completeness and Termination

Removing the assumptions

Complexity
Complexity of algorithm on empty TBox

**Theorem**

*Satisfiability of $\mathcal{ALCN}$-concepts and consistency checking of $\mathcal{ALCN}$-ABoxes is $\mathsf{PSpace}$-complete for acyclic TBoxes.*
Complexity of algorithm on empty TBox

Theorem
Satisfiability of $\text{ALCN}$-concepts and consistency checking of $\text{ALCN}$-ABoxes is $\text{PSpace}$-complete for acyclic TBoxes.

Proof (Sketch).

In $\text{PSpace}$: If we alter the algorithm accordingly:

(i) Apply $\forall$-, $\sqcap$- and $\sqcup$-rules as long as possible, and look for clashes of the form $\bot(x)$ and $A(x), \neg A(x)$.

(ii) Generate new individuals with the $\exists$- and $\geq$-rules.

(iii) Identify equivalences with the $\leq$-rule, and check for $\leq$-clashes.

(iv) Successively handle the successors in the same way. Generated successors can be treated separately, so we only need to store one path of the tree. Furthermore, we do not need to generate all $n$ individuals for every $(\geq n R(x))$. 

$\text{PSpace}$-hard: Can be reduced to validity of Quantified Boolean Formula.
Complexity of algorithm on empty TBox

Theorem

Satisfiability of ALCN-concepts and consistency checking of ALCN-ABoxes is PSPACE-complete for acyclic TBoxes.

Proof (Sketch).

In PSPACE: If we alter the algorithm accordingly:

(i) Apply ∀-, ∩- and ∪-rules as long as possible, and look for clashes of the form ⊥(x) and A(x), ¬A(x).

(ii) Generate new individuals with the ∃- and ≥-rules.
Complexity of algorithm on empty TBox

Theorem

Satisfiability of $\mathcal{ALCN}$-concepts and consistency checking of $\mathcal{ALCN}$-ABoxes is PSPACE-complete for acyclic TBoxes.

Proof (Sketch).

In PSPACE: If we alter the algorithm accordingly:

(i) Apply $\forall$-, $\sqcap$- and $\sqcup$-rules as long as possible, and look for clashes of the form $\bot(x)$ and $A(x), \neg A(x)$.

(ii) Generate new individuals with the $\exists$- and $\geq$-rules.

(iii) Identify equivalences with the $\leq$-rule, and check for $\leq$-clashes.
Complexity of algorithm on empty TBox

Theorem

Satisfiability of\( ALCN \)-concepts and consistency checking of\( ALCN \)-ABoxes is \( \text{PSpace-complete} \) for acyclic TBoxes.

Proof (Sketch).

In \( \text{PSpace} \): If we alter the algorithm accordingly:

(i) Apply \( \forall \)-, \( \sqcap \)- and \( \sqcup \)-rules as long as possible, and look for clashes of the form \( \bot(x) \) and \( A(x), \neg A(x) \).

(ii) Generate new individuals with the \( \exists \)- and \( \geq \)-rules.

(iii) Identify equivalences with the \( \leq \)-rule, and check for \( \leq \)-clashes.

(iv) Successively handle the successors in the same way.
Complexity of algorithm on empty TBox

Theorem
Satisfiability of $\text{ALCN}$-concepts and consistency checking of $\text{ALCN}$-ABoxes is $\text{PSPACE}$-complete for acyclic TBoxes.

Proof (Sketch).

In $\text{PSPACE}$: If we alter the algorithm accordingly:

(i) Apply $\forall$-, $\sqcap$- and $\sqcup$-rules as long as possible, and look for clashes of the form $\bot(x)$ and $A(x), \neg A(x)$.

(ii) Generate new individuals with the $\exists$- and $\geq$-rules.

(iii) Identify equivalences with the $\leq$-rule, and check for $\leq$-clashes.

(iv) Successively handle the successors in the same way.

Generated successors can be treated separately, so we only need to store one path of the tree.
Complexity of algorithm on empty TBox

Theorem

Satisfiability of $\mathcal{ALCN}$-concepts and consistency checking of $\mathcal{ALCN}$-ABoxes is $\text{PSPACE}$-complete for acyclic TBoxes.

Proof (Sketch).

In $\text{PSPACE}$: If we alter the algorithm accordingly:

(i) Apply $\forall$, $\sqcap$- and $\sqcup$-rules as long as possible, and look for clashes of the form $\bot(x)$ and $A(x), \neg A(x)$.

(ii) Generate new individuals with the $\exists$- and $\geq$-rules.

(iii) Identify equivalences with the $\leq$-rule, and check for $\leq$-clashes.

(iv) Successively handle the successors in the same way.

Generated successors can be treated separately, so we only need to store one path of the tree. Furthermore, we do not need to generate all $n$ individuals for every $(\geq n R)(x)$. 
**Complexity of algorithm on empty TBox**

**Theorem**

Satisfiability of $\mathcal{ALCN}$-concepts and consistency checking of $\mathcal{ALCN}$-ABoxes is $\text{PSPACE}$-complete for acyclic TBoxes.

**Proof (Sketch).**

In $\text{PSPACE}$: If we alter the algorithm accordingly:

(i) Apply $\forall$-, $\cap$- and $\sqcup$-rules as long as possible, and look for clashes of the form $\bot(x)$ and $A(x), \neg A(x)$.

(ii) Generate new individuals with the $\exists$- and $\geq$-rules.

(iii) Identify equivalences with the $\leq$-rule, and check for $\leq$-clashes.

(iv) Successively handle the successors in the same way.

Generated successors can be treated separately, so we only need to store one path of the tree. Furthermore, we do not need to generate all $n$ individuals for every $(\geq n \mathcal{R})(x)$.

$\text{PSPACE}$-hard: Can be reduced to validity of Quantified Boolean Formula.
### More complexity results

<table>
<thead>
<tr>
<th>DL</th>
<th>Combined complexity</th>
<th>Data complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{ALC}$</td>
<td>ExpTime-complete</td>
<td>NP-complete</td>
</tr>
<tr>
<td>$\mathcal{ALCN}$</td>
<td>ExpTime-complete</td>
<td>NP-complete</td>
</tr>
<tr>
<td>$\mathcal{SHIQ}$</td>
<td>ExpTime-complete</td>
<td>NP-complete</td>
</tr>
<tr>
<td>$\mathcal{SHOIN}(D)$</td>
<td>NExpTime-complete</td>
<td>NP-hard</td>
</tr>
<tr>
<td>$\mathcal{SROIQ}$</td>
<td>N2ExpTime-complete</td>
<td>NP-hard</td>
</tr>
<tr>
<td>$\mathcal{EL}$</td>
<td>P-complete</td>
<td>P-complete</td>
</tr>
<tr>
<td>$\mathcal{RL}$</td>
<td>P-complete</td>
<td>P-complete</td>
</tr>
<tr>
<td>$\mathcal{DL}_{\text{Lite}}$</td>
<td>$\log P$</td>
<td>$\log \text{LOGSPACE}$</td>
</tr>
</tbody>
</table>
For more info see:

- F. Baader and W. Nutt’s chapter *Basic Description Logics* from *The Description Logic Handbook*.


Thanks for listening!