Principles of Knowledge Representation and Reasoning

5. Semantic Networks and Description Logics

5.5 Description Logics – Algorithms

Bernhard Nebel

• Motivation
• Structural Subsumption Algorithms
• Tableaux Algorithms
Reasoning Problems & Algorithms

- Satisfiability or subsumption of concept descriptions
Reasoning Problems & Algorithms

- *Satisfiability* or *subsumption* of concept descriptions
- *Satisfiability* or *instance relation* in ABoxes
Reasoning Problems & Algorithms

- Satisfiability or subsumption of concept descriptions
- Satisfiability or instance relation in ABoxes

Structural subsumption algorithms
Reasoning Problems & Algorithms

- *Satisfiability* or *subsumption* of concept descriptions
- *Satisfiability* or *instance relation* in ABoxes

⇒ **Structural subsumption algorithms**
  - *Normalization* of concept descriptions and *structural comparison*
Reasoning Problems & Algorithms

- *Satisfiability* or *subsumption* of concept descriptions
- *Satisfiability* or *instance relation* in ABoxes

Structural subsumption algorithms
- *Normalization* of concept descriptions and *structural comparison*
- very fast, but can only be used for small DLs
Reasoning Problems & Algorithms

- Satisfiability or subsumption of concept descriptions
- Satisfiability or instance relation in ABoxes

Structural subsumption algorithms
  - Normalization of concept descriptions and structural comparison
  - very fast, but can only be used for small DLs

Tableaux algorithms
**Reasoning Problems & Algorithms**

- *Satisfiability* or *subsumption* of concept descriptions
- *Satisfiability* or *instance relation* in ABoxes

→ **Structural subsumption algorithms**
  - *Normalization* of concept descriptions and *structural comparison*
  - very fast, but can only be used for small DLs

→ **Tableaux algorithms**
  - Similar to modal tableaux methods
Reasoning Problems & Algorithms

- *Satisfiability* or *subsumption* of concept descriptions
- *Satisfiability* or *instance relation* in ABoxes

~~ Structural subsumption algorithms
  - *Normalization* of concept descriptions and *structural comparison*
  - very fast, but can only be used for small DLs

~~ Tableaux algorithms
  - Similar to modal tableaux methods
  - Meanwhile the method of choice
Structural Subsumption Algorithms

- Small Logic $\mathcal{FL}^-$
Structural Subsumption Algorithms

• Small Logic $\mathcal{FL}^-$
  - $C \cap D$
  - $\forall r. C$
  - $\exists r$ (simple existential quantification)
Structural Subsumption Algorithms

- **Small Logic** $\mathcal{FL}^-$
  - $C \sqcap D$
  - $\forall r. C$
  - $\exists r$ (simple existential quantification)

- **Idea**
  1. In the conjunction, collect all *universally quantified expressions* (also called *value restrictions*) with the same role and build *complex value restriction*:

$$\forall r. C \sqcap \forall r. D \rightarrow \forall r. (C \sqcap D).$$
Structural Subsumption Algorithms

- **Small Logic** $FL^{-}$
  - $C \sqcap D$
  - $\forall r.C$
  - $\exists r$ (simple existential quantification)

- **Idea**
  1. In the conjunction, collect all *universally quantified expressions* (also called *value restrictions*) with the same role and build *complex value restriction*:

\[
\forall r.C \sqcap \forall r.D \rightarrow \forall r. (C \sqcap D).
\]

  2. Compare all conjuncts with each other.
Structural Subsumption Algorithms

- **Small Logic** $\mathcal{FL}^-$
  - $C \sqcap D$
  - $\forall r. C$
  - $\exists r$ (simple existential quantification)

- **Idea**
  1. In the conjunction, collect all *universally quantified expressions* (also called *value restrictions*) with the same role and build *complex value restriction*:

$$\forall r. C \sqcap \forall r. D \implies \forall r. (C \sqcap D).$$

  2. Compare all conjuncts with each other. For each conjunct in the subsuming concept there should be a *corresponding one* in the subsumed one.
Example

\[ D = \text{Human} \cap \exists \text{has-child} \cap \forall \text{has-child.Human} \cap \forall \text{has-child.} \exists \text{has-child} \]

\[ C = \text{Human} \cap \text{Female} \cap \exists \text{has-child} \cap \forall \text{has-child.}(\text{Human} \cap \text{Female} \cap \exists \text{has-child}) \]
\[ D = \text{Human} \sqcap \exists \text{has-child} \sqcap \forall \text{has-child.} \text{Human} \sqcap \forall \text{has-child.} \exists \text{has-child} \]

\[ C = \text{Human} \sqcap \text{Female} \sqcap \exists \text{has-child} \sqcap \forall \text{has-child.} (\text{Human} \sqcap \text{Female} \sqcap \exists \text{has-child}) \]

Check: \( C \subseteq D \)
Example

\[ D = \text{Human} \cap \exists \text{has-child} \cap \forall \text{has-child}. \text{Human} \cup \forall \text{has-child}. \exists \text{has-child} \]

\[ C = \text{Human} \cap \text{Female} \cap \exists \text{has-child} \cap \forall \text{has-child}. (\text{Human} \cap \text{Female} \cap \exists \text{has-child}) \]

Check: \( C \subseteq D \)

1. Collect value restrictions in \( D \): \( \ldots \forall \text{has-child}. (\text{Human} \cap \exists \text{has-child}) \)
Example

\[ D = \text{Human} \cap \exists \text{has-child} \cap \forall \text{has-child}.\text{Human} \cap \forall \text{has-child}.\exists \text{has-child} \]
\[ C = \text{Human} \cap \text{Female} \cap \exists \text{has-child} \cap \forall \text{has-child}.(\text{Human} \cap \text{Female} \cap \exists \text{has-child}) \]

Check: \( C \subseteq D \)

1. **Collect value restrictions in** \( D \): \( \ldots \forall \text{has-child}.(\text{Human} \cap \exists \text{has-child}) \)

2. **Compare:**
   2.1. **For** \text{Human} **in** \( D \), **we have** \text{Human} **in** \( C \)
Example

\[ D = \text{Human} \cap \exists \text{has-child} \cap \forall \text{has-child.} \text{Human} \cap \forall \text{has-child.} \exists \text{has-child} \]

\[ C = \text{Human} \cap \text{Female} \cap \exists \text{has-child} \cap \forall \text{has-child.} (\text{Human} \cap \text{Female} \cap \exists \text{has-child}) \]

Check: \( C \subseteq D \)

1. **Collect value restrictions in** \( D \): \( \ldots \forall \text{has-child.}(\text{Human} \cap \exists \text{has-child}) \)

2. **Compare:**
   
   2.1. For \text{Human} in \( D \), we have \text{Human} in \( C \)

   2.2. For \( \exists \text{has-child} \) in \( D \), we have \ldots
Example

\[D = \text{Human} \cap \exists \text{has-child} \cap \forall \text{has-child}. \text{Human} \cap \forall \text{has-child}. \exists \text{has-child}\]

\[C = \text{Human} \cap \text{Female} \cap \exists \text{has-child} \cap \forall \text{has-child}.(\text{Human} \cap \text{Female} \cap \exists \text{has-child})\]

Check: \(C \subseteq D\)

1. **Collect value restrictions in** \(D\): \(\ldots \forall \text{has-child}.(\text{Human} \cap \exists \text{has-child})\)

2. **Compare**:
   2.1 For \text{Human} in \(D\), we have \text{Human} in \(C\)
   2.2 For \(\exists \text{has-child}\) in \(D\), we have \ldots
   2.3 For \(\forall \text{has-child}.(\ldots)\) in \(D\), we have \ldots
Example

\[ D \ = \ \text{Human} \cap \exists \text{has-child} \cap \forall \text{has-child} \cdot \text{Human} \cap \forall \text{has-child} \cdot \exists \text{has-child} \]

\[ C \ = \ \text{Human} \cap \text{Female} \cap \exists \text{has-child} \cap \forall \text{has-child} \cdot (\text{Human} \cap \text{Female} \cap \exists \text{has-child}) \]

Check: \( C \sqsubseteq D \)

1. Collect value restrictions in \( D \): \( \ldots \forall \text{has-child} \cdot (\text{Human} \cap \exists \text{has-child}) \)
2. Compare:
   2.1 For \( \text{Human} \) in \( D \), we have \( \text{Human} \) in \( C \)
   2.2 For \( \exists \text{has-child} \) in \( D \), we have \( \ldots \)
   2.3 For \( \forall \text{has-child} \cdot (\ldots) \) in \( D \), we have \( \ldots \)
       2.3.1 For \( \text{Human} \) \( \ldots \)
Example

\[ D = \text{Human} \cap \exists \text{has-child} \cap \forall \text{has-child}. \text{Human} \cap \forall \text{has-child}. \exists \text{has-child} \]

\[ C = \text{Human} \cap \text{Female} \cap \exists \text{has-child} \cap \forall \text{has-child}. (\text{Human} \cap \text{Female} \cap \exists \text{has-child}) \]

Check: \( C \subseteq D \)

1. Collect value restrictions in \( D \): ...\( \forall \text{has-child}. (\text{Human} \cap \exists \text{has-child}) \)

2. Compare:
   2.1 For \( \text{Human} \) in \( D \), we have \( \text{Human} \) in \( C \)
   2.2 For \( \exists \text{has-child} \) in \( D \), we have ...
   2.3 For \( \forall \text{has-child}. (...) \) in \( D \), we have ...
      2.3.1 For \( \text{Human} \) ...
      2.3.2 For \( \exists \text{has-child} \)...
Example

\[ D = \text{Human} \cap \exists \text{has-child} \cap \forall \text{has-child.} \text{Human} \cap \forall \text{has-child.} \exists \text{has-child} \]

\[ C = \text{Human} \cap \text{Female} \cap \exists \text{has-child} \cap \forall \text{has-child.}(\text{Human} \cap \text{Female} \cap \exists \text{has-child}) \]

Check: \( C \subseteq D \)

1. **Collect value restrictions in** \( D \): \( \ldots \forall \text{has-child.}(\text{Human} \cap \exists \text{has-child}) \)

2. **Compare:**
   
   2.1 For \( \text{Human} \) in \( D \), we have \( \text{Human} \) in \( C \)
   
   2.2 For \( \exists \text{has-child} \) in \( D \), we have \( \ldots \)
   
   2.3 For \( \forall \text{has-child.}(\ldots) \) in \( D \), we have \( \ldots \)
      
      2.3.1 For \( \text{Human} \ldots \)
      
      2.3.2 For \( \exists \text{has-child} \ldots \)

\( \leadsto C \) is subsumed by \( D \)!
Subsumption Algorithm

SUB($C, D$) algorithm:

1. Reorder terms (*commutativity, associativity* and *value restriction law*):

   \[
   C = \bigcap A_i \cap \bigcap \exists r_j \cap \bigcap \forall r_k : C_k
   \]
   \[
   D = \bigcap B_l \cap \bigcap \exists s_m \cap \bigcap \forall s_n : D_n
   \]
Subsumption Algorithm

**SUB**(*C, D*) algorithm:

1. Reorder terms (commutativity, associativity and value restriction law):

   \[
   C = \prod A_i \cap \prod \exists r_j \cap \prod \forall r_k : C_k
   \]
   \[
   D = \prod B_l \cap \prod \exists s_m \cap \prod \forall s_n : D_n
   \]

2. For each \(B_l\) in \(D\), is there an \(A_i\) in \(C\) with \(A_i = B_l\)?
**Subsumption Algorithm**

**$	ext{SUB}(C, D)$ algorithm:**

1. Reorder terms (*commutativity, associativity* and *value restriction law*):

   \[
   C = \prod A_i \cap \prod \exists r_j \cap \prod \forall r_k : C_k \\
   D = \prod B_l \cap \prod \exists s_m \cap \prod \forall s_n : D_n
   \]

2. For each $B_l$ in $D$, is there an $A_i$ in $C$ with $A_i = B_l$?

3. For each $\exists s_m$ in $D$, is there an $\exists r_j$ in $C$ with $s_m = r_j$?
Subsumption Algorithm

\textbf{SUB}(C, D) algorithm:

1. Reorder terms (\textit{commutativity, associativity} and \textit{value restriction law}):

\[
C = \bigcap A_i \cap \bigcap \exists r_j \cap \bigcap \forall r_k : C_k \\
D = \bigcap B_l \cap \bigcap \exists s_m \cap \bigcap \forall s_n : D_n
\]

2. For each \(B_l\) in \(D\), is there an \(A_i\) in \(C\) with \(A_i = B_l\)?

3. For each \(\exists s_m\) in \(D\), is there an \(\exists r_j\) in \(C\) with \(s_m = r_j\)?

4. For each \(\forall s_n : D_n\) in \(D\), is there a \(\forall r_k : C_k\) in \(C\) such that \(C_k \sqsubseteq D_n\) and \(s_n = r_k\)?
Subsumption Algorithm

**SUB**(*C, D*) algorithm:

1. Reorder terms (*commutativity, associativity* and *value restriction law*):

   \[
   C = \bigcap A_i \cap \bigcap \exists r_j \cap \bigcap \forall r_k : C_k
   \]

   \[
   D = \bigcap B_l \cap \bigcap \exists s_m \cap \bigcap \forall s_n : D_n
   \]

2. For each *B_l* in *D*, is there an *A_i* in *C* with *A_i = B_l*?

3. For each *∃s_m* in *D*, is there an *∃r_j* in *C* with *s_m = r_j*?

4. For each *∀s_n : D_n* in *D*, is there a *∀r_k : C_k* in *C* such that *C_k ⊑ D_n* and *s_n = r_k*?

   \[
   \leadsto C \sqsubseteq D \text{ iff all questions are answered positively}
   \]
Soundness

\[ \text{SUB}(C, D) \Rightarrow C \sqsubseteq D \]

**Proof sketch**

*Reordering of terms (1):*

a) Commutativity and associativity are trivial
Soundness

\[ \text{SUB}(C, D) \Rightarrow C \sqsubseteq D \]

Proof sketch

Reordering of terms (1):

a) Commutativity and associativity are trivial

b) Value restriction law. We show: 

\[
(\forall r. (C \sqcap D))^\mathcal{I} = (\forall r. C \sqcap \forall r. D)^\mathcal{I}
\]
Soundness

\[
\text{SUB}(C, D) \Rightarrow C \sqsubseteq D
\]

**Proof sketch**

*Reordering of terms (1):*

a) Commutativity and associativity are trivial

b) Value restriction law. We show: 
\[
(\forall r. (C \sqcap D))^I = (\forall r. C \sqcap \forall r. D)^I
\]

**Assumption:** 
\[
d \in (\forall r. (C \sqcap D))^I
\]
Soundness

\[ \text{SUB}(C, D) \Rightarrow C \subseteq D \]

Proof sketch

*Reordering of terms (1):*

a) Commutativity and associativity are trivial

b) Value restriction law. We show: \( (\forall r.(C \sqcap D))^I = (\forall r.C \sqcap \forall r.D)^I \)

**Assumption:** \( d \in (\forall r.(C \sqcap D))^I \)

**Case 1:** \( \forall e : (d, e) \in r^I \)
Soundness

\[ \text{SUB}(C, D) \Rightarrow C \subseteq D \]

Proof sketch

Reordering of terms (1):

a) Commutativity and associativity are trivial

b) Value restriction law. We show:

\[ (\forall r. (C \cap D))^I = (\forall r. C \cap \forall r. D)^I \]

**Assumption:** \( d \in (\forall r. (C \cap D))^I \)

**Case 1:** \( \exists e : (d, e) \in r^I \) \( \sqrt{\} \)
Soundness

$\text{SUB}(C, D) \Rightarrow C \subseteq D$

Proof sketch

Reordering of terms (1):

a) Commutativity and associativity are trivial

b) Value restriction law. We show: $(\forall r. (C \sqcap D))^I = (\forall r. C \sqcap \forall r. D)^I$

Assumption: $d \in (\forall r. (C \sqcap D))^I$

Case 1: $\not\exists e : (d, e) \in r^I \quad \checkmark$

Case 2: $\exists e : (d, e) \in r^I \Rightarrow e \in (C \sqcap D)^I \Rightarrow e \in C^I, \ e \in D^I$
Soundness

\[ \text{SUB}(C, D) \Rightarrow C \subseteq D \]

Proof sketch

Reordering of terms (1):

a) Commutativity and associativity are trivial

b) Value restriction law. We show: 

\[ (\forall r. (C \cap D))^I = (\forall r. C \cap \forall r. D)^I \]

Assumption: \( d \in (\forall r. (C \cap D))^I \)

Case 1: \( \forall e \in (d,e) \in r^I \) \( \sqrt{\} \)

Case 2: \( \exists e \in (d,e) \in r^I \Rightarrow e \in (C \cap D)^I \Rightarrow e \in C^I, e \in D^I \)

Since \( e \) is arbitrary: \( d \in (\forall r. C)^I, d \in (\forall r. D)^I \) then \( d \) must also be conjunction
Soundness

\[ \text{SUB}(C, D) \Rightarrow C \subseteq D \]

Proof sketch

Reordering of terms (1):

a) Commutativity and associativity are trivial

b) Value restriction law. We show: \( (\forall r. (C \cap D))^I = (\forall r. C \cap \forall r. D)^I \)

Assumption: \( d \in (\forall r. (C \cap D))^I \)

Case 1: \( \forall e : (d, e) \in r^I \quad \sqrt{\phantom{a}} \)

Case 2: \( \exists e : (d, e) \in r^I \Rightarrow e \in (C \cap D)^I \Rightarrow e \in C^I, e \in D^I \)

Since \( e \) is arbitrary: \( d \in (\forall r. C)^I, d \in (\forall r. D)^I \) then \( d \) must also be conjunction, i.e., \( (\forall r. (C \cap D))^I \subseteq (\forall r. C \cap \forall r. D)^I \)
Soundness

\[ \text{SUB}(C, D) \Rightarrow C \subseteq D \]

Proof sketch

*Reordering of terms (1):*

a) Commutativity and associativity are trivial

b) Value restriction law. We show: \((\forall r. (C \cap D))^I = (\forall r. C \cap \forall r. D)^I\)

**Assumption:** \(d \in (\forall r. (C \cap D))^I\)

**Case 1:** \(\not\exists e : (d, e) \in r^I\) \(\checkmark\)

**Case 2:** \(\exists e : (d, e) \in r^I \Rightarrow e \in (C \cap D)^I \Rightarrow e \in C^I, e \in D^I\)

Since \(e\) is arbitrary: \(d \in (\forall r. C)^I, d \in (\forall r. D)^I\) then \(d\) must also be conjunction, i.e., \((\forall r. (C \cap D))^I \subseteq (\forall r. C \cap \forall r. D)^I\)

Other direction is similar
Soundness

\[ \text{SUB}(C, D) \Rightarrow C \subseteq D \]

**Proof sketch**

*Reordering of terms (1):*

a) Commutativity and associativity are trivial

b) Value restriction law. We show: \((\forall r. (C \cap D))^I = (\forall r. C \cap \forall r. D)^I\)

**Assumption:** \(d \in (\forall r. (C \cap D))^I\)

**Case 1:** \(\forall e: (d, e) \in r^I \Rightarrow \sqrt{\ }\)

**Case 2:** \(\exists e: (d, e) \in r^I \Rightarrow e \in (C \cap D)^I \Rightarrow e \in C^I, \ e \in D^I\)

Since \(e\) is arbitrary: \(d \in (\forall r. C)^I, \ d \in (\forall r. D)^I\) then \(d\) must also be conjunction, i.e., \((\forall r. (C \cap D))^I \subseteq (\forall r. C \cap \forall r. D)^I\)

Other direction is similar

*(2+3+4):* Induction on the nesting depth of \(\forall\)-expressions
Completeness

\[ C \sqsubseteq D \Rightarrow \text{SUB}(C, D) \]
Completeness

\[ C \subseteq D \Rightarrow \text{SUB}(C, D) \]

One shows the contrapositive:

\[ \neg \text{SUB}(C, D) \Rightarrow C \nsubseteq D \]
Completeness

\[ C \subseteq D \Rightarrow \text{SUB}(C, D) \]

One shows the contrapositive:

\[ \neg \text{SUB}(C, D) \Rightarrow C \nsubseteq D \]

**Idea:** If one of the rules leads to a negative answer, we use this to construct an interpretation with a special element \( d \) such that

\[ d \in C^\mathcal{I}, \text{ but } d \notin D^\mathcal{I} \]
Completeness

\[ C \subseteq D \Rightarrow \text{SUB}(C, D) \]

One shows the contrapositive:

\[ \neg \text{SUB}(C, D) \Rightarrow C \nsubseteq D \]

**Idea:** If one of the rules leads to a negative answer, we use this to construct an interpretation with a special element \( d \) such that

\[ d \in C^\mathcal{I}, \text{ but } d \notin D^\mathcal{I} \]

**Note:** Strictly speaking, the value restrictions do not need to be collected in the subsuming concepts.
Generalizing the Algorithm

Extensions of $\mathcal{FL}^\neg$ by

- $\neg A$ (atomic negation),
Generalizing the Algorithm

Extensions of $\mathcal{FL}^-$ by

- $\neg A$ (*atomic negation*),
- $(\leq n.r), (\geq n.r)$ (*cardinality restrictions*),
Generalizing the Algorithm

Extensions of $\mathcal{FL}^-$ by

- $A$ (atomic negation),
- $(\leq n \cdot r), (\geq n \cdot r)$ (cardinality restrictions),
- $r \circ s$ (role composition)
Generalizing the Algorithm

Extensions of $\mathcal{FL}^-$ by

- $\neg A$ (atomic negation),
- $(\leq n\, r), (\geq n\, r)$ (cardinality restrictions),
- $r \circ s$ (role composition)

does not lead to any problems.
Generalizing the Algorithm

Extensions of $\mathcal{FL}^-$ by

- $\neg A$ (atomic negation),
- $(\leq n \cdot r), (\geq n \cdot r)$ (cardinality restrictions),
- $r \circ s$ (role composition)

does not lead to any problems.

**However**: If we use full existential restrictions, then it is very unlikely that we can come up with a simple structural subsumption algorithm – having the same flavor as the one above.
Generalizing the Algorithm

Extensions of $FL^-$ by

- $\neg A$ (atomic negation),
- $(\leq nr), (\geq nr)$ (cardinality restrictions),
- $r \circ s$ (role composition)

does not lead to any problems.

**However:** If we use full existential restrictions, then it is very unlikely that we can come up with a *simple* structural subsumption algorithm – having the same flavor as the one above.

**More precisely:** There is (most probably) no algorithm that uses polynomially many reorderings and simplifications and allows for a simple structural comparison.
Generalizing the Algorithm

Extensions of $\mathcal{FL}^-$ by

- $\neg A$ (atomic negation),
- $(\leq n \cdot r), (\geq n \cdot r)$ (cardinality restrictions),
- $r \circ s$ (role composition)

does not lead to any problems.

However: If we use full existential restrictions, then it is very unlikely that we can come up with a simple structural subsumption algorithm – having the same flavor as the one above.

More precisely: There is (most probably) no algorithm that uses polynomially many reorderings and simplifications and allows for a simple structural comparison

Reason: Subsumption for $\mathcal{FL}^- + \exists r. C$ is NP-hard (Nutt).
Idea: \textit{abstraction} + \textit{classification}
ABox Reasoning

Idea: abstraction + classification

- Complete ABox by propagating value restrictions to role fillers
ABox Reasoning

Idea: abstraction + classification

- Complete ABox by propagating value restrictions to role fillers
- Compute for each object its most specialized concepts
ABox Reasoning

Idea: abstraction + classification

- Complete ABox by propagating value restrictions to role fillers

- Compute for each object its most specialized concepts

- These can then be handled using the ordinary subsumption algorithm
Tableaux Method

- **Logic** $\mathcal{ALC}$
  - $C \cap D$
  - $C \sqcup D$
  - $\neg C$
  - $\forall r.C$
  - $\exists r.C$
Tableaux Method

- **Logic** $\mathcal{ALC}$
  - $C \cap D$
  - $C \sqcup D$
  - $\neg C$
  - $\forall r. C$
  - $\exists r. C$

- **Idea**: Decide (un-)satisfiability of a concept description $C$ by trying to **systematically construct** a model for $C$. If that is successful, $C$ is satisfiable. Otherwise $C$ is unsatisfiable.
Example: Subsumption in a TBox

TBox

Hermaphrodite = Male \sqcap Female
Example: Subsumption in a TBox

TBox

Hermaphrodite ⊑ Male △ Female

Parents-of-sons-and-daughters ⊑

∃has-child.Male △ ∃has-child.Female
Example: Subsumption in a TBox

**TBox**

Hermaphrodite $= \text{Male} \sqcap \text{Female}$

Parents-of-sons-and-daughters $= \exists \text{has-child}. \text{Male} \sqcap \exists \text{has-child}. \text{Female}$

Parents-of-hermaphrodite $= \exists \text{has-child}. \text{Hermaphrodite}$
Example: Subsumption in a TBox

TBox

Hermaphrodite = Male \sqcap Female

Parents-of-sons-and-daughters =
\exists has\text{-}child.\text{Male} \sqcap \exists has\text{-}child.\text{Female}

Parents-of-hermaphrodite = \exists has\text{-}child.\text{Hermaphrodite}

Query

Parents-of-sons-and-daughters \sqsubseteq_T \text{Parents-of-hermaphrodites}
Reductions

1. *Unfolding*

\[ \exists \text{has-child}. \text{Male} \sqcap \exists \text{has-child}. \text{Female} \subseteq \exists \text{has-child}. (\text{Male} \sqcap \text{Female}) \]
Reductions

1. Unfolding
   \[ \exists \text{has-child}. \text{Male} \sqcap \exists \text{has-child}. \text{Female} \sqsubseteq \exists \text{has-child}. (\text{Male} \sqcap \text{Female}) \]

2. Reduction to unsatisfiability
   Is
   \[ \exists \text{has-child}. \text{Male} \sqcap \exists \text{has-child}. \text{Female} \sqcap \neg (\exists \text{has-child}. (\text{Male} \sqcap \text{Female})) \]
   unsatisfiable?
Reductions

1. **Unfolding**
   \[ \exists \text{has-child}. \text{Male} \sqcap \exists \text{has-child}. \text{Female} \sqsubseteq \exists \text{has-child}. (\text{Male} \sqcap \text{Female}) \]

2. **Reduction to unsatisfiability**
   Is
   \[ \exists \text{has-child}. \text{Male} \sqcap \exists \text{has-child}. \text{Female} \sqcap \neg(\exists \text{has-child}. (\text{Male} \sqcap \text{Female})) \]
   unsatisfiable?

3. **Negation normal form** (move negations inside):
   \[ \exists \text{has-child}. \text{Male} \sqcap \exists \text{has-child}. \text{Female} \sqcap \forall \text{has-child}. (\neg \text{Male} \sqcup \neg \text{Female}) \]
Reductions

1. **Unfolding**
   \[ \exists \text{has-child}. \text{Male} \sqcap \exists \text{has-child}. \text{Female} \sqsubseteq \exists \text{has-child}. (\text{Male} \sqcap \text{Female}) \]

2. **Reduction to unsatisfiability**
   Is
   \[ \exists \text{has-child}. \text{Male} \sqcap \exists \text{has-child}. \text{Female} \sqcap \neg (\exists \text{has-child}. (\text{Male} \sqcap \text{Female})) \]
   unsatisfiable?

3. **Negation normal form** (move negations inside):
   \[ \exists \text{has-child}. \text{Male} \sqcap \exists \text{has-child}. \text{Female} \sqcap \forall \text{has-child}. (\neg \text{Male} \sqcup \neg \text{Female}) \]

4. **Try to construct a model**
1. **Assumption**: There exists an object $x$ in the interpretation of our concept:

$$x \in (\exists \ldots)^I$$
1. **Assumption**: There exists an object \(x\) in the interpretation of our concept:

\[ x \in (\exists \ldots)^I \]

2. This implies that \(x\) is in the interpretation of all conjuncts:

\[ x \in (\exists \text{has-child}.\text{Male})^I \]
\[ x \in (\exists \text{has-child}.\text{Female})^I \]
\[ x \in (\forall \text{has-child}.(\neg \text{Male} \sqcup \neg \text{Female}))^I \]
1. **Assumption**: There exists an object $x$ in the interpretation of our concept:

$$x \in (\exists \ldots)^\mathcal{I}$$

2. This implies that $x$ is in the interpretation of all conjuncts:

$$x \in (\exists \text{has-child}.\text{Male})^\mathcal{I}$$
$$x \in (\exists \text{has-child}.\text{Female})^\mathcal{I}$$
$$x \in (\forall \text{has-child}.(\neg \text{Male} \sqcup \neg \text{Female}))^\mathcal{I}$$

3. This implies that there should be objects $y$ and $z$ such that

$$(x, y) \in \text{has-child}^\mathcal{I}$$
1. **Assumption**: There exists an object \( x \) in the interpretation of our concept:

\[
x \in (\exists \ldots)^\mathcal{I}
\]

2. This implies that \( x \) is in the interpretation of all conjuncts:

\[
x \in (\exists \text{has-child}.\text{Male})^\mathcal{I}
\]

\[
x \in (\exists \text{has-child}.\text{Female})^\mathcal{I}
\]

\[
x \in (\forall \text{has-child}.(\neg \text{Male} \sqcup \neg \text{Female}))^\mathcal{I}
\]

3. This implies that there should be objects \( y \) and \( z \) such that

\[
(x,y) \in \text{has-child}^\mathcal{I}, (x,z) \in \text{has-child}^\mathcal{I}
\]
1. **Assumption**: There exists an object $x$ in the interpretation of our concept:

$$x \in (\exists \ldots)^I$$

2. This implies that $x$ is in the interpretation of all conjuncts:

$$x \in (\exists \text{has-child}.\text{Male})^I$$

$$x \in (\exists \text{has-child}.\text{Female})^I$$

$$x \in (\forall \text{has-child}.(\neg \text{Male} \sqcup \neg \text{Female}))^I$$

3. This implies that there should be objects $y$ and $z$ such that

$$(x, y) \in \text{has-child}^I, (x, z) \in \text{has-child}^I, y \in \text{Male}^I \text{ and } z \in \text{Female}^I \text{ and }$$...
Model Construction (2)

\[ x : \exists \text{has-child}. \text{Male} \]
\[ x : \exists \text{has-child}. \text{Female} \]
$x : \exists \text{has-child}.\text{Male}$

$x : \exists \text{has-child}.\text{Female}$

$x : \forall \text{hat-child}. (\neg \text{Male} \sqcup \neg \text{Female})$
Model Construction (4)

\[ x : \exists \text{has-child. Male} \]
\[ x : \exists \text{has-child. Female} \]
\[ x : \forall \text{hat-child. (}\neg\text{Male} \sqcup \neg\text{Female}) \]
\[ y : \neg\text{Male} \]
Modellkonstruktion (5)

\[ x : \exists \text{has-child}. \text{Male} \]
\[ x : \exists \text{has-child}. \text{Female} \]
\[ x : \forall \text{hat-child.}(\neg \text{Male} \sqcup \neg \text{Female}) \]
\[ y : \neg \text{Female} \]
\[ z : \neg \text{Male} \]
Modellkonstruktion (5)

\[
x : \exists \text{has-child}. \text{Male} \\
x : \exists \text{has-child}. \text{Female} \\
x : \forall \text{hat-child}. (\neg \text{Male} \sqcup \neg \text{Female}) \\
y : \neg \text{Female} \\
z : \neg \text{Male}
\]

\[\neg \text{Male or } \neg \text{Female} \]

\[\neg \text{Female} \quad \neg \text{Male}\]

\[\models \text{Model constructed!}\]
Tableaux Method (1)

\[ C \equiv D \text{ iff } C \subseteq D \text{ and } D \subseteq C. \]
Tableaux Method (1)

\[ C \equiv D \text{ iff } C \subseteq D \text{ and } D \subseteq C. \]

Now we have the following equivalences:

\[ \neg(C \cap D) \equiv \neg C \cup \neg D \]
Tableaux Method (1)

\[ C \equiv D \text{ iff } C \subseteq D \text{ and } D \subseteq C. \]

Now we have the following equivalences:

\[ \neg(C \cap D) \equiv \neg C \cup \neg D \]
\[ \neg(C \cup D) \equiv \neg C \cap \neg D \]
\( C \equiv D \) iff \( C \subseteq D \) and \( D \subseteq C \).

Now we have the following equivalences:

\[
\neg(C \cap D) \equiv \neg C \cup \neg D \\
\neg(C \cup D) \equiv \neg C \cap \neg D \\
\neg\neg C \equiv C
\]
Tableaux Method (1)

$C \equiv D$ iff $C \sqsubseteq D$ and $D \sqsubseteq C$.

Now we have the following equivalences:

$\neg(C \cap D) \equiv \neg C \cup \neg D$

$\neg(C \cup D) \equiv \neg C \cap \neg D$

$\neg \neg C \equiv C$

$\neg(\forall r.C) \equiv \exists r.\neg C$
$C \equiv D$ iff $C \subseteq D$ and $D \subseteq C$.

Now we have the following equivalences:

\[
\begin{align*}
\neg(C \cap D) & \equiv \neg C \cup \neg D \\
\neg(C \cup D) & \equiv \neg C \cap \neg D \\
\neg\neg C & \equiv C \\
\neg(\forall r. C) & \equiv \exists r. \neg C \\
\neg(\exists r. C) & \equiv \forall r. \neg C
\end{align*}
\]
Tableaux Method (1)

$C \equiv D$ iff $C \subseteq D$ and $D \subseteq C$.

Now we have the following equivalences:

\[
\neg (C \cap D) \equiv \neg C \cup \neg D \\
\neg (C \cup D) \equiv \neg C \cap \neg D \\
\neg \neg C \equiv C \\
\neg (\forall r. C) \equiv \exists r. \neg C \\
\neg (\exists r. C) \equiv \forall r. \neg C
\]

These equivalences can be used to move all negations signs to the inside, resulting in concept description where only concept names are negated: **negation normal form (NNF)**
Tableaux Method (1)

\[ C \equiv D \text{ iff } C \sqsubseteq D \text{ and } D \sqsubseteq C. \]

Now we have the following equivalences:

\[
\begin{align*}
\neg(C \cap D) & \equiv \neg C \cup \neg D \\
\neg(C \cup D) & \equiv \neg C \cap \neg D \\
\neg \neg C & \equiv C \\
\neg(\forall r. C) & \equiv \exists r. \neg C \\
\neg(\exists r. C) & \equiv \forall r. \neg C
\end{align*}
\]

These equivalences can be used to move all negations signs to the inside, resulting in concept description where only concept names are negated:

**negation normal form (NNF)**

**Theorem.** The negation normal form of a \( \mathcal{ALC} \) concept can be computed in polynomial time.
Tableaux Method (2)

A constraint is a syntactical object of the form: $x: C$ or $xy$, where $C$ is a concept description in NNF, $r$ is a role name and $x$ and $y$ are variable names.
Tableaux Method (2)

A **constraint** is a syntactical object of the form: $x: C$ or $xry$, where $C$ is a concept description in NNF, $r$ is a role name and $x$ and $y$ are **variable names**. Let $\mathcal{I}$ be an interpretation. An **$\mathcal{I}$-assignment** $\alpha$ is a function that maps each variable symbol to an object of the universe $\mathcal{D}$. 
Tableaux Method (2)

A constraint is a syntactical object of the form: \( x: C \) or \( xry \), where \( C \) is a concept description in NNF, \( r \) is a role name and \( x \) and \( y \) are variable names.

Let \( \mathcal{I} \) be an interpretation. An \( \mathcal{I} \)-assignment \( \alpha \) is a function that maps each variable symbol to an object of the universe \( D \).

A constraint \( x: C \ (xry) \) is satisfied by an \( \mathcal{I} \)-assignment \( \alpha \), if \( \alpha(x) \in C^\mathcal{I} \) \( ((\alpha(x), \alpha(y)) \in r^\mathcal{I}) \).
Tableaux Method (2)

A constraint is a syntactical object of the form: $x : C$ or $x r y$, where $C$ is a concept description in NNF, $r$ is a role name and $x$ and $y$ are variable names.

Let $I$ be an interpretation. An $I$-assignment $\alpha$ is a function that maps each variable symbol to an object of the universe $D$.

A constraint $x : C \ (x r y)$ is satisfied by an $I$-assignment $\alpha$, if $\alpha(x) \in C^I$ and $((\alpha(x), \alpha(y)) \in r^I)$.

Ein constraint system $S$ is a finite, non-empty set of constraints. A $I$-assignment $\alpha$ satisfies $S$ if $\alpha$ satisfies each constraint in $S$. 
A constraint is a syntactical object of the form: \(x: C\) or \(xry\), where \(C\) is a concept description in NNF, \(r\) is a role name and \(x\) and \(y\) are variable names.

Let \(\mathcal{I}\) be an interpretation. An \(\mathcal{I}\)-assignment \(\alpha\) is a function that maps each variable symbol to an object of the universe \(\mathcal{D}\).

A constraint \(x: C\ (xry)\) is satisfied by an \(\mathcal{I}\)-assignment \(\alpha\), if \(\alpha(x) \in C^{\mathcal{I}}\) \(((\alpha(x), \alpha(y)) \in r^{\mathcal{I}})\).

Ein constraint system \(S\) is a finite, non-empty set of constraints. A \(\mathcal{I}\)-assignment \(\alpha\) satisfies \(S\) if \(\alpha\) satisfies each constraint in \(S\). \(S\) is satisfiable if there exists \(\mathcal{I}\) and \(\alpha\) such that \(\alpha\) satisfies \(S\).
A constraint is a syntactical object of the form: \( x : C \) or \( x r y \), where \( C \) is a concept description in NNF, \( r \) is a role name and \( x \) and \( y \) are variable names.

Let \( \mathcal{I} \) be an interpretation. An \( \mathcal{I} \)-assignment \( \alpha \) is a function that maps each variable symbol to an object of the universe \( D \).

A constraint \( x : C (xry) \) is satisfied by an \( \mathcal{I} \)-assignment \( \alpha \), if \( \alpha(x) \in C^\mathcal{I} \)(\( ((\alpha(x), \alpha(y)) \in r^\mathcal{I}) \).

Ein constraint system \( S \) is a finite, non-empty set of constraints. A \( \mathcal{I} \)-assignment \( \alpha \) satisfies \( S \) if \( \alpha \) satisfies each constraint in \( S \). \( S \) is satisfiable if there exists \( \mathcal{I} \) and \( \alpha \) such that \( \alpha \) satisfies \( S \).

**Theorem.** An \( \mathcal{ALC} \) concept \( C \) in NNF is satisfiable iff the system \( \{ x : C \} \) is satisfiable.
Tableaux Method (3)

**Transformation rules:**

1. $S \rightarrow \{x: C_1, x: C_2\} \cup S$
   
   if $(x: C_1 \cap C_2) \in S$ and either $(x: C_1)$ or $(x: C_2)$ or both are not in $S$. 
Transformation rules:

1. \( S \rightarrow \cap \{ x : C_1, x : C_2 \} \cup S \)
   if \( (x : C_1 \cap C_2) \in S \) and either \( (x : C_1) \) or \( (x : C_2) \) or both are not in \( S \).

2. \( S \rightarrow \sqcup \{ x : D \} \cup S \)
   if \( (x : C_1 \cup C_2) \in S \) and neither \( (x : C_1) \in S \) nor \( (x : C_2) \in S \) and \( D = C_1 \) or \( D = C_2 \).
Tableaux Method (3)

Transformation rules:

1. \( S \rightarrow \cap \{x: C_1, x: C_2\} \cup S \)
   if \((x: C_1 \cap C_2) \in S\) and either \((x: C_1)\) or \((x: C_2)\) or both are not in \(S\).

2. \( S \rightarrow \cup \{x: D\} \cup S \)
   if \((x: C_1 \sqcup C_2) \in S\) and neither \((x: C_1) \in S\) nor \((x: C_2) \in S\) and \(D = C_1\) or \(D = C_2\).

3. \( S \rightarrow \exists \{xry, y: C\} \cup S \)
   if \((x: \exists r. C) \in S\), \(y\) is a fresh variable, and there is no \(z\) s.t. \((xz) \in S\) and \((z: C) \in S\).
Transformation rules:

1. $S \rightarrow \cap \{x: C_1, x: C_2\} \cup S$
   if $(x: C_1 \cap C_2) \in S$ and either $(x: C_1)$ or $(x: C_2)$ or both are not in $S$.

2. $S \rightarrow \sqcup \{x: D\} \cup S$
   if $(x: C_1 \sqcup C_2) \in S$ and neither $(x: C_1) \in S$ nor $(x: C_2) \in S$ and $D = C_1$ or $D = C_2$.

3. $S \rightarrow \exists \{xry, y: C\} \cup S$
   if $(x: \exists r. C) \in S$, $y$ is a fresh variable, and there is no $z$ s.t. $(xrz) \in S$ and $(z: C) \in S$.

4. $S \rightarrow \forall \{y: C\} \cup S$
   if $(x: \forall r. C), (xry) \in S$ and $(y: C) \not\in S$. 
Tableaux Method (3)

Transformation rules:

1. $S \rightarrow \cap \{x: C_1, x: C_2\} \cup S$
   if $(x: C_1 \cap C_2) \in S$ and either $(x: C_1) \in S$ or $(x: C_2) \in S$ or both are not in $S$.

2. $S \rightarrow \cup \{x: D\} \cup S$
   if $(x: C_1 \cup C_2) \in S$ and neither $(x: C_1) \in S$ nor $(x: C_2) \in S$ and $D = C_1$ or $D = C_2$.

3. $S \rightarrow \exists \{xry, y: C\} \cup S$
   if $(x: \exists r.C) \in S$, $y$ is a fresh variable, and there is no $z$ s.t. $(xrz) \in S$ and $(z: C) \in S$.

4. $S \rightarrow \forall \{y: C\} \cup S$
   if $(x: \forall r.C), (xry) \in S$ and $(y: C) \not\in S$.

Deterministic rules (1,3,4) vs. non-deterministic (2).
Tableaux Method (3)

Transformation rules:

1. \( S \rightarrow \cap \{x: C_1, x: C_2\} \cup S \)
   
   if \((x: C_1 \cap C_2) \in S\) and either \((x: C_1)\) or \((x: C_2)\) or both are not in \(S\).

2. \( S \rightarrow \sqcup \{x: D\} \cup S \)
   
   if \((x: C_1 \sqcup C_2) \in S\) and neither \((x: C_1)\in S\) nor \((x: C_2)\in S\) and \(D = C_1\) or \(D = C_2\).

3. \( S \rightarrow \exists \{xry, y: C\} \cup S \)
   
   if \((x: \exists r.C) \in S\), \(y\) is a fresh variable, and there is no \(z\) s.t. \((xrz) \in S\) and \((z: C) \in S\).

4. \( S \rightarrow \forall \{y: C\} \cup S \)
   
   if \((x: \forall r.C), (xry) \in S\) and \((y: C) \notin S\).

Deterministic rules (1,3,4) vs. non-deterministic (2).
Generating rules (3) vs. non-generating (1,2,4).
Theorem (Invariance). Let $S$ and $T$ be constraint systems:

1. If $T$ has been generated by applying a deterministic rule to $S$, then $S$ is satisfiable iff $T$ is satisfiable.
**Theorem (Invariance).** Let $S$ and $T$ be constraint systems:

1. If $T$ has been generated by applying a deterministic rule to $S$, then $S$ is satisfiable iff $T$ is satisfiable.

2. If $T$ has been generated by applying a non-deterministic rule to $S$, then $S$ is satisfiable if $T$ is satisfiable. Furthermore, if a non-deterministic rule can be applied to $S$, then it can be applied such that $S$ is satisfiable iff the resulting system $T$ is satisfiable.
**Tableaux Method (4)**

**Theorem (Invariance).** Let $S$ and $T$ be constraint systems:

1. If $T$ has been generated by applying a deterministic rule to $S$, then $S$ is satisfiable iff $T$ is satisfiable.

2. If $T$ has been generated by applying a non-deterministic rule to $S$, then $S$ is satisfiable if $T$ is satisfiable. Furthermore, if a non-deterministic rule can be applied to $S$, then it can be applied such that $S$ is satisfiable iff the resulting system $T$ is satisfiable.

**Theorem (Termination).** Let $C$ be an $\mathcal{ALC}$ concept description in NNF. Then there exists no infinite chain of transformations starting from the constraint system $\{x: C\}$. 
Tableaux Method (5)

A constraint system is called *closed* if no transformation rule can be applied.
A constraint system is called **closed** if no transformation rule can be applied.

An **clash** is a pair of constraints of the form $x: A$ and $x: \neg A$, where $A$ is a concept name.
A constraint system is called **closed** if no transformation rule can be applied.

An **clash** is a pair of constraints of the form $x: A$ and $x: \neg A$, where $A$ is a concept name.

**Theorem.** A closed constraint system is satisfiable iff it does not contain a clash.
A constraint system is called **closed** if no transformation rule can be applied.

An **clash** is a pair of constraints of the form $x: A$ and $x: \neg A$, where $A$ is a concept name.

**Theorem.** A closed constraint system is satisfiable iff it does not contain a clash.

**Proof Idea.** $\Rightarrow$: obvious. $\Leftarrow$: Construct a model by using the concept labels.
Space Requirements

Because the tableaux method is *non-deterministic* (→⊔ rule) ... there could be exponentially many closed constraint systems in the end.
Space Requirements

Because the tableaux method is *non-deterministic* ($\rightarrow_\emptyset$ rule) . . . there could be exponentially many closed constraint systems in the end.

Interestingly, even one constraint system can have *exponential size*.
Space Requirements

Because the tableaux method is non-deterministic (→⊔ rule) . . . there could be exponentially many closed constraint systems in the end.

Interestingly, even one constraint system can have exponential size.

Example:

\[
\exists r. A \land \exists r. B \\
\forall r. \left( \exists r. A \land \exists r. B \\
\forall r. (\exists r. A \land \exists r. B \\
\forall r. (\ldots)) \right)
\]
Space Requirements

Because the tableaux method is *non-deterministic* ($\rightarrow_{\downarrow}$ rule) ... there could be exponentially many closed constraint systems in the end.

Interestingly, even one constraint system can have *exponential size*.

**Example:**

$$\exists r. A \land \exists r. B \land$$
$$\forall r. (\exists r. A \land \exists r. B \land$$
$$\forall r. (\exists r. A \land \exists r. B \land$$
$$\forall r. (...) )$$

**However:** One can modify the algorithm so that it needs only poly. space.

**Idea:** Generating a $y$ only for one $\exists r. C$ and then proceeding into the depth.
ABox Reasoning

ABox satisfiability can also be decided using the tableaux method if we can add constraints of the form $x \neq y$ (for $UNA$)
ABox Reasoning

ABox satisfiability can also be decided using the tableaux method if we can add constraints of the form $x \neq y$ (for UNA):

- Normalize and unfold and add inequalities for all pairs of objects mentioned in the ABox.
ABox Reasoning

ABox satisfiability can also be decided using the tableaux method if we can add constraints of the form $x \neq y$ (for UNA):

- **Normalize** and **unfold** and add inequalities for all pairs of objects mentioned in the ABox.

- Strictly speaking, in $\mathcal{ALC}$ we do not need this because we are never **forced** to identify two objects.
Literature


