Principles of Knowledge Representation and Reasoning
Semantic Networks and Description Logics IV: Description Logics – Algorithms

1 Motivation

Motivation
Structural Subsumption Algorithms
Tableau Subsumption Method
Literature

Reasoning problems & algorithms

Reasoning problems:
- Satisfiability or subsumption of concept descriptions
- Satisfiability or instance relation in ABoxes

Solving techniques presented in this chapter:
- Structural subsumption algorithms
  - Normalization of concept descriptions and structural comparison
  - very fast, but can only be used for small DLs
- Tableau algorithms
  - Similar to modal tableau methods
  - Often the method of choice
2 Structural Subsumption Algorithms

- Idea
- Example
- Algorithm
- Soundness
- Completeness
- Generalizations
- ABox Reasoning

Structural subsumption algorithms

In what follows we consider the rather small logic $FL^-$:

- $C \sqcap D$
- $\forall r.C$
- $\exists r$ (simple existential quantification)

To solve the subsumption problem for this logic we apply the following 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.
   For each conjunct in the subsuming concept there should be a corresponding one in the subsumed one.

Example

$D = \text{Human} \sqcap \exists \text{has-child} \sqcap \forall \text{has-child.} \text{Human} \sqcap \forall \text{has-child.} \text{has-child}$

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

Check: $C \sqsubseteq D$

1. Collect value restrictions in $D$:
   \[
   \ldots \forall \text{has-child.}(\text{Human} \sqcap \exists \text{has-child})
   \]

2. Compare:
   1. For Human in $D$, we have Human in $C$.
   2. For $\exists \text{has-child} \text{in } D$, we have $\exists \text{has-child} \text{in } C$.
   3. For $\forall \text{has-child.}(\ldots)$ in $D$, we have Human and $\exists \text{has-child} \text{in } C$.

$\Rightarrow C \sqsubseteq D$ !

Subsumption algorithm

SUB($C, D$) algorithm:

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

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

\[
D = \prod B_i \prod \exists s_m \prod \forall s_n : D_n
\]

2. For each $B_i$ in $D$, is there an $A_i$ in $C$ with $A_i = B_i$?
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 $s_n = r_k$ and $C_k \sqsubseteq D_n$ (i.e., check $\text{SUB}(C_k, D_n)$)?

$\Rightarrow C \sqsubseteq D$ iff all questions are answered positively.
Soundness

**Theorem (Soundness)**

\[ SUB(C, D) \Rightarrow C \sqsubseteq D \]

**Proof sketch.**

Reordering of terms step (1):

1. Commutativity and associativity are trivial.
2. Value restriction law. We show:
   \[ (\forall r. (C \land D))^I = (\forall r. C \land \forall r. D)^I \]
   Assume \( d \in (\forall r. (C \land D))^I \).
   If there is no \( e \in D \) with \((d, e) \in r^I\) it follows trivially that \( d \in (\forall r. C \land \forall r. D)^I \).
   If there is an \( e \in D \) with \((d, e) \in r^I\) it follows \( e \in (C \land D)^I = C^I \land D^I \).
   Since \( e \) is arbitrary, we have \( d \in (\forall r. C)^I \) and \( d \in (\forall r. D)^I \), i.e., \( (\forall r. (C \land D))^I \subseteq (\forall r. C \land \forall r. D)^I \).
   The other direction is similar.

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


Completeness

**Theorem (Completeness)**

\[ C \sqsubseteq D \Rightarrow SUB(C, D). \]

**Proof idea.**

One shows the contrapositive:

\[ \neg SUB(C, D) \Rightarrow C \not\sqsubseteq D \]

**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

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

November 30, 2015 Nebel, Lindner, Engesser – KR&R 12 / 34

Generalizing the algorithm

**Extensions of \( FL^- \) by**

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

**do 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 \( FL^- + \exists r. C \) is NP-hard (Nutt).

November 30, 2015 Nebel, Lindner, Engesser – KR&R 13 / 34

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.

November 30, 2015 Nebel, Lindner, Engesser – KR&R 14 / 34
Example: Subsumption in a TBox

Example

TBox:

- Hermaphrodite ≡ Male ⊓ Female
- Parent-of-sons-and-daughters ≡ ∃ has-child.Male ⊓ ∃ has-child.Female
- Parent-of-hermaphrodite ≡ ∃ has-child.Hermaphrodite

Query:

- Parent-of-sons-and-daughters ⊑ Parent-of-hermaphrodites
Assumption: There exists an object \( x \) in the interpretation of our concept:
\[
x \in (\exists \ldots)^I
\]

This implies that \( x \) is in the interpretation of all conjuncts:
\[
x \in (\exists \text{has-child}.\text{Male})^I \
x \in (\exists \text{has-child}.\text{Female})^I \
x \in (\forall \text{has-child}.(\neg \text{Male} \sqcup \neg \text{Female}))^I
\]

This implies that there should be objects \( y \) and \( z \) such that
\[
(x,y) \in \text{has-child}^I, (x,z) \in \text{has-child}^I, y \in \text{Male}^I \text{ and } z \in \text{Female}^I, \text{ and }...
\]
Tableau method (1): NNF

We write: \( C \equiv D \) iff \( C \subseteq D \) and \( D \subseteq C \). Now we have the following equivalences:

\[
\neg (C \sqcap D) \equiv \neg C \sqcup \neg D \\
\neg (\forall r. C) \equiv \exists r. \neg C \\
\neg (\exists r. C) \equiv \forall r. \neg C \\
\neg \neg C \equiv C
\]

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).

Theorem (NNF)

The negation normal form of an \( \mathcal{ALC} \) concept can be computed in polynomial time.

Tableau method (2): Constraint systems

A constraint is a syntactical object of the form:

\[ x : C \text{ or } x ry, \]

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 with universe \( D \). 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 \( \alpha(x) \in C^\mathcal{I} \) (resp. \( (\alpha(x), \alpha(y)) \in r^\mathcal{I} \)).

Tableau method (3): Constraint systems

Definition

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

Theorem

An \( \mathcal{ALC} \) concept \( C \) in NNF is satisfiable if and only if the system \( \{ x : C \} \) is satisfiable.
Tableau method (4): Transforming constraint systems

**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 \(S\).

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

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

**Notice:** Deterministic rules (1,3,4) vs. non-deterministic (2).

Generating rules (3) vs. non-generating (1,2,4).

Tableau method (5): Invariances

**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 if and only if \( T \) 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 if and only if the resulting system \( T \) is satisfiable.

**Theorem (Termination)**

Let \( C \) be an \( ALC \) concept description in NNF. Then there exists no infinite chain of transformations starting from the constraint system \( \{ x: C \} \).

Tableau method (6): Soundness and completeness

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

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

**Theorem (Soundness and Completeness)**

A closed constraint system is satisfiable if and only if it does not contain a clash.

**Proof idea.**

\( \Rightarrow \): obvious. \( \Leftarrow \): Construct a model by using the concept labels.

Space requirements

Because the tableau method is **non-deterministic** (\( \rightarrow \cup \) rule), there could be exponentially many closed constraint systems in the end.

Interestingly, applying the rules on a single constraint system can lead to constraint systems of **exponential size**.

**Example**

\[
\exists rA \land \exists rB \land \forall r.( \exists \forall A \land \exists \forall B \land \forall r.( \exists rA \land \exists rB \land \forall r.( \ldots )))
\]

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

**Idea:** Generate \( A \) only for one \( \exists r.C \) and then proceed into the depth.
ABox reasoning

ABox satisfiability can also be decided using the tableau 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

- **Literature I**

- **Literature II**