Subsumption algorithms for description logics

G. Falquet

Université de Genève

November 16, 2011

Objectives

- infer implicit knowledge
- determine subsumption ($C \sqsubseteq D$)
- determine satisfiability (there is a non empty model for C)

Remarks

1. $C \sqsubseteq D$ if and only if $C \sqcap \neg D$ is not satisfiable.
2. C is subsumed by D iff for any domain $\triangle$ and any extension function I over $\triangle$

$$I(C) \subseteq I(D)$$

A structural algorithm

Works only for $\mathcal{FL}^-$
$\mathcal{FL}^-$ is limited to $A \mid C \sqcap D \mid \forall R.C \mid \exists R$
2-phases algorithm:

1. Normalization
2. Recursive comparison

Normalization

Flatten all embedded conjunctions :

$$A \sqcap (B \sqcap C) \longrightarrow A \sqcap B \sqcap C$$

Factorize all conjunctions of universal quantifiers over the same role

$$\forall R.C \sqcap \forall R.D \longrightarrow \forall R.(C \sqcap D)$$

The Subsumes(C, D) algorithm

Let $C = C_1 \sqcap \cdots \sqcap C_n$ and $D = D_1 \sqcap \cdots \sqcap D_m$
Subsumes(C, D) returns **true** iff for all C_i :

1. if C_i is atomic or of the form $\exists R$ then there exists D_j such that $C_i = D_j$;
2. if C_i is of the form $\forall R.C'$ then there exists D_j of the form $\forall R.D'$ such that Subsumes(C', D')

Exercise

Use the algorithm to check

- $\text{Adult} \sqcap \text{Male} \sqsubseteq \text{Adult}$
- $\text{Adult} \sqcap \text{Male} \sqcap \text{Rich} \sqsubseteq \text{Rich} \sqcap \text{Adult}$
- $\forall \text{child}.(\text{Adult} \sqcap \text{Male}) \sqsubseteq \forall \text{child}.\text{Adult}$
- $\forall \text{child}.\text{Adult} \sqcap \exists \text{child} \sqsubseteq \forall \text{child}.\text{Adult}$
- $\forall \text{child}.\text{Adult} \not\sqsubseteq \exists \text{child}$
- $\exists \text{child} \not\sqsubseteq \forall \text{child}.\text{Adult}$

Properties of the algorithm

Time complexity $O(|C| \times |D|)$

Soundness The algorithm is sound. Whenever is answers "yes" then C is subsumed by D.

Completeness Whenever $C \sqsubseteq D$ the algorithm answers "yes"

Limits of structural algorithms

- Algorithms based on a syntactic analysis cannot handle more complex logics.
- For instance, $A \sqcup \neg A$ subsumes any concept C even if C does not mention A.

Tableau algorithms

Tableau algorithm prove the non satisfiability of a concept by trying to build a model.
They take advantage of the "tree model property": if there is a model then there is a model that has a tree shape (the object-relation graph is a tree)

TBox and ABox

- N_C : set of concept names
- N_R : role names
- N_I : individual names

ABox : set of assertions of the form

- $C(a)$, C is a concept expression, a an individual
- $r(a, b)$, r is a role name

Model of an ABox

Interpretation I of the roles and concept such that

- I assigns to each individual a an object $I(a) \in \Delta$
- if $C(a)$ is in the ABox then $I(a) \in I(C)$
- if $r(a, b)$ is in the ABox then $(I(a), I(b)) \in I(r)$

Consistency An ABox is consistent if it has a model.

Instance An individual a is an instance of C if in every model I of the ABox A, $I(a) \in I(C)$. Notation $A \models C(a)$

Reformulation $A \models C(a)$ iff $A \cup \{\neg C(a)\}$ is inconsistant

From acyclic TBoxes to ABoxes

If a TBox has no circular definition it is always possible to rewrite every concept definition

$$C \equiv Expr$$

as

$$C \equiv Expr'$$

where $Expr'$ contains only basic (not defined) concept names.
Then if the ABox contains $C(a)$ it can be rewritten as $Expr'(a)$. This is a way to empty the TBox

Satisfiability Algorithm

To test the satisfiability of C.
The algorithm tries to build a model I in which $I(C)$ is not empty.

1. put C in negative normal form (all negations beside atomic concept)
2. crate an initial set of ABoxes: $\{\{C(a)\}\}$
3. exhaustively apply the production rules
4. if there is an open ABox (without contradiction of the type $P(x)$ et $\neg P(x)$) in the resulting ABox set, C is satisfiable.

Rules

For an ABox $\mathcal{A}$ generate one or two new ABoxes $\mathcal{A}'$ and $\mathcal{A}''$

- $\rightarrow_{\sqcap}$ rule if $\mathcal{A}$ contains $(C \sqcap D)(x)$ but not $C(x)$ and $D(x)$
  then $A' = \mathcal{A}' = \mathcal{A} \cup \{C(x), D(x)\}$.
- $\rightarrow_{\sqcup}$ rule if $\mathcal{A}$ contains $(C \sqcup D)(x)$ but neither $C(x)$ nor $D(x)$
  then $\mathcal{A}' = \mathcal{A} \cup \{C(x)\}$ and $\mathcal{A}'' = \mathcal{A}A \cup \{D(x)\}$.
- $\rightarrow_{\exists}$ rule if $\mathcal{A}$ contains $(\exists r.C)(x)$ but no individual name z such that
  $C(z)$ and $r(x, z)$ are in A
  then $\mathcal{A}' = \mathcal{A} \cup \{C(y), r(x, y)\}$.
- $\rightarrow_{\forall}$ rule if $\mathcal{A}$ contains $(\forall r.C)(x)$ and $r(x, y)$ but not $C(y)$
  then $\mathcal{A}' = \mathcal{A} \cup \{C(y)\}$.

Properties of the algorithm

1. rule application always terminates (no infinite loop).
2. if $\mathcal{A}'$ has been produced from $\mathcal{A}$ by a rule application then $\mathcal{A}$ is consistent iff $\mathcal{A}'$ is consistent.
3. every cloased ABox (containing $P(x)$ and $\neg P(x)$) in inconsistent.
4. every complete and open ABox is consistent.

Complexity

The size of the ABox set generated during the process may be exponential in the size of C.
e.g. for the following family of ABoxes

$$C_1 := \exists r.A \sqcap \exists r.B,$$

$$C_{n+1} := \exists r.A \sqcap \exists r.B \sqcap \forall r.C_n$$

For general TBoxes

Remark. A TBox

$$\{C_1 \sqsubseteq D_1, \ldots, C_n \sqsubseteq D_n\}$$

is equivalent to a TBox

$$\{\top \sqsubseteq ((\neg C_1 \sqcup D_1) \sqcap \cdots \sqcap (\neg C_n \sqcup D_n))\}$$

Thus we can consider a TBox with a single axiom of the form

$$\top \sqsubseteq C$$

.i.e. every object of the domain must belong to the interpretation of C

Additional rule

To represent the TBox axiom $\top \sqsubseteq C$ we add a new rule

$\longrightarrow_{\top \sqsubseteq C}$-rule if the individual name x appears in the ABox and $C(x)$ is not present, add $C(x)$ to the ABox

Blocking

To avoid infinite loops for cyclic TBoxes
If the TBox is cyclic, the $\longrightarrow_\exists$-rule may create infinite sequences of individuals connected through roles.
The application of the $\longrightarrow_\exists$-rule to an individual x is blocked by an individual y if

- $x$is younger than y: x has been introduced by an $\longrightarrow_\exists$-rule after the introduction of y
- x has no more constraints than y, i.e. $\{C : C(x) \in \text{ABox}\} \subseteq \{C : C(y) \in \text{ABox}\}$

The idea is that we can use y instead of x.