

Computing the Least Common Subsumer w.r.t. a Background Terminology*

Franz Baader Barış Sertkaya Anni-Yasmin Turhan

Theoretical Computer Science
TU Dresden, Germany

JELIA 2004

* Supported by DFG project BA 1122/4-3 and DFG grant GRK 334/3

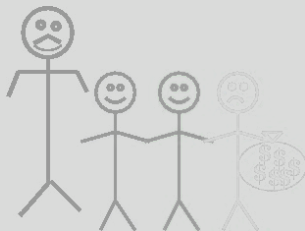


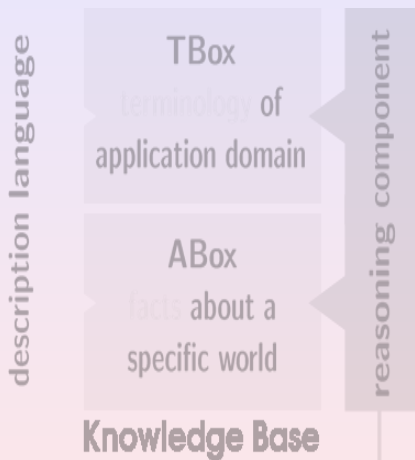
What are they?

Class of Knowledge Representation formalisms

- **concepts**: interpreted as sets of objects
- **roles**: interpreted as binary relations on objects
- **set of constructors**: build complex concepts from atomic ones

Example


$$\begin{aligned} \text{HappyFather} &\equiv \\ \text{Male} \sqcap (\exists \text{hasChild}.\text{Blue}) \sqcap \\ &(\exists \text{hasChild}.\text{Green}) \sqcap \\ &(\forall \text{hasChild}.\text{Happy} \sqcup \text{Rich}) \end{aligned}$$

Description language

- logic-based, formal semantics
- $\mathcal{AL}\mathcal{E}$:
 $\neg A, C \sqcap D, \exists r.C, \forall r.C$
- $\mathcal{AL}\mathcal{C}$: $\neg C, C \sqcap D, C \sqcup D, \exists r.C, \forall r.C$

Standard inferences

- Subsumption: $C \sqsubseteq D$
- Instance: $C(a)$



Most specific concept (msc)

$C = msc(a)$ iff

- 1 $a \in C^{\mathcal{I}}$ (a is an instance of C)
- 2 if $a \in D^{\mathcal{I}}$ then $C \sqsubseteq D$ (C is least)

Least common subsumer (lcs)

$D = lcs(C_1, C_2)$ iff

- 1 $C_1, C_2 \sqsubseteq D$ (D subsumes both C_1 and C_2)
- 2 if $C_1, C_2 \sqsubseteq E$ then $D \sqsubseteq E$ (D is least)



Example



Bottom-up Construction Steps

- 1 User selects similar ABox individuals
- 2 Automatic generalization of individuals into concept descriptions (compute msc)
- 3 Automatic extraction of commonalities (compute LCS)
- 4 User inspects the concept, modifies it and adds it to the terminology



Example

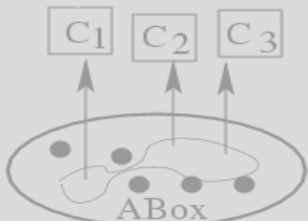


Bottom-up Construction Steps

- 1 User selects similar ABox individuals
- 2 Automatic generalization of individuals into concept descriptions (compute msc)
- 3 Automatic extraction of commonalities (compute LCS)
- 4 User inspects the concept, modifies it and adds it to the terminology



Example

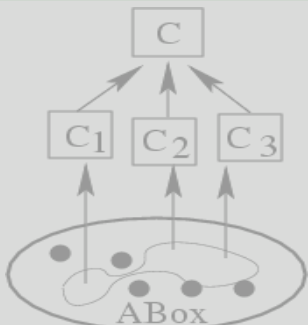


Bottom-up Construction Steps

- 1 User selects similar ABox individuals
- 2 Automatic generalization of individuals into concept descriptions (compute msc)
- 3 Automatic extraction of commonalities (compute LCS)
- 4 User inspects the concept, modifies it and adds it to the terminology



Example

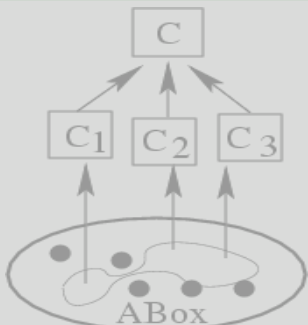


Bottom-up Construction Steps

- 1 User selects similar ABox individuals
- 2 Automatic generalization of individuals into concept descriptions (compute msc)
- 3 Automatic extraction of commonalities (compute LCS)
- 4 User inspects the concept, modifies it and adds it to the terminology



Example



Bottom-up Construction Steps

- 1 User selects similar ABox individuals
- 2 Automatic generalization of individuals into concept descriptions (compute msc)
- 3 Automatic extraction of commonalities (compute LCS)
- 4 User inspects the concept, modifies it and adds it to the terminology



Aim

- There are KBs based on very expressive DLs
- Allow the user to extend, maintain, restructure these KBs.
- Still support the Bottom-up Approach

Problems

- LCS computation restricted to inexpressive DLs
- LCS yields nothing interesting for DLs with disjunction

Solution

- Use a less expressive DL for LCS computation
- But allow using names from a TBox defined in an expressive DL



Aim

- There are KBs based on very expressive DLs
- Allow the user to extend, maintain, restructure these KBs.
- Still support the Bottom-up Approach

Problems

- LCS computation restricted to inexpressive DLs
- LCS yields nothing interesting for DLs with disjunction

Solution

- Use a less expressive DL for LCS computation
- But allow using names from a TBox defined in an expressive DL



Aim

- There are KBs based on very expressive DLs
- Allow the user to extend, maintain, restructure these KBs.
- Still support the Bottom-up Approach

Problems

- LCS computation restricted to inexpressive DLs
- LCS yields nothing interesting for DLs with disjunction

Solution

- Use a less expressive DL for LCS computation
- But allow using names from a TBox defined in an expressive DL



Background Terminology \mathcal{T}

- basic notions of the application domain
- expressive DL (\mathcal{ALC})

User Terminology

- refines the background terminology
- less expressive DL ($\mathcal{AL}\mathcal{E}$)
- $\mathcal{AL}\mathcal{E}(\mathcal{T})$ -concepts: concept descriptions containing names defined in \mathcal{T}

Example

- \mathcal{ALC} -TBox $\mathcal{T} := \{A \equiv P \sqcup Q\}$
- without using names from \mathcal{T} : $\text{lcs}_{\mathcal{AL}\mathcal{E}(\mathcal{T})}(P, Q) = \top$
- with using names from \mathcal{T} : $\text{lcs}_{\mathcal{AL}\mathcal{E}(\mathcal{T})}(P, Q) = A$
- **more specific result**



Theorem

Let \mathcal{T} be an \mathcal{ALC} -TBox. The LCS of $\mathcal{AL}\mathcal{E}$ (\mathcal{T})-concept descriptions w.r.t. \mathcal{T} *always exists* and can *effectively be computed*.

- Proof based on the role depth of concept descriptions (max. nesting of \forall and \exists restrictions)

Brute-force approach

lcs of $\mathcal{AL}\mathcal{E}$ (\mathcal{T})-concepts C_1, \dots, C_n bounded by role depth k

- compute the set of all $\mathcal{AL}\mathcal{E}$ (\mathcal{T})-concept descriptions of role depth $k + 1$
- check which of them are common subsumers of C_1, \dots, C_n w.r.t \mathcal{T}
- build the conjunction of them



Good common subsumer

Problems with the brute-force approach

- a very large number of concept descriptions must be considered, not feasible
- it is not crucial to use the *least* common subsumer
- using it may even result in over-fitting
- instead, use a "good" common subsumer

Good common subsumer

- a common subsumer which is not too general
- modify the LCS algorithm
- use only subsumption information from the background terminology, not the concept definitions



LCS algorithm for $\mathcal{AL}\mathcal{E}$

$$\text{lcs}_{\mathcal{AL}\mathcal{E}}(C, D) = \bigsqcap_{A \in \text{names}(C) \cap \text{names}(D)} A \sqcap \bigsqcap_{\neg B \in \overline{\text{names}(C) \cap \text{names}(D)}} \neg B \dots$$

where $\text{names}(C)$ is the set of concept names in top level conjunction

GCS algorithm for $\mathcal{AL}\mathcal{E}$

- more information from the background terminology can be incorporated
- take the smallest conjunction that subsumes both

$$\bigsqcap_{A \in \text{names}(C)} A \sqcap \bigsqcap_{\neg B \in \overline{\text{names}(C)}} \neg B \quad \text{and} \quad \bigsqcap_{A' \in \text{names}(D)} A' \sqcap \bigsqcap_{\neg B' \in \overline{\text{names}(D)}} \neg B'$$



LCS algorithm for $\mathcal{AL}\mathcal{E}$

$$\text{lcs}_{\mathcal{AL}\mathcal{E}}(C, D) = \bigwedge_{A \in \text{names}(C) \cap \text{names}(D)} A \sqcap \bigwedge_{\neg B \in \overline{\text{names}(C) \cap \text{names}(D)}} \neg B \dots$$

where $\text{names}(C)$ is the set of concept names in top level conjunction

GCS algorithm for $\mathcal{AL}\mathcal{E}$

- more information from the background terminology can be incorporated
- take the smallest conjunction that subsumes both

$$\bigwedge_{A \in \text{names}(C)} A \sqcap \bigwedge_{\neg B \in \overline{\text{names}(C)}} \neg B \quad \text{and} \quad \bigwedge_{A' \in \text{names}(D)} A' \sqcap \bigwedge_{\neg B' \in \overline{\text{names}(D)}} \neg B'$$



Hierarchy of conjunctions of (negated) names

We need

subsumption hierarchy of conjunctions of concept names and negated names

But!

- requires 2^{2n} subsumption tests for n concepts
- each subsumption test is expensive
- can we compute it without checking all pairs of conjunctions for subsumption?

Solution

Formal Concept Analysis, Attribute Exploration Algorithm



Hierarchy of conjunctions of (negated) names

We need

subsumption hierarchy of conjunctions of concept names and negated names

But!

- requires 2^{2n} subsumption tests for n concepts
- each subsumption test is expensive
- can we compute it without checking all pairs of conjunctions for subsumption?

Solution

Formal Concept Analysis, Attribute Exploration Algorithm



Formal Concept Analysis

- applied lattice theory for data analysis/knowledge processing
- represent objects/attributes in a cross-table; **formal context**
- classify objects according to their attributes; **formal concept**
- order and represent them in a lattice; **concept lattice**
- query and process the knowledge by means of this lattice

Attribute Exploration

- interactive algorithm to compute the concept lattice
- based on **implications** between attributes
- produces a minimal representation of the concept lattice



Hierarchy of conjunctions of (negated) names with AE

- define a formal context whose concept lattice is isomorphic to this hierarchy [Baader95, BaaderMolitor02, BaaderSertkaya04]
- a decision procedure for subsumption can act as the expert

Attribute exploration with background knowledge

- avoid asking too many questions to the expert
- incorporate background knowledge
- $A \sqcap \neg A \rightarrow \perp$
- $C \sqsubseteq_{\mathcal{T}} D$ implies $\neg D \sqsubseteq_{\mathcal{T}} \neg C$
- $C \sqsubseteq_{\mathcal{T}} D$ implies $C \rightarrow D$ on FCA side



Experimental results

- depend heavily on the shape of the TBox, not just the size
- for 9 concept names (2^{18} conjunctions), 50 minutes
- several seconds for a handcrafted TBox with more concepts
- background knowledge decreases calls to the expert
- but increases overall time, since the expert is highly optimized
- reasoning with background is unoptimized



Summary

- bottom-up approach for KBs in very expressive DLs
- but LCS is restricted to inexpressive DLs
- **LCS w.r.t a background terminology**
- brute-force approach not feasible; instead of LCS, use **GCS**
- hierarchy of conjunctions of (negated) names **using FCA**

Future work

- test attribute exploration with/without background knowledge
- cost and gain of using more background knowledge
- how good is our GCS?

