A Bottom-Up Approach to Designing Ontologies

Anni-Yasmin Turhan

Technische Universität Dresden

The TONES Consortium:
- Free University of Bozen-Bolzano
- Università di Roma "La Sapienza"
- The University of Manchester
- Technische Universität Dresden
- Technische Universität Hamburg-Harburg

http://www.tonesproject.org/

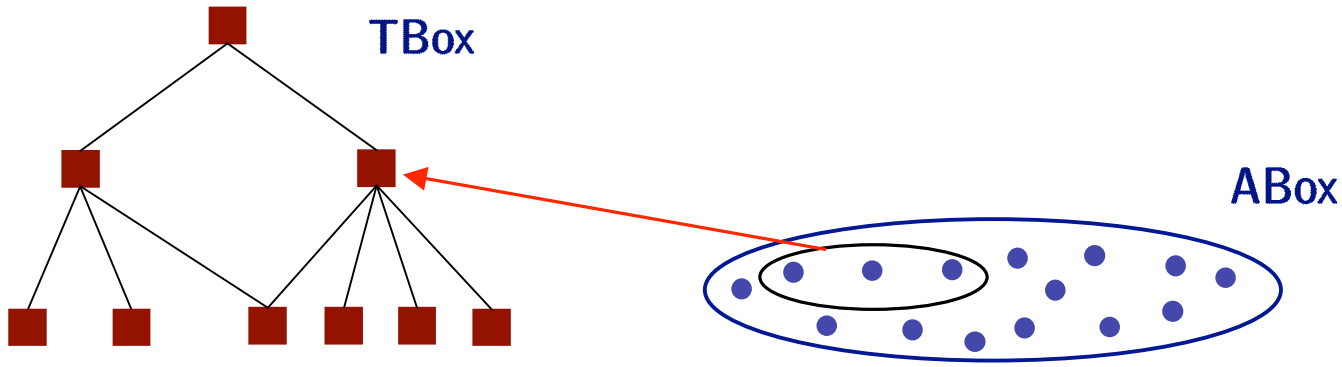
Outline

- Why bottom-up construction?
- How does bottom-up construction work?
  - most specific concept (MSC)
  - least common subsumer (LCS)
- Populating Ontologies
- LCS for expressive languages
  - concept approximation
  - good common subsumers

Building Ontologies

Top-down approach:

first model concepts in the TBox and then model individuals in the ABox



(+) supported by standard reasoning

(−) requires a good knowledge of DLs!

Building Ontologies (II)

- It happens often that a knowledge engineer has a notion in mind, but cannot "code" it in DLs

- Working with users in chemical process engineering showed that it is sometimes easier to first introduce "typical" individuals and then generalize them into a concept

 building ontology in a bottom-up way

Idea: support this method (semi-)automatically

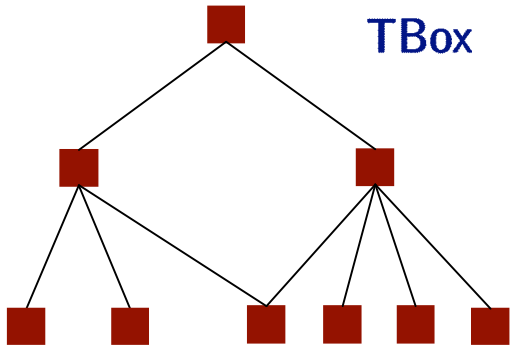
Bottom-Up Construction

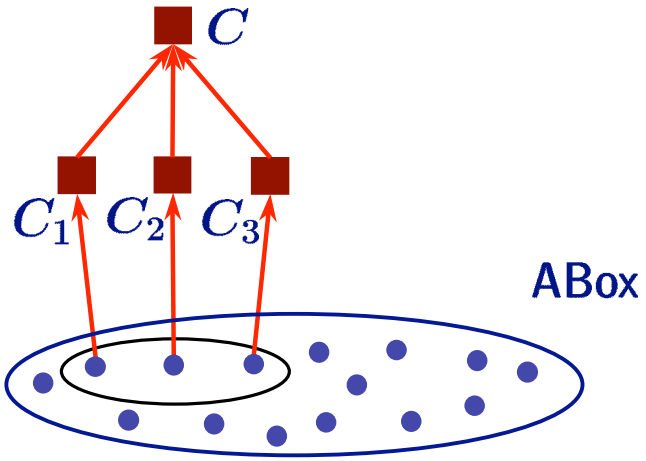
1. User selects similar ABox individuals

2. Automatic generalization of individuals into concept descriptions.

3. Automatic generalization of concept descriptions into one concept description.

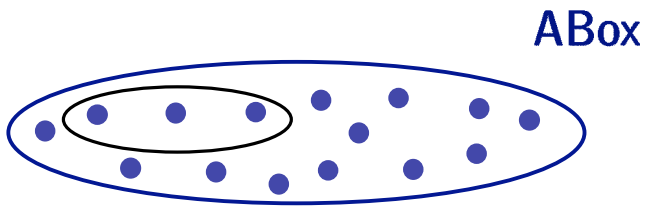
4. User inspects the concept, modifies it and adds it to the terminology





Bottom-Up Construction (Step by Step)

1. User selects similar ABox individuals



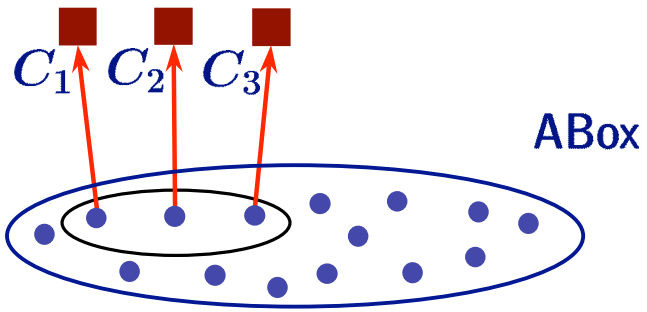
Bottom-Up Construction (Step by Step)

1. User selects similar ABox individuals

2. Automatic generalization of individuals into concept descriptions. Compute the most specific concept (MSC)

The MSC of an individual a in an ABox $\mathcal{A}$ is the concept C such that:

- it follows from $\mathcal{A}$ that a is an instance of C ($\mathcal{A} \models a : C$)
- for each D with $\mathcal{A} \models a : D$, C is subsumed by D



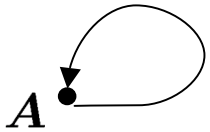
Most Specific Concept (MSC)

Intuitively, computing MSC of an individual a in an ABox $\mathcal{A}$ is building a complex concept description C

- the input individual is an instance of
- that is the best-fitting concept description for input individual

Properties of MSC:

- Available for unfoldable TBoxes
- Exists in $\mathcal{FL}_0$, but not in $\mathcal{EL}$:



$\Longrightarrow$ consider only acyclic ABoxes or compute approximations

- For $\mathcal{ALE}$ we compute k-approximations

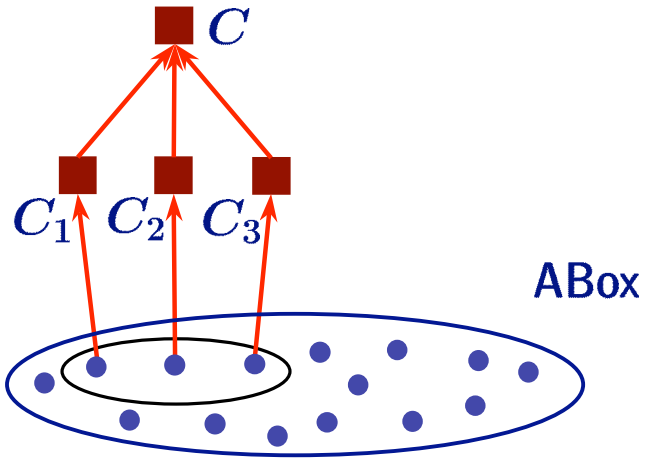
Demo MSC!

Bottom-Up Construction (Step by Step)

1. User selects similar ABox individuals
2. Automatic generalization of individuals into concept descriptions. Compute the most specific concept (MSC)
3. Automatic generalization of concept descriptions into one concept description. Compute the least common subsumer(LCS)

Least common subsumer (LCS)
of input concept descriptions $C_1, \ldots, C_n$ in DL $\mathcal{L}$ is the $\mathcal{L}$-concept description C such that

- C subsumes all input concepts C_i, and
- C is least w.r.t. subsumption



Least Common Subsumer (LCS)

Intuitively, computing LCS of a set of concepts defined in a TBox $\mathcal{T}$ is building a complex concept description C

- that subsumes each of the input concepts
- that is the best-fitting concept description for the input concepts.

Properties of LCS:

- Available for unfoldable TBoxes
- The most expressive logic for which it is implemented is $\mathcal{ALEN}$

Demo LCS!

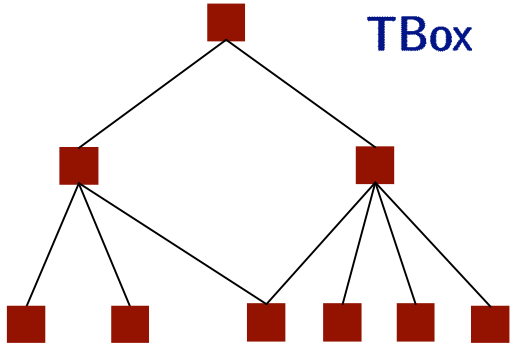
Bottom-Up Construction (Step by Step)

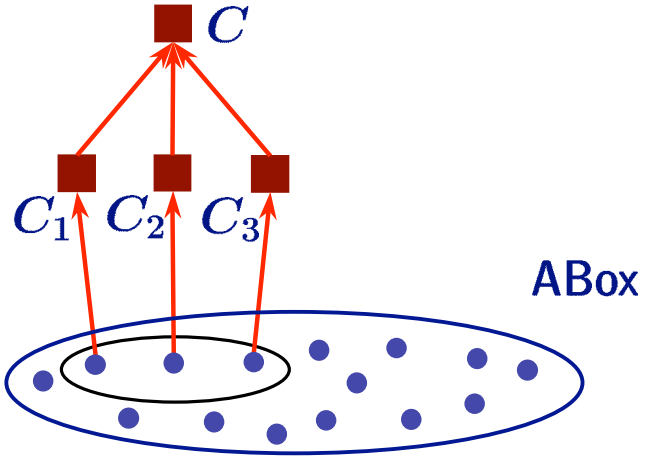
1. User selects similar ABox individuals

2. Automatic generalization of individuals into concept descriptions.
   Compute the most specific concept

3. Automatic generalization of concept descriptions into one concept description.
   Compute the least common subsumer.

4. User inspects the concept, modifies it and adds it to the terminology





Rewriting

Concept descriptions returned by MSC and LCS can grow very large in practice!

Remedy:

Minimal Rewriting:

For a concept description C compute a concept description C', s.t.

- C' is equivalent to C w.r.t. TBox
- C' has minimal size.

Post-processing step in the bottom-up approach.

Demo Generalization!

MSC for populating TBoxes

Problem: rich ABox given, but no concept definitions in TBox

Solution:

- Pick individual with rich description from the ABox
- Apply MSC to it
- (edit and) add obtained concept description to TBox

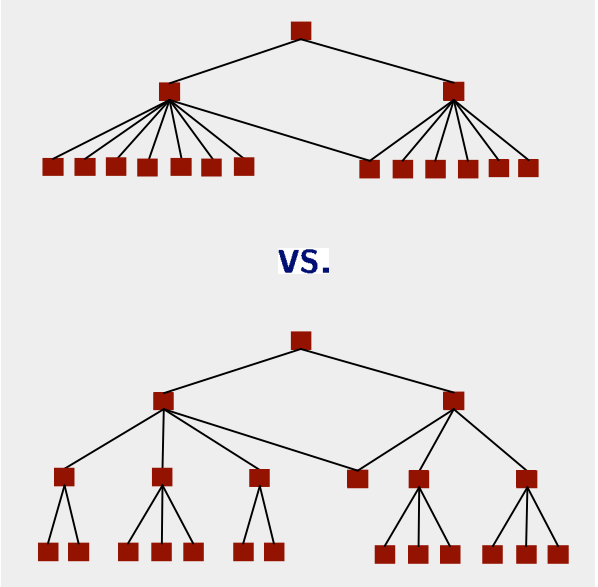
MSC can be used to populate the TBox with complex descriptions!

Demo MSC!

LCS for extending TBoxes

Sometime referred to as "Extended bottom-up approach"

Problem: flat concept hierarchies are not informative



Solution:

- Pick sibling concepts from concept hierarchy
- Apply LCS to them
- (edit and) add obtained concept description to TBox

LCS can be used to obtain a deeper concept hierarchy!

Problem: LCS & disjunction

Application ontologies

- often use more expressive DLs
- use DLs that offer all Boolean operators as concept constructors

Problem with disjunction and LCS:

$$\text{lcs}_{\mathcal{ALC}}(C_1, C_2, \cdots, C_n) = C_1 \sqcup C_2 \sqcup \cdots \sqcup C_n$$

- ➳ not informative
- ➳ does not describe commonalities
- ➳ not a good starting point for editing

Two solutions:

1. approximation-based approach
2. customization-based approach

Approximation-based approach

Approach:

1. Modify input concepts:
   - eliminate disjunction from input concepts
   - preserve as much information as possible

   } concept approximation

2. Compute LCS of modified input concepts in DL without disjunction

Concept Approximation

Concept approximation is a "translation"
from expressive DL $\mathcal{L}_1$ to less expressive DL $\mathcal{L}_2$

$\mathcal{L}_2$-concept approximation of $\mathcal{L}_1$-concept description C is an $\mathcal{L}_2$-concept description that

- subsumes C and
- is least w.r.t. subsumption

Demo Concept Approximation

Customization-based Approach

Frame work:

- user customizes background ontology $\mathcal{T}$ for local application by user ontology

- Background ontology: written in expressive DL — $\mathcal{ALC}$
  - defines basic notions of the application domain.
  - built by knowledge engineer.

- User ontology:
  - refines the background terminology.
  - built by domain experts.
  - written in DL without disjunction — $\mathcal{ALE}$.
  - refers to concept names from background ontology — $\mathcal{ALE}(\mathcal{T})$.
  - is unfoldable.

LCS w.r.t. a background Ontology

Consider: $\mathcal{L}_2 : \mathcal{ALC}$ $(\sqcap, \sqcup, \neg, \forall, \exists)$
$\mathcal{L}_1 : \mathcal{EL}$ $(\sqcap, \exists)$

TBox: $\mathcal{T} := \{A \equiv P \sqcup Q\}$

LCS:
- in $\mathcal{EL}$: $\mathsf{LCS}(P, Q) = \top$
- in $\mathcal{EL}$ $(\mathcal{T})$: $\mathsf{LCS}(P, Q) = A$

Even with Approximation and Rewriting!

Good common subsumers

- No constructive, i.e. feasible method known to compute $\mathcal{ALE}(\mathcal{T})$-lcs in practice!

- We propose good common subsumers instead.

Subsumption-based common subsumer (SCS)

- Not all information from the background knowledge base is used
- Idea: use only the subsumption hierarchy from the background knowledge base
- If concepts in background knowledge base are by chance $\mathcal{ALE}$, full information is used.

Demo SCS

The SONIC System

SONIC

- implements a collection of generalization inferences
- available as a plugin for RacerPorter
- Uses RacerPro as background reasoner
- So far only available for Linux platform RacerPorter

upcoming version (end of November):

- Plug-in for Protege 4.0 RacerPorter

Thank you!