Modularity in Ontologies: Introduction (Part B)

Thomas Schneider[1] Dirk Walther[2]

[1]Department of Computer Science, University of Bremen, Germany

[2]Center for Advancing Electronics, Technical University of Dresden, Germany

ESSLLI, 12 August 2013

Plan for the rest of today's lecture

1. A case for modularity of ontologies
2. Overview and comparison of modularisation approaches
3. Overview of the remainder of this course

And now . . .

1. A case for modularity of ontologies
2. Overview and comparison of modularisation approaches
3. Overview of the remainder of this course

What can I do with my ontology?

Ontology users and engineers use ontologies to

- represent and archive knowledge
- compute inferences from that knowledge (quickly)
  e.g., classification, query answering, explanation

Modularity can help with these tasks.

What can I do with my ontology?

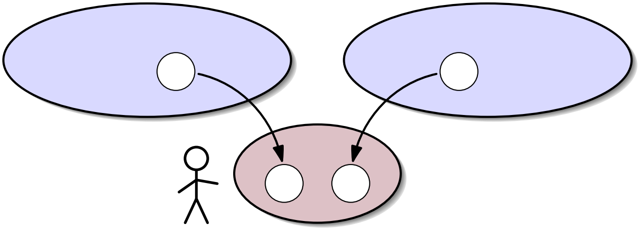
Building and using an ontology can be eased by

- frequent and fast reasoning
  classification, explanations;
  expressivity $\leftrightarrow$ complexity, optimisations, incremental reasoning
- reusing knowledge from existing ontologies
  efficient import
- exposing the logical structure of the represented knowledge
  comprehension
- collaborative development
- version control

Modularity can help with these tasks.

An import/reuse scenario

"Borrow" knowledge from external ontologies



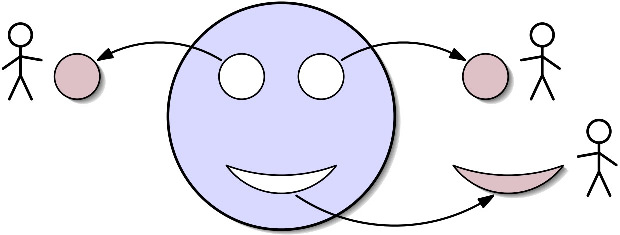
- Provides access to well-established knowledge
- Doesn't require expertise in external disciplines

This scenario is well-understood and implemented.

➽ *Tuesday* + *Wednesday*

A collaboration scenario

Collective ontology development

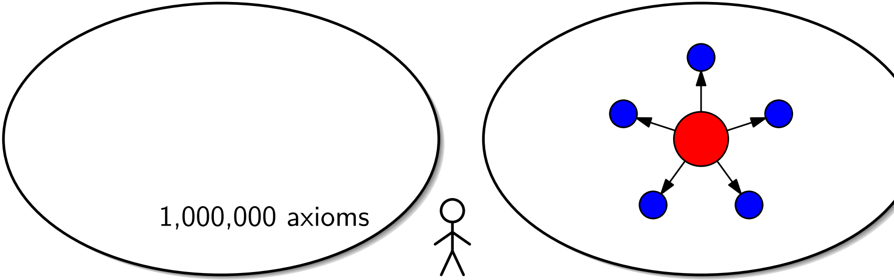


- Developers work (edit, invoke reasoning) locally
- Extra care at re-combination
- Prescriptive/analytic behaviour

This approach is mostly understood, but not implemented yet.

Understanding and/or structuring an ontology

Compute the logical structure of an ontology



This is work in progress. ➳ *Thursday*

And now . . .

1 A case for modularity of ontologies

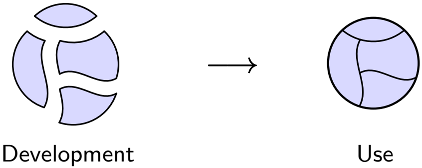
2 Overview and comparison of modularisation approaches

3 Overview of the remainder of this course

A priori vs. a posteriori

A priori modularisation approaches

- First, a modular structure for the ontology $\mathcal{O}$ is decided on.
- Then, $\mathcal{O}$ is developed and used according to that structure.



A-priori modularisation approaches

- Provide a framework to develop an ontology modularly from the start
- Provide means to "bridge" between the modules
  dependency of modules/signature, flow of knowledge
- Often consist of extensions of (description) logics
- Sometimes allow for distributed reasoning
- Generally, don't guarantee that modules are logically closed
  in some cases, this is deliberately so

A-priori: different files with imports

- Used to develop large ontologies about different domains
- Each domain expert (team) maintains “their” file $\mathcal{F}_i$
- The overall ontology $\mathcal{O}$ imports all files:
  $\mathcal{O} = \mathcal{F}_1 \cup \dots \cup \mathcal{F}_n$
- Example: $\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3$ about diseases, anatomy and drugs
- Problems?
  - The $\mathcal{F}_i$ are not necessarily logically closed
  - Experts’ knowledge interferes with each other,
    e.g.: diseases are located in body parts and treated by drugs

  $\leadsto$ Maintenance of $\mathcal{O}$ as difficult as in the monolithic case
  - Reasoning or reuse might still require the whole ontology
- Still used to develop and maintain ontologies!
  see e.g. `http://bioportal.bioontology.org`

Package-based description logics (PB-DLs)

[Bao et al. 2006, 2009]

- Extension of standard DLs
- Domain-specific files are called *packages*
- Semantic import links between packages (explicit dependency)
- Terms annotated with “home package”
- Semantics local w.r.t. each package
- Reasoning controlled by the links
- Translation to “plain” DLs yields implicit decision procedures
- Problems?
  - Reasoning or reuse might still require the whole ontology

Distributed description logics

[Borgida and Serafini 2003] [Serafini and Tamilin 2009]

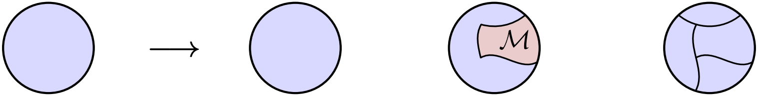
- Similar to PB-DLs
- Replace import links by "bridge rules":
  subconcept relations between (complex) concepts from different packages
- Distributed decision procedures exist

Related notion: E-connections
[Kutz et al. 2001]

A priori vs. a posteriori

A posteriori modularisation approaches

- The ontology $\mathcal{O}$ is built and used as a monolithic entity.
- A module is extracted or $\mathcal{O}$ is decomposed into modules.



(standalone) (module extraction) (decomposition)

Development Use

A-posteriori modularisation approaches

- Regard an ontology $\mathcal{O}$ as a monolithic entity
  remember: $\mathcal{O}$ is a set of axioms
- Module: subset $\mathcal{M} \subseteq \mathcal{O}$
- Extract one module (e.g., for reuse) or
  decompose $\mathcal{O}$ into several modules (e.g., for comprehension)
- Often, a **signature** (set of terms) $\Sigma \subseteq \text{sig}(\mathcal{O})$ is specified and the module extracted using Σ as a parameter
- Ideally, modules **encapsulate** knowledge in some form
  e.g., all consequences of $\mathcal{O}$ in Σ
- Not all a-posteriori module notions guarantee encapsulation

Graph-based a-posteriori modularisation approaches

- Are based on a graph representation of the ontology
  usually concept/role hierarchy, sometimes enriched with disjointness
- Start with a signature Σ
- Traverse the graph and "harvest" entities and axioms
  follow subconcept relation and/or restrictions ($\exists$, domain, range)
- Resulting module = set of harvested axioms
- Examples
  - Ontology segmentation [Seidenberg and Rector 2006, 2009]
  - Traversals [Noy and Musen 2003, 2009]
  - More general framework [d'Aquin et al. 2007]

Pro and contra graph-based approaches

Pro

- Modules can usually be extracted efficiently
  time polynomial in the size of $\mathcal{O}$ $\leadsto$ robustly scalable
- Easy to implement
- Applicable to many logics

Contra

- Heuristic, no characterisation of the expected module contents
- In particular, no logical guarantees such as entailment preservation

$\leadsto$ Modules typically lose knowledge from $\mathcal{O}$

A-posteriori approaches with coverage

Coverage

$\mathcal{M} \subseteq \mathcal{O}$ **covers** $\mathcal{O}$ **for** Σ if

all Σ-consequences of $\mathcal{O}$ already follow from $\mathcal{M}$.

- i.e., $\mathcal{M}$ preserves all knowledge in $\mathcal{O}$ about α
  ➽ *Tuesday*
- This guarantee is needed, e.g., for ontology reuse or reasoning
  ➽ *Tuesday + Wednesday*
- Of course, $\mathcal{O}$ is always covering

Problems

- *Minimal* covering modules are, in general, hard to extract
  ➽ *Tuesday*

Coverage-providing module notions

- Restricted to logics where coverage can be decided efficiently
  e.g., MEX for acyclic $\mathcal{EL}$ ⤁ *Wednesday*
  [Konev et al. 2008]
- Or use a tractable condition sufficient for coverage, leading to modules that always contain minimal modules
  Examples:
  - Modules obtained from partitions based on E-connections
    [Cuenca Grau et al. 2006]
  - Locality-based modules ⤁ *Wednesday*
    [Cuenca Grau et al. 2007, 2009]
  - Reachability-based modules ⤁ *Friday*
    [Suntisrivaraporn 2008]

Comparison of a-posteriori module extraction approaches

Module notion	Covrg.	Min.	Covered DLs	Complexity
All axioms referencing Σ	✘		any	easy
Graph-based	✘		any	easy
The whole ontology	✓	✘✘	any	easy
Minim. mod. with coverage*	✓	✓	few	hard
MEX*	✓	✓	acyclic $\mathcal{EL}$	easy
E-connections based mod.	✓	✘	OWL	easy
Locality-based mod.*	✓	✘	OWL	easy
Reachability-based mod.*	✓	✘	almost OWL	easy
(Modules with rewriting)	✓?	✓✓?	few?	hard?

*Will be covered here ➻ *Tuesday, Wednesday, Friday*

And now . . .

1. A case for modularity of ontologies
2. Overview and comparison of modularisation approaches
3. Overview of the remainder of this course

Course overview

2. Module extraction and its formal foundations
   - A case scenario: modular reuse
   - Logical guarantees required
   - Conservative extensions, inseparability, robustness
3. Module extraction
   - Efficient module notions (locality, MEX)
   - Module extraction algorithms and tools
4. Decomposing ontologies
   - Atomic decomposition
5. Recent advances/current work
   - Forgetting and interpolation
   - Reachability-based modules
   - Incremental reasoning
   - Modular reasoning