Working Modularly with Ontologies

Bijan Parsia    Uli Sattler    Thomas Schneider

School of Computer Science, University of Manchester, UK

30 November 2009
About the project

**Title**

*Composing and decomposing ontologies: a logic-based approach*

**People involved/interested**

- Uli Sattler, Bijan Parsia, Thomas Schneider  (Manchester)
- Frank Wolter, Boris Konev, Dirk Walther  (Liverpool)
- Ian Horrocks, Bernardo Cuenca Grau, Yevgeny Kazakov  (Oxford)
- Carsten Lutz  (Bremen)
- Michael Zakharyaschev, Roman Kontchakov  (London)
And now...

1. Ontologies and Description Logic
2. Why modularity?
3. A reuse scenario
4. Understanding ontologies via modules
Ontology

= collection of statements about a **domain** (*axioms*)

- Language used: usually logic, often *description logic* (*DL*)
- *Inferences* can be drawn from axioms

Domains:
biology, medicine, chemistry, business processes, natural language, ...
Example axioms + inferences

\[ \forall x \left( \text{Duck}(x) \rightarrow \exists y \left( \text{feedsOn}(x, y) \land \text{Grass}(y) \right) \right) \]
Example axioms + inferences

- $\text{Duck} \sqsubseteq \exists \text{feedsOn}. \text{Grass}$

  $\forall x (\text{Duck}(x) \rightarrow \exists y (\text{feedsOn}(x, y) \land \text{Grass}(y)))$

- $\text{Bird} \equiv \text{Duck} \sqcup \text{Chicken}$

  $\forall x (\text{Bird}(x) \leftrightarrow (\text{Duck}(x) \lor \text{Chicken}(x)))$
Example axioms + inferences

- Duck ⊑ ∃ feedsOn . Grass

  \[ \forall x \left( \text{Duck}(x) \rightarrow \exists y \left( \text{feedsOn}(x, y) \land \text{Grass}(y) \right) \right) \]

- Bird ≡ Duck ⊔ Chicken

  \[ \forall x \left( \text{Bird}(x) \leftrightarrow (\text{Duck}(x) \lor \text{Chicken}(x)) \right) \]

- \[ \models \text{Bird} \sqcap \neg \text{Chicken} \sqsubseteq \exists \text{feedsOn}.\text{Grass} \]

  \[ \forall x \left( (\text{Bird}(x) \land \neg \text{Chicken}(x)) \rightarrow \exists y \left( \text{feedsOn}(x, y) \land \text{Grass}(y) \right) \right) \]
Example axioms + inferences

- Duck ⊑ ∃ feedsOn . Grass

\[ \forall x \left( \text{Duck}(x) \rightarrow \exists y \left( \text{feedsOn}(x, y) \land \text{Grass}(y) \right) \right) \]

- Bird ≡ Duck ⊔ Chicken

\[ \forall x \left( \text{Bird}(x) \leftrightarrow (\text{Duck}(x) \lor \text{Chicken}(x)) \right) \]
Example axioms + inferences

- \( \text{Duck} \sqsubseteq \exists \text{feedsOn} \cdot \text{Grass} \)

\[
\forall x \left( \text{Duck}(x) \rightarrow \exists y \left( \text{feedsOn}(x, y) \land \text{Grass}(y) \right) \right)
\]

- \( \text{Bird} \equiv \text{Duck} \sqcup \text{Chicken} \)

\[
\forall x \left( \text{Bird}(x) \leftrightarrow \left( \text{Duck}(x) \lor \text{Chicken}(x) \right) \right)
\]

- \( \text{Tweety} : \text{Duck} \)

\( \text{Duck} \)(Tweety)
Example axioms + inferences

- **Duck** \(_\sqsubseteq\exists\text{feedsOn}.\text{Grass}\)

\[
\forall x \left( \text{Duck}(x) \rightarrow \exists y \left( \text{feedsOn}(x, y) \land \text{Grass}(y) \right) \right)
\]

- **Bird** \(\equiv\) **Duck** \(_\sqcup\)** **Chicken**

\[
\forall x \left( \text{Bird}(x) \leftrightarrow \left( \text{Duck}(x) \lor \text{Chicken}(x) \right) \right)
\]

- **Tweety** : **Duck**

\[
\text{Duck}(\text{Tweety})
\]

\[
\models \text{Tweety} : \exists \text{feedsOn}.\text{Grass}
\]

\[
\exists y \left( \text{feedsOn}(\text{Tweety}, y) \land \text{Grass}(y) \right)
\]
Reasoning tasks

- **Consistency:**
  Does ontology $\mathcal{O}$ have a model?

- **Satisfiability:**
  Is there a model of $\mathcal{O}$ that interprets concept $C$ as nonempty?

- **Subsumption:**
  Does $C \sqsubseteq D$ hold in every model of $\mathcal{O}$?

- **Instance checking:**
  Is individual $x$ an instance of $C$ in every model of $\mathcal{O}$?

Inter-reducible; optimised reasoners available
The Web Ontology Language OWL

- W3C-recommended standard since 2004
- OWL 2 published on 27 Oct.
The Web Ontology Language OWL

- W3C-recommended standard since 2004
- OWL 2 published on 27 Oct.

**OWL Full**
Consistency?, **Reasoning**

**OWL DL**
Based on DL **SROIQ**
∃, ∀, counting, role chains and hierarchies, transitivity, inverse roles, nominals

**OWL EL, QL, RL**
Sub-profiles for efficient reasoning and application orientation
And now . . .

1. Ontologies and Description Logic
2. Why modularity?
3. A reuse scenario
4. Understanding ontologies via modules
A case for modularity

Common practice in software engineering

Modular software development allows for:
- Importing/reusing modules
- Collaborative development
- Understanding the code from the interaction between the modules

Wouldn’t it be nice . . .

. . . to have this for ontology development as well?
Three scenarios

- **Import/reuse**
- **Collaboration**
- **Understanding**
Three scenarios

Import/reuse

Collaboration

Understanding
Scenario 1: Import/reuse

“Borrow” knowledge about certain terms from external ontologies
Scenario 1: Import/reuse

“Borrow” knowledge about certain terms from external ontologies

- Provides access to well-established knowledge
- Doesn’t require expertise in external disciplines
Scenario 1: Import/reuse

“Borrow” knowledge about certain terms from external ontologies

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

This scenario is well-understood and implemented.
Scenario 2: Collaboration

Collaborative ontology development
Scenario 2: Collaboration

Collaborative ontology development

- Developers work (edit, classify) locally
- Extra care at re-combination
Scenario 2: Collaboration

Collaborative ontology development

- Developers work (edit, classify) locally
- Extra care at re-combination

This approach is understood, but not implemented yet.
Scenario 3: Understanding

Visualise the modular structure of an ontology

1,000,000
Scenario 3: Understanding

Visualise the modular structure of an ontology
Scenario 3: Understanding

Visualise the modular structure of an ontology

We’re still playing with this.
Summing up

Import/reuse

Collaboration

Understanding
Summing up

Import/reuse

Collaboration

Understanding
And now ...

1. Ontologies and Description Logic
2. Why modularity?
3. A reuse scenario
4. Understanding ontologies via modules
A reuse scenario

Import/reuse one external ontology
A reuse scenario

Import/reuse one external ontology

Animals

knowledge about “Bird”

Farm
A reuse scenario

Import/reuse one external ontology

How much of Animals do we need?
A reuse scenario

Import/reuse a part of an external ontology

How much of **Animals** do we need?

- **Coverage:** Import *everything* relevant for the chosen terms.
- **Economy:** Import *only* what’s relevant for them.
  Compute that part quickly.
A reuse scenario

Import/reuse parts of several external ontologies

Bird → Animals

Farm

Barn → Buildings
The *Health-e-Child* project

- Arthropathy
  - Arthritis
    - Atrophic Arthritis
    - Polyarthritis
      - Juvenile Chronic Polyarthritis
  - Autoimmune Disease
  - Rheumatologic Disorder
    - Juvenile Rheumatoid Arthritis
The Health-e-Child project

- Arthropathy
  - Arthritis
    - Atrophic Arthritis
    - Polyarthritis
  - Autoimmune Disease
  - Rheumatologic Disorder
  - Rheumatoid Arthritis
  - Juvenile Chronic Polyarthritis
  - Juvenile Rheumatoid Arthritis

- NCI
  - JRAO
  - Galen

- Drugs
- Joints

- C1
- C7

- affects

- isTreatedBy
A working cycle

1. Edit your ontology \( O \)
2. Import a module

Why modularity? Reuse Understanding ontologies
A working cycle

1. Edit your ontology $O$
2. Load an external ontology $\mathcal{E}$
3. Specify terms from $\mathcal{E}$ to be reused
4. Get module from $\mathcal{E}$
5. Import this module into $O$
A working cycle

1. Edit your ontology $\mathcal{O}$
2. Load an external ontology $\mathcal{E}$
3. Specify terms from $\mathcal{E}$ to be reused
4. Get module from $\mathcal{E}$
5. Import this module into $\mathcal{O}$

Examples:
- Farm
- Animals
- Animal, feedsOn
- Animals'
- Farm $\cup$ Animals'
A working cycle

Edit your ontology \( O \)

Load an external ontology \( \mathcal{E} \)

Specify terms from \( \mathcal{E} \) to be reused

Get module from \( \mathcal{E} \)

Import this module into \( O \)

\[ \text{Farm } \cup \text{ Animals' } \]

\[ \text{Buildings} \]

\[ \text{DuckHousing, Silo} \]

\[ \text{Buildings'} \]

\[ \text{Farm} \cup \text{ Animals'} \cup \text{ Buildings'} \]
A working cycle

1. Edit your ontology $O$
2. Load an external ontology $E$
3. Specify terms from $E$ to be reused
4. Get module from $E$
5. Import this module into $O$

Module Coverage
Module coverage

**Goal:** Import everything the external ontology knows about the topic that consists of the specified terms.
Module coverage

**Goal:** Import everything the external ontology knows about the topic that consists of the specified terms.

**Example 1:**

- **Topic:** Fox, Bird, feedsOn
- **On-topic:**
  - Fox \sqsubseteq \forall feedsOn.Bird
  - Fox \sqcap Bird \sqsubseteq \exists feedsOn.T
  - Bird \sqsubseteq \neg Fox
  - Bird \sqsubseteq Bird \sqcup Fox
- **Off-topic:**
  - Duck \sqsubseteq Bird

\[ \text{Goal} = \text{preserve all on-topic knowledge} \]
Module coverage

**Goal:** Import everything the external ontology knows about the topic that consists of the specified terms.

**Question:** Which axioms do we need to import?
Module coverage

Goal: Import everything the external ontology knows about the topic that consists of the specified terms.

Question: Which axioms do we need to import?

Example 2:

\[\text{Animal} \equiv \text{Bird} \sqcup \text{Fox}\]
\[\text{Bird} \equiv \text{Duck} \sqcup \text{Chicken}\]
\[\text{Duck} \sqsubseteq \exists \text{feedsOn}.\text{Grass}\]
\[\text{Chicken} \sqsubseteq \exists \text{feedsOn}.\text{Worm}\]
\[\text{Fox} \sqsubseteq \exists \text{feedsOn}.\text{Bird}\]

\[\text{Farm} \sqcup \text{Animals}\]
\[\models\]
\[\text{Animal} \sqsubseteq \exists \text{feedsOn}.\top\]
Goal: Import everything the external ontology knows about the topic that consists of the specified terms.

Question: Which axioms do we need to import?

Example 2:

\[
\begin{align*}
\text{Animal} & \equiv \text{Bird} \sqcup \text{Fox} \\
\text{Bird} & \equiv \text{Duck} \sqcup \text{Chicken} \\
\text{Duck} & \sqsubseteq \exists \text{feedsOn.} \text{Grass} \\
\text{Chicken} & \sqsubseteq \exists \text{feedsOn.} \text{Worm} \\
\text{Fox} & \sqsubseteq \exists \text{feedsOn.} \text{Bird} \\
\end{align*}
\]

\[\text{Farm} \sqcup \text{Animals}_1 \]

\[\not\sqsubseteq \]

\[\text{Animal} \sqsubseteq \exists \text{feedsOn.} \top \]
**Goal:** Import everything the external ontology knows about the topic that consists of the specified terms.

**Question:** Which axioms do we need to import?

**Example 2:**

\[
\text{Animal} \equiv \text{Bird} \sqcup \text{Fox} \\
\text{Bird} \equiv \text{Duck} \sqcup \text{Chicken} \\
\text{Duck} \sqsubseteq \exists \text{feedsOn. Grass} \\
\text{Chicken} \sqsubseteq \exists \text{feedsOn. Worm} \\
\text{Fox} \sqsubseteq \exists \text{feedsOn. Bird}
\]

\[
\text{Farm} \cup \text{Animals}_2 \\
\not\equiv \\
\text{Animal} \sqsubseteq \exists \text{feedsOn. } \top
\]
Module coverage

**Goal:** Import everything the external ontology knows about the topic that consists of the specified terms.

**Question:** Which axioms do we need to import?

**Example 2:**

\[
\begin{align*}
\text{Animal} & \equiv \text{Bird} \sqcup \text{Fox} \\
\text{Bird} & \equiv \text{Duck} \sqcup \text{Chicken} \\
\text{Duck} & \sqsubseteq \exists \text{feedsOn}.\text{Grass} \\
\text{Chicken} & \sqsubseteq \exists \text{feedsOn}.\text{Worm} \\
\text{Fox} & \sqsubseteq \exists \text{feedsOn}.\text{Bird} \\
\text{Farm} & \sqcup \text{Animals}_3 \\
\text{Farm} & \not\sqsubseteq \text{Animal} \sqsubseteq \exists \text{feedsOn}.\top
\end{align*}
\]
**Goal:** Import everything the external ontology knows about the topic that consists of the specified terms.

**Question:** Which axioms do we need to import?

**Example 2:**

\[
\begin{align*}
\text{Animal} &\equiv \text{Bird} \sqcup \text{Fox} \\
\text{Bird} &\equiv \text{Duck} \sqcup \text{Chicken} \\
\text{Duck} &\sqsubseteq \exists \text{feedsOn. Grass} \\
\text{Chicken} &\sqsubseteq \exists \text{feedsOn. Worm} \\
\text{Fox} &\sqsubseteq \exists \text{feedsOn. Bird}
\end{align*}
\]

\[
\text{Farm} \sqcup \text{Animals}_4 \\
\models \\
\text{Animal} \sqsubseteq \exists \text{feedsOn. T}
\]
Module coverage

- Module $\mathcal{E}'$ covers ontology $\mathcal{E}$ for the specified topic $\mathcal{T}$ if for all concepts $C, D$ built from terms in $\mathcal{T}$:
  
  if $O \cup \mathcal{E} \models C \sqsubseteq D$, then $O \cup \mathcal{E}' \models C \sqsubseteq D$.

- Coverage $\widehat{\models}$ preserving entailments
Module coverage

- Module $\mathcal{E}'$ covers ontology $\mathcal{E}$ for the specified topic $\mathcal{T}$ if for all concepts $C, D$ built from terms in $\mathcal{T}$:
  
  \[
  \text{if } O \cup \mathcal{E} \models C \sqsubseteq D, \text{ then } O \cup \mathcal{E}' \models C \sqsubseteq D.
  \]

- Coverage $\hat{=} \text{ preserving entailments}$

- No coverage $\not\leadsto$ no encapsulation $\not\leadsto$ no module

- With coverage: trade-off minimality $\leftrightarrow$ computation time
Module coverage

- Module $\mathcal{E}'$ covers ontology $\mathcal{E}$ for the specified topic $\mathcal{T}$ if for all concepts $C, D$ built from terms in $\mathcal{T}$:
  
  if $O \cup \mathcal{E} \models C \sqsubseteq D$, 
  then $O \cup \mathcal{E}' \models C \sqsubseteq D$.

- Coverage $\models$ preserving entailments

- Minimal covering modules via conservative extensions (CEs)
  - CEs hard to impossible to decide
  - Tractable approximation: syntactic locality
A working cycle

1. Edit your ontology $\mathcal{O}$
2. Load an external ontology $\mathcal{E}$
3. Specify terms from $\mathcal{E}$ to be reused
4. Get module from $\mathcal{E}$
5. Import this module into $\mathcal{O}$

- Safety
- Module Coverage
Safety

**Goal:** Don’t change the meaning of imported terms.

Don’t add new knowledge about the imported topic.

**Question:** Which axioms are we allowed to write?
Safety

**Goal:** Don’t change the meaning of imported terms. 
= Don’t add new knowledge about the imported topic.

**Question:** Which axioms are we allowed to write?

**Example:**

Tweety: Duck, ¬Flies

Duck ⊑ Bird

Animals

Bird ⊑ Flies

Farm
Safety

**Goal:** Don’t change the meaning of imported terms.
Don’t add new knowledge about the imported topic.

**Question:** Which axioms are we allowed to write?

**Example:**

```
Tweety : Duck, ¬Flies
Duck ⊑ Bird
```

```
Animals

Bird ⊑ Flies
```

```
Farm
```
Goal: Don’t change the meaning of imported terms.

⇒ Don’t add new knowledge about the imported topic.

Question: Which axioms are we allowed to write?

Example:

\[ \text{Tweety: Duck, } \neg \text{Flies} \]
\[ \text{Duck } \sqsubseteq \text{ Bird} \]

\(\text{Farm } \cup \text{ Animals } \models \text{ Bird } \sqsubseteq \text{ Flies}\)

but

\(\text{Animals } \not\models \text{ Bird } \sqsubseteq \text{ Flies}\)
Safety

- Our ontology $O$ uses the imported terms safely if for all concepts $C, D$ built from the imported terms:
  
  \[
  \text{if } \mathcal{E}' \not\models C \sqsubseteq D, \text{ then } O \cup \mathcal{E}' \not\models C \sqsubseteq D,
  \]

- Safety $\hat{=} \text{ preserving non-entailments}$
## Comparison of different approaches

<table>
<thead>
<tr>
<th>Kind of “module”</th>
<th>Covrg.</th>
<th>Min.</th>
<th>Covered DLs</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>All ax’s referencing $T$</td>
<td>✗</td>
<td>any</td>
<td>any</td>
<td>easy</td>
</tr>
<tr>
<td>Seidenberg/Rector</td>
<td>✗</td>
<td>any</td>
<td></td>
<td>easy</td>
</tr>
<tr>
<td>Prompt</td>
<td>✗</td>
<td>?</td>
<td></td>
<td>easy</td>
</tr>
</tbody>
</table>
## Comparison of different approaches

<table>
<thead>
<tr>
<th>Kind of “module”</th>
<th>Covrg.</th>
<th>Min.</th>
<th>Covered DLs</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>All ax’s referencing $\mathcal{T}$</td>
<td>✗</td>
<td></td>
<td>any</td>
<td>easy</td>
</tr>
<tr>
<td>Seidenberg/Rector</td>
<td>✗</td>
<td></td>
<td>any</td>
<td>easy</td>
</tr>
<tr>
<td>Prompt</td>
<td>✗</td>
<td>✗</td>
<td>?</td>
<td>easy</td>
</tr>
<tr>
<td>The whole ontology</td>
<td>✓</td>
<td>✗</td>
<td>any</td>
<td>easy</td>
</tr>
<tr>
<td>conserv.-based mod.</td>
<td>✓</td>
<td>✓</td>
<td>few</td>
<td>hard</td>
</tr>
<tr>
<td>MEX (Liverpool)</td>
<td>✓</td>
<td>✓</td>
<td>acyclic $\mathcal{EL}$</td>
<td>easy</td>
</tr>
<tr>
<td>locality-based mod.</td>
<td>✓</td>
<td>✗</td>
<td>$\approx$ OWL 2 DL</td>
<td>easy</td>
</tr>
<tr>
<td>E-connections</td>
<td>✓</td>
<td>✗</td>
<td>OWL 1 DL</td>
<td>easy</td>
</tr>
</tbody>
</table>
## Comparison of different approaches

<table>
<thead>
<tr>
<th>Kind of “module”</th>
<th>Covrg.</th>
<th>Min.</th>
<th>Covered DLs</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>All ax’s referencing $\mathcal{T}$</td>
<td>✗</td>
<td>any</td>
<td>any</td>
<td>easy</td>
</tr>
<tr>
<td>Seidenberg/Rector</td>
<td>✗</td>
<td>any</td>
<td>any</td>
<td>easy</td>
</tr>
<tr>
<td>Prompt</td>
<td>✗</td>
<td>?</td>
<td>any</td>
<td>easy</td>
</tr>
<tr>
<td>The whole ontology</td>
<td>✓</td>
<td>✓ ✓</td>
<td>any</td>
<td>easy</td>
</tr>
<tr>
<td>conserv.-based mod.</td>
<td>✓</td>
<td>✓</td>
<td>few</td>
<td>hard</td>
</tr>
<tr>
<td>MEX (Liverpool)</td>
<td>✓</td>
<td>✓</td>
<td>acyclic $\mathcal{EL}$</td>
<td>easy</td>
</tr>
<tr>
<td>locality-based mod.</td>
<td>✓</td>
<td>✗</td>
<td>$\simeq$ OWL 2 DL</td>
<td>easy</td>
</tr>
<tr>
<td>E-connections</td>
<td>✓</td>
<td>✗</td>
<td>OWL 1 DL</td>
<td>easy</td>
</tr>
<tr>
<td>interpolants-based (no subsets!)</td>
<td>✓</td>
<td>✓ ✓</td>
<td>few</td>
<td>hard</td>
</tr>
</tbody>
</table>
Module extraction in Protégé 4

Nightly build:

http://owl.cs.manchester.ac.uk/2008/iswc-modtut/equinox.zip

- Realises import scenario
- Provides coverage via locality-based modules
- We’re working on safety . . .
- To be released as Protégé 4 plugin soon

(Thanks to Matthew Horridge.)
## Web service for module extraction

http://owl.cs.manchester.ac.uk/modularity

### OWL Module Extractor

**Ontology source**

Paste your ontology, or enter a URL of a document, into the text box below.

http://www.codi.org/ontologies/pizza/pizza.owl

**Signature**

Enter a signature. Put each entity name on a new line. (Accepts full URIs or URI fragments)

Pizza

**Modularity type**

Select the module type

- Top (lower) module
- Bottom (upper) module
- Bottom-of-top (upper-of-lower) module
- Top-of-bottom (lower-of-upper) module

[Show axioms view (instead of outputting RDF/XML)]

**Extract module**

### Selected signature

**Pizza**

**Module metrics**

- Number of axioms: 112
- Number of logical axioms: 112
- Number of classes: 35
- Number of object properties: 7
- Number of data properties: 0
- Number of individuals: 5

### Module axioms

- CheeseTopping SubClassOf PizzaTopping
- CheeseTopping DisjointWith FishTopping
- FishTopping DisjointWith FruitTopping
- CheeseTopping DisjointWith HerbSidebarTopping
- CheeseTopping DisjointWith MeatTopping
- CheeseTopping DisjointWith NutTopping
- CheeseTopping DisjointWith SauceTopping
- CheeseTopping DisjointWith VegetableTopping
- CheesePizza EquivalentTo Pizza and (hasTopping some CheeseTopping)
- Country EquivalentTo DomainConcept and (America , England , France , Germany , Italy)
- DeepPanBase SubClassOf PizzaBase
- DeepPanBase DisjointWith ThinAndCrispyBase
- DomainConcept DisjointWith ValuePartition
- FishTopping SubClassOf PizzaTopping
- FishTopping DisjointWith HasBottonsome Mix
- FishTopping DisjointWith FruitTopping
- FishTopping DisjointWith HerbSidebarTopping
- FishTopping DisjointWith NutTopping
- FishTopping DisjointWith SauceTopping
- FishTopping DisjointWith VegetableTopping
- Food SubClassOf DomainConcept
- FruitTopping SubClassOf PizzaTopping
- FruitTopping DisjointWith HerbSidebarTopping
- FruitTopping DisjointWith NutTopping
- FruitTopping DisjointWith SauceTopping
- FruitTopping DisjointWith VegetableTopping
- HerbSidebarTopping SubClassOf PizzaTopping
And now ...
We bet Robert Stevens ...

- Ontology about periodic table of the chemical elements
- What is its modular structure?
- What is “the meat” of it?
- We can find it using locality-based modules.
Impetus for the “Meat” idea

Partition of koala.owl via E-connections in Swoop

- Importing part
- Imported but non-importing part
- Isolated part

“Imports vocabulary from”
Partition for the periodic table ontology

- Importing part
- Imported but non-importing part
- Isolated part

→ “imports vocabulary from”
“Meat” via locality-based modules

Hopes:

- Finer-grained analysis
- Guidance for users to choose the right topic(s)
  (module signature $\neq T$)
- Draw conclusions on characteristics of an ontology:
  topicality, connectedness, axiomatic richness, superfluous parts, modelling
“Meat” via locality-based modules

Problem:

- Ontologies of size $n$ can have up to $2^n$ modules
- But do real-life ontologies fall into the worst case?
## Results so far

- **Highly optimised algorithm to extract all modules**

<table>
<thead>
<tr>
<th>Ontology</th>
<th>#Ax</th>
<th>#Terms</th>
<th>#mods</th>
<th>Theor. Max.</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Koala</td>
<td>42</td>
<td>25</td>
<td>3660</td>
<td>33 554 432</td>
<td>9s</td>
</tr>
<tr>
<td>Mereology</td>
<td>44</td>
<td>25</td>
<td>1952</td>
<td>33 554 432</td>
<td>3min</td>
</tr>
</tbody>
</table>
Results so far

- Highly optimised algorithm to extract all modules

<table>
<thead>
<tr>
<th>Ontology</th>
<th>#Ax</th>
<th>#Terms</th>
<th>#mods</th>
<th>Theor. Max.</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Koala</td>
<td>42</td>
<td>25</td>
<td>3660</td>
<td>33 554 432</td>
<td>9s</td>
</tr>
<tr>
<td>Mereology</td>
<td>44</td>
<td>25</td>
<td>1952</td>
<td>33 554 432</td>
<td>3min</td>
</tr>
</tbody>
</table>

- Not scalable

- Single module numbers don’t say much
For 8 ontologies, we modularised randomly generated subontologies.

Mostly “negative” results

Trendline equation: $y = O(1.5^x)$, confidence 0.96
Estimate the number of all modules more precisely

Proportion of “genuine” modules

Relation between module number and justificatory structure of an ontology

...
Outlook 2

- Collaborative ontology development using modules
- Modules that are no subsets
- Connections between modularity and explanations of entailments
- Modularity of specifications
Collaborative ontology development using modules

Modules that are no subsets

Connections between modularity and explanations of entailments

Modularity of specifications

Thank you.