Working Modularly with OWL

Thomas Schneider

School of Computer Science, University of Manchester, UK

27 January 2009
Why modularity?

Reuse

Background

Tools

Comparison

Understanding

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  (Oxford)
- Carsten Lutz  (Bremen)
And now . . .

1. Why modularity?
2. A reuse scenario
3. Logical background
4. Tools
5. Comparison of modularisation approaches
6. 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 + inference

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

- **Duck** ⊆ ∃ feedsOn . Grass

- **Bird** ≡ Duck ⊔ Chicken

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

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

- **Duck** ⊆ ∃ feedsOn . Grass
  - class
  - property
  - class
  - class
  - \( \forall x \left( \text{Duck}(x) \rightarrow \exists y \left( \text{feedsOn}(x, y) \land \text{Grass}(y) \right) \right) \)

- **Bird** ≡ Duck ⊔ Chicken
  - \( \forall x \left( \text{Bird}(x) \leftrightarrow (\text{Duck}(x) \lor \text{Chicken}(x)) \right) \)

- **Tweety** : Duck
  - Duck(Tweety)
  - individual
Example axioms + inference

- **Duck** ⊆ ∃喂食On. Grass

- **Bird** ≡ Duck ⊔ Chicken

- **Tweety** : Duck

- **Tweety** : Bird

- **Tweety** : ∃喂食On. 草
Reasoning tasks

- **Inference:** Does axiom $\alpha$ follow from ontology $O$?
- **Satisfiability:**
  Is there a model of $O$ that interprets class $C$ as nonempty?
- **Instance checking:**
  Is individual $x$ an instance of $C$ in every model of $O$?

Inter-reducible; optimised reasoners available
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

1. **Import/reuse**
   - Reuse

2. **Collaboration**
   - Collaboration

3. **Understanding**
   - 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

Collective ontology development
Scenario 2: Collaboration

Collective ontology development

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

Collective 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. Why modularity?
2. A reuse scenario
3. Logical background
4. Tools
5. Comparison of modularisation approaches
6. 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.
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.

How to achieve coverage and economy?
A reuse scenario

Import/reuse parts of several external ontologies

Bird → Animals
Barn → Buildings
Farm
The *Health-e-Child* project
Why modularity?

Reuse

Background

Tools

Comparison

Understanding

The *Health-e-Child* project

Arthritis

- Arthropathy
- Autoimmune Disease
- Rheumatologic Disorder

- Arthritis
  - Atrophic Arthritis
  - Polyarthritis
  - Juvenile Chronic Polyarthritis
  - *Juvenile Rheumatoid Arthritis*

- Rheumatoid Arthritis

**NCI**

**JRAO**

**Galen**

**Drugs**

**Joints**

Affects

isTreatedBy

*C_1, C_7*
A working cycle

1. Edit your ontology $O$
2. Import a module
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 \( O \)
A working cycle

Edit your ontology $O$

Load an external ontology $E$

Specify terms from $E$ to be reused

Get module from $E$

Import this module into $O$

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 $O$

- $\text{Farm} \cup \text{Animals}'$
- Buildings
- DuckHousing, Silo
- Buildings'
- $\text{Farm} \cup \text{Animals}' \cup \text{Buildings}'$
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 $\mathcal{O}$

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 ⊑ ∀ feedsOn.Bird
  - Fox ⊔ Bird ⊑ ∃ feedsOn.T
  - Bird ⊑ ¬Fox
  - Bird ⊑ Bird ⊔ Fox
- Off-topic:
  - Duck ⊑ Bird

Goal = 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:**

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

\[\text{Farm} \sqcup \text{Animals} \]
\[\models \]
\[\text{Animal} \sqsubseteq \exists \text{feedsOn. } \top\]
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:

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

\[
\text{Farm} \sqcup \text{Animals}_1 \\
\not\sqsubseteq \\
\text{Animal} \sqsubseteq \exists \text{feedsOn. } \top
\]
Why modularity?

Reuse

Background

Tools

Comparison

Understanding

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:

\[
\text{Animal} \equiv \text{Bird} \sqcup \text{Fox} \\
\text{Bird} \equiv \text{Duck} \sqcup \text{Chicken} \\
\text{Duck} \sqsubseteq \exists \text{feedsOn}.\text{Grass} \\
\text{Chicken} \sqsubseteq \exists \text{feedsOn}.\text{Worm} \\
\text{Fox} \sqsubseteq \exists \text{feedsOn}.\text{Bird} \\
\text{Farm} \sqcup \text{Animals}_2 \\
\
\text{Animal} \sqsubseteq \exists \text{feedsOn}.\top
\]
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:

\[
\begin{align*}
\text{Animal} & \equiv \text{Bird} \sqcup \text{Fox} \\
\text{Bird} & \equiv \text{Duck} \sqcup \text{Chicken} \\
\text{Duck} & \sqsubseteq \exists \text{feedsOn}.\text{Grass} \\
\text{Chicken} & \sqsubseteq \exists \text{feedsOn}.\text{Worm} \\
\text{Fox} & \sqsubseteq \exists \text{feedsOn}.\text{Bird} \\
\text{Animals}_3 & \subseteq \exists \text{feedsOn}.\top
\end{align*}
\]
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:**

`Animal ≡ Bird ⊔ Fox`

`Bird ≡ Duck ⊔ Chicken`

`Duck ⊑ ∃ feedsOn.Grass`

`Chicken ⊑ ∃ feedsOn.Worm`

`Fox ⊑ ∃ feedsOn.Bird`

```
Animals₃
```

```
Farm ⊔ Animals₄

╞

Animal ⊑ ∃ feedsOn.⊤
```

```
Animals₃
```

```
Farm
```

```
The module $\mathcal{E}'$ covers the ontology $\mathcal{E}$ for the specified topic $T$ if for all classes $A, B$ built from terms in $T$:

if $\mathcal{O} \cup \mathcal{E} \models A \sqsubseteq B$, then $\mathcal{O} \cup \mathcal{E}' \models A \sqsubseteq B$.

Coverage $\triangleq$ preserving entailments
Module coverage

- The module $E'$ covers the ontology $E$ for the specified topic $T$ if for all classes $A, B$ built from terms in $T$:
  
  \[
  \text{if } O \cup E \models A \sqsubseteq B, \\
  \text{then } O \cup E' \models A \sqsubseteq B.
  \]

- Coverage $\triangleq$ preserving entailments

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

- With coverage: trade-off minimality $\leftrightarrow$ computation time
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?
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, \(\neg\)Flies
\[\text{Duck} \sqsubseteq \text{Bird}\]

Animals

Bird \(\sqsubseteq\) 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:  

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:**

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

\[
\text{Duck} \subseteq \text{Bird}
\]

\[
\text{Farm} \cup \text{Animals} \models \text{Bird} \subseteq \text{Flies}
\]

but

\[
\text{Animals} \notmodels \text{Bird} \subseteq \text{Flies}
\]
Safety

- Our ontology $O$ uses the imported terms safely if for all classes $A, B$ built from the imported terms:
  
  If $\mathcal{E}' \not\models A \sqsubseteq B$, then $O \cup \mathcal{E}' \not\models A \sqsubseteq B$,

- Safety $\hat{=} \text{preserving non-entailments}$
And now . . .

1. Why modularity?
2. A reuse scenario
3. Logical background
4. Tools
5. Comparison of modularisation approaches
6. Understanding ontologies via modules
Module coverage

- The module $\mathcal{E}'$ covers the ontology $\mathcal{E}$ for the specified topic $\mathcal{T}$ if for all classes $A, B$ built from terms in $\mathcal{T}$:
  
  if $\mathcal{O} \cup \mathcal{E} \models A \subseteq B$, 
  then $\mathcal{O} \cup \mathcal{E}' \models A \subseteq B$.

- Coverage $\models$ preserving entailments
The module $\mathcal{E}'$ covers the ontology $\mathcal{E}$ for the specified topic $\mathcal{T}$ if for all classes $A, B$ built from terms in $\mathcal{T}$:

if $O \cup \mathcal{E} \models A \sqsubseteq B$,
then $O \cup \mathcal{E}' \models A \sqsubseteq B$.

Coverage $\models$ preserving entailments

$O$ may allow “more” interpretations of imported terms than $\mathcal{E}$.
If so, include more “restricting” axioms into $\mathcal{E}'$.
Finish when all terms $\not\in \mathcal{E}'$ can be interpreted as $\bot$ or $\top$.
Locality says whether this is possible.
Notions of covering modules

- **Minimal coverage-providing modules**
  - based on conservative extensions
  - hard to compute (intractable/undecidable)

- **Locality-based modules**
  - based on the above considerations
  - not minimal, hard to compute

- **Modules based on syntactic locality**
  - not minimal, easy to compute (tractable)
Notions of covering modules

- Minimal coverage-providing modules
  based on conservative extensions
  hard to compute (intractable/undecidable)

- Locality-based modules
  based on the above considerations
  not minimal, hard to compute

- Modules based on syntactic locality
  not minimal, easy to compute (tractable)

- Computation:

  \[ \mathcal{T} \leftarrow \text{topic}; \quad M \leftarrow \emptyset \]
  While there is non-local axiom \( \alpha \) w.r.t. \( \mathcal{T} \cup \text{sig}(M) \) do:
  \[ M \leftarrow M \cup \{ \alpha \} \]
  extended topic
Notions of covering modules

- Minimal coverage-providing modules
  - based on conservative extensions
  - hard to compute (intractable/undecidable)

- Locality-based modules
  - based on the above considerations
  - not minimal, hard to compute

- Modules based on *syntactic locality*
  - not minimal, easy to compute (tractable)

- Computation:

  \[ T \leftarrow \text{topic}; \quad M \leftarrow \emptyset \]
  \[ \text{While there is non-local axiom } \alpha \text{ w.r.t. } T \cup \text{sig}(M) \text{ do:} \]
  \[ M \leftarrow M \cup \{ \alpha \} \]
  extended topic

- We often extract the $T$-module of the $\bot$-module of $E$. 
And now . . .

1. Why modularity?
2. A reuse scenario
3. Logical background
4. Tools
5. Comparison of modularisation approaches
6. Understanding ontologies via modules
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
- Will soon provide safety too . . .
- To be released as Protégé 4 plugin in the near future

(Thanks to Matthew Horridge.)
Web-based 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.co-ode.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

Module: http://www.co-ode.org/ontologies/pizza/pizza.owl_module.owl

Selected signature
Pizza (http://www.co-ode.org/ontologies/pizza/pizza.owl#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
- CheeseTopping DisjointWith FruitTopping
- CheeseTopping DisjointWith HerbSaltPepperTopping
- CheeseTopping DisjointWith MeatTopping
- CheeseTopping DisjointWith NutTopping
- CheeseTopping DisjointWith SauceTopping
- CheeseTopping DisjointWith VegetableTopping
- CheesyPizza EquivalentTo Pizza and hasTopping some CheeseTopping
- Country EquivalentTo DomainConcept and (American, England, France, Germany, Italy)
- DeepPanBase SubClassOf PizzasBase
- DeepPanBase SubClassOf ThinAndCrispyBase
- DomainConcept DisjointWith ValuePartition
- FishTopping SubClassOf PizzaTopping
- FishTopping DisjointWith HerbSaltPepperTopping
- FishTopping DisjointWith MeatTopping
- FishTopping DisjointWith NutTopping
- FishTopping DisjointWith SauceTopping
- FishTopping DisjointWith VegetableTopping
- Food SubClassOf DomainConcept
- FruitTopping SubClassOf PizzaTopping
- FruitTopping DisjointWith HerbSaltPepperTopping
- FruitTopping DisjointWith MeatTopping
- FruitTopping DisjointWith NutTopping
- FruitTopping DisjointWith SauceTopping
- FruitTopping DisjointWith VegetableTopping
- HerbSaltPepperTopping SubClassOf PizzaTopping
Web-based module extraction

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

Try it! 😊

- Ontology: http://www.co-ode.org/ontologies/pizza/pizza.owl
- Signature “Pizza”, “VegetarianPizza”, or “Country”
- Select “Show axioms view”

(Thanks to Matthew Horridge.)

This tool currently ignores non-logical axioms (annotations etc.).
And now . . .

1. Why modularity?
2. A reuse scenario
3. Logical background
4. Tools
5. Comparison of modularisation approaches
6. Understanding ontologies via modules
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>
</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 $T$</td>
<td>✗</td>
<td>any</td>
<td>easy</td>
<td></td>
</tr>
<tr>
<td>Seidenberg/Rector</td>
<td>✗</td>
<td>any</td>
<td>easy</td>
<td></td>
</tr>
<tr>
<td>Prompt</td>
<td>✗</td>
<td>?</td>
<td>easy</td>
<td></td>
</tr>
<tr>
<td>The whole ontology</td>
<td>✓</td>
<td>xx</td>
<td>any</td>
<td>easy</td>
</tr>
<tr>
<td>MEX (Liverpool) conserv.-based mod.</td>
<td>✓</td>
<td>✓</td>
<td>acyclic EL</td>
<td>easy</td>
</tr>
<tr>
<td>locality-based mod.</td>
<td>✓</td>
<td>✗</td>
<td>≈ OWL 1 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 $T$</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>MEX (Liverpool) conserv.-based mod.</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 1 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>
And now . . .

1. Why modularity?
2. A reuse scenario
3. Logical background
4. Tools
5. Comparison of modularisation approaches
6. Understanding ontologies via modules
We bet Robert …

- Ontology about periodic table of the chemical elements
- 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

- **Hope**: finer-grained analysis
- **Difficulties**: Computation harder, interpretation unclear
“Meat” via locality-based modules

- **Hope:** finer-grained analysis
- **Difficulties:** Computation harder, interpretation unclear

**Results so far**
- 416 modules for all $\approx$ 800 singleton topics
- Sizes 0, $\ldots$, 2800; average 1600 ($\approx$ 4%)
- Found small modelling irregularity
Why modularity?

Reuse

Background

Tools

Comparison

Understanding

“Meat” via locality-based modules

- Hope: finer-grained analysis
- Difficulties: Computation harder, interpretation unclear

Results so far

- 416 modules for all \( \approx 800 \) singleton topics
- Sizes 0, \ldots, 2800; average 1600 (\( \approx 4\% \))
- Found small modelling irregularity

Struggle with visualisation

Blowup-free methodology for bigger modules?

What does the collection of all modules tell us?

Modules for topics of axioms?