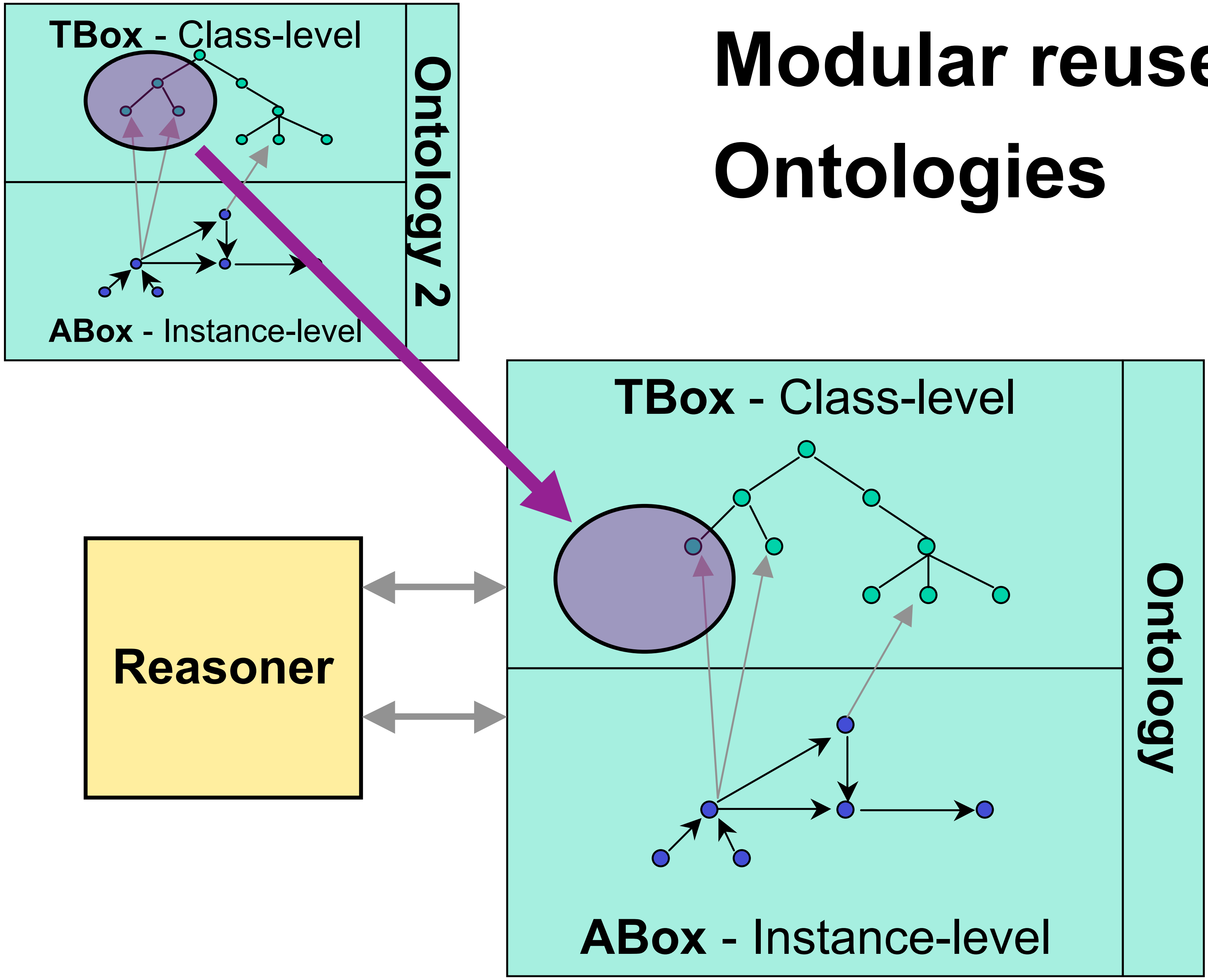
Reasoning for Ontology Engineering and Reuse

Part 3
Modularisation and Explanation

Matthew Horridge, Uli Sattler
University of Manchester

Modularisation

Modular reuse of Ontologies

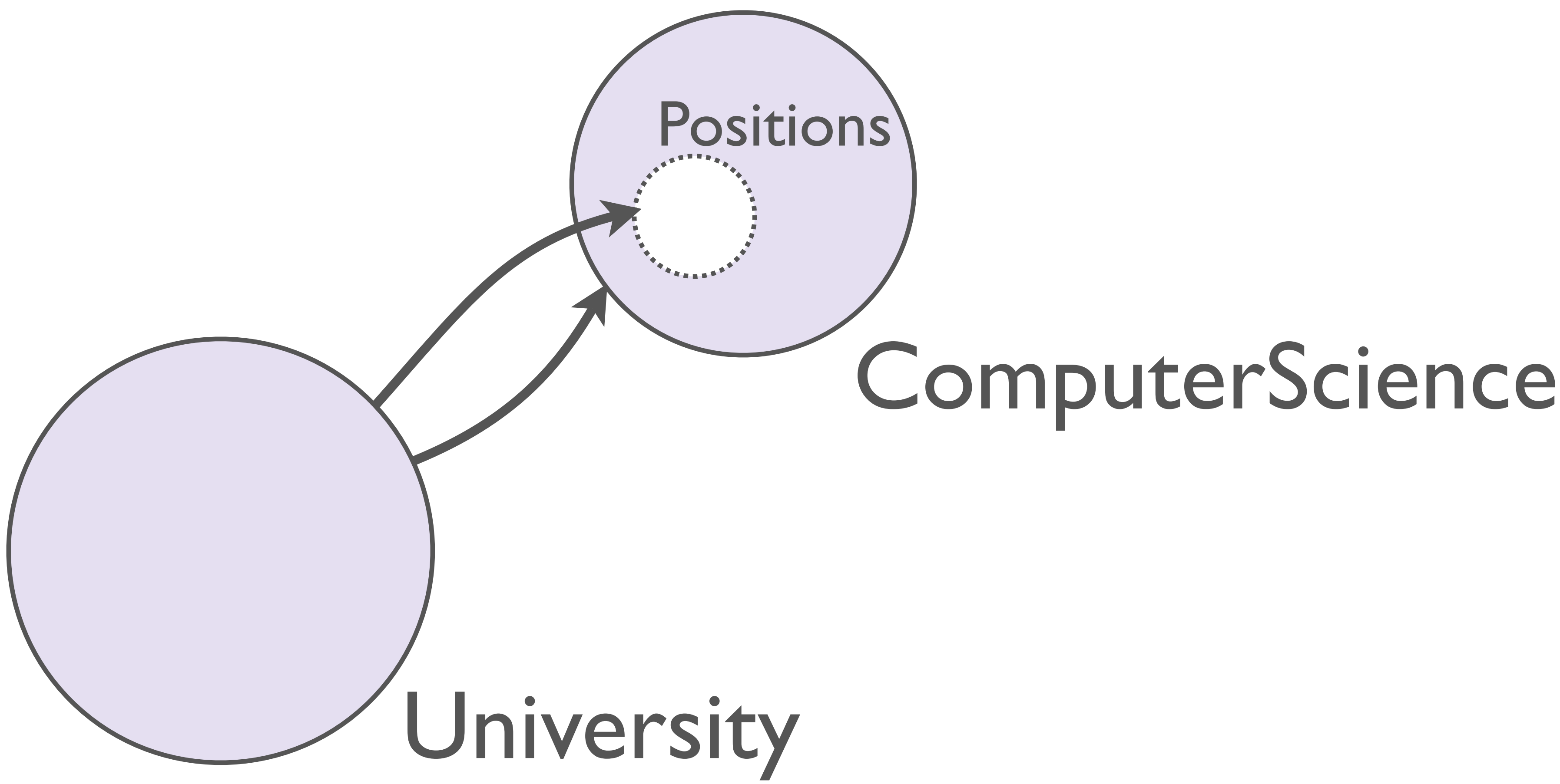


Why Modules & Reuse?

Many good reasons:

- common practice in software engineering
- we can borrow terms from other ontologies
  - to cover topics that we aren't experts in
  - to safe time
  - to ensure common understanding
- modularize our ontology
  - to enable collaborative development
  - to gain insight into its structure & dependencies

Imports/Reuse Scenario



Coverage: Import everything relevant for the given terms

Economy: Import only what is relevant for them

Methodology

Edit working ontology O1	University
Load external ontology O2	ComputerScience
Select terms from O2 to be reused	Person (and subclasses)
Get module from O2	ComputerScienceMod
Import module into O1	University ∪ ComputerScience

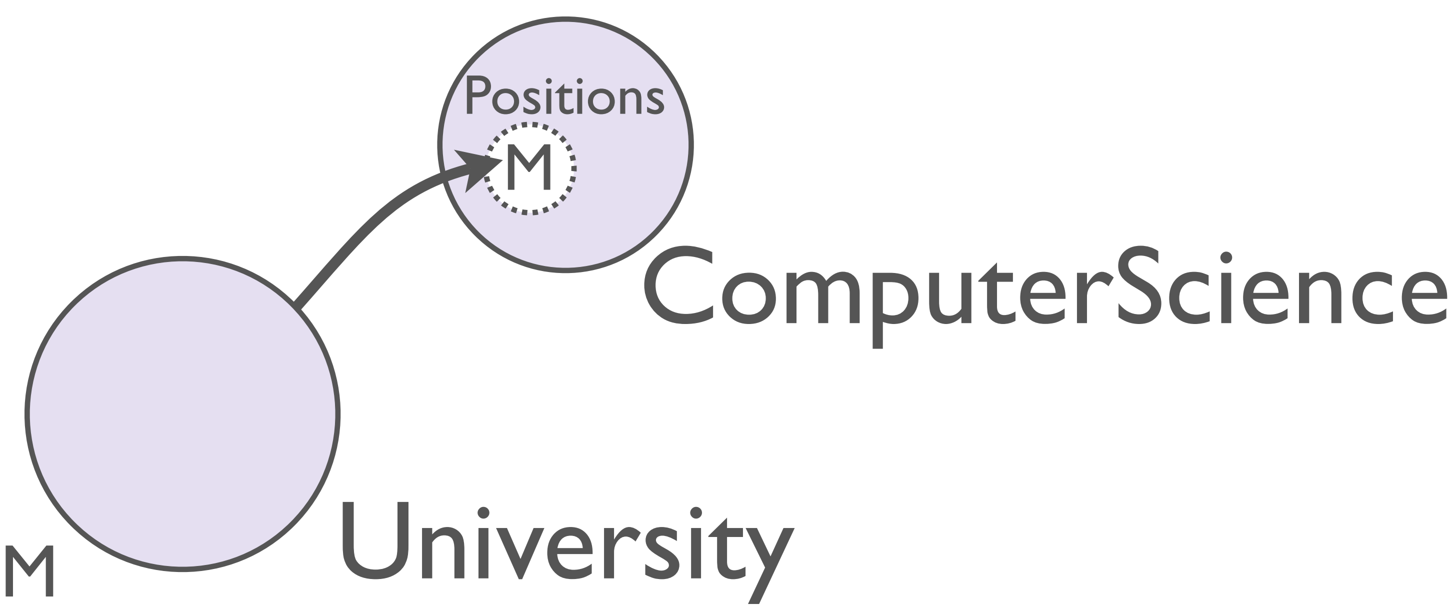
Coverage

Goal: Import everything the external ontology knows about the topic that consists of the specified terms (but hopefully not the whole ontology)

A module, $M \subseteq E$ covers E for the specified terms if for all class expressions C, D built from these terms:

If $\quad O \cup E \models C \sqsubseteq D$

then $\quad O \cup M \models C \sqsubseteq D$



Coverage:
Preserving entailments
No difference between using E or M

Coverage

- How to guarantee coverage?
  - In general, undecidable
  - Closely related to "conservative extensions"
- We use a syntactic approximation of a semantic approximation
  - Fast!
  - Quite good so far - modules are not minimal in size, but guarantee coverage

Safety

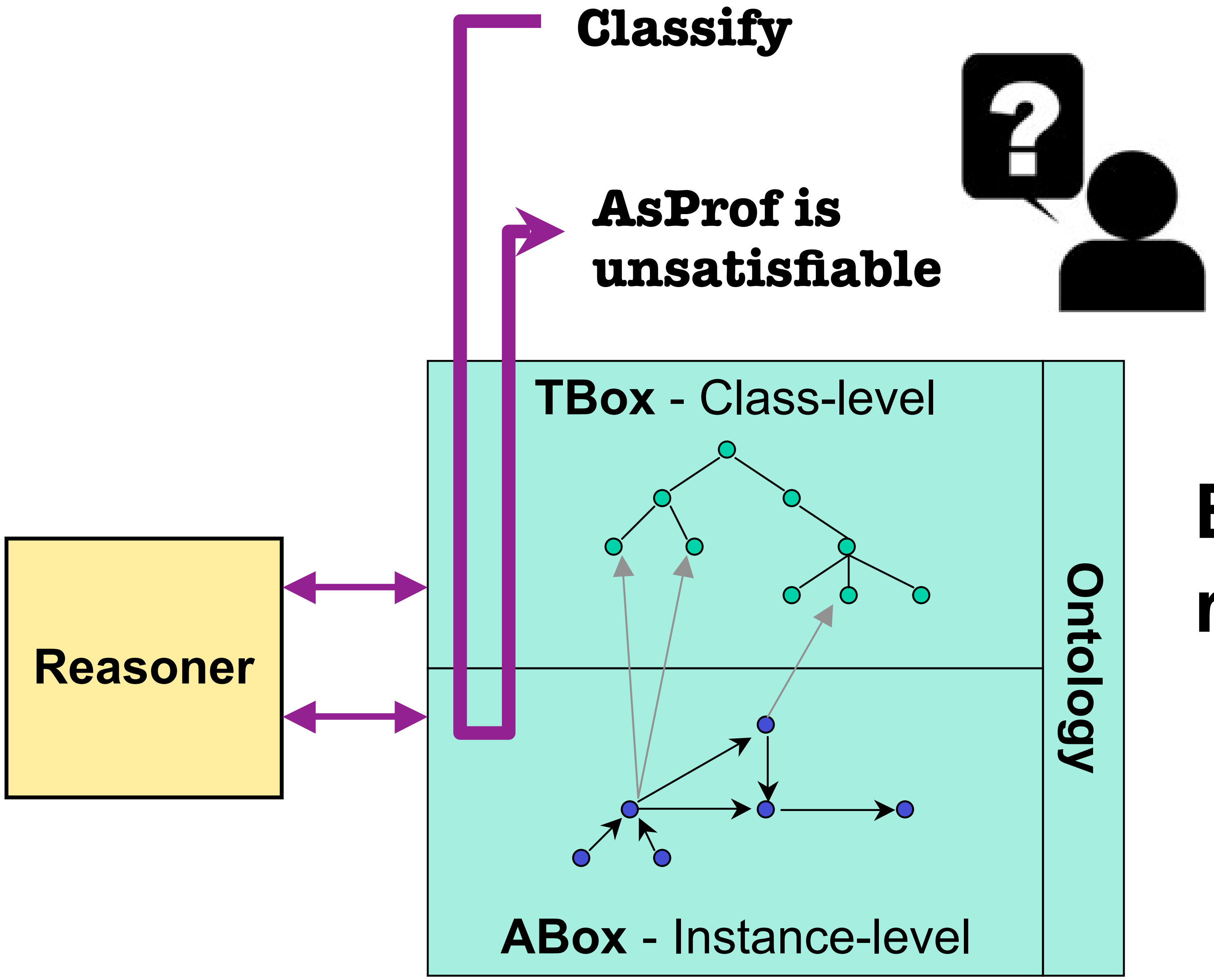
- Do you want to preserve meaning of terms imported?
    - e.g., because you are not an expert in this topic
    - also closely related to "conservative extensions"
- Subject to on-going research and development
    - please stay tuned!

Module Extraction in Protégé

You can follow this demo using the

- version of Protégé and
- example ontologies from the tutorial web page

http://owl.cs.manchester.ac.uk/2008/iswc-tones/



Explanation of reasoning

Root Unsatisfiable Classes

- How do we know which unsatisfiable classes to focus on?

Root Unsatisfiable Classes

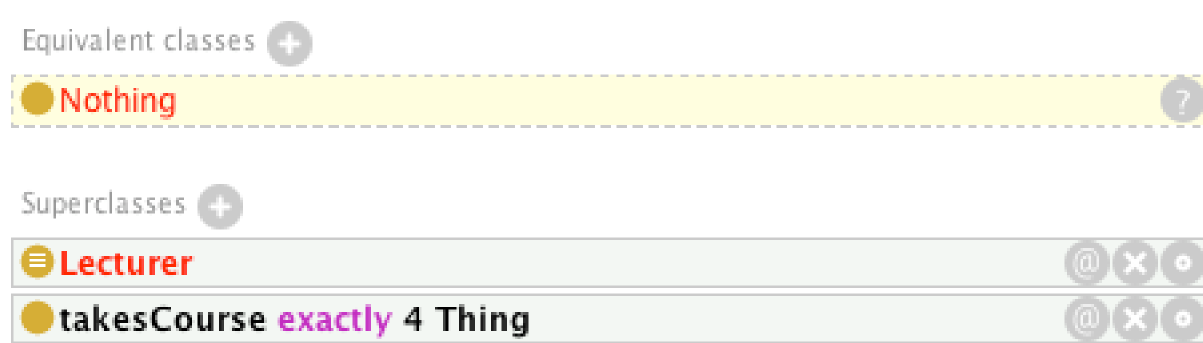
(Side example)

A published ontology, the TAMBIS ontology, contains **144** unsatisfiable classes

Root/Derived Unsatisfiable Classes

- How do we know where to start?
- The satisfiability of one class may depend on the satisfiability of another class
- The tools show unsatisfiable class names in red

LecturerTaking4Courses



Root/Derived Unsatisfiable Classes

- How do we know where to start?
- The satisfiability of one class may depend on the satisfiability of another class
- The tools show unsatisfiable class names in red

CS_Course

Root/Derived Unsatisfiable Classes

- How do we know where to start?
- The satisfiability of one class may depend on the satisfiability of another class
- The tools show unsatisfiable class names in red
- Manual tracing can be very time consuming

Root/Derived Unsatisfiable Classes

- A class whose satisfiability depends on another class is known as a derived unsatisfiable class
- An unsatisfiable class that is not a derived unsatisfiable class is a root unsatisfiable class

Root unsatisfiable classes should be examined and fixed first

Finding Root Unsatisfiable Classes in Protégé

Justifications

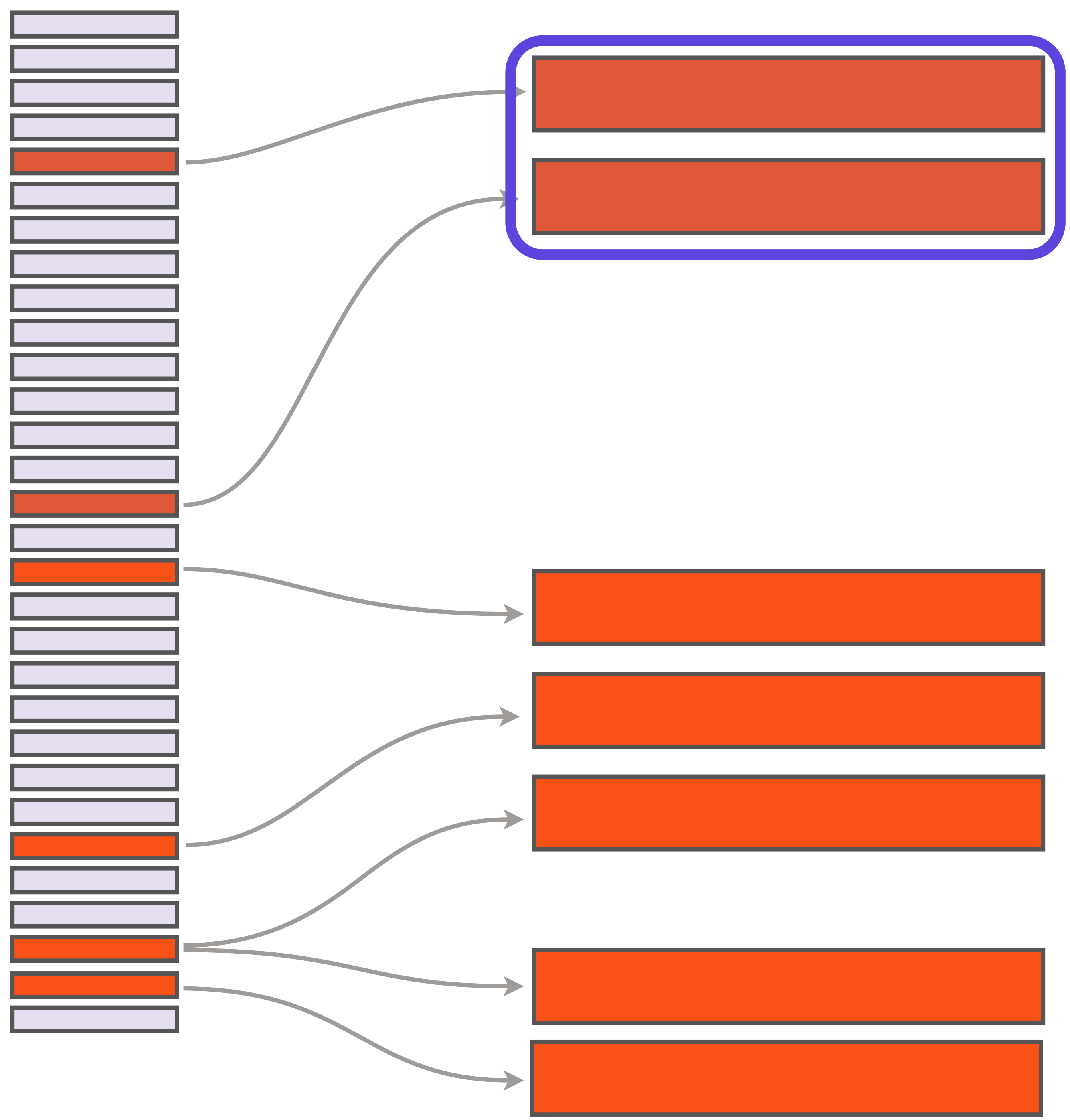
- Justifications are a kind of explanation
- Justifications are minimal subsets of an ontology that are sufficient for a given entailment to hold
- Also known as MUPS, MinAs

Justifications

$$\mathcal{O} = \{\alpha_1, \alpha_2 \ldots \alpha_n\} \qquad \mathcal{O} \models \eta$$

$$\boxed{\begin{array}{ll} J \subseteq \mathcal{O} & J \models \eta \\ \forall J' \subset J & J' \not\models \eta \end{array}}$$

Justifications



Justifications

- There may be multiple justifications for an entailment
- For a given entailment, if there are multiple justifications they may overlap
- Removing one axiom from each justification breaks the justifications so that the entailment is no longer supported by the remaining axioms. This is a repair.

Root/Derived Unsatisfiable Classes

- A class is a derived unsatisfiable class if it has a justification that is a superset of a justification for some other unsatisfiable class.
- An unsatisfiable class that is not derived is a root unsatisfiable class, i.e., none of its justifications contains a justification of another unsatisfiable class.

Root/Derived Unsatisfiable Classes

- Partially derived unsatisfiable classes - derived unsatisfiable classes for which there is at least one justification that is not a superset of justifications for other unsatisfiable classes
- Purely derived unsatisfiable classes - unsatisfiable classes for which all of the justifications are supersets of justifications for other unsatisfiable classes

Justifications in Protégé

Computing Justifications

- Implementations of a service for computing justifications can be split into two main categories:
  - Glass-box
  - Black-box

Glass-box

- Glass-box techniques are specific to a particular reasoner
- For an existing reasoner, implementing glass box tracing requires a thorough and non-trivial modification of the reasoner internals
- Examples: Pellet, CEL

Black-box

- Does not depend on a particular reasoner
- All that we require is that we can ask the reasoner whether a class expression is satisfiable - i.e. satisfiability checking

Entailments to Unsatisfiable Expressions

$$\mathcal{O} \models C \sqsubseteq D$$

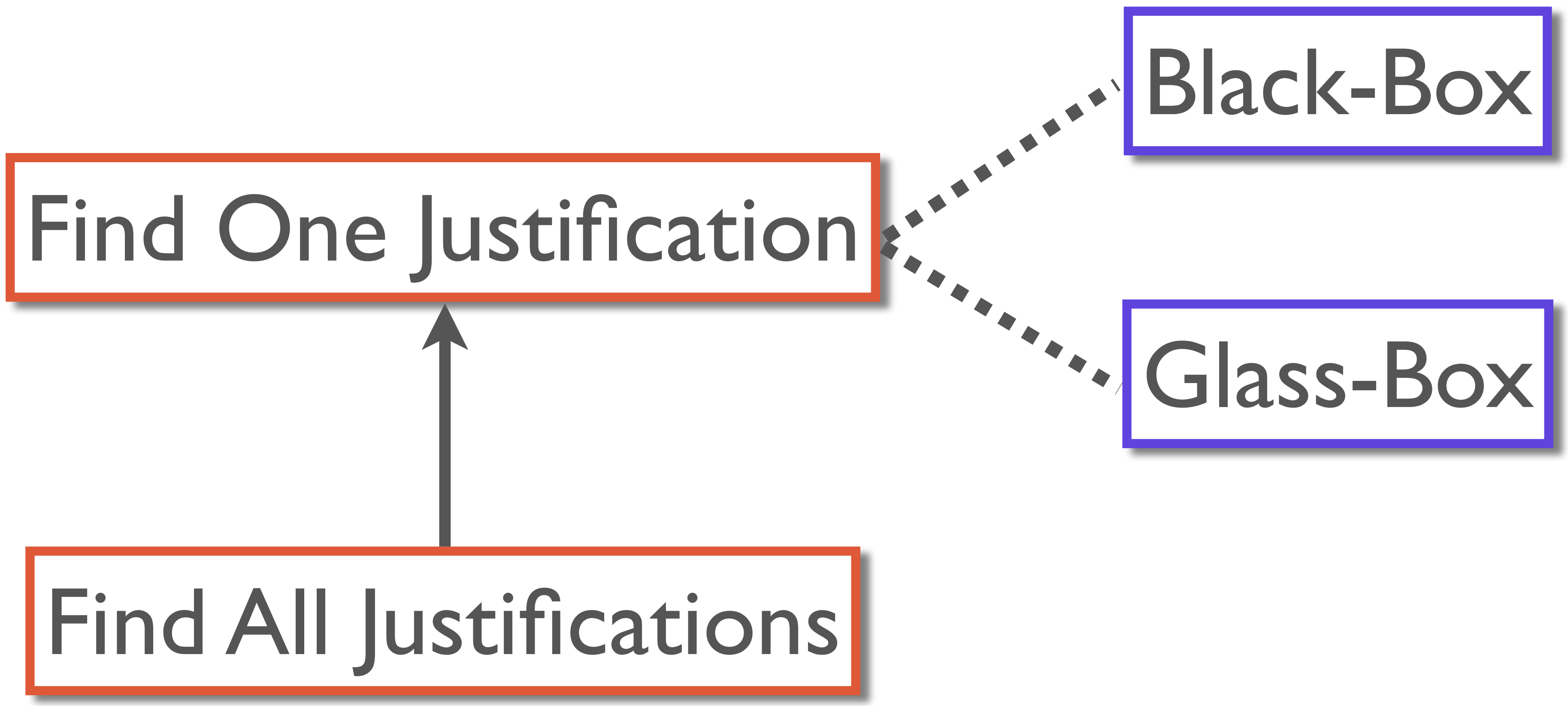
$$\downarrow$$

$$\mathcal{O} \models \boxed{C \sqcap \neg D} \equiv \bot$$

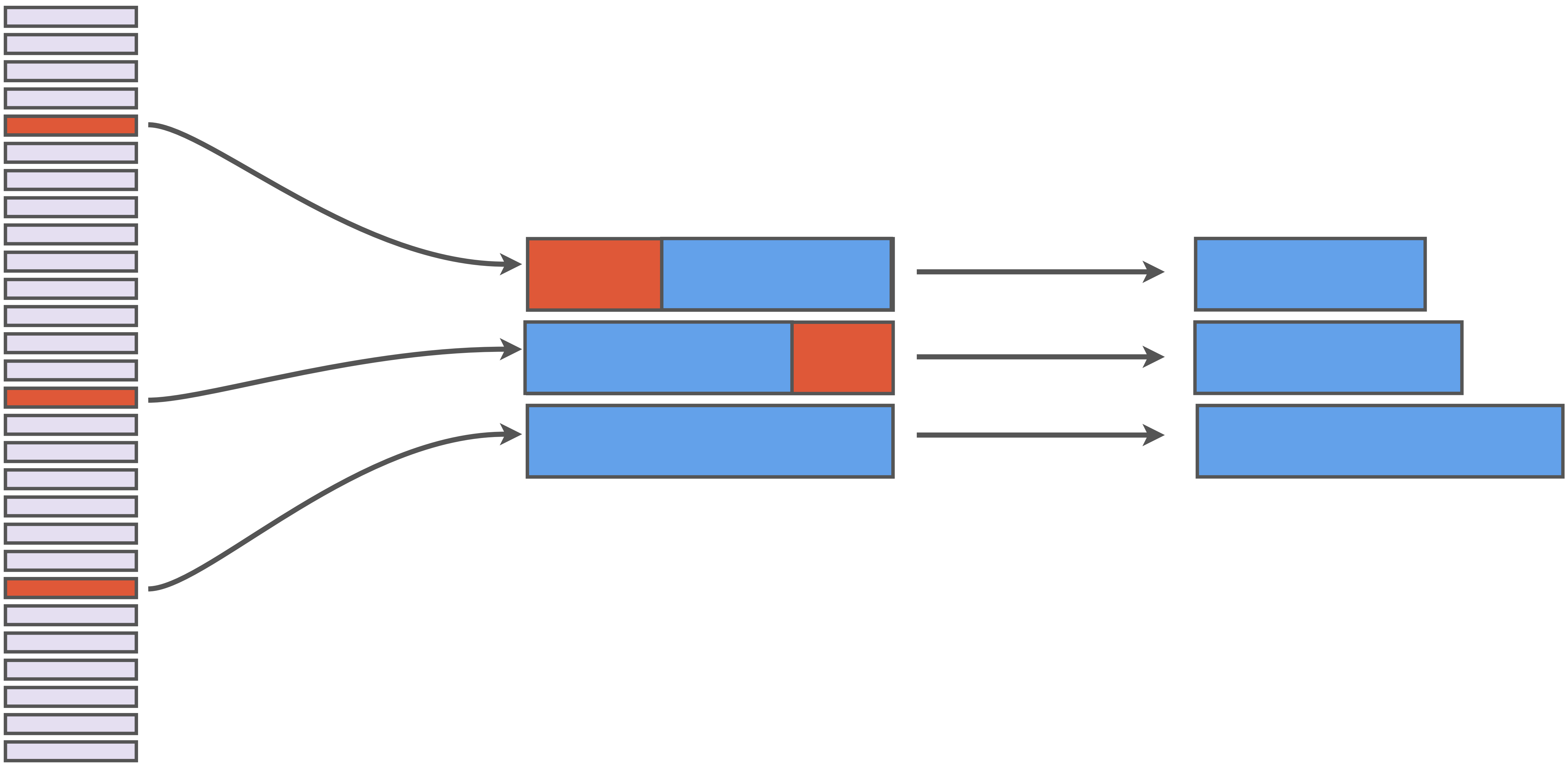
Black-box

- Typically uses an expand-contract strategy
  - Create an empty ontology
  - Expand until expression is unsatisfiable
  - Prune until the expression is satisfiable
  - Several optimisations, including the use of use modularity

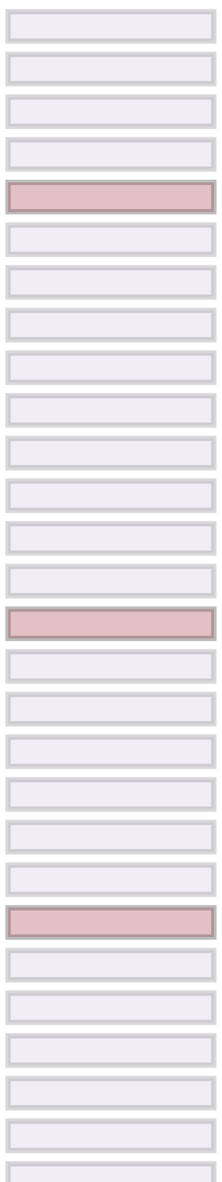
Computing Justifications



Superfluousness



Superfluousness

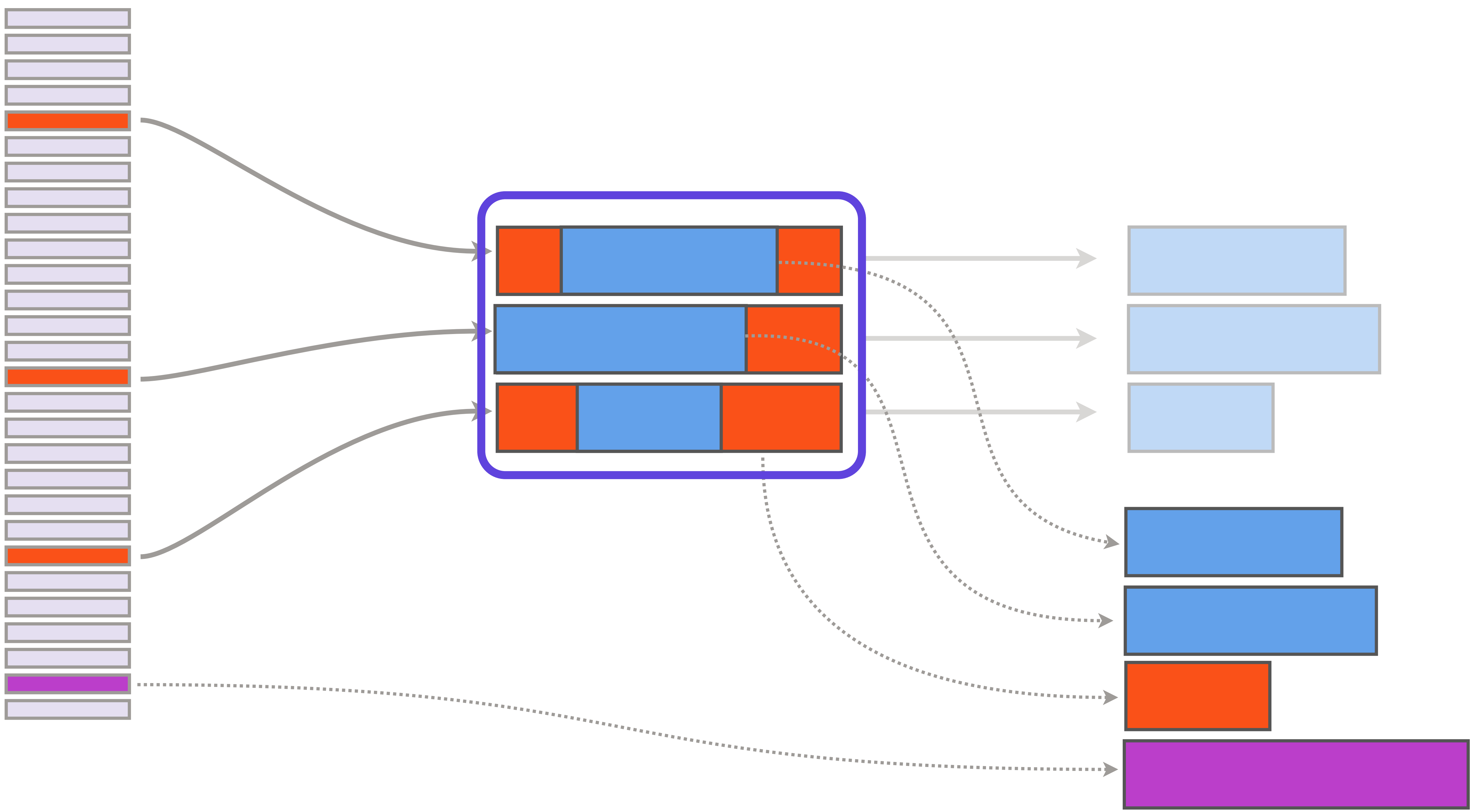


$$E \sqcup D \sqsubseteq B \longrightarrow D \sqsubseteq B$$

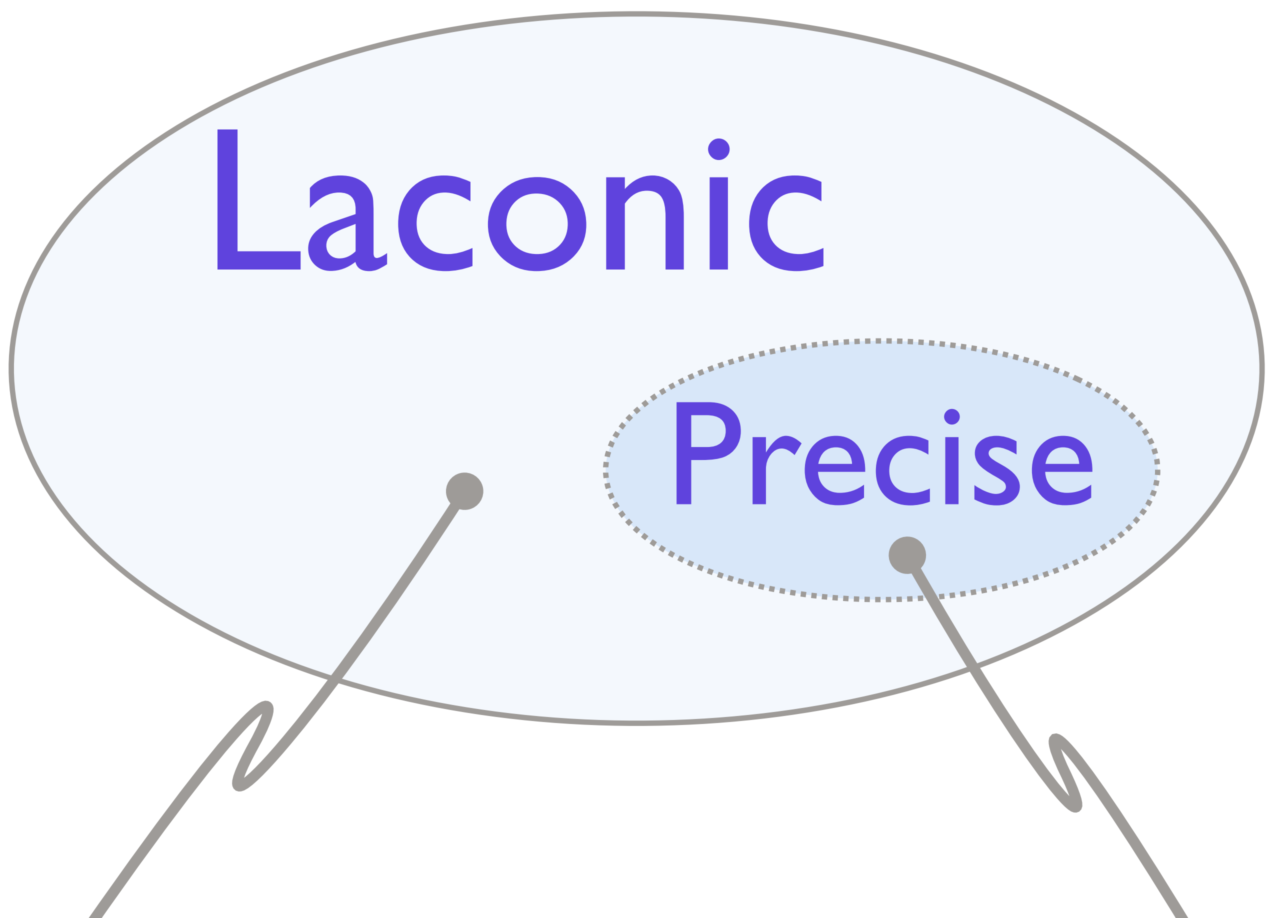
$$B \sqsubseteq \exists R.C \longrightarrow B \sqsubseteq \exists R$$

$$B \sqsubseteq \forall R.\bot \longrightarrow B \sqsubseteq \forall R.\bot$$

External Masking



Fine-grained Justifications



No superfluous parts

All parts as weak as possible

Primarily geared towards repair

Each axiom is a minimal repair

Example

$$\mathcal{O} = \{ A \sqsubseteq D \sqcap = 1R.C \sqcap B$$
$$D \sqsubseteq \forall R.C \sqcap F$$
$$E \equiv \exists R.C \sqcap \forall R.C \} \models A \sqsubseteq E$$

$$A \sqsubseteq D \sqcap \geq 1R$$
$$D \sqsubseteq \forall R.C$$
$$\exists R.C \sqcap \forall R.C \sqsubseteq E$$

Laconic Justifications in Protégé

Internal Masking

$$\mathcal{O} = \{A \sqsubseteq B \sqcap \exists R.C \sqcap \forall R.C$$
$$F \equiv \exists R.C\} \models A \sqsubseteq F$$

1) $A \sqsubseteq B \sqcap \exists R.C \sqcap \forall R.C$

(plus $F \equiv \exists R.C$)

1) $A \sqsubseteq B \sqcap \exists R.C \sqcap \forall R.C$

(plus $F \equiv \exists R.C$)

Wrap Up

- Modules for re-use
- Root/derived unsatisfiable classes
- Justifications
- Fine-grained Justifications
  - Laconic justifications
  - Precise justifications
- Tools available as plugins for Protégé 4