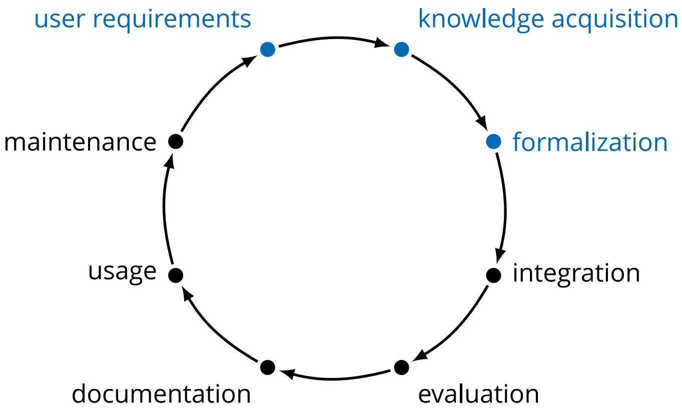
TECHNISCHE UNIVERSITÄT DRESDEN

Stefan Borgwardt
Institute of Theoretical Computer Science, Chair of Automata Theory

Logic-Based Ontology Engineering

Part 2: Ontology Creation

The Ontology Life Cycle



Outline

2.1 Knowledge Acquisition

Engineers and Experts

Creating an ontology requires both ontology engineers and domain experts.

They need to formalize knowledge about a domain of interest, with the goal of automating certain tasks.

The ontology engineer can formalize the domain knowledge, but does not have (enough) domain knowledge.

The domain expert has the domain knowledge, but does not know how to formalize it.

The ontology engineer needs to guide the expert in the creation of the ontology.

Roadmap to Create an Ontology

1. Find out what the goals of the application are.
2. Define the scope of the ontology.
3. Gather knowledge about the domain.
4. Define all entities of the ontology.
5. Define all axioms of the ontology.
6. Evaluate the ontology against the goals.

Competency Questions

"Questions that an ontology must be able to answer" (Grüninger, Fox, 1995)

- Informal description of the goals of the ontology
- First insights into the domain vocabulary
- The final ontology must answer the competency questions (evaluation).

Example competency questions:

What kinds of biological processes are there?

Which genes are involved in biological processes in the brain?

List all genes that are involved in three or more biological processes.

Which are the high-value customers?

Are basic kinship relations like parentage, grandparents, siblings represented?

Formalizing Competency Questions

To evaluate competency questions over the final ontology, they need to be formalized.

What are the subclasses of BiologicalProcess?

What are the instances of $\exists$annotatedWith.$\exists$annotationProcess.$\exists$situatedIn.Brain?

What are the instances of $(\geq 3\,$annotatedWith$.\top)$?

What are the instances of HighValueCustomer?

Does the ontology contain the relations hasParent, hasGrandparent, hasSibling?

This can only be done once the vocabulary of the ontology is determined.

Knowledge Acquisition

We (the ontology engineers) need to acquire knowledge from:

- domain experts:
  - know a lot of the domain (coverage)
  - are highly reliable (accuracy)
  - typically don't know much about ontology engineering
- documents:
  - cover parts of the domain
  - may be outdated
  - cannot answer questions
  - need to be interpreted
- databases:
  - large amount of knowledge about individuals
  - restricted knowledge about classes, relations, and axioms
  - easy to import automatically

Knowledge Elicitation: Extract relevant knowledge in dialog with the domain experts.

Knowledge Elicitation

"There are known knowns; there are things we know we know. We also know there are known unknowns; that is to say we know there are some things we do not know. But there are also unknown unknowns – the ones we don't know we don't know." (Donald Rumsfeld, 2002)

Problem: How to obtain all relevant knowledge from the domain expert, if we don't know in advance what that knowledge encompasses?

Simply asking the expert to write all knowledge down has several problems:

- They know too much.
- They know too little about the application goals and the constraints of OWL and DLs.
- Much of what they know is tacit (e.g., common assumptions that nobody talks about). The knowledge is there, but hard to access and to describe.
- Even their knowledge may be incomplete, although they are better at acquiring it.
- Their time is valuable.

Challenges of Knowledge Elicitation

Knowledge elicitation techniques must

- minimize the time required
- minimize the requirement for the domain expert to learn ontology engineering
- capture all essential knowledge
- capture tacit knowledge
- support multiple sources (building consensus between multiple experts)
- allow for the ontology engineers to learn the domain knowledge, so they can understand enough of it

Knowledge elicitation works mostly in the realm of natural language.

The formalization of this knowledge comes later.

Elicitation Techniques

Pre-representation:

- The starting point, before any ontology exists
- Focuses on generating protocols of interactions with the domain experts: recording of interviews, reports, non-interview observations, other documents
- Convert protocols into proto-formalizations

Post-representation:

- An initial formalization already exists
- Domain experts interact with it (guided by the ontology engineer)
- Interaction generates new questions, directions to expand

Proto-Formalization: Protocol Analysis

After we have obtained the protocols:

- Find the key terms
- Discard terms that are out of scope (irrelevant for the goals)
- Harmonize the terms (capitalization, pluralization, orthography, etc.)
- Distinguish significant terms
- Explore the terms to discover new ones

Example: Animals

Task: Generate a small ontology to describe the content of a children's book about animals.

Competency Questions:

- What are the types of animals?
- Where do the animals live?
- What do they eat?
- Are they dangerous?
- Are they big or small?
- Basic anatomy: legs, wings? feathers, fur?
- . . . whatever else may be in the book

Example: Terms Extracted from Protocols

Horse	Bear
Wild	Wolf
Trout	Goldfish
Farmed	Sheep
Grass	Shark
Cow	Herring
Dangerous	Human
Cat	Pet
Wheat	Dog
Carnivorous	Tree

Card Sorting

An informal procedure that identifies similarities between terms:

- Write down each concept on a card
- Organize them into piles
- Identify what the pile represents $\rightarrow$ new concept $\rightarrow$ new card
- Link the piles together
- Record the reasons for the sorting and the links
- Repeat

This works best in small groups, because everyone has a different idea how to sort the piles.

Example: Animals

Animal:

- Horse
- Cat
- Wolf
- Cow
- Bear
- Dog
- Sheep

Plant:

- Wheat
- Grass
- Tree

Fish:

- Herring
- Shark
- Trout
- Goldfish

Property:

- Pet
- Dangerous
- Carnivorous
- Wild
- Farmed

Links:

Fish are also animals.

Some animals eat plants.

Three Cards Trick

- Select 3 cards at random
- Identify which 2 cards are the most similar
- Write down why $\rightarrow$ new term?
- Write down why the 3rd is different $\rightarrow$ new term?

Again, doing this in a small group is better. Each person may have different ideas about the similarity.

Example: Animals

Shark Wolf Goldfish

New terms:

- Scales / Fur
- Fins / Feet
- Water / Land
- Swimming / Walking
- Carnivorous / Herbivorous
- Large / Small
- Gray / Gold

20 Questions

(like the game)

- The ontology engineer picks a concept.
- The domain expert tries to guess it, by asking a series of yes/no questions.

Different from the game, only the questions and their order are of interest.

It forces the domain experts to reveal the taxonomy of the domain.

Example: Animals

Is it a living thing? yes

Is it a plant? no

Is it an animal? yes

Is it a mammal? . . .

We now know that plants and animals are living things, they are possibly disjoint, and mammals are animals.

Outline

2.2 Formalization

Organize Terms

- Identify classes (nouns and adjectives) and relations (verbs)
- Organize terms into rough categories

General Animal Categories: Animal, Mammal

Specific Animals: Cat, Dog, Horse, Trout, Shark

Properties of Animals: Wild, Dangerous, Carnivorous, Pet

Plants: Tree, Grass, Wheat

Relations: eats, hasBodyPart, hasColor

Class Hierarchy

- Before adding complex axioms, first define the class hierarchy (**SubClassOf**/$\sqsubseteq$ axioms).
- Flesh out the hierarchy with common superconcepts, missing siblings.
- Ideally, much of this information was already elicited, otherwise we have to ask the domain experts again.

Example: Class Hierarchy

Living Thing

- Animal
  - Mammal
    - Cat
    - ...
  - Fish
    - Trout
    - ...
  - Carnivorous
  - Herbivorous
  - Omnivorous
- Plant
  - Tree
  - Grass
  - Wheat

Once the class hierarchy is fixed, we can add definitions.

Class Definitions

Identify which terms should be defined:

- Depends on the goals of the ontology.
- General terms like “Living Thing” probably don’t need a definition.
- Some terms are easier to define than others, e.g., “Cat” vs. “Carnivorous”.
- For some terms, the information about their place in the class hierarchy is enough.

Intensional definitions consist of the superclass(es) and any distinguishing characteristics.

> A cat is a mammal that has claws, 4 legs, and a tail.
>
> A carnivore is an animal that eats only meat.
>
> A pet is a domesticated animal that lives with humans.

Extensional definitions:

$$\mathsf{EU} \equiv \{\mathsf{UK}\} \sqcup \{\mathsf{France}\} \sqcup \{\mathsf{Germany}\} \sqcup \dots$$

Definitions (II)

Distinguish between full definitions ($\equiv$) and partial definitions ($\sqsubseteq$)!

$\mathsf{Animal} \equiv \mathsf{LivingThing} \sqcap \exists \mathsf{eats}.\mathsf{LivingThing}$

$\mathsf{Pet} \equiv \mathsf{Animal} \sqcap \exists \mathsf{livesWith}.\mathsf{Human}$

$\mathsf{Herbivorous} \equiv \mathsf{Animal} \sqcap \forall \mathsf{eats}.\mathsf{Plant}$

$\mathsf{Cow} \sqsubseteq \mathsf{Mammal} \sqcap \forall \mathsf{eats}.\mathsf{Grass}$

Class Hierarchy (II)

In general, the class hierarchy is not a simple tree, but a directed acyclic graph (there is multiple inheritance).

$\mathsf{Cow} \sqsubseteq \mathsf{Mammal} \qquad \mathsf{Cow} \sqsubseteq \mathsf{Herbivorous}$

(Mammal and Herbivorous are unrelated)

Instead of specifying all subclass-superclass relationships, it is easier to specify only a tree and let the reasoner infer the implicit ones.

$\mathsf{Grass} \sqsubseteq \mathsf{Plant}$

$\mathsf{Herbivorous} \equiv \mathsf{Animal} \sqcap \forall \mathsf{eats}.\mathsf{Plant}$

$\mathsf{Cow} \sqsubseteq \mathsf{Mammal} \sqcap \forall \mathsf{eats}.\mathsf{Grass}$

This entails $\mathsf{Cow} \sqsubseteq \mathsf{Herbivorous}$, so we do not have to explicitly add this axiom to the ontology.

Definitions can affect the (inferred) class hierarchy.

“Some” Does Not Mean “Only”

When writing definitions, it is not trivial to find the correct one.

A common modeling error is to swap $\forall$ and $\exists$:

$\mathsf{Grass} \sqsubseteq \mathsf{Plant}$

$\mathsf{Herbivorous} \equiv \mathsf{Animal} \sqcap \forall \mathsf{eats}.\mathsf{Plant}$

$\mathsf{Cow} \sqsubseteq \mathsf{Mammal} \sqcap \exists \mathsf{eats}.\mathsf{Grass}$

Cow is not subsumed by Herbivorous!

(A cow must eat “at least 1 Grass”, but could eat other things.)

“Only” Does Not Mean “Some”

$\mathsf{Cow} \sqsubseteq \forall \mathsf{eats}.\mathsf{Grass}$

Cow is not subsumed by $\exists \mathsf{eats}.\mathsf{Grass}$, not even $\exists \mathsf{eats}.\top$.

(A cow can eat only Grass, but does not have to eat anything.)

$\mathsf{Animal} \equiv \mathsf{LivingThing} \sqcap \exists \mathsf{eats}.\mathsf{LivingThing}$

$\mathsf{Mammal} \sqsubseteq \mathsf{Animal}$

$\mathsf{Cow} \sqsubseteq \mathsf{Mammal} \sqcap \forall \mathsf{eats}.\mathsf{Grass}$

entails $\mathsf{Cow} \sqsubseteq \exists \mathsf{eats}.\mathsf{Grass}$.

"And" Does Not Mean "Or"

"Cows eat grass and grain."

$\mathsf{Cow} \sqsubseteq \forall \mathsf{eats}.(\mathsf{Grass} \sqcap \mathsf{Grain}) \qquad \mathsf{Grass} \sqsubseteq \neg\mathsf{Grain}$

Cow and $\exists\mathsf{eats}.\top$ are disjoint!

(A cow can eat only things that are at the same time Grass and Grain, which do not exist.)

$\mathsf{Cow} \sqsubseteq \forall \mathsf{eats}.(\mathsf{Grass} \sqcup \mathsf{Grain}) \qquad \mathsf{Grass} \sqsubseteq \neg\mathsf{Grain}$

General Axioms and Annotations

- Declare disjoint classes!
- Declare domains and ranges, transitivity, . . . for object properties!
- Encode more specific knowledge, e.g., GCIs and complex role inclusions.
- Comment all entities and axioms, justify design choices (shared conceptualization)!

$\mathrm{Dom}(\mathsf{eats}) \sqsubseteq \mathsf{LivingThing}$

$\mathrm{Ran}(\mathsf{eats}) \sqsubseteq \mathsf{LivingThing}$

rdfs:comment "We ignore the difference between living things and parts of living things that are eaten (e.g., meat)."

Domain and Range Restrictions

Be careful of declared domains and ranges. They affect all class expressions using the property:

$\mathrm{Ran}(\mathsf{eats}) \sqsubseteq \mathsf{LivingThing} \qquad (\top \sqsubseteq \forall \mathsf{eats}.\mathsf{LivingThing})$

$\mathsf{Bird} \sqsubseteq \exists \mathsf{eats}.\mathsf{Stone}$

means that some stones are living things (those that are eaten by birds).

If Stone and LivingThing are disjoint, then the ontology is inconsistent.

$\mathrm{Dom}(\mathsf{eats}) \sqsubseteq \mathsf{LivingThing} \qquad (\exists \mathsf{eats}.\top \sqsubseteq \mathsf{LivingThing})$

$\mathsf{StoneEater} \sqsubseteq \exists \mathsf{eats}.\mathsf{Stone}$

entails $\mathsf{StoneEater} \sqsubseteq \mathsf{LivingThing}$.

If StoneEater and LivingThing are disjoint, then StoneEater is unsatisfiable.

Mereology

Mereology: The theory of parts, wholes, and their relations.

> Partonomies (part-of hierarchies), are as important as taxonomies (class hierarchies).
>
> A partonomy is modeled by a dedicated object property (**partOf**).

Generally accepted properties:

> Ref(**partOf**)   Tra(**partOf**)   Asy(**partOf**)

In other words, **partOf** is a partial order.

However, not all part-of relations are the same:

> "The tail is part of the cat."
>
> "The tree is part of the garden."
>
> "The wheat is part of the bread."

Properties of Part-of Relations

Part-of relations can be classified according to the following properties:

Functional: Are parts restricted by their function or placement?

Homeomeric: Are parts the same kinds of things as the whole?

Separable: Can the parts be removed from the whole?

Part-of Relation	Functional	Homeomeric	Separable
Component-Object	✓	–	✓
Material-Object	✓	–	–
Portion-Object	✓	✓	✓
Place-Area	✓	✓	–
Member-Collection	–	–	✓
Member-Partnership	–	–	–

Functional, Non-Homeomeric

Component-Object relation:

- separable
- "What are its parts?"

branch - tree

scene - movie

Material-Object relation:

- non-separable
- "What is it made of?"

wood - tree

flour - bread

Functional, Homeomeric

Portion-Object relation:

- separable
- not integral, but measurable parts; “some of”

slice - bread

meter - kilometer

Place-Area relation:

- non-separable
- usually between places and locations

garden - estate

Dresden - Germany

Non-Functional, Non-Homeomeric

Member-Collection relation:

- separable
- characterized by spatial/social/temporal proximity, not by similarity

tree - forest

Stefan Borgwardt - TU Dresden

Member-Partnership relation:

- non-separable
- members are defining parts of the whole

Ernie - Bert and Ernie

John - married couple

Several Part-of Relations

In an ontology, these different relations are given informative names to distinguish them, e.g., by referring to their domains.

$\mathsf{bodyPartOf} \qquad \mathsf{partOfRegion} \qquad \mathsf{memberOfUniversity}$

Generally, transitivity holds only along the same part-of relation.

$(\mathsf{Stefan\ Borgwardt}, \mathsf{TU\ Dresden}) : \mathsf{memberOfUniversity}$

$(\mathsf{TU\ Dresden}, \mathsf{Dresden}) : \mathsf{partOfRegion}$

. . . but sometimes also among different part-of relations.

$\mathsf{materialOf} \circ \mathsf{partOf} \sqsubseteq \mathsf{materialOf}$

Sometimes it is more convenient to use the inverse (has-part) relation.

$\mathsf{Car} \sqsubseteq \exists \mathsf{hasPart}.\mathsf{Engine} \qquad \text{vs.} \qquad \mathsf{Engine} \sqsubseteq \exists \mathsf{partOf}.\mathsf{Car}$

Not Part-of Relations

What is not a part-of relation:

Topological inclusion: “The wine is in the cellar.”

Class inclusion: “Frying is part of cooking.”

Attachment: “Fingers are part of the hand.” vs. “Earrings are part of the ear.”

Ownership: “A bicycle has wheels.” vs. “I have a bicycle.”

Direct Part-of

Often, it is more useful to refer to the direct parts only, instead of all (indirect) sub-parts.

$\mathsf{Piston} \sqsubseteq \exists \mathsf{directPartOf}.\mathsf{Engine} \qquad \mathsf{Engine} \sqsubseteq \exists \mathsf{directPartOf}.\mathsf{Car}$

directPartOf is not transitive!

$\mathsf{directPartOf} \sqsubseteq \mathsf{partOf} \qquad \mathsf{Tra}(\mathsf{partOf})$

This separation allows us to use directPartOf in number restrictions.

$\top \sqsubseteq \leq 1\, \mathsf{directPartOf}.\top \qquad \mathsf{Car} \sqsubseteq \leq 4\, \mathsf{hasDirectPart}.\mathsf{Wheel} \sqcap \dots$

This is not possible for partOf, since non-simple roles are not allowed in number restrictions!

When to Introduce a New Class?

If a class expression is used often, introduce a new named class:

$\mathsf{Shark} \sqsubseteq \mathsf{Fish} \sqcap \exists \mathsf{hasRisk}.(\exists \mathsf{hasSeverity}.\mathsf{Deadly})$

$\mathsf{Lion} \sqsubseteq \mathsf{Mammal} \sqcap \exists \mathsf{hasRisk}.(\exists \mathsf{hasSeverity}.\mathsf{Deadly})$

$\mathsf{Shark} \sqsubseteq \mathsf{Fish} \sqcap \mathsf{Dangerous}$

$\mathsf{Lion} \sqsubseteq \mathsf{Mammal} \sqcap \mathsf{Dangerous}$

$\mathsf{Dangerous} \equiv \exists \mathsf{hasRisk}.\mathsf{DeadlyRisk}$

$\mathsf{DeadlyRisk} \equiv \exists \mathsf{hasSeverity}.\mathsf{Deadly}$

This keeps the ontology more readable, but is not necessary for more specific characteristics of individuals:

$\mathsf{Scout} : \mathsf{Horse} \qquad \mathsf{Scout} : \mathsf{Agressive}$

There is no need for AgressiveHorse, unless this concept plays an important role in the ontology.

When To Stop Modeling?

It is easy to get lost trying to define all things in the smallest detail.

$$\text{Dangerous} \equiv \exists\text{hasRisk}.(\exists\text{hasSeverity}.(\text{Deadly} \sqcap \exists\text{hasCause}.\ldots))$$

For this to be meaningful, the new terms have to be used in the ontology.

For example, every animal should be assigned a risk level, cause, etc.

Making this consistent through a large ontology gets harder with every property that is added.

Decide whether the new class expression is necessary to achieve the goals (check competency questions).

Often a particular expressivity is targeted, e.g., one of the tractable OWL 2 Profiles.

Decide whether the new class expression is worth leaving this profile.

When to Use Classes vs. Individuals

Individuals are the most specific concepts represented in an ontology.

Daisy : **Cow**   **Felix** : **Cat**

If the application does not need to talk about specific animals, individuals may instead be breeds or even the animal species themselves.

Shetland Cattle : **Cow**   or   **Cow** : **Animal**

Individuals

Many ontologies do not have individuals, especially top-level ontologies.

Medical ontologies contain definitions of diseases, but no patient data.

However, many applications require reasoning about individuals.

$$\mathsf{Felix} : \mathsf{Cat} \quad (\mathsf{Felix}, \mathsf{Bob}) : \mathsf{livesWith} \quad \mathsf{Bob} : \mathsf{Human} \quad \models \mathsf{Felix} : \mathsf{Pet}$$

If the amount of individuals is much larger than the amount of classes and properties in the ontology, then we need automated techniques to

- import assertions from legacy data sources (e.g., databases).
- extract assertions automatically from text (lower quality).

Importing assertions from databases is often done by so-called mappings:

$$\mathsf{PetOwner}(x, y) \rightsquigarrow x : \mathsf{Human},\ (y, x) : \mathsf{livesWith}$$

where PetOwner is a table (predicate) in the database.

In general, this could be an arbitrary SQL (FO) query.

Database Mappings

There are two ways to use mappings:

1. Create a complete ABox by a one-time import from the database.
2. Use the mappings in addition to the ontology for reasoning.

The second option is often taken by OWL 2 QL/RL ontologies, where the mappings can be used to translate all reasoning tasks into SQL/Datalog queries over the existing database.

This is well-suited for databases that change often.

Such a translation to SQL is not possible in general for OWL 2 DL ontologies, since they are too expressive.

In such a case, we need to take Approach 1.

This means that either

- the ontology is used as the new primary data store, or
- the data is re-imported every time the database is updated.

Ontology Reuse

It is important to reuse as much of existing ontologies as possible.

- Identify relevant existing top-level/core/domain ontologies.
- Evaluate whether they are compatible with the goals.
- Import the ontologies.

Advantages:

- Can use existing axioms for reasoning
- Enforce some structure (**Process**, **TemporalRegion**, **Agent**, . . .)
- Interoperability with other ontologies

Drawbacks:

- Imports the whole ontology, which may be more than needed
- Cannot adapt existing axioms

Alternatives: import only the vocabulary (using prefixes), or record links to other ontologies in annotations (see Part 3: Ontology Integration)

Outline

2.3 Ontology Learning

Ontology Learning

So far, we have seen guidelines on how to construct an ontology by hand.

In Ontology Learning, one tries to (semi-)automatically construct an ontology from available resources.

Ontology Learning from Text:

- See lecture Semantic Computing given by Dr. Dagmar Gromann
- Extract assertions and basic taxonomies (**SubClassOf**)

Concept Learning in DLs:

- Tries to learn concept definitions from assertions

$$\mathsf{Bob} : \mathsf{Father} \qquad (\mathsf{Bob}, \mathsf{Fred}) : \mathsf{hasChild} \qquad \rightsquigarrow \qquad \mathsf{Father} \equiv \exists \mathsf{hasChild}.\top$$

- As with most learning methods, this needs enough data
- Danger of overfitting: If the data contains much more sons than daughters, we may end up with $\mathsf{Father} \equiv \exists \mathsf{hasChild}.\mathsf{Male}$.
- Humans still need to determine whether the definitions make sense

The Concept Learning Problem

Let $\mathcal{O}$ be a consistent ontology, $\mathbf{I}(\mathcal{O})$ be the set of individual names occurring in $\mathcal{O}$, and $A \in \mathbf{C}$ be the target concept name.

We denote the set of instances of A w.r.t. $\mathcal{O}$ by

$$\mathbf{I}_A(\mathcal{O}) := \{a \in \mathbf{I}(\mathcal{O}) \mid \mathcal{O} \models a : A\}.$$

The learning problem:

Given $\mathcal{O}$ and A, find a concept description $C_A \not\equiv_{\mathcal{O}} A$, such that

- $\mathcal{O} \models a : C_A$ for all $a \in \mathbf{I}_A(\mathcal{O})$
- $\mathcal{O} \not\models a : C_A$ for all $a \in \mathbf{I}_{\neg A}(\mathcal{O})$

(Lehmann, Hitzler, 2010)

- $\mathbf{I}_A(\mathcal{O})$ are the positive examples, $\mathbf{I}_{\neg A}(\mathcal{O})$ the negative examples for A in $\mathcal{O}$.
- Positive/negative examples may be explicitly contained in $\mathcal{O}$, e.g., $a : A, b : \neg A \in \mathcal{O}$, or entailed by other axioms.

Solving the Learning Problem

Given $\mathcal{O}$ and A, find a concept description $C_A \not\equiv_{\mathcal{O}} A$, such that

- $\mathcal{O} \models a : C_A$ for all $a \in \mathbf{I}_A(\mathcal{O})$
- $\mathcal{O} \not\models a : C_A$ for all $a \in \mathbf{I}_{\neg A}(\mathcal{O})$

To find C_A, we can restrict the search to the vocabulary of $\mathcal{O}$ ($\mathbf{I}(\mathcal{O}) \cup \mathbf{C}(\mathcal{O}) \cup \mathbf{R}(\mathcal{O})$), which is finite.

However, the number of candidates for C_A is still infinite.

An exact, but useless, solution is $\displaystyle\bigsqcup_{a \in \mathbf{I}_A(\mathcal{O})} \{a\}$.

To avoid overfitting (and simplify the search for C_A), we restrict the syntax of C_A.

Then, an exact solution may not exist, but we look for approximations.

Interlude: $\mathcal{ALC}$

We consider the description logic $\mathcal{ALC}$.

In $\mathcal{ALC}$,

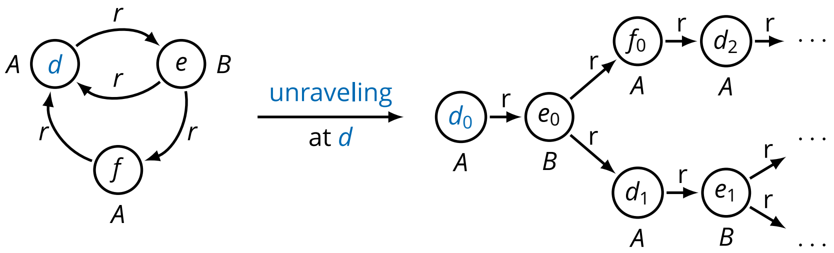
- only role names are allowed as roles (no inverse roles),
- concepts can be built only from concept names, $\top$, $\bot$, $\sqcap$, $\sqcup$, $\neg$, $\exists$, and $\forall$ (no data properties, self restrictions, number restrictions, or nominals),
- only concept axioms, concept and role assertions are allowed (no role axioms or individual (in)equalities).

Interlude: Tree Model Property in $\mathcal{ALC}$

$\mathcal{ALC}$ has the tree model property, i.e., every concept C that is satisfiable w.r.t. an $\mathcal{ALC}$ ontology $\mathcal{O}$ has a tree model $\mathcal{I}$:

- $\mathcal{I}$ is a model of $\mathcal{O}$,
- the directed graph $G_{\mathcal{I}} = (\Delta^{\mathcal{I}}, \bigcup_{r \in \mathbf{R}} r^{\mathcal{I}})$ is a tree, and
- the root of the tree belongs to $C^{\mathcal{I}}$.

Example: a model of $A \sqcap \neg B \sqcap \exists r.(B \sqcap \forall r.A)$



Interlude: Negation Normal Form in $\mathcal{ALC}$

An $\mathcal{ALC}$ concept is in negation normal form (NNF) if it contains negation ($\neg$) only directly in front of concept names.

Every $\mathcal{ALC}$ concept is equivalent (w.r.t. $\mathcal{O} = \emptyset$) to an $\mathcal{ALC}$ concept in NNF.

We can use the following normalization rules to transform subconcepts that are not in NNF:

$$\begin{aligned}
\neg\neg C &\rightsquigarrow C \\
\neg(C \sqcap D) &\rightsquigarrow \neg C \sqcup \neg D \\
\neg(C \sqcup D) &\rightsquigarrow \neg C \sqcap \neg D \\
\neg\exists r.C &\rightsquigarrow \forall r.\neg C \\
\neg\forall r.C &\rightsquigarrow \exists r.\neg C
\end{aligned}$$

Interlude: Concept Size in $\mathcal{ALC}$

The size of an $\mathcal{ALC}$ concept C is inductively defined as follows:

- $\text{size}(A) = 1$ for all $A \in \mathbf{C}$ (including $\top$ and $\bot$),
- $\text{size}(C \sqcap D) = \text{size}(C \sqcup D) = 1 + \text{size}(C) + \text{size}(D)$ for all concepts C, D,
- $\text{size}(\neg C) = \text{size}(\exists r.C) = \text{size}(\forall r.C) = 1 + \text{size}(C)$ for all $r \in \mathbf{R}$ and concepts C.

“the number of symbols it takes to write the concept”

$\text{size}\big(\exists r.(\exists s.A \sqcap \exists r.\exists s.\top)\big) = 7$

Interlude: Role Depth in $\mathcal{ALC}$

The role depth of an $\mathcal{ALC}$ concept C is inductively defined as follows:

- $\mathrm{rd}(A) = 0$ for all $A \in \mathbf{C}$ (including $\top$ and $\bot$),
- $\mathrm{rd}(\neg C) = \mathrm{rd}(C)$ for all concepts C,
- $\mathrm{rd}(C \sqcap D) = \mathrm{rd}(C \sqcup D) = \max\{\mathrm{rd}(C), \mathrm{rd}(D)\}$ for all concepts C, D,
- $\mathrm{rd}(\exists r.C) = \mathrm{rd}(\forall r.C) = 1 + \mathrm{rd}(C)$ for all $r \in \mathbf{R}$ and concepts C.

"the maximal nesting depth of role restrictions in the concept"

$\mathrm{rd}\big(\exists r.(\exists s.A \sqcap \exists r.\exists s.\top)\big) = 3$

Back to the Learning Problem

Given $\mathcal{O}$ and A, find a concept description $C_A \not\equiv_{\mathcal{O}} A$, such that

- $\mathcal{O} \models a : C_A$ for all $a \in \mathbf{I}_A(\mathcal{O})$
- $\mathcal{O} \not\models a : C_A$ for all $a \in \mathbf{I}_{\neg A}(\mathcal{O})$

Recall that we are interested in approximations.

Basic approach:

- Generate candidates for C_A, called hypotheses
- Evaluate and rank the hypotheses
- Let the ontology engineer choose among the best hypotheses
- Add the new concept definition $A \equiv C_A$ to $\mathcal{O}$

Evaluating Hypotheses

Before describing how to generate hypotheses, we consider criteria for their evaluation:

- Should have almost the same positive/negative instances as A
- Should be as short as possible, to be easier to understand (and also to reduce overfitting)

Let $\mathcal{O}$ be an ontology, $A \in \mathbf{C}$ and C_A be a concept.

$$\mathrm{fn}(C_A) := \#\{a \in \mathbf{I}_A(\mathcal{O}) \mid \mathcal{O} \not\models a : C_A\}$$ false negatives

$$\mathrm{fp}(C_A) := \#\{a \in \mathbf{I}_{\neg A}(\mathcal{O}) \mid \mathcal{O} \models a : C_A\}$$ false positives

$$\mathrm{acc}(C_A) := 1 - \frac{\mathrm{fn}(C_A) + \mathrm{fp}(C_A)}{\#\mathbf{I}_A(\mathcal{O}) + \#\mathbf{I}_{\neg A}(\mathcal{O})}$$ accuracy

$$\mathrm{score}(C_A) := \mathrm{acc}(C_A) - \beta \cdot \mathrm{size}(C_A)$$

How to Find Good Hypotheses?

Idea from Inductive Logic Programming (ILP):
Start with $C_A = \top$, and iteratively refine the concept.

A downward refinement operator ρ (w.r.t. $\mathcal{O}$) maps each concept C to a set of concepts $\rho(C) \subseteq \{D \mid D \sqsubseteq_{\mathcal{O}} C\}$.

We write $C \rightarrow_\rho D$ if $D \in \rho(C)$.

We denote with $\rightarrow_\rho^*$ the reflexive transitive closure of $\rightarrow_\rho$.

We say that D can be reached from C via $\rightarrow_\rho$ if $C \rightarrow_\rho^* D$.

$\top \rightarrow_\rho B \rightarrow_\rho B \sqcap \exists r.\top \rightarrow_\rho \ldots$

Properties of Refinement Operators

Problem: There are infinitely many concept descriptions.

How to traverse the search space efficiently towards a good hypothesis?

A downward refinement operator ρ is . . .

- (locally) finite if $\rho(C)$ is finite for all concepts C.
- proper if $C \to_\rho D$ implies $C \not\equiv_{\mathcal{O}} D$.
- complete if $C \sqsubset_{\mathcal{O}} D$ implies that $D \to_\rho^* E$ for some concept $E \equiv_{\mathcal{O}} C$.
- ideal if it is finite, proper, and complete.

Intuitively,

- only finitely many new hypotheses need to be evaluated at each step
- new hypotheses are more specific than previous ones
- all more specific concepts can be reached (modulo equivalence)

Are there ideal refinement operators?

Ideal Operators in $\mathcal{ALC}$

Does $\mathcal{ALC}$ have finite, proper, and complete refinement operators?

Lemma

Every $\mathcal{ALC}$ ontology $\mathcal{O}$ has a complete, finite refinement operator.

Proof: Define $\rho_1(C) := \{C \sqcap \top\} \cup \{D \mid \text{size}(D) \leq \text{size}(C) \text{ and } D \sqsubset_{\mathcal{O}} C\}$.

- Finite: There are only finitely many concepts up to a given size.
- Complete: To reach any $D \sqsubset_{\mathcal{O}} C$, we first increase the size of C:

$$C \to_{\rho_1} C \sqcap \top \to_{\rho_1} \dots \to_{\rho_1} C \sqcap \top \sqcap \dots \sqcap \top$$

until $\text{size}(C \sqcap \top \sqcap \dots \sqcap \top) \geq \text{size}(D)$, and then $C \sqcap \top \sqcap \dots \sqcap \top \to_{\rho_1} D$. $\square$

Lemma

Every $\mathcal{ALC}$ ontology $\mathcal{O}$ has a complete, proper refinement operator.

Proof: Define $\rho_2(C) := \{D \mid D \sqsubset_{\mathcal{O}} C\}$. $\square$

These operators are not ideal, and also not efficient.

No Ideal Operators in $\mathcal{ALC}$ (I)

Theorem

For $\mathcal{ALC}$, there is no ideal refinement operator w.r.t. $\mathcal{O} = \emptyset$.

Proof: Assume that ρ is an ideal refinement operator. Since ρ is finite and proper, $\rho(\top) = \{C_1, \ldots, C_m\}$ is finite and $C_i \sqsubset \top$ for all i, $1 \leq i \leq m$.

We construct a concept $C \notin \rho(\top)$ for which $C \sqsubset \top$, but there exists no C_i such that $C \sqsubseteq C_i \sqsubset \top$. This means that C cannot be reached from any element of $\rho(\top)$, which contradicts the assumption that ρ is complete.

Let $n := \max\{\mathrm{rd}(C_i) \mid 1 \leq i \leq m\} + 1$ and $\exists r^n$ be an abbreviation for n nested "$\exists r$" expressions. We set $C := \neg\exists r^n.\top \sqcup \exists r^{n+1}.\top$.

Because $\mathrm{rd}(C) = n + 1$, we have $C \notin \rho(\top)$. Moreover, $C \sqsubset \top$: For $\mathcal{I}$ with $\Delta^{\mathcal{I}} = \{d_0, \ldots, d_n\}$ and $r^{\mathcal{I}} = \{(d_i, d_{i+1}) \mid 0 \leq i < n\}$, we have $d_0 \notin C^{\mathcal{I}}$.

Claim

There exists no concept D with $C \sqsubseteq D \sqsubset \top$ and $\mathrm{rd}(D) < n$.

No Ideal Operators in $\mathcal{ALC}$ (II)

$$C := \neg\exists r^n.\top \sqcup \exists r^{n+1}.\top$$

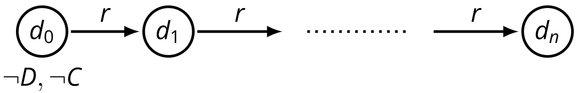
Claim

There exists no concept D with $C \sqsubseteq D \sqsubset \top$ and $\mathrm{rd}(D) < n$.

Proof: Assume that such a concept D exists. Since $D \sqsubset \top$, we know that $\neg D$ is satisfiable, i.e., there is a tree model $\mathcal{I}$ with root d_0 such that $d_0 \notin D^{\mathcal{I}}$.

Since $C \sqsubseteq D$, we know that $d_0 \notin C^{\mathcal{I}}$, i.e., $d_0 \in (\exists r^n.\top)^{\mathcal{I}}$ and $d_0 \notin (\exists r^{n+1}.\top)^{\mathcal{I}}$.

Thus, there is an r-path of length n in $\mathcal{I}$ starting from d_0, but no r-path of length $n+1$:

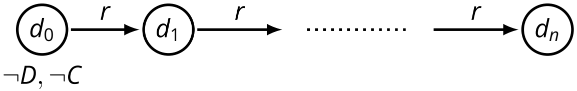


No Ideal Operators in $\mathcal{ALC}$ (III)

$$C := \neg\exists r^n.\top \sqcup \exists r^{n+1}.\top$$

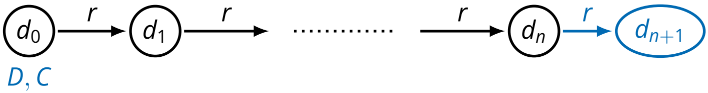
Claim

There exists no concept D with $C \sqsubseteq D \sqsubset \top$ and $\mathrm{rd}(D) < n$.



We create $\mathcal{I}'$ from $\mathcal{I}$ by adding d_{n+1} and setting $r^{\mathcal{I}'} := r^{\mathcal{I}} \cup \{(d_n, d_{n+1})\}$.

Now d_0 has an r-path of length $n + 1$, and thus satisfies C and D in $\mathcal{I}'$:



Since D has role depth $< n$, it can only refer to domain elements that are up to n steps away in the tree (Proof: Exercise). Since $\mathcal{I}$ and $\mathcal{I}'$ only differ in elements that are more than n steps away from d_0, the facts $d_0 \notin D^{\mathcal{I}}$ and $d_0 \in D^{\mathcal{I}'}$ contradict each other. $\square$

Refinement Operators in $\mathcal{ALC}$

We have shown:

Theorem

For $\mathcal{ALC}$, there is no ideal refinement operator w.r.t. $\mathcal{O} = \emptyset$.

In particular, any complete and proper operator needs to have infinitely many refinements of $\top$:

$$\{\exists r.\top, \neg\exists r.\top \sqcup \exists r^2.\top, \dots, \neg\exists r^n.\top \sqcup \exists r^{n+1}.\top, \dots\}$$

We assumed that **R** contains at least one role name.

But without role names, $\mathcal{ALC}$ is essentially propositional logic: the concept names $A \in$ **C** can be seen as propositional variables, $\sqcap$ as $\wedge$, $\sqcup$ as $\vee$, and $\sqsubseteq$ as $\rightarrow$. Without roles, different individuals cannot interact.

Since we want to learn $\mathcal{ALC}$ concepts and not propositional formulas, it is reasonable to assume that **R** is not empty.

Non-Ideal Operators for $\mathcal{ALC}$

Which of the three properties (finite, complete, proper) do we give up?

Idea: Instead of imposing finiteness, generate only hypotheses up to size n, and increase this limit only if there are no hypotheses that are good enough.

$$\rho_3^n(C) := \{D \mid D \sqsubset_{\mathcal{O}} C \text{ and } \mathrm{size}(D) \leq n\} \quad ?$$

We need a more practical operator that expands the concepts step-by-step, and takes into account the existing axioms in $\mathcal{O}$.

Next:

- We define the complete downward refinement operator ρ_c.
- We adapt it to a complete and proper operator.
- We discuss how to generate smaller hypotheses first.

(Lehmann, Hitzler, 2010)

Definitions

The relation $\sqsubseteq_{\mathcal{O}}$ can be seen as a partial order on $\mathbf{C}$. In the following, the terms “minimal” and “maximal” refer to this partial order.

The set $\downarrow(A)$ collects the lower neighbors of $A \in \mathbf{C}$ w.r.t. $\mathcal{O}$, i.e., the maximal concept names $A' \in \mathbf{C}$ with $\mathcal{O} \models A' \sqsubset A$.

The set $\uparrow(A)$ of upper neighbors is defined similarly.

Given $r \in \mathbf{R}$, the atomic domain $\mathsf{ADom}(r) \in \mathbf{C}$ is the unique minimal concept name such that $\mathcal{O} \models \mathsf{Dom}(r) \sqsubseteq \mathsf{ADom}(r)$.

The atomic range $\mathsf{ARan}(r)$ is defined similarly.

Note that $\mathsf{ADom}(r)$ and $\mathsf{ARan}(r)$ may be $\top$.

However, often the domain or range of a role is defined to be a single concept name, e.g., $\mathsf{Dom}(\textbf{hasParent}) \sqsubseteq_{\mathcal{O}} \textbf{Human}$, or $\mathsf{Ran}(\textbf{participatesIn}) \sqsubseteq_{\mathcal{O}} \textbf{BiologicalProcess}$.

Refinement with Context

If we want to refine the concept $\exists r.D$, we could replace D with a more specific concept E. However, E should not be disjoint with the atomic range of r, as this would yield an unsatisfiable concept:

$$\exists r.E \equiv_{\mathcal{O}} \exists r.(\mathrm{ARan}(r) \sqcap E) \equiv_{\mathcal{O}} \exists r.\bot \equiv_{\mathcal{O}} \bot$$

For this reason, we define a family of refinement operators ρ_B relative to a context $B \in \mathbf{C}$.

Intuitively, the context B is implicitly present in the concept D that is refined. That is, instead of D we are actually dealing with $B \sqcap D$.

For refining D in $\exists r.D$, the context is $\mathrm{ARan}(r)$.

Initially, however, the context is $\top$: We set $\rho_c(C) := \rho_\top(C) \cup \{\bot\}$.

Next, we define $\rho_B(C)$ by induction on the structure of C.

To simplify this, we ensure that all refined concepts are in NNF.

$\rho_B(\bot)$

The concept $\bot$ cannot be refined anymore, so we set $\rho_B(\bot) := \emptyset$.

$\rho_B(\top)$

For $\top$, there are several refinement options:

(a) concept names $A \in {\downarrow}(\top)$ that are not disjoint with B ($B \sqcap A \not\sqsubseteq_{\mathcal{O}} \bot$);

(b) $\neg A$, where $A \in {\uparrow}(\bot)$ and $\neg A$ is not disjoint with B;

(c) $\exists r.\top$ or $\forall r.\top$, where $\mathrm{ADom}(r)$ is not disjoint with B, or $\forall r.\bot$.

We collect the concepts from (a)–(c) into the set M_B.

Intuitively, these are the maximal concepts below $\top$ that are compatible with the context B and do not contain disjunctions.

We then consider all disjunctions of concepts from M_B:

$$\rho_B(\top) := \{\bot\} \cup \{C_1 \sqcup \cdots \sqcup C_n \mid C_1, \ldots, C_n \in M_B\}$$

- In $C_1 \sqcup \cdots \sqcup C_n$, elements of M_B may occur multiple times.
- $\bot$ can be seen as the empty disjunction.
- $\forall r.\top$ can be used in later refinement steps to generate concepts of the form $\forall r.C$.

$\rho_B(C)$

For all other concepts C, we set $\rho_B(C) := \rho'_B(C) \cup \{C \sqcap \top\}$, where

$$\begin{aligned}
\rho'_B(A) &:= \{A' \mid A' \in {\downarrow}(A),\ B \sqcap A' \not\sqsubseteq_{\mathcal{O}} \bot\} \\
\rho'_B(\neg A) &:= \{\neg A' \mid A' \in {\uparrow}(A),\ B \sqcap \neg A' \not\sqsubseteq_{\mathcal{O}} \bot\} \\
\rho'_B(\exists r.D) &:= \{\exists r.E \mid E \in \rho_{\mathsf{ARan}(r)}(D)\} \\
\rho'_B(\forall r.D) &:= \{\forall r.E \mid E \in \rho_{\mathsf{ARan}(r)}(D)\} \\
\rho'_B(C_1 \sqcap C_2) &:= \{C_1 \sqcap D_2 \mid D_2 \in \rho_B(C_2)\} \cup \{D_1 \sqcap C_2 \mid D_1 \in \rho_B(C_1)\} \\
\rho'_B(C_1 \sqcup C_2) &:= \{C_1 \sqcup D_2 \mid D_2 \in \rho_B(C_2)\} \cup \{D_1 \sqcup C_2 \mid D_1 \in \rho_B(C_1)\}
\end{aligned}$$

Lemma

ρ_c is a downward refinement operator.

Proof: Exercise

ρ_C is Complete

Theorem

ρ_C is complete.

Proof: Assume that $C \sqsubseteq_{\mathcal{O}} D$, which means that $D \sqcap C \equiv_{\mathcal{O}} C$.

It is enough to show that a concept equivalent to $D \sqcap C$ can be reached via $\rho_\top$ from D.

Since $D \sqcap \top \in \rho_\top(D)$, we only need to show that a concept equivalent to C can be reached via $\rho_\top$ from $\top$.

In the case that $C \equiv_{\mathcal{O}} \bot$, we know that $\bot \in \rho_\top(\top)$ by the definition of $\rho_\top$.

Claim (Weak Completeness)

For all concepts C and all $B \in \mathbf{C}$ with $B \sqcap C \not\sqsubseteq_{\mathcal{O}} \bot$, we have $\top \rightarrow^*_{\rho_B} E$ for some concept E with $B \sqcap E \equiv_{\mathcal{O}} B \sqcap C$.

Instantiating B with $\top$, we get exactly what we need: If $C \not\sqsubseteq_{\mathcal{O}} \bot$, then a concept equivalent to C can be reached via $\rho_\top$ from $\top$. □

Weak Completeness for Concept Names

Claim (Weak Completeness)

For all concepts C and all $B \in \mathbf{C}$ with $B \sqcap C \not\sqsubseteq_{\mathcal{O}} \bot$, we have $\top \rightarrow^*_{\rho_B} E$ for some concept E with $B \sqcap E \equiv_{\mathcal{O}} B \sqcap C$.

Proof: We prove the claim by induction on the structure of C. (Note that C can be neither $\top$ nor $\bot$.)

- **If C is a concept name**, then we can reach it using the operator $\downarrow(\cdot)$:

  $$\top = A_0 \sqsupset_{\mathcal{O}} A_1 \sqsupset_{\mathcal{O}} \cdots \sqsupset_{\mathcal{O}} A_n = C$$

  where $A_i \in \downarrow(A_{i-1})$ for all $i \in \{1, \dots, n\}$.
- No A_i can be disjoint with B, since then

  $$B \sqcap C \sqsubseteq_{\mathcal{O}} B \sqcap A_i \sqsubseteq_{\mathcal{O}} \bot.$$

  Thus, $\top \rightarrow_{\rho_B} A_1 \rightarrow_{\rho_B} A_2 \rightarrow_{\rho_B} \dots \rightarrow_{\rho_B} A_n = C$.
- The claim for **negated concept names** can be shown similarly.

Weak Completeness for Existential Restrictions

- If C is of the form $\exists r.D$, for the induction hypothesis we assume that, for all $B' \in \mathbf{C}$ with $B' \sqcap D \not\sqsubseteq_{\mathcal{O}} \bot$, we have $\top \rightarrow^*_{\rho_{B'}} E'$, where $B' \sqcap E' \equiv_{\mathcal{O}} B' \sqcap D$.
- $\mathsf{ADom}(r)$ cannot be disjoint with B, since otherwise
$$B \sqcap \exists r.D \sqsubseteq_{\mathcal{O}} B \sqcap \exists r.\top \sqsubseteq_{\mathcal{O}} B \sqcap \mathsf{ADom}(r) \sqsubseteq_{\mathcal{O}} \bot.$$
Thus, $\top \rightarrow_{\rho_B} \exists r.\top$.
- Similarly, $\mathsf{ARan}(r)$ cannot be disjoint with D, since otherwise
$$B \sqcap \exists r.D \sqsubseteq_{\mathcal{O}} \exists r.D \sqsubseteq_{\mathcal{O}} \exists r.(\mathsf{ARan}(r) \sqcap D) \sqsubseteq_{\mathcal{O}} \exists r.\bot \sqsubseteq_{\mathcal{O}} \bot.$$
- Choosing $B' := \mathsf{ARan}(r)$, the induction hypothesis yields that $\top \rightarrow^*_{\rho_{\mathsf{ARan}(r)}} E'$ for some E' with $\mathsf{ARan}(r) \sqcap E' \equiv_{\mathcal{O}} \mathsf{ARan}(r) \sqcap D$.
- By the definition of ρ_B, this means that $\exists r.\top \rightarrow^*_{\rho_B} \exists r.E'$.
- We have shown that $\top \rightarrow^*_{\rho_B} E$ for $E := \exists r.E'$, and
$B \sqcap E = B \sqcap \exists r.E' \equiv_{\mathcal{O}} B \sqcap \exists r.(B' \sqcap E') \equiv_{\mathcal{O}} B \sqcap \exists r.(B' \sqcap D) \equiv_{\mathcal{O}} B \sqcap \exists r.D = B \sqcap C.$
- The claim for **value restrictions** $\forall r.D$ can be shown similarly.

Weak Completeness for Conjunctions

- If C is of the form $C_1 \sqcap \cdots \sqcap C_n$, for the induction hypothesis we assume that, for all $B_i \in \mathbf{C}$ with $B_i \sqcap C_i \not\sqsubseteq_{\mathcal{O}} \bot$, we have $\top \to^*_{\rho_{B_i}} E_i$ for some E_i with $B_i \sqcap E_i \equiv_{\mathcal{O}} B_i \sqcap C_i$ (for all $i \in \{1, \dots, n\}$).
- B cannot be disjoint with any C_i, since otherwise
$$B \sqcap C \sqsubseteq_{\mathcal{O}} B \sqcap C_i \sqsubseteq_{\mathcal{O}} \bot.$$
- Choosing $B_i := B$, the induction hypothesis yields that $\top \to^*_{\rho_B} E_i$ for concepts E_i with $B \sqcap E_i \equiv_{\mathcal{O}} B \sqcap C_i$ (for all $i \in \{1, \dots, n\}$).
- By the definition of ρ_B, we get
$$\top \to^*_{\rho_B} E_1 \to_{\rho_B} E_1 \sqcap \top \to^*_{\rho_B} E_1 \sqcap E_2 \to_{\rho_B} \dots \to^*_{\rho_B} E_1 \sqcap \cdots \sqcap E_n,$$
where $E := E_1 \sqcap \cdots \sqcap E_n$, and
$$B \sqcap E \equiv_{\mathcal{O}} (B \sqcap E_1) \sqcap \cdots \sqcap (B \sqcap E_n) \equiv_{\mathcal{O}} (B \sqcap C_1) \sqcap \cdots \sqcap (B \sqcap C_n) \equiv_{\mathcal{O}} B \sqcap C.$$

Weak Completeness for Disjunctions

- If C is of the form $C_1 \sqcup \cdots \sqcup C_n$, for the induction hypothesis we assume that, for all $B_i \in \mathbf{C}$ with $B_i \sqcap C_i \not\sqsubseteq_{\mathcal{O}} \bot$, we have $\top \to^*_{\rho_{B_i}} E_i$ for some E_i with $B_i \sqcap E_i \equiv_{\mathcal{O}} B_i \sqcap C_i$ (for all $i \in \{1, \dots, n\}$).
- We discard all C_i that are disjoint with B. We assume that there exists $m \in \{1, \dots, n\}$ such that exactly $C_1, \dots, C_m$ are not disjoint with B.
- For all $i \in \{1, \dots, m\}$, we have $\top \to^*_{\rho_B} E_i$ for some E_i with $B \sqcap E_i \equiv_{\mathcal{O}} B \sqcap C_i$.
- $$\begin{aligned} B \sqcap (E_1 \sqcup \cdots \sqcup E_m) &\equiv_{\mathcal{O}} (B \sqcap E_1) \sqcup \cdots \sqcup (B \sqcap E_m) \sqcup \bot \sqcup \cdots \sqcup \bot \\ &\equiv_{\mathcal{O}} (B \sqcap C_1) \sqcup \cdots \sqcup (B \sqcap C_m) \sqcup (B \sqcap C_{m+1}) \sqcup \cdots \sqcup (B \sqcap C_n) \\ &\equiv_{\mathcal{O}} B \sqcap (C_1 \sqcup \cdots \sqcup C_n) \end{aligned}$$
- If there is an $i \in \{1, \dots, m\}$ for which $E_i = \top$, then $B \sqcap \top \equiv_{\mathcal{O}} B \sqcap C$, and thus we can choose $E := \top$.
- Otherwise, there exist E_i' such that $\top \to_{\rho_B} E_i' \to^*_{\rho_B} E_i$ (for all $i \in \{1, \dots, m\}$).
- By the definition of ρ_B, we get

$$\top \to_{\rho_B} E_1' \sqcup \cdots \sqcup E_m' \to^*_{\rho_B} E_1 \sqcup \cdots \sqcup E_m.$$

ρ_c is Complete, But Not Proper

We have shown:

Claim (Weak Completeness)

For all concepts C and all $B \in \mathbf{C}$ with $B \sqcap C \not\sqsubseteq_{\mathcal{O}} \bot$, we have $\top \to^*_{\rho_B} E$ for some concept E with $B \sqcap E \equiv_{\mathcal{O}} B \sqcap C$.

Theorem

ρ_c is complete.

But ρ_c is obviously not proper. For example, $\rho_c(\exists r.D)$ contains $\exists r.D \sqcap \top$, which is equivalent to $\exists r.D$.

A Complete and Proper Operator

The operator $\rho_c^{\downarrow}$ is defined as follows: We have $D \in \rho_c^{\downarrow}(C)$ iff

$$C \to_{\rho_c} C_1 \to_{\rho_c} \ldots \to_{\rho_c} C_n \to_{\rho_c} D,$$

where $C \equiv_{\mathcal{O}} C_1 \equiv_{\mathcal{O}} \cdots \equiv_{\mathcal{O}} C_n$ and $C \not\equiv_{\mathcal{O}} D$.

Theorem

$\rho_c^{\downarrow}$ is a complete and proper downward refinement operator.

Since $\rho_c^{\downarrow}$ is not finite, we want to compute all refinements up to some size n, and increase this number if necessary.

Can all D with $C \to_{\rho_c^{\downarrow}} D$ and $\mathrm{size}(D) \leq n$ be computed in finite time?

Size Restriction (I)

Lemma

For all $D \in \rho_c(C)$, we have $\mathrm{size}(D) \geq \mathrm{size}(C)$.

The length of chains $C_1 \to_{\rho_c} \dots \to_{\rho_c} C_n$ with $\mathrm{size}(C_1) = \dots = \mathrm{size}(C_n)$ is bounded polynomially in the size of C_1 and $\mathcal{O}$.

Proof: Most refinement steps increase the size of the concept. The only exception are those that replace a concept name (including $\top$) by another concept name. But the size is never decreased.

A (negated) concept name A is refined by replacing it with a lower (upper) neighbor in the concept hierarchy. For each A, this can be done at most $\mathrm{size}(\mathcal{O})$ times.

There are at most $\mathrm{size}(C_1)$ concept names in C_1.

After at most $\mathrm{size}(C_1) \cdot \mathrm{size}(\mathcal{O})$ refinement steps with ρ_c, the size of the concept must be increased. $\square$

Size Restriction (II)

Lemma

For all concepts C in NNF and all $n \in \mathbb{N}$, the set $\{D \in \rho_c^{\downarrow}(C) \mid \mathrm{size}(D) \leq n\}$ can be computed in finite time.

Proof: There are only finitely many ρ_c-refinements of C up to length n, and each of them can be reached by polynomially many refinement steps via ρ_c.

In $\rho_c^{\downarrow}$, we only skip some steps that result in equivalent concepts. $\square$

The Search Algorithm

We are looking for good hypotheses for the learning problem.

The search tree contains nodes of the form (C, n), where C is the current hypothesis and n the bound on the size.

Algorithm (Concept Learning in DL-Learner (Lehmann, Hitzler, 2010))

Input: Ontology $\mathcal{O}$, concept name $A \in \mathbf{C}$, parameters β, γ

Output: A list of candidates for C_A, ranked by their score

- Start with the single node $(\top, 0)$
- While there is a node (C, n) with $\mathrm{fn}(C) < \gamma \cdot \#\mathbf{I}_A(\mathcal{O})$:
  - Choose a node (C, n) with maximal score
  - Add all nodes $(D, n+1)$ with $D \in \rho_{\mathcal{C}}^{\downarrow}(C)$ and $\mathrm{size}(D) = n+1$ as children of (C, n)
  - Replace (C, n) by $(C, n+1)$
- Stop the algorithm at any time, and return all computed concepts ranked by their score

Optimizations

Avoid redundancy, e.g.,

$\top \rightarrow_{\rho_c} A \rightarrow_{\rho_c} A \sqcap \top \rightarrow_{\rho_c} A \sqcap B$ and $\top \rightarrow_{\rho_c} B \rightarrow_{\rho_c} B \sqcap \top \rightarrow_{\rho_c} B \sqcap A$

$A \sqcap B \rightarrow_{\rho_c} A \sqcap B \sqcap \top \rightarrow_{\rho_c} A \sqcap B \sqcap A$

$\top \rightarrow_{\rho_c} \exists r.\top \sqcup \exists r.\top \rightarrow^{*}_{\rho_c} \exists r.C \sqcup \exists r.D$ and $\top \rightarrow_{\rho_c} \exists r.\top \rightarrow^{*}_{\rho_c} \exists r.(C \sqcup D)$

Restrict the allowed constructors ($\sqcap$, $\sqcup$, $\neg$, $\exists$, $\forall$) to decrease the search space

In $\mathcal{EL}$ (the sublogic of $\mathcal{ALC}$ restricted to $\sqcap$ and $\exists$), every ontology has an ideal downward refinement operator.