Knowledge Graphs

Knowledge graphs are used in many places:

- Wikidata and DBPedia reflect knowledge of Wikipedia
- Search Engines: Google, Bing, Yahoo, etc.
- Question answering systems: WolframAlpha, Siri, Alexa
- Social Networks: Facebook, LinkedIn

Having explicit representations of knowledge has also advantages for AI systems

Knowledge graphs on their own are powerful, but they also have limitations:

- we only have the explicit knowledge
- graphs may be incomplete
- it is not straightforward to link different knowledge sources

Ontology as a Discipline of Philosophy

Ontology *philosophy* the study of the nature of being [Greek *on* being + logy]

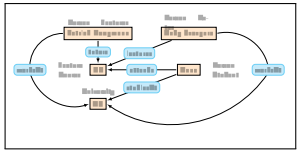
Branch of metaphysics

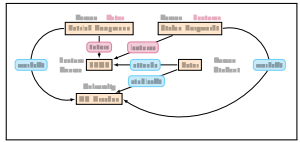
- What kinds of things/entities exist?
- What does it mean to exist?
- How are entities related?
- What are their basic categories? (eg. substances, property, relation, event,)
- Which entities are the most fundamental?
- "Ontology" is concerned with rather general and fundamental concepts.

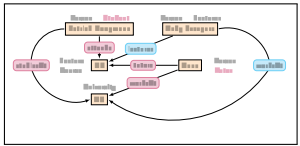
Conceptualizations (after Gruber 1994)

A conceptualization is given by:

- a domain of discourse — not fixed
- a set of conceptual relations — concepts + roles
- for different states of the world, and different possible worlds — interpretations







• • •

Formal, Explicit Specifications

~~Extensional specification:~~

- ~~explicitly state the elements of the conceptual relations~~

Intensional specification:

- constrain concepts and roles in the different possible worlds
- *axiomatize*
- formal languages make these specifications *precise*
- dedicated languages make them human readable

$$\begin{aligned} &\mathit{Lecture} \sqsubseteq \mathit{Course} \\ &\mathit{TA} \equiv \exists \mathit{teaches}.\mathit{Course} \\ &\mathit{TA} \sqsubseteq \exists \mathit{worksAt}.\mathit{University} \\ &\vdots \end{aligned}$$

Formal, Explicit Specifications

> "An ontology is a formal, explicit specification of a shared conceptualization." (Gruber 1994; Staab, Studer, 2009)

Ontologies allow to share a fixed conceptualization with different parties

- AI system and user
- different user groups/companies
- different systems (communication via the web)

Ontologies in Practice

- Many knowledge graphs are used without or with very simple ontologies
  - in Wikidata, a lot of relevant knowledge is already there
  - terms are often not used coherently enough to allow for logical reasoning
- Existing ontologies vary a lot in size and expressivity
  - some ontologies are just taxonomies
  - others allow for complex logical inferencing
- In the context of this course, we are interested in the more expressive ontology formalisms

Ontology Languages

There are very different ontology languages used in practice:

- Higher order logic (SUMO)
  - relatively rare
  - limited practicality
- RDFS
  - "**R**ich **D**ata **F**ormat **S**chema"
  - very common for knowledge graphs in RDF
  - limited expressivity
  - complicated semantics
  - covered in Masters course "Knowledge Representation in the Web"
- Datalog
  - Rule-based language
  - Common systems are RDFOx, VLog and Nemo
- OWL
  - "**W**eb **O**ntology **L**anguage"
  - Most expressive decidable formalism in this list
  - Based on description logics

Description Logics

Description logics (DLs)

- are decidable fragments of first-order logic
- are restricted to unary and binary predicates
- use a special syntax for formulas
- have the specification of ontologies as main use case

Vocabulary

Formally the syntax of DLs is based on the following infinite, disjoint sets:

concept names	role names	individual names
$\mathbf{C} = \{A, B, \dots\}$	$\mathbf{R} = \{r, s, \dots\}$	$\mathbf{I} = \{a, b, \dots\}$

- Together, they are called the vocabulary.
- Concept names A describe sets.

  The concept name Person denotes the set of all persons.

- Role names describe binary relations.

  belongsTo denotes the set of all pairs (x, y) where x belongs to y.

- Individual names describe individual objects.

  vrijeUniversiteit denotes this university.

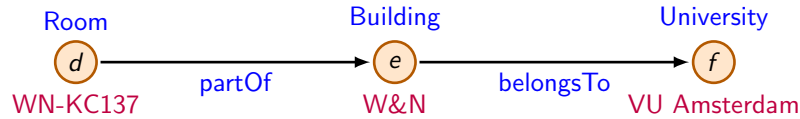
Interpretations

The semantics of DLs is based on (first-order) interpretations.

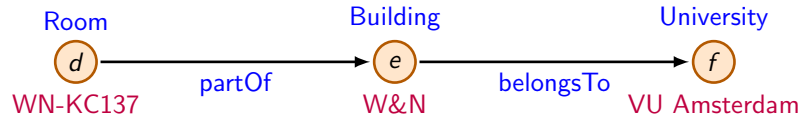
A DL interpretation is a tuple $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where

- $\Delta^{\mathcal{I}}$ is a non-empty set, called the domain of $\mathcal{I}$,
- $\cdot^{\mathcal{I}}$ is the interpretation function that assigns meanings to names:
  - each $A \in \mathbf{C}$ is interpreted as a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$,
  - each $r \in \mathbf{R}$ is interpreted as a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$,
  - each $a \in \mathbf{I}$ is interpreted as an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$.

Interpretations can be represented as labeled graphs:



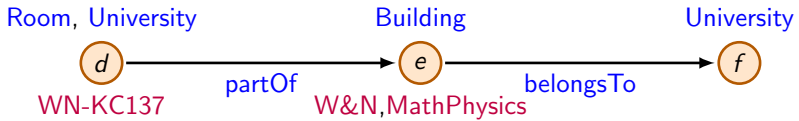
Important Facts about Interpretations



- The domain $\Delta^{\mathcal{I}}$ can be infinite.
- Elements of $\Delta^{\mathcal{I}}$ are called (domain) elements, individuals, or objects.
- $\mathit{Room}^{\mathcal{I}} = \{d\}$ is called the extension of Room
- The individual e is different from the individual name $\mathit{W\&N}$.
- f is called belongsTo-successor of e.
- e is called belongsTo-predecessor of f.

Example: Another Interpretation

Interpretations can be completely arbitrary:



- Different individual names can be interpreted as the same individual, e.g. *W&N* and *Math&Physics* are interpreted as e.
- There can be unnamed individuals, e.g. f, which do not have a corresponding individual name.
- Rooms can be universities, which intuitively does not make sense.

- The axioms in an ontology restrict the set of interpretations, e.g. by stating that rooms cannot be lectures.

Concepts

Before we introduce axioms, we introduce complex concepts

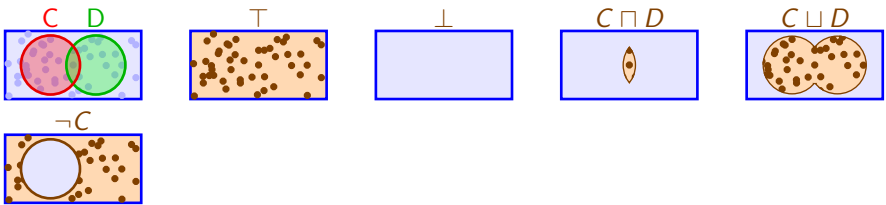
- also: *concept descriptions*, *compound concepts*, or just *concepts*
- describe sets of objects in an interpretation
- central elements in ontologies and ontology axioms
- the semantics of concepts is captured by the interpretation function:
  - $\cdot^{\mathcal{I}}$ is extended to map concepts C to subsets $C^{\mathcal{I}}$ of the domain $\Delta^{\mathcal{I}}$

Concepts

We define $\mathcal{ALC}$ concepts C and their semantics $C^{\mathcal{I}}$ inductively.

- every concept name A is a concept with semantics $A^{\mathcal{I}}$
- if C, D are concepts, then the following are also concepts:

Name:	top	bottom	conjunction	disjunction	complement
Syntax:	$\top$	$\bot$	$C \sqcap D$	$C \sqcup D$	$\neg C$
Semantics:	$\Delta^{\mathcal{I}}$	$\emptyset$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$

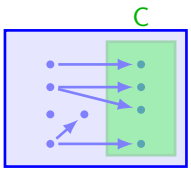


Concepts using Roles

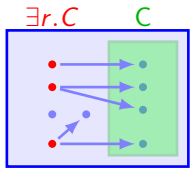
Additionally, we can describe outgoing role connections by specifying the concepts of the role successors.

- If C is a concept and r is a role, then the following are also concepts:

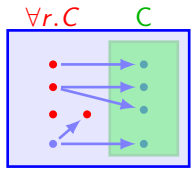
Name:	existential restriction	value restriction
Syntax:	$\exists r.C$	$\forall r.C$
Semantics:	$\{d \mid \exists e.(d,e) \in r^{\mathcal{I}} \wedge e \in C^{\mathcal{I}}\}$	$\{d \mid \forall e.(d,e) \in r^{\mathcal{I}} \rightarrow e \in C^{\mathcal{I}}\}$



Some r-successor in C: $\exists r.C$



All r-successors in C: $\forall r.C$

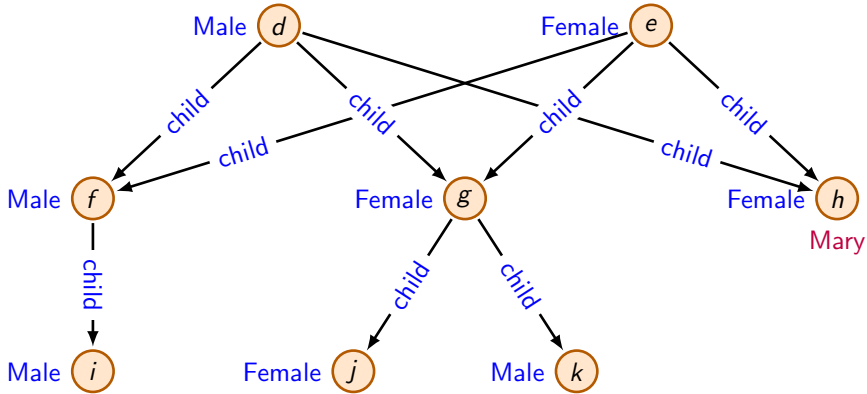


Syntax of $\mathcal{ALC}$ Concepts

$\mathcal{ALC}$ is the description logic that allows the concept constructors $\top$, $\bot$, $\sqcap$, $\sqcup$, $\neg$, $\exists$, and $\forall$ to build concepts.

Other DLs may use different constructors.

Example: Interpreting Concepts



$(\exists child.\top)^{\mathcal{I}} = \{d, e, f, g\}$

$(Female \sqcap \exists child.\top)^{\mathcal{I}} = \{e, g\}$

$(\neg\exists child.\top)^{\mathcal{I}} = \{h, i, j, k\}$

$(\exists child.Female)^{\mathcal{I}} = \{d, e, g\}$

$(\forall child.Female)^{\mathcal{I}} = \{h, i, j, k\}$

$(\exists child.\bot)^{\mathcal{I}} = \emptyset$

From Concepts to Ontologies

- Interpretations are arbitrary
  - We usually do not know which interpretation is the correct one
- Only some will be models of our conceptualization
- To specify what is a model, we use axioms
- Axioms
  - express constraints on interpretations
  - relate concepts to other concepts
  - relate concepts to individuals
- An ontology is then a collection of axioms.

From Concepts to Ontologies

- DL ontologies are sets of axioms
- Axioms put constraints on interpretations
- We distinguish two types:
  - terminological axioms put concepts in relation
  - assertions relate individual names with concepts and roles
- They respectively form the TBox and the ABox of an ontology
- An interpretation satisfying such an axiom/ontology is a *model*
  - $\mathcal{I}$ *satisfies* α: $\mathcal{I} \models \alpha$

Terminological Axioms

If C and D are concepts, then the following is a (terminological) axiom:

Name:	general concept inclusion (GCI)	equivalence axiom
Syntax:	$C \sqsubseteq D$	$C \equiv D$
Semantics:	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$	$C^{\mathcal{I}} \equiv D^{\mathcal{I}}$

$\mathit{Lecture} \sqsubseteq \mathit{Course} \qquad \mathit{Room} \sqsubseteq \neg \mathit{Lecture}$

$\mathit{Room} \sqsubseteq \mathit{Structure} \sqcap \exists \mathit{partOf}.\mathit{Building}$

$\mathit{TA} \equiv \mathit{Person} \sqcap \exists \mathit{teaches}.\mathit{Course}$

$\mathit{Lecturer} \equiv \mathit{Person} \sqcap \exists \mathit{teaches}.\mathit{Lecture}$

Assertions

Assertions (also called facts) are axioms about named individuals.

Given $a, b \in \mathbf{I}$, a concept C, and $r \in \mathbf{R}$, the following are assertions:

Name:	concept assertion	role assertion
Syntax:	$a : C$	$(a, b) : r$
Semantics:	$a^{\mathcal{I}} \in C^{\mathcal{I}}$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$

$WN\text{-}KC137 : Room \qquad (W\&N, VUAmsterdam) : belongsTo$

Ontologies

An ontology is a set $\mathcal{O} = \mathcal{A} \cup \mathcal{T}$, where

- $\mathcal{A}$ is an ABox, a finite set of assertions,
- $\mathcal{T}$ is a TBox, a finite set of GCIs and equivalence axioms,

An interpretation is a model of $\mathcal{O}$ (written $\mathcal{I} \models \mathcal{O}$) if it is a model of all axioms in $\mathcal{O}$.

- The ABox contains facts about named individuals (data), the TBox contains (terminological) knowledge that applies to all individuals.

Entailment is a quite general reasoning task.

With ontologies, we usually have more specific questions we want to answer.

Typical reasoning tasks are the following:

- Subsumption-relationships between concepts
- Individuals in a given concept
- Consistency and coherence of an ontology

Reasoning: Relationships between Concepts

Let C, D be concepts and $a \in \mathbf{I}$.

- If $\mathcal{O} \models C \sqsubseteq D$, we say that C is subsumed by D wrt. $\mathcal{O}$. $C \sqsubseteq_{\mathcal{O}} D$
- If $\mathcal{O} \models C \equiv D$, we say that C is equivalent to D wrt. $\mathcal{O}$. $C \equiv_{\mathcal{O}} D$
- If $C \sqsubseteq_{\mathcal{O}} D$ and $C \not\equiv_{\mathcal{O}} D$, C is strictly subsumed by D wrt. $\mathcal{O}$. $C \sqsubset_{\mathcal{O}} D$
- If $\mathcal{O} \models C \sqcap D \sqsubseteq \bot$, we say that C and D are disjoint wrt. $\mathcal{O}$.

$\{\mathit{AILecture} \sqsubseteq \mathit{Lecture},\ \mathit{Lecture} \sqsubseteq \mathit{Course}\} \models \mathit{AILecture} \sqsubseteq \mathit{Course}$

$\{\mathit{AILecture} \sqsubseteq \mathit{Lecture},\ \mathit{Lecture} \sqcap \mathit{Room} \sqsubseteq \bot\} \models \mathit{AILecture} \sqcap \mathit{Room} \sqsubseteq \bot$

If $C \sqsubset D$, then C is more specific than D (w.r.t. $\mathcal{O}$), and D is more general than C (w.r.t. $\mathcal{O}$).

Reasoning: Consistency

$\mathcal{O}$ is consistent if it has a model.

- An inconsistent ontology contains contradictory statements.
- An inconsistent ontology entails all axioms, even if they are nonsensical.

Theorem: If $\mathcal{O}$ is inconsistent, then $\mathcal{O} \models \alpha$ for any axiom α.

Theorem: An ontology $\mathcal{O}$ is inconsistent iff $\mathcal{O} \models \top \sqsubseteq \bot$.

This means that checking consistency is a kind of (non-)entailment test.

Reasoning: Consistency

Theorem: If $\mathcal{O}$ is inconsistent, then $\mathcal{O} \models \alpha$ for any axiom α.

Proof: Assume $\mathcal{O}$ is inconsistent and α is an arbitrary axiom. $\mathcal{O}$ has no model. Therefore, there is no model $\mathcal{I}$ s.t. $\mathcal{I} \not\models \alpha$. This means $\mathcal{I} \models \alpha$ in every model of $\mathcal{O}$. □

Theorem: An ontology $\mathcal{O}$ is inconsistent iff $\mathcal{O} \models \top \sqsubseteq \bot$.

Proof: ($\Rightarrow$) By the previous theorem, if $\mathcal{O}$ is inconsistent, $\mathcal{O} \models \top \sqsubseteq \bot$.

($\Leftarrow$) Assume $\mathcal{O} \models \top \sqsubseteq \bot$. Then, for every model $\mathcal{I}$ of $\mathcal{O}$, $\mathcal{I} \models \top \sqsubseteq \bot$. This means, $\Delta^{\mathcal{I}} \subseteq \emptyset$. By definition, $\Delta^{\mathcal{I}}$ is never empty in an interpretation. Therefore, there cannot be any model $\mathcal{I}$ of $\mathcal{O}$, and $\mathcal{O}$ must be inconsistent. □

Other Reasoning Problems

- If $\mathcal{O} \models a : C$, then a is an instance of C w.r.t. $\mathcal{O}$.
- If $\mathcal{O} \not\models C \sqsubseteq \bot$, then C is satisfiable w.r.t. $\mathcal{O}$.
- If all concept names in $\mathcal{O}$ are satisfiable w.r.t. $\mathcal{O}$, then $\mathcal{O}$ is coherent.
- Classification is the task of computing all entailments of the form $\mathcal{O} \models A \sqsubseteq B$, where $A, B \in \mathbf{C}$.
- Materialization is the task of computing all entailments of the form $\mathcal{O} \models a : A$ and $\mathcal{O} \models (a, b) : r$, where $a, b \in \mathbf{I}$, $A \in \mathbf{C}$, and $r \in \mathbf{R}$.

- Classification and materialization make explicit much of the knowledge that is implicitly given by the ontology.

Example: Consistency and Coherence

$$\{\mathit{Felix} : \mathit{Cat},\quad \mathit{Cat} \sqsubseteq \mathit{Animal},\quad (\mathit{Felix}, \mathit{Toby}) : \mathit{hasFather},\quad \exists \mathit{hasFather}.\top \sqsubseteq \mathit{Human},\quad \mathit{Human} \sqcap \mathit{Animal} \sqsubseteq \bot\}$$

$$\{\mathit{Human} \sqcap \mathit{Animal} \sqsubseteq \bot,\quad \mathit{Werewolf} \sqsubseteq \mathit{Human} \sqcap \mathit{Wolf},\quad \mathit{Wolf} \sqsubseteq \mathit{Animal}\}$$

Disjointness axioms ($C \sqsubseteq \neg D$, or equivalently $C \sqcap D \sqsubseteq \bot$) are very useful for debugging ontologies, because they can expose hidden contradictions.

Open World and Open Domain

DLs make the open-world assumption, i.e. facts that are not entailed are not necessarily false, but simply unknown.

$\{(Bob, Fred) : hasChild\}$ does not entail $Bob : Father$ or $Bob : \neg Father$.

They also make the open-domain assumption, i.e. there may be individuals we do not know, and that have no name.

$\mathcal{O} = \{ Human \sqsubseteq \exists hasParent.Mother,\ peter : Human\}$

$\models peter : \exists hasParent.Mother$

$\not\models a : Mother$ for any $a \in \mathbf{I}$

Binary vs. n-ary Relations

Description logics can only express unary and binary relations.

n-ary relations with $n \geq 3$ can be simulated, but not without loss of generality.

$hasDiagnosis(Bob, Flu, High)$

could be reformulated as

$Diagnosis(d112)$, $hasDiagnosis(Bob, d112)$,
$associatedDisease(d112, Flu)$, $associatedProbability(d112, High)$

This process is called reification.

This can lead to more complex axioms, because one has to refer to several roles.

$\exists hasDiagnosis.\exists associatedDisease.InfectiousDisease \sqsubseteq Infectious$

Flashback: Reasoning

Reasoning allows us to discover new insights from the knowledge represented in the ontology.

The central reasoning task is entailment:

> $\mathcal{O}$ entails an axiom α ($\mathcal{O} \models \alpha$) if every model of $\mathcal{O}$ is also a model of α.

- We need to consider what all models have in common.
- This is the same as in propositional and first-order logic.

Deciding $\mathcal{O} \models \alpha$

Reasoning looks harder than in propositional logic:

- In propositional logic, we only have a finite number of interpretations to consider
  - interpretations are truth valuations
  - we can go through the interpretations one by one and check
- In description logics, there are infinitely many interpretations!
  - interpretations involve different domain elements
  - their number is arbitrary (up infinitely many)
- But we are better off than in first-order logic
  - entailment in description logics is decidable
  - entailment for first-order logic is only semi-decidable

No Equivalence Axioms

To keep the following simpler, we assume that our TBoxes contain no equivalence axioms.

If the TBox contains equivalence axioms $C \equiv D$, we can replace each such axiom by the two axioms $C \sqsubseteq D$ and $D \sqsubseteq C$.

Reasoning in Description Logics

Reasoning with $\mathcal{ALC}$ is truely harder than for propositional logic

- we have to consider different options for different elements
- it may require exponential time in the size of the ontology
- differently to propositional logic ($P = NP$ question), this is certain from a theoretically perspective

Before we look at $\mathcal{ALC}$, we look at an easier description logic

A practical fragment of $\mathcal{ALC}$

- $\mathcal{ALC}$ is just one example of a DL
- Different DLs differ in
  - What concept operators they allow
  - What axiom types they allow
- A very important DL is $\mathcal{EL}$, which is a fragment of $\mathcal{ALC}$
  - $\mathcal{EL}$ only allows the concept operators $\top$, $\sqcap$ and $\exists$ $\forall$
  - $\mathcal{EL}$ does allow all axiom types we have seen so far.
- $\mathcal{EL}$ is used predominantly in many large ontologies
  - Very often, most axioms in an ontology are $\mathcal{EL}$ axioms
  - A lot of ontologies are pure $\mathcal{EL}$ ontologies (or in friendly extensions of $\mathcal{EL}$)
  - SNOMED CT, the large medical ontology mentioned in Lecture 3, is one such example

Reasoning in $\mathcal{EL}$

- Some reasoning tasks are not interesting in $\mathcal{EL}$:
  - We do not have $\bot$ or $\neg$
  - $\Rightarrow$ We cannot create contradictions
  - $\Rightarrow$ every $\mathcal{EL}$ ontology is consistent and coherent
- Other reasoning tasks are more interesting:
  - Subsumption: $\mathcal{O} \models C \sqsubseteq D$
  - Instance checking: $\mathcal{O} \models a : C$
  - Classification: Determine all $\mathcal{O} \models A \sqsubseteq B$ where $A, B \in \mathbf{C}$
  - Materialization: Determine all $\mathcal{O} \models a : B$ where $a \in \mathbf{I}$ and $B \in \mathbf{C}$
- We first look at an algorithm for subsumption

The $\mathcal{EL}$ Subsumption Algorithm

The idea:

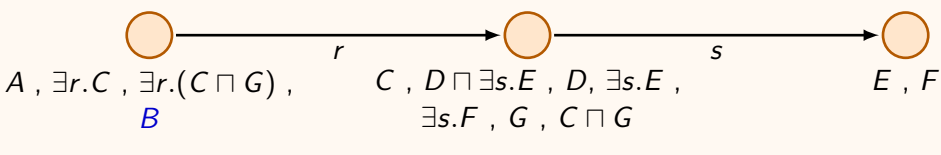
- We want to determine whether $\mathcal{O} \models C \sqsubseteq D$
- We iteratively construct a special model of $\mathcal{O}$ with an element $d \in C^{\mathcal{I}}$
- In this model, d will satisfy all concepts D' for which $\mathcal{O} \models C \sqsubseteq D'$
  - The model is thus called a canonical model
- We start with the single element d that should satisfy C
- Throughout the algorithm, elements are marked with concepts they (should) satisfy
- Special rules are applied towards satisfying those concepts
- If we eventually assign D to the initial element, then $\mathcal{O} \models C \sqsubseteq D$

The $\mathcal{EL}$ Subsumption Algorithm: Example

We first look at an example:

$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq \exists r.C, \qquad C \sqsubseteq D \sqcap \exists s.E, \qquad E \sqsubseteq F,$$
$$\exists s.F \sqsubseteq G, \qquad \exists r.(C \sqcap G) \sqsubseteq B \quad \}$$

We want to determine whether $\mathcal{O} \models A \sqsubseteq B$



The $\mathcal{EL}$ Subsumption Algorithm: Example

But $\mathcal{O} \models A \sqsubseteq B$ only if $\mathcal{I} \models A \sqsubseteq B$ in every model $\mathcal{I}$ of $\mathcal{O}$.

Did we really show that $\mathcal{I} \models A \sqsubseteq B$ in every model of $\mathcal{O}$?

Yes:

- We added what needs to be in every model $\mathcal{I}$ of $\mathcal{O}$ where $A^{\mathcal{I}} \neq \emptyset$
- Only exception: role-successors
  - other models might use different elements as role-successors
  - but for every role-successor we added, there has to be a corresponding role successor in the other models that satisfies at least the same concepts

$\Rightarrow$ $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$ also on every other model $\mathcal{I}$ of $\mathcal{O}$ where $A^{\mathcal{I}} \neq \emptyset$

- If $A^{\emptyset}$, we also have $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$

$\Rightarrow$ $\mathcal{I} \models A \sqsubseteq B$ in every model $\mathcal{I}$ of $\mathcal{O}$ $\square$

$\mathcal{EL}$ Inference Rules

The idea: To decide whether $\mathcal{O} \models C_0 \sqsubseteq D_0$, we start with an element d_0, assign C_0 to it, and check whether we can apply the follwoing rules so that D_0 gets eventually assigned:

- $\top$-rule: add $\top$ to d.
- $\sqcap$-rule 1: If d has $C \sqcap D$ assigned, assign also C and D to d.
- $\sqcap$-rule 2: If d has C and D assigned, assign also $C \sqcap D$ to d.
- $\exists$-rule 1: If d has $\exists r.C$ assigned, add a new r-successor to d and assign C to it.
- $\exists$-rule 2: If d has an r-successor with C assigned, add $\exists r.C$ to d.
- $\sqsubseteq$-rule: If d has C assigned and $C \sqsubseteq D \in \mathcal{T}$, then also assign D to d.

Remaining Challenges

If we just use the rules like that, our algorithm will never stop:

1. $\sqcap$-rule will keep generating new concepts.
2. $\exists$-rule 1 will keep adding elements.

To have a decision procedure we have to know when to stop.

- How else would we ever know if $\mathcal{O} \not\models C_0 \sqsubseteq D_0$?

Fixing Problem 1: Unbounded Introduction of Concepts

All inference rules need to be applied with the following side condition:

- If we assign a concept C, then C must occur somewhere in our input
  - in the ontology or in the entailment $C_0 \sqsubseteq D_0$
  - "occur in" includes nested concepts
- Idea:
  - If a concept does not occur at least nested in the TBox, then it will never be needed to trigger the $\sqsubseteq$-rule
  - Other concepts are only relevant if they occur in D_0

Fixing Problem 2: Unbounded Introduction of Individuals

- For every individual, we remember the initial concept
- We modify the $\exists$-rule 1 as follows:

> $\exists$-rule 1: If d has $\exists r.C$ assigned:
>
> 1. If there is some e with initial concept $\underline{C}$, make e the r-successor of d
> 2. Otherwise, add a new r-successor to d, and assign to it as initial concept $\underline{C}$

The Final $\mathcal{EL}$-Completion Rules

$\top$-rule: Add $\top$ to any individual.

$\sqcap$-rule 1: If d has $C \sqcap D$ assigned, assign also C and D to d.

$\sqcap$-rule 2: If d has C and D assigned, assign also $C \sqcap D$ to d.

$\exists$-rule 1: If d has $\exists r.C$ assigned:

1. If there is an element e with initial concept $\underline{C}$ assigned, make e the r-successor of d.
2. Otherwise, add a new r-successor to d, and assign to it as initial concept $\underline{C}$.

$\exists$-rule 2: If d has an r-successor with C assigned, add $\exists r.C$ to d.

$\sqsubseteq$-rule: If d has C assigned and $C \sqsubseteq D \in \mathcal{T}$, then also assign D to d

The $\mathcal{EL}$-Completion Algorithm

Decide whether $\mathcal{O} \models C_0 \sqsubseteq D_0$

1. Start with initial element d_0, assign to C_0 to it as initial concept
2. Set `changed` := **true**
3. While `changed` = **true** :
   - 3.1 Set `changed` := **false**
   - 3.2 For every element d in the current interpretation:
     - 3.2.1 Apply all the rules on d in all possible ways so that only concepts from the input get assigned
     - 3.2.2 If a new element was added or a new concept assigned, set `changed` = **true**
4. If D_0 was assigned to d_0, return `YES`, otherwise return `NO`

Concepts from the input: occur, possibly nested, explicitly in $\mathcal{O}$, C_0 or D_0

Termination

Lemma: The algorithm stops after at most n^2 steps, where n is the number of concepts in the input

Proof:

- We add at most one element for each concept in the input: at most n elements.
  - Because we never introduce two elements with the same initial concept.
- For each element, we only add concepts occurring in the input: at most n concepts per element.
- In each step, we either add an element or assign a new concept to an existing element: at most $n \times n$ steps. □

Note: This is better than for propositional logic!

Completeness

Lemma: If the algorithm returns NO, then $\mathcal{O} \not\models C_0 \sqsubseteq D_0$.

Proof (sketch):

- The final interpretation $\mathcal{I}$ is a model of the TBox $\mathcal{T}$.
  - The $\top$-rule, $\sqcap$-rules and $\exists$-rules make sure that for every concept C occurring in the input, and for every $d \in \Delta^{\mathcal{I}}$, d is assigned to C iff $c \in C^{\mathcal{I}}$.
  - For every $C \sqsubseteq D \in \mathcal{O}$, if C is assigned to d, then D is also assigned to d. (by the $\sqsubseteq$-rule).

  $\Rightarrow \mathcal{I} \models C \sqsubseteq D$ for every $C \sqsubseteq D \in \mathcal{T}$.
- We have $d_0 \in C_0^{\mathcal{I}}$ and $d_0 \notin D_0^{\mathcal{I}}$.
  - by the previous observation

$\Rightarrow \mathcal{I} \models \mathcal{T}$ and $\mathcal{I} \not\models C_0 \sqsubseteq D_0$, but what about the ABox?

- The ABox is not relevant: we can extend any model of $\mathcal{O}$ by adding the stuff in $\mathcal{I}$! □

Soundness

Lemma: If the algorithm returns `YES`, then $\mathcal{O} \models C_0 \sqsubseteq D_0$

Proof Idea:

- Assume the algorirthm returns `YES`.
- We need to show that for every model $\mathcal{I}$ of $\mathcal{O}$, $\mathcal{I} \models C_0 \sqsubseteq D_0$.
- Take any model $\mathcal{I}'$ of $\mathcal{O}$ in which there is some $d_0' \in C_0^{\mathcal{I}}$.
- Now simulate the steps of the algorithm on the elements in $\mathcal{I}'$, starting with d.
  - Only difference: $\exists$-rule 1 now uses a corresponding, existing role successor in $\mathcal{I}'$.
- One can show that, whenever the algorithm would assign C to d, then also $d \in C^{\mathcal{I}'}$.
- The algorithm assigns D_0 to d_0', and therefore $d_0' \in D_0^{\mathcal{I}'}$.
- $\mathcal{I}'$ was an arbitrary model, and d_0' an arbitrary element of C_0', so the same would happen for all models and all elements in C_0'.
- Hence, $\mathcal{O} \models C \sqsubseteq D$. □

Solving the Other Reasoning Tasks

We can use the same algorithm with little adaptations for other tasks:

- Instance checking: determine whether $\mathcal{O} \models a : C$
  - Same procedure, but start with the ABox
- Classification: determine all $\mathcal{O} \models A \sqsubseteq B$ where $A, B \in \mathbf{C}$
  - Start with a node for every concept name
- Materialization: determine all $\mathcal{O} \models a : A$ where $a \in \mathbf{I}$ and $A \in \mathbf{C}$
  - Same algorithm as for instance checking.
  - *(Note: role assertions cannot be inferred from other axioms in $\mathcal{EL}$)*

Note: Algorithms always takes at most n^2 steps!

- Better complexity as propositional logic.
- Modern $\mathcal{EL}$ reasoners like ELK process 10,000s of axioms in seconds.

Beyond $\mathcal{EL}$?

There are some extensions to $\mathcal{EL}$ that still allow reasoning in polynomial time, even if we allow $\bot$.

However, any other concept constructor we have seen until now makes reasoning even harder than for propositional logic:

- disjunction: $\sqcup$
- negation: $\neg$
- value restrictions: $\forall$

Allowing any of these three constructs makes reasing ExpTime-hard, which means reasoning may require a number of steps that is exponential in the size of the ontology (indepedently of whether $P = NP$).

Challenges with $\mathcal{ALC}$

First challenge: Disjunction $C \sqcup D$

- For instances of $C \sqcup D$, we do not know whether we need to satisfy C or D .

$\Rightarrow$ Case distinction required

$\Rightarrow$ There is no canonical model as for $\mathcal{EL}$

Challenges with $\mathcal{ALC}$

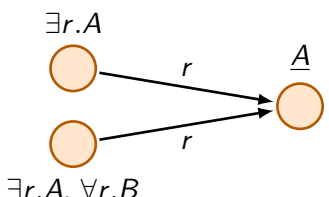
Second challenge: Value restrictions $\forall r.C$

- We need a rule like the following:

> $\forall$-rule: If d as $\forall r.C$ assigned and e is an r-succesor of d, then assign C to e

$\Rightarrow$ Additional concepts come from predecessors of a node.

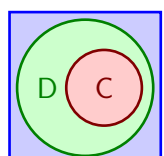
$\Rightarrow$ We cannot reuse individuals as before.



Challenges with $\mathcal{ALC}$

Third challenge: Negation $\neg C$

- Concepts can be unsatisfiable $\rightarrow$ model construction can fail
  - cannot have A and $\neg A$ at the same time
- Nested expressions
  - What do we do with $\neg(A \sqcap \neg(B \sqcup C))$?
- Different ways to express the same thing:
  - $C \sqsubseteq D$
  - $\neg D \sqsubseteq \neg C$
  - $C \sqcap \neg D \sqsubseteq \bot$
  - $\top \sqsubseteq \neg C \sqcup D$

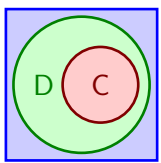


Overview of the Tableaux Method for $\mathcal{ALC}$

This time, it is easier to focus on concept satisfiability

> Given an ontology $\mathcal{O}$ and a concept C, C is satisfiable w.r.t. $\mathcal{O}$ iff $\mathcal{O}$ has a model $\mathcal{I}$ in which $C^{\mathcal{I}} \neq \emptyset$ (a model of C and $\mathcal{O}$).

- For $\mathcal{EL}$, this wouldn't have made sense, since every concept is satisfiable
- In $\mathcal{ALC}$, we can reduce many problems to it:
  - To decide $\mathcal{O} \models C \sqsubseteq D$, we check whether $C \sqcap \neg D$ is unsatisfiable
  - To decide consistency of $\mathcal{O}$, we check whether $\top$ is satisfiable



Overview of Tableaux Method for $\mathcal{ALC}$

Idea:

- We first normalize the ontology and concept
- We then try, on different branches, to find a model for $\mathcal{O}$ and C
- Again rules to add concepts and individuals
- A branch might get a clash in case it is inconsistent
  - We then go to the next branch
- If we complete a branch without clash, the concept is satisfiable
  - The branch corresponds to a model
- If all branches clash, the concept is unsatisfiable

Normalization

- A concept C is in negation normal form (NNF) iff the negation symbol $\neg$ only occurs in front of concept names.
- A TBox is in NNF if every axiom is of the form $\top \sqsubseteq C$, where C is in NNF
- An ABox is in NNF if every concept in it is in NNF
- An ontology is in NNF if TBox and ABox are in NNF

The Tableaux procedure

- We assume everything is normalized.
- Branches are represented as ABoxes.
- We start with the assertion $a : C$.
- We then step-wise apply a set of $\mathcal{ALC}$ expansion rules to construct different ABoxes on different branches.

$\mathcal{ALC}$ Expansion Rules

$X \Longrightarrow Y$ reads: If X is on the current branch and not Y, add Y

- $\sqcap$-rule: $a : C \sqcap D$
  $\Longrightarrow$ $a : C$, $a : D$
- $\sqcup$-rule: $a : C \sqcup D$ and we have neither $a : C$ nor $a : D$
  $\Longrightarrow$ $a : C$ or $a : D$
- $\exists$-rule: $a : \exists r.C$, no $\langle a, b \rangle : r$ s.t. $b : C$
  $\Longrightarrow \langle a, b \rangle : r, b : C$ for a new individual b
- $\forall$-rule: $a : \forall r.C$, $\langle a, b \rangle : r$
  $\Longrightarrow b : C$
- $\mathcal{T}$-rule: $\top \sqsubseteq C \in \mathcal{T}$
  $\Longrightarrow$ $a : C$ for any individual a we already introduced

Differences to the Precompletion Method for $\mathcal{EL}$

1. Rules only go in "one direction"
   - We only break concepts into simpler ones
   - We never introduce conjunction, disjunction or role restrictions
   - We want to decide satisfiability not subsumption
   - We don't have to deal with concepts on the left-hand side of an axiom (all axioms are of the form $\top \sqsubseteq C$)
   - In the $\mathcal{EL}$ method, every concept has to already occur in the input
   - For $\mathcal{ALC}$, this would be to restrictive
   - Since concepts only get smaller, we only introduce finitely many concepts
2. No special treatment of initial concepts
   - In the $\mathcal{EL}$ method, we would reuse individuals based on their initial concepts
   - We already saw that this does not work together with the $\forall$-rule
   - We will need another mechanism to deal with cyclic TBoxes such as $\{A \sqsubseteq \exists r.A\}$

Termination Criterion

A branch has a clash if for some individual a, we either have $a : \bot$ or for some concept name A, we have both $a : A$ and $a : \neg A$.

- Once we found a clash, we know that we cannot find a model anymore on the current branch.
- We then close the branch, and continue on the next.
- If all branches are closed with a clash, we know that the concept is unsatisfiable.

A branch is complete if it has no clash and no more rule can be applied.

- Once we have a complete branch, we know that our concept is satisfiable.

Example

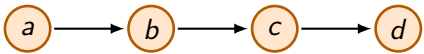
$$\mathcal{O} = \mathcal{T} = \{ \quad A \sqsubseteq B \sqcup C, \qquad C \sqsubseteq D \sqcap \exists r.F, \qquad A \sqcap \exists r.\top \sqsubseteq \neg D \quad \}$$

We want to decide whether $\mathcal{O} \models A \sqsubseteq B$.

1. Reduce to satisfiability problem:
   - Decide whether $A \sqcap \neg B$ is unsatisfiable.
2. Transform Input into NNF:

$$\begin{aligned}
\mathcal{T}_1 &= \{ \top \sqsubseteq \neg A \sqcup (B \sqcup C),\ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F),\ \top \sqsubseteq \neg(A \sqcap \exists r.\top) \sqcup \neg D \ \} \\
\mathcal{T}_2 &= \{ \top \sqsubseteq \neg A \sqcup (B \sqcup C),\ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F),\ \top \sqsubseteq (\neg A \sqcup \neg \exists r.\top) \sqcup \neg D \} \\
\mathcal{T}_3 &= \{ \top \sqsubseteq \neg A \sqcup (B \sqcup C),\ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F),\ \top \sqsubseteq (\neg A \sqcup \forall r.\neg \top) \sqcup \neg D \} \\
\mathcal{T}_4 &= \{ \top \sqsubseteq \neg A \sqcup (B \sqcup C),\ \top \sqsubseteq \neg C \sqcup (D \sqcap \exists r.F),\ \top \sqsubseteq (\neg A \sqcup \forall r.\bot) \sqcup \neg D \ \} \\
&= \mathcal{T}'
\end{aligned}$$

Ensuring Termination



- Every individual d has a path from the individual a from which we started
- The other individuals on this path (including the a) are called ancestors of d

- An individual a is blocked by an ancestor b if for every $a : C$, we also have $b : C$.
- An indivudal is blocked if it is blocked by some ancestor, or if some ancestor of it is blocked.

- No rule can be applied on a blocked individual
- Idea: any expansion rule that applies to a also applies to b, so there is no reason to apply it also on a
- Note: only ancestors can block

Example: Blocking

$$\mathcal{T} = \{ \quad \top \sqsubseteq \neg A \sqcup \forall r.B, \qquad \top \sqsubseteq \neg B \sqcup \exists r.A \quad \}$$

Task: check satisfiability of $A \sqcap \exists r.A$ wrt. $\mathcal{T}$

$a : A \sqcap \exists r.A$	$a : \neg B \sqcup \exists r.A$	$\langle b, c \rangle : r$
$a : A$	$a : \neg B$	$c : A$
$a : \exists r.A$	$b : B$	$c : B$
$a : \neg A \sqcup \forall r.B$	$b : \neg A \sqcup \forall r.B$	
$a : \forall r.B$	$b : \forall r.B$	
$\langle a, b \rangle : r$	$b : \neg B \sqcup \exists r.A$	
$b : A$	$b : \exists r.A$	c is blocked by b!

$\Rightarrow$ $A \sqcap \exists r.A$ is satisfiable!

Decision Procedure

- One can show that, with the blocking condition, the tableaux method always terminates
  - every branch is closed or complete after finitely many steps
  - only finitely many branches are introduced
- It is also easy to see that the method is sound
  - If it returns "satisfiable", then the concept is satisfiable.
  - Reason: we can easily transform the branch into a model
  - For the blocked nodes, we add a loop
- In fact, the algorithm is also complete
  - For every satisfiable concept, it returns "satisfiable"
  - Idea: If C is satisfiable, then there is a model $\mathcal{I}$ for it
  - We can use this model to "guide" the tableaux procedure $\rightarrow$ determine which rules to apply how

Decision Procedure

Theorem: The tableaux algorithm is a decision procedure for $\mathcal{ALC}$ concept satisfiability.

- This means: it terminates, it is sound and it is complete
- However, the complexity is much worse as for the $\mathcal{EL}$ procedure
  - It needs much more inference steps in relation to the ontology size
- In fact, reasoning with $\mathcal{ALC}$ is provably harder than for $\mathcal{EL}$
  - Exponential time instead of polynomial time complexity
- Modern DL reasoners try to exploit the "$\mathcal{EL}$-like" parts of the ontology as much as possible.

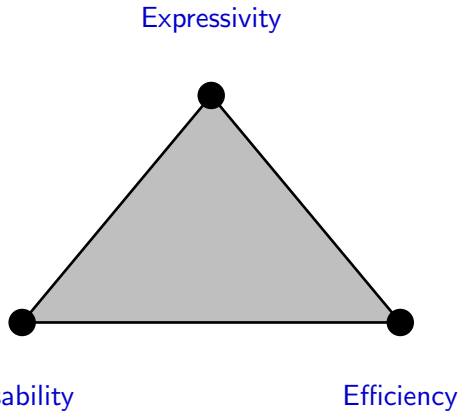
More Expressive DLs

Motivation

- For many realistic applications, $\mathcal{ALC}$ is too limited
- Consider the following axioms:
  - A human has two hands with 5 fingers each.
  - An AjaxFan is a fan of the club AFC Ajax.
  - hasChild is the inverse of hasParent.
  - hasIngredient is transitive
  - A Margherita has 1,120 kcal and costs 12.90€ .
  - The enemies of my friends are also my enemies.
- Modelling these with $\mathcal{ALC}$ would be awkward to impossible.

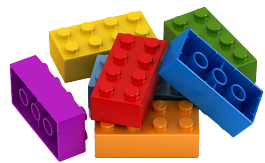
The Limits of Expressivity

- A central feature of DLs is not only the syntax, but also *decidability*.
- A challenge is to stay *decidable*, while offering sufficient expressivity.



Very Expressive Description Logics

- $\mathcal{SROIQ}(D)$ is one of the most expressive logics of the description logic family that is still decidable. *The DL underlying OWL!*
- The name describes its main additional features to $\mathcal{ALC}$:
  - transitive roles ($\mathcal{S}$),
  - complex **R**ole axioms ($\mathcal{R}$),
  - role **H**ierarchies ($\mathcal{H}$, contained in $\mathcal{R}$)
  - n**O**minals ($\mathcal{O}$),
  - **I**nverse roles ($\mathcal{I}$),
  - (**Q**ualified) number restrictions ($\mathcal{Q}$),
  - Concrete **D**omains (D).



- DLs between $\mathcal{EL}$ and $\mathcal{SROIQ}(D)$ follow the same naming scheme: $\mathcal{ELHO}$, $\mathcal{ALCI}$, $\mathcal{SOQ}$, etc.

Number Restrictions

(Qualified) number restrictions restrict the number of outgoing role connections.

For all role names r, concepts C, and $n \geq 0$, the following are concepts:

Name	at-least restriction	at-most restriction
Syntax	$\geq nr.C$	$\leq nr.C$
Semantics	$\{d \mid \#\{e \in C^{\mathcal{I}} \mid (d,e) \in r^{\mathcal{I}}\} \geq n\}$	$\{d \mid \#\{e \in C^{\mathcal{I}} \mid (d,e) \in r^{\mathcal{I}}\} \leq n\}$

$\mathit{Student} \sqcap (\geq 2 \mathit{attends}.\mathit{Lecture}) \qquad \leq 1 \mathit{belongsTo}.\top$

- We can also write $=nr.C := (\geq nr.C) \sqcap (\leq nr.C)$

$\mathit{Cow} \sqsubseteq =4 \mathit{hasBodyPart}.\mathit{Leg}$

$\mathit{Hand} \sqsubseteq =5 \mathit{hasBodyPart}.\mathit{Finger}$

Inverse Roles

For all role names r, the following is also a role, and can be used in all places where a role name can be used:

Name:	inverse role
Syntax:	r^-
Semantics:	$(r^-)^{\mathcal{I}} = \{(d, e) \mid (e, d) \in r^{\mathcal{I}}\}$

belongsTo$^-$ $\quad \top \sqsubseteq \forall \mathit{hasChild}^-.\mathit{Parent}$

Nominals and Self-Love

For all individual names a and role names r, the following are concepts:

Name:	nominal	local reflexivity
Syntax:	$\{a\}$	$\exists r.\text{Self}$
Semantics:	$\{a^{\mathcal{I}}\}$	$\{x \mid (x,x) \in r^{\mathcal{I}}\}$

$\exists employedBy.\{VUAmsterdam\}$ $\exists loves.\text{Self}$

Extensional Definitions

Nominals allow to provide extensional definitions in addition to the intensional ones.

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 instead list the elements of the class.

$$EUMember \equiv \{France\} \sqcup \{Germany\} \sqcup \{Italy\} \sqcup \ldots$$

Additional Assertions

For all individual names a, b and role names r, the following are assertions:

Name	equality	inequality	negated role assertion
Syntax	$a \approx b$	$a \not\approx b$	$(a, b) : \neg r$
Semantics	$a^{\mathcal{I}} = b^{\mathcal{I}}$	$a^{\mathcal{I}} \neq b^{\mathcal{I}}$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \notin r^{\mathcal{I}}$

$anna \not\approx tom$ $\quad$ $morningStar \approx eveningStar$ $\quad$ $(Ernie, Bert) : \neg hasBrother$

In fact, all assertions are now syntactic sugar:

$$a : C \iff \{a\} \sqsubseteq C \qquad (a, b) : r \iff \{a\} \sqsubseteq \exists r.\{b\}$$
$$a \approx b \iff \{a\} \sqsubseteq \{b\} \qquad (a, b) : \neg r \iff \{a\} \sqsubseteq \forall r.\neg\{b\}$$
$$a \not\approx b \iff \{a\} \sqsubseteq \neg\{b\}$$

Role Axioms

With the concept constructors so far, a range of role axioms can be expressed:

Name	Syntax	Meaning
Domain	$\mathsf{dom}(r) \sqsubseteq C$	$\exists r.\top \sqsubseteq C$
Range	$\mathsf{ran}(r) \sqsubseteq C$	$\exists r^{-}.\top \sqsubseteq C, \quad \top \sqsubseteq \forall r.C$
Functionality	$\mathsf{fun}(r)$	$\top \sqsubseteq \leq 1 r.\top$
Reflexivity	$\mathsf{Ref}(r)$	$\top \sqsubseteq \exists r.\mathsf{Self}$

Role Axioms

In $\mathcal{SROIQ}(D)$, an ontology consists of three parts $\mathcal{O} = \mathcal{A} \cup \mathcal{T} \cup \mathcal{R}$, where $\mathcal{R}$ is an RBox, i.e. a finite set of role axioms.

If $r, s, s_1, \ldots, s_n$ are roles, then the following are role axioms:

Name:	role inclusion	complex role inclusion	role disjointness
Syntax:	$r \sqsubseteq s$	$s_1 \circ \cdots \circ s_n \sqsubseteq r$	$\mathsf{dis}(r, s)$
Semantics:	$r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$	$s_1^{\mathcal{I}} \circ \cdots \circ s_n^{\mathcal{I}} \subseteq r^{\mathcal{I}}$	$r^{\mathcal{I}} \cap s^{\mathcal{I}} = \emptyset$

$r \circ s$ represents the concatenation of r and s:

Role Axioms

Things we can express in an RBox:

- sub-roles and chains:

  $hasMother \sqsubseteq hasParent$ $\quad$ $hasParent \circ hasMother \sqsubseteq hasGrandMother$

- "Inverse of":

  $hasParent \sqsubseteq hasChild^{-}$ $\quad$ $hasChild^{-} \sqsubseteq hasParent$

- Transitivity:

  $partOf \circ partOf \sqsubseteq partOf$

- Other features:

  $\mathsf{dis}(hasDaughter, hasSon)$ $\quad$ $\mathsf{Ref}(hasRelative)$

Additional Axioms: Syntactic Sugar

Name	Syntax	Defined as
disjointness	$\mathsf{dis}(C, D)$	$C \sqsubseteq \neg D$ or $D \sqsubseteq \neg C$ or $C \sqcap D \sqsubseteq \bot$
role equivalence	$r \equiv s$	$r \sqsubseteq s$, $s \sqsubseteq r$
domain restriction	$\mathsf{dom}(r) \sqsubseteq C$	$\top \sqsubseteq \forall r^-.C$ or $\exists r.\top \sqsubseteq C$
range restriction	$\mathsf{ran}(r) \sqsubseteq C$	$\top \sqsubseteq \forall r.C$ or $\exists r^-.\top \sqsubseteq C$
role irreflexivity	$\mathsf{irr}(r)$	$\exists r.\mathsf{Self} \sqsubseteq \bot$
role functionality	$\mathsf{fun}(r)$	$\top \sqsubseteq \leq 1 r.\top$
role symmetry	$\mathsf{sym}(r)$	$r \sqsubseteq r^-$
role asymmetry	$\mathsf{asy}(r)$	$\mathsf{dis}(r, r^-)$
role transitivity	$\mathsf{tra}(r)$	$r \circ r \sqsubseteq r$

Note: Domain and Range Restrictions

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

$\mathrm{ran}(\mathit{eats}) \sqsubseteq \mathit{Organism}$ (equivalent to $\top \sqsubseteq \forall \mathit{eats}.\mathit{Organism}$)
$\mathit{Bird} \sqsubseteq \exists \mathit{eats}.\mathit{Stone}$

implies that some stones are organisms (those that are eaten by birds).

If *Stone* and *Organism* are disjoint, then Bird is unsatisfiable.

$\mathrm{dom}(\mathit{eats}) \sqsubseteq \mathit{Organism}$ (equivalent to $\exists \mathit{eats}.\top \sqsubseteq \mathit{Organism}$)
$\mathit{Tornado} \sqsubseteq \exists \mathit{eats}.\mathit{House}$

entails $\mathit{Tornado} \sqsubseteq \mathit{Organism}$.

If *Tornado* and *Organism* are disjoint, then *Tornado* becomes unsatisfiable.

Concrete Domains

Examples of concrete domains are strings, numbers, dates.

In DLs with concrete domains (including $\mathcal{SROIQ}(D)$), there are special role names (attributes, or data properties in OWL terminology) that refer to elements in the concrete domain.

In addition, we have special predicates, that can be used to refer to sets in the concrete domain.

- For example: numbers larger than 10, strings starting with "Mrs. ", etc.

$\exists \mathit{hasAge}.\leq_{18} \qquad \exists \mathit{hasPrice}.\geq_{1{,}000€} \qquad \exists \mathit{hasSize}.\{30\}$

The formal definition is a bit involved, which is why we skip it here.

Decidability

Apart from adding constructors and axioms to $\mathcal{ALC}$, $\mathcal{SROIQ}(D)$ imposes several restrictions on the use of roles, to retain decidability.

- The RBox must be regular.
- Number restrictions, self restrictions, and disjoint role axioms can only contain simple roles.

Regular RBoxes

Intuitively, an RBox is regular if there are no cyclic dependencies between role names.

The RBox $\{\mathit{hasFather} \circ \mathit{hasBrother} \sqsubseteq \mathit{hasUncle}, \mathit{hasChild} \circ \mathit{hasUncle} \sqsubseteq \mathit{hasBrother}\}$ is not regular.

Certain cycles are however allowed:

- $r \circ r \sqsubseteq r$ to express transitivity
- $r_1 \circ \ldots \circ r_n \circ r \sqsubseteq r$ (role directly at the end of the chain)
- $r \circ r_1 \circ \ldots \circ r_n \sqsubseteq r$ (role directly at the beginning of the chain)

Non-regular RBoxes make reasoning undecidable and are therefore forbidden in $\mathcal{SROIQ}(D)$!

Simple Roles

Apart from adding constructors and axioms to $\mathcal{ALC}$, $\mathcal{SROIQ}(D)$ imposes several restrictions on the use of roles, to retain decidability.

- The RBox must be regular.
- Number restrictions, self restrictions, and disjoint role axioms can only contain simple roles.

The set of non-simple roles is inductively defined as follows:

- If $r_1 \circ \cdots \circ r_n \sqsubseteq r \in \mathcal{R}$ with $n \geq 2$, then r and r^- are non-simple.
- If $s \sqsubseteq r \in \mathcal{R}$ and s is non-simple, then r and r^- are non-simple.

All other roles are simple.

Transitive roles and roles that have transitive subroles are not simple.

Example: Partonomies

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

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

directPartOf is not transitive, but has a transitive super role.

$$\mathit{directPartOf} \sqsubseteq \mathit{partOf} \qquad \mathsf{tra}(\mathit{partOf})$$

This separation allows us to use *directPartOf* in number restrictions.

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

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

End of Part I

DLs are a very important formalism for KR

- Many use cases of ontologies to work with data and knowledge
- High expressiveness within the boundaries of decidability
- Fast reasoning with modern DL reasoners
  - Important examples: ELK (for $\mathcal{ELH}$), HermiT and Konclude (the fastest reasoner)

But DLs are not the formalism of choice for every use case

- Another important formalism are rule-based languages (datalog, ASP)

Some limitations come through the foundation on first-order logic:

- do not deal well with contradictions
- cannot represent probabilities