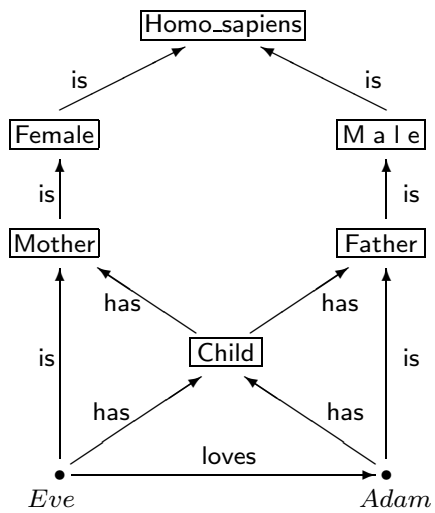
Knowledge Representation



We would like to be able to ask a computer questions regarding this knowledge.

Problems: What is the *precise* meaning of this 'semantic network', e.g. what is the difference between *Adam* and Male or what is the difference between a loves- and an is-arrow?

Figure taken from: D. Gabbay *et al.*, *Many-Dimensional Modal Logics*, 2002. Forthcoming.

Formalising Semantic Networks

It seems useful to distinguish the following:

- Specific *objects* (or ***individuals***) such as *Adam* and *Eve*.
- *Classes* or *sets* of objects, such as Female (later called ***concepts***).
- The use of is as a *subset-relation* between concepts (like between Mother and Female).
- The use of is as a *membership-relation* between individuals and concepts (like between *Adam* and Father).
- Other binary *relations* (which we will call ***roles***) between individuals, such as loves between *Eve* and *Adam*.
- The use of the same kinds of *relations (roles) between individuals and concepts* (e.g. has). In that case we may want to distinguish (at least) whether the relation is meant to hold for *all* of the individuals belonging to the concept or just for *some* of them.

Description Logics

History. Description logics (DL) have been developed in the late 80s/early 90s to provide sound logical foundations for semantic networks and similar semi-formal knowledge representation (KR) languages.

Many logics. There is not just one DL, but many. In this course, however, we will only introduce the most important one: $\mathcal{ALC}$ (generally considered the 'standard' DL).

We will define its syntax and semantics, see how it can be used for KR purposes, and give a Tableaux-based calculus to reason in $\mathcal{ALC}$.

A hot topic. This is currently a very active research area (which also includes people at King's). People investigate (for example) more expressive DLs, faster reasoning algorithms, combinations of DLs with other logics, ...

Syntax and Semantics of Concepts

$\mathcal{ALC}$ *concept formulas* are built up from basic *concept names* and *roles*. The semantics of a concept is defined in terms of a *domain* $\mathcal{D}$ and an *interpretation function* $\mathcal{I}$. Concept names are interpreted as subsets of $\mathcal{D}$ and roles are interpreted as subsets of $\mathcal{D} \times \mathcal{D}$.

Syntax and semantics of complex concept formulas:

$\neg C$	$\mathcal{D} \setminus C^{\mathcal{I}}$
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
$\forall R.C$	$\{x \in \mathcal{D} \mid \{y \in \mathcal{D} \mid (x,y) \in R^{\mathcal{I}}\} \subseteq C^{\mathcal{I}}\}$
$\exists R.C$	$\{x \in \mathcal{D} \mid \{y \in \mathcal{D} \mid (x,y) \in R^{\mathcal{I}}\} \cap C^{\mathcal{I}} \neq \varnothing\}$

TBoxes and ABoxes

We can distinguish *terminological knowledge*, i.e. knowledge about concepts, from *assertional knowledge*, i.e. knowledge about individuals. Accordingly, a DL knowledge base consists of a *TBox* and an *ABox*.

TBox. A *TBox* is a set of *concept definitions:*

$$CN \dot{\sqsubseteq} C \qquad\qquad CN \doteq C$$

Here, CN is a concept name and C is a concept formula.

ABox. An *ABox* is a set of *instantiational* and *relational assertions*:

$$a : C \qquad\qquad (a, b) : R$$

Here, a and b are *individuals*, C is a concept formula, and R is a role.

Exercise

Translate the following text into a set of ABox and TBox formulas.

> *Britney is a superstar. She dislikes Christina. A superstar is a person who is famous, attractive, and either very talented or managed by a clever person. Also, every superstar dislikes at least one rich but untalented superstar. Anyone who manages Britney is bound to be very rich.*

Use the following vocabulary:

- **individuals:** *britney*, *christina*
- **roles:** *dislikes*, *managed-by*
- **concept names:** *Attractive*, *Clever*, *Famous*, *Person*, *Rich*, *Superstar*, *Talented*

Truth in a Model

Recall that the interpretation function $\mathcal{I}$ maps concepts to subsets of a domain $\mathcal{D}$ and roles to sets of pairs of elements of $\mathcal{D}$. Furthermore, let $\mathcal{I}$ map individual names a to elements $a^{\mathcal{I}}$ of $\mathcal{D}$.

The truth of ABox and TBox formulas in a model $\mathcal{M} = (\mathcal{D}, \mathcal{I})$ is defined as follows:

- $\mathcal{M} \models a : C$ iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$
- $\mathcal{M} \models (a, b) : R$ iff $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$
- $\mathcal{M} \models C \dot{\sqsubseteq} D$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
- $\mathcal{M} \models C \doteq D$ iff $C^{\mathcal{I}} = D^{\mathcal{I}}$

Reasoning Tasks

A DL system may typically offer some of the following inference services:

- *Concept satisfiability:* Is the given concept formula C satisfiable/consistent?
  [check $C^{\mathcal{I}} \neq \{\}$ for some model $\mathcal{M} = (\mathcal{D}, \mathcal{I})$]
- *Concept subsumption:* Given concept formulas C and D, is C more specific than (or equivalent to) D?
  [check $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all models $\mathcal{M} = (\mathcal{D}, \mathcal{I})$]
- *ABox satisfiability:* Is the given ABox $\mathcal{A}$ satisfiable/consistent?
  [check $\mathcal{M} \models \mathcal{A}$ (every formula in $\mathcal{A}$) for some model $\mathcal{M}$]
- *Instance checking:* Given ABox $\mathcal{A}$, can we infer that the individual a must be a member of the concept C?
  [check $\mathcal{M} \models a : C$ for all models $\mathcal{M}$ with $\mathcal{M} \models \mathcal{A}$]

TBox Reasoning

Typically, reasoning that involves a TBox is more complex than pure ABox reasoning or reasoning about just one or two concept formulas.

We can formulate the reasoning tasks from the previous slide also with respect to a TBox (but won't learn how to actually do this kind of stuff in this course). Example:

- *Concept satisfiability:* Is the given concept formula C satisfiable, given the information in TBox $\mathcal{T}$?
  [check $C^{\mathcal{I}} \neq \{\}$ for some model $\mathcal{M} = (\mathcal{D}, \mathcal{I})$ with $\mathcal{M} \models \mathcal{T}$]

TBox unfolding. If we have only one TBox formula $CN \doteq \ldots$ per concept name CN and the TBox is *acyclic*, then we can *unfold* the concept definitions. In that case, the TBox may be regarded as a simple list of abbreviations for complex concept formulas.

Tableaux for $\mathcal{ALC}$ ABoxes

We shall give a Tableaux calculus to decide the satisfiability of a given ABox in $\mathcal{ALC}$.

The basic idea is the same as for FOL: we have a proof tree with formulas on it, which we analyse according to certain expansion rules. We succeed (i.e. prove unsatisfiability of the input ABox) iff we can close all branches.

Formulas here are $\mathcal{ALC}$ ABox formulas, i.e. instantiational assertions of the form $a : C$ and relational assertions of the form $(a, b) : R$.

Tableaux Rules for $\mathcal{ALC}$ ABoxes

Alpha Rules

$$\frac{a : C \sqcap D}{\begin{array}{c} a : C \\ a : D \end{array}} \qquad \frac{a : \neg(C \sqcup D)}{\begin{array}{c} a : \neg C \\ a : \neg D \end{array}}$$

$\neg\neg$-Elim.

$$\frac{a : \neg\neg C}{a : C}$$

Beta Rules

$$\frac{a : C \sqcup D}{a : C \mid a : D} \qquad \frac{a : \neg(C \sqcap D)}{a : \neg C \mid a : \neg D}$$

Closing Branches

$$\frac{\begin{array}{c} a : C \\ a : \neg C \end{array}}{\times}$$

Remark. Alternatively, we may use KE-style rules, i.e. PB and non-branching beta rules. These actually have been shown to be more efficient (but: only *empirically* and only for $\mathcal{ALC}$).

Tableaux Rules for $\mathcal{ALC}$ ABoxes (2)

Gamma Rules

$$\frac{\begin{array}{c} a : \forall R.C \\ (a,b) : R \end{array}}{b : C} \qquad \frac{\begin{array}{c} a : \neg\exists R.C \\ (a,b) : R \end{array}}{b : \neg C}$$

Delta Rules

$$\frac{a : \exists R.C}{\begin{array}{c} (a,b) : R \\ b : C \end{array}} \qquad \frac{a : \neg\forall R.C}{\begin{array}{c} (a,b) : R \\ b : \neg C \end{array}}$$

- Gamma formulas need to be analysed for every individual b with $(a,b) : R$ on the branch.
- The b in the delta rules is a *new* individual name.
- We don't need to apply the delta rule if we already have a *'witness'* on the branch.

Soundness, Completeness and Termination

We can prove the following theorems:

Theorem 1 (Termination) *For any input, the $\mathcal{ALC}$ Tableaux algorithm will terminate after a finite number of steps.*

Theorem 2 (Soundness) *Whenever all branches in an $\mathcal{ALC}$ tableau can be closed, then the input ABox is indeed unsatisfiable.*

Theorem 3 (Completeness) *Whenever an $\mathcal{ALC}$ tableau cannot be closed completely, then the input ABox is indeed satisfiable.*

Decidability. The crucial difference between FOL and $\mathcal{ALC}$ is that for the latter we have termination, i.e. we will always get an answer. This means, the ABox satisfiability problem for $\mathcal{ALC}$ is *decidable*.

Termination Proof

Proof sketch. The central idea is to show that the number of different ABox formulas that could possibly be generated is limited and therefore the algorithm will stop at some point (because any further rule applications would be redundant).

Observation 1. Any concept formula added to a branch will be a subformula of a formula appearing in the input ABox.

Observation 2. The number of new individuals introduced via the delta rule is limited by the 'quantifier depth' of concept formulas in the input.

$\Rightarrow$ There can only be finitely many formulas on any one branch.

Observation 3. As the number of disjunctive formulas is limited as well, there can only be finitely many branches.

Soundness Proof

Proof sketch. Very similar to the FOL case. All we need to do is to show that none of the proof rules can 'destroy' satisfiability. In other words: applying any of the rules to a satisfiable branch (a branch whose formulas form a satisfiable ABox) will always result in another satisfiable branch.

Completeness Proof

Proof sketch. Because of termination, for any input we either get a closed tableau or one with an open saturated branch. This makes the completeness proof much easier than for FOL.

All we need to show is that, given an open saturated branch, we can construct a model for the input ABox (and thereby show its satisfiability). The idea is very similar to that of deriving countermodels from (terminating) unsuccessful KE proofs in FOL:

We define as the domain $\mathcal{D}$ the set of individual names appearing on the branch. Instantiational assertions of the form $a : CN$ (where CN is a concept name) give rise to the interpretation of concept names; relational assertions tell us how to interpret role names.

Then we use structural induction to show that this is indeed a model for the input ABox.

Using $\mathcal{ALC}$ Tableaux

We can use $\mathcal{ALC}$ Tableaux for the following reasoning tasks:

- *ABox satisfiability:* To show ABox $\mathcal{A}$ is not satisfiable, show the tableau for $\mathcal{A}$ closes.
- *Instance checking:* To show a is an instance of C given ABox $\mathcal{A}$, show the tableau for $\mathcal{A} \cup \{a : \neg C\}$ closes.
- *Concept satisfiability:* To show concept formula C is not satisfiable, show the tableau for $\{a : C\}$ closes.
- *Concept subsumption:* To show that concept C is more specific than (or equivalent to) concept D, show the tableau for $\{a : C \sqcap \neg D\}$ closes.

Remark. It is also possible to incorporate TBoxes into Tableaux. In the case of a simple acyclic TBox we could just unfold. In the general case things get more complicated (the problem has to do with termination), but there is a way (keyword *'blocking'*) ...

Standard Translation

Translating concepts. We can translate $\mathcal{ALC}$ concepts into FOL formulas with a single free variable x.

During translation we will replace each concept name CN with a unary predicate CN' and each role R with a binary predicate R' (in practice we can do without the $'$).

The standard translation of concept formulas is defined inductively:

$$\begin{aligned}
CN^* &= CN'(x) \\
(\neg C)^* &= \neg C^* \\
(C \sqcap D)^* &= C^* \wedge D^* \\
(C \sqcup D)^* &= C^* \vee D^* \\
(\forall R.C)^* &= (\forall y)(R'(x,y) \rightarrow C^*[x \leftarrow y]) \\
(\exists R.C)^* &= (\exists y)(R'(x,y) \wedge C^*[x \leftarrow y])
\end{aligned}$$

Standard Translation (2)

Translating TBoxes. The standard translation of TBox formulas:

$$\begin{aligned}(C \dot{\sqsubseteq} D)^* &= (\forall x)(C^* \rightarrow D^*)\\ (C \doteq D)^* &= (\forall x)(C^* \leftrightarrow D^*)\end{aligned}$$

Translating ABoxes. If we extend the translation to ABox formulas we replace every individual a with a constant symbol a'.

The standard translation of ABox formulas is defined as follows:

$$\begin{aligned}(a : C)^* &= C^*[x \leftarrow a']\\ ((a, b) : R)^* &= R'(a', b')\end{aligned}$$

That is, TBox and ABox formulas are mapped to FOL *sentences*.

Remark. All of this means that $\mathcal{ALC}$ corresponds to a *fragment of first-order logic*.