Beschreibungssprachen (Description Logics)

A family of logic based Knowledge Representation formalisms

- Descendants of semantic networks and KL-ONE
- Describe domain in terms of concepts (classes), roles (relationships) and individuals

Distinguished by:

- Formal semantics (typically model theoretic)
  - Decidable fragments of FOL
  - Closely related to Propositional Modal & Dynamic Logics
- Provision of inference services
  - Sound and complete decision procedures for key problems
  - Implemented systems (highly optimised)

- Einfache Sprache zum Start: \( \mathcal{ALC} \) (Attributive Language with Complement)
- Im Semantic Web wird \( \mathcal{SHOIN(D)} \) eingesetzt. Hierauf basiert die Semantik von OWL DL.

Geschichte

- Ihre Entwicklung wurde inspiriert durch semantische Netze und Frames.
- Frühere Namen:
  - KL-ONE like languages
  - terminological logics
- Ziel war eine Wissensrepräsentation mit formaler Semantik.

Literatur

- Ian Horrocks, Peter F. Patel-Schneider and Frank van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language.
Ontology/KR languages aim to model (part of) world

Terms in language correspond to entities in world

Meaning given by, e.g.:
- Mapping to another formalism, such as FOL, with own well defined semantics
- or a Model Theory (MT)

MT defines relationship between syntax and interpretations
- There can be many interpretations (models) of one piece of syntax
- Models supposed to be analogue of (part of) world
  - E.g., elements of model correspond to objects in world
- Formal relationship between syntax and models
  - Structure of models reflect relationships specified in syntax
- Inference (e.g., subsumption) defined in terms of MT
  - E.g., $T \models A \subseteq B$ iff in every model of $T$, $\text{ext}(A) \subseteq \text{ext}(B)$

Many logics (including standard First Order Logic) use a model theory based on (Zermelo-Frankel) set theory

The domain of discourse (i.e., the part of the world being modelled) is represented as a set (often referred as $\Delta$)

Objects in the world are interpreted as elements of $\Delta$
- Classes/concepts (unary predicates) are subsets of $\Delta$
- Properties/roles (binary predicates) are subsets of $\Delta \times \Delta$ (i.e., $\Delta^2$)
- Ternary predicates are subsets of $\Delta^3$ etc.

The sub-class relationship between classes can be interpreted as set inclusion.

Formally, the vocabulary is the set of names we use in our model of (part of) the world
- \{Daisy, Cow, Animal, Mary, Person, Z123ABC, Car, drives, \ldots\}

An interpretation $\mathcal{I}$ is a tuple $\langle \Delta, \mathcal{I} \rangle$
- $\Delta$ is the domain (a set)
- $\mathcal{I}$ is a mapping that maps
  - Names of objects to elements of $\Delta$
  - Names of unary predicates (classes/concepts) to subsets of $\Delta$
  - Names of binary predicates (properties/roles) to subsets of $\Delta \times \Delta$
  - And so on for higher arity predicates (if any)
DL Architecture

Knowledge Base

Tbox (schema)
- Man = Human \& Male
- Happy-Father = Man \& \exists has-child Female \& ...

Abox (data)
- John : Happy-Father
- (John, Mary) : has-child

DL Semantics

Interpretation function \( I \) extends to concept expressions in the obvious way, i.e.:

\[
\begin{align*}
(C \cap D)^I &= C^I \cap D^I \\
(C \cup D)^I &= C^I \cup D^I \\
(\neg C)^I &= \Delta^I \setminus C^I \\
\{x\}^I &= \{x^I\} \\
(\exists R.C)^I &= \{x \mid \exists y \langle x, y \rangle \in R^I \land y \in C^I\} \\
(\forall R.C)^I &= \{x \mid \forall y \langle x, y \rangle \in R^I \Rightarrow y \in C^I\} \\
(\leq n R)^I &= \{x \mid \#\{y \mid \langle x, y \rangle \in R^I\} \leq n\} \\
(\geq n R)^I &= \{x \mid \#\{y \mid \langle x, y \rangle \in R^I\} \geq n\}
\end{align*}
\]

DL Knowledge Base

DL Knowledge Base (KB) normally separated into 2 parts:
- **TBox** is a set of axioms describing structure of domain (i.e., a conceptual schema), e.g.:
  - HappyFather = Man \& \exists hasChild Female \& ...
  - Elephant = Animal \& Large \& Grey
  - transitive(ancestor)
- **ABox** is a set of axioms describing a concrete situation (data), e.g.:
  - John:HappyFather
  - (John, Mary):hasChild

Separation has no logical significance
- But may be conceptually and implementationally convenient

DL Knowledge Bases (Ontologies)

A DL Knowledge Base is of the form \( \mathcal{K} = (T, A) \)
- **T** (Tbox) is a set of axioms of the form:
  - \( C \subseteq D \) (concept inclusion)
  - \( C \equiv D \) (concept equivalence)
  - \( R \subseteq S \) (role inclusion)
  - \( R \equiv S \) (role equivalence)
  - \( R^+ \subseteq R \) (role transitivity)
- **A** (Abox) is a set of axioms of the form
  - \( x \in D \) (concept instantiation)
  - \( \langle x, y \rangle \in R \) (role instantiation)

Two sorts of Tbox axioms often distinguished
- “Definitions”
  - \( C \subseteq D \) or \( C \equiv D \) where \( C \) is a concept name
- General Concept Inclusion axioms (GCIs)
  - \( C \subseteq D \) where \( C \) is an arbitrary concept
Knowledge Base Semantics

An interpretation \( I \) satisfies (models) an axiom \( A (I \models A) \):
- \( I \models C \subseteq D \iff C^I \subseteq D^I \)
- \( I \models C \equiv D \iff C^I = D^I \)
- \( I \models R \subseteq S \iff R^I \subseteq S^I \)
- \( I \models R^+ \subseteq R \iff (R^I)^+ \subseteq R^I \)
- \( I \models x \in D \iff x^I \in D^I \)
- \( I \models (x,y) \in R \iff (x^I,y^I) \in R^I \)

\( I \) satisfies a Tbox \( T (I \models T) \) iff \( I \) satisfies every axiom \( A \) in \( T \)

\( I \) satisfies an Abox \( A (I \models A) \) iff \( I \) satisfies every axiom \( A \) in \( A \)

\( I \) satisfies an KB \( K (I \models K) \) iff \( I \) satisfies both \( T \) and \( A \)

Inference Tasks

Knowledge is correct (captures intuitions)
- \( C \) subsumes \( D \) w.r.t. \( K \) iff for every model \( I \) of \( K \), \( C^I \subseteq D^I \)

Knowledge is minimally redundant (no unintended synonyms)
- \( C \) is equivalent to \( D \) w.r.t. \( K \) iff for every model \( I \) of \( K \), \( C^I = D^I \)

Knowledge is meaningful (classes can have instances)
- \( C \) is satisfiable w.r.t. \( K \) iff there exists some model \( I \) of \( K \) s.t. \( C^I \neq \emptyset \)

Querying knowledge
- \( x \) is an instance of \( C \) w.r.t. \( K \) iff for every model \( I \) of \( K \), \( x^I \in C^I \)
- \( (x,y) \) is an instance of \( R \) w.r.t. \( K \) iff for every model \( I \) of \( K \), \( (x^I,y^I) \in R^I \)

Knowledge base consistency
- A KB \( K \) is consistent iff there exists some model \( I \) of \( K \)

Syntax für DLs (ohne concrete domains)

<table>
<thead>
<tr>
<th>Concepts</th>
<th>Atomic</th>
<th>A, B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not</td>
<td>¬C</td>
<td></td>
</tr>
<tr>
<td>And</td>
<td>C ( \cap ) D</td>
<td></td>
</tr>
<tr>
<td>Or</td>
<td>C ( \cup ) D</td>
<td></td>
</tr>
<tr>
<td>Exists</td>
<td>( \exists R.C )</td>
<td></td>
</tr>
<tr>
<td>For all</td>
<td>( \forall R.C )</td>
<td></td>
</tr>
<tr>
<td>At least</td>
<td>( \geq ) m R.C</td>
<td>(( \geq ) m R)</td>
</tr>
<tr>
<td>At most</td>
<td>( \leq ) m R.C</td>
<td>(( \leq ) m R)</td>
</tr>
<tr>
<td>Nominal</td>
<td>{i_1, \ldots, i_n}</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Roles</th>
<th>Atomic</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Inverse</td>
<td>R^-</td>
</tr>
</tbody>
</table>

S = ALC + Transitivity

OWL DL = SHOIN(D) (D: concrete domain)

The Description Logic ALC: Syntax

Atomic types: concept names \( A, B, \ldots \) (unary predicates)
role names \( R, S, \ldots \) (binary predicates)

Constructors:
- \( \neg C \) (negation)
- \( C \cap D \) (conjunction)
- \( C \cup D \) (disjunction)
- \( \exists R.C \) (existential restriction)
- \( \forall R.C \) (value restriction)

Abbreviations:
- \( C \rightarrow D = \neg C \cup D \) (implication)
- \( C \leftrightarrow D = C \rightarrow D \) \( \cap D \rightarrow C \) (bi-implication)
- \( T = (A \cup \neg A) \) (top concept)
- \( \bot = A \cap \neg A \) (bottom concept)
Examples

- Person \( \sqcap \) Female
- Person \( \sqcap \) \( \exists \) attends. Course
- Person \( \sqcap \) \( \forall \) attends. (Course \( \rightarrow \) ~Easy)
- Person \( \sqcap \) \( \exists \) teaches. (Course \( \sqcap \) \( \forall \) attended-by. (Bored \( \sqcup \) Sleeping))

Interpretations

Semantics based on interpretations \((\Delta^I, \cdot^I)\), where
- \(\Delta^I\) is a non-empty set (the domain)
- \(\cdot^I\) is the interpretation function mapping
  each concept name \(A\) to a subset \(A^I\) of \(\Delta^I\) and
  each role name \(R\) to a binary relation \(R^I\) over \(\Delta^I\).

Intuition: interpretation is complete description of the world

Technically: interpretation is first-order structure
with only unary and binary predicates

Semantics of Complex Concepts

\((-C)^I = \Delta^I \setminus C^I\)
\((C \sqcap D)^I = C^I \cap D^I\)
\((C \sqcup D)^I = C^I \cup D^I\)

\((\exists R.C)^I = \{d \mid \text{there is an } e \in \Delta^I \text{ with } (d, e) \in R^I \text{ and } e \in C^I\}\)

\((\forall R.C)^I = \{d \mid \text{for all } e \in \Delta^I, (d, e) \in R^I \text{ implies } e \in C^I\}\)
Capture an application’s terminology means defining concepts

**TBoxes** are used to store concept definitions:

**Syntax:**
- finite set of concept equations $A \sqsubseteq C$
- with $A$ concept name and $C$ concept
- left-hand sides must be unique!

**Semantics:**
- interpretation $\mathcal{I}$ satisfies $A \sqsubseteq C$ iff $A^\mathcal{I} = C^\mathcal{I}$
- $\mathcal{I}$ is model of $\mathcal{T}$ if it satisfies all definitions in $\mathcal{T}$

E.g.: Lecturer $\sqsubseteq$ Person $\sqcap \exists$teaches.Course

Yields two kinds of concept names: defined and primitive

**TBox: Example**

*TBoxes* are used as ontologies:

Woman $\sqsubseteq$ Person $\sqcap$ Female

Man $\sqsubseteq$ Person $\sqcap \neg$Woman

Lecturer $\sqsubseteq$ Person $\sqcap \exists$teaches.Course

Student $\sqsubseteq$ Person $\sqcap \exists$attends.Course

BadLecturer $\sqsubseteq$ Person $\sqcap \forall$teaches.(Course $\rightarrow$ Boring)

**TBox: Example II**

A TBox restricts the set of admissible interpretations.

Lecturer $\sqsubseteq$ Person $\sqcap \exists$teaches.Course

Student $\sqsubseteq$ Person $\sqcap \exists$attends.Course

Reasoning Tasks — Subsumption

$C$ subsumed by $D$ w.r.t. $\mathcal{T}$ (written $C \sqsubseteq^\mathcal{T} D$)

iff

$C^\mathcal{I} \sqsubseteq D^\mathcal{I}$ holds for all models $\mathcal{I}$ of $\mathcal{T}$

Intuition: If $C \sqsubseteq^\mathcal{T} D$, then $D$ is more general than $C$

Example:

Lecturer $\sqsubseteq$ Person $\sqcap \exists$teaches.Course

Student $\sqsubseteq$ Person $\sqcap \exists$attends.Course

Then

Lecturer $\sqcap \exists$attends.Course $\sqsubseteq^\mathcal{T}$ Student
Reasoning Tasks — Classification

Classification: arrange all defined concepts from a TBox in a hierarchy w.r.t. generality

Woman ⩾ Person ⩾ Female
Man ⩾ Person ⩾ ¬Woman
MaleLecturer ⩾ Man ⩾ ∃ teaches.Course

Can be computed using multiple subsumption tests
Provides a principled view on ontology for browsing, maintaining, etc.

A Concept Hierarchy

Excerpt from a process engineering ontology

Reasoning Tasks — Satisfiability

$C$ is satisfiable w.r.t. $T$ iff $T$ has a model with $C \models \bot$

Intuition: If unsatisfiable, the concept contains a contradiction.

Example: Woman ⩾ Person ⩾ Female
Man ⩾ Person ⩾ ¬Woman

Then ∃ sibling.Man ⩾ ∃ sibling.Woman is unsatisfiable w.r.t. $T$

Subsumption can be reduced to (un)satisfiability and vice versa:
- $C \sqsubseteq_T D$ if $C \sqsubseteq_T \neg D$ is not satisfiable w.r.t. $T$
- $C$ is satisfiable w.r.t. $T$ if not $C \sqsubseteq_T \bot$

Many reasoners decide satisfiability rather than subsumption.
Definitional TBoxes

A **primitive interpretation** for TBox $\mathcal{T}$ interpretes:
- the **primitive** concept names in $\mathcal{T}
- all role names

A TBox is called **definitional** if every primitive interpretation for $\mathcal{T}$ can be uniquely extended to a model of $\mathcal{T}$. i.e., primitive concepts (and roles) uniquely determine defined concepts.

Not all TBoxes are definitional:

$$\text{Person} \equiv \exists\text{parent}\cdot\text{Person}$$

Non-definitional TBoxes describe constraints, e.g. from background knowledge.

Acyclic TBoxes

TBox $\mathcal{T}$ is **acyclic** if there are no definitional cycles:

- $\text{Lecturer} \equiv \exists\text{teaches}\cdot\text{Course}$
- $\text{Course} \equiv \exists\text{has-title}\cdot\text{Title} \equiv \exists\text{taught-by}\cdot\text{Lecturer}$

Expansion of acyclic TBox $\mathcal{T}$:
- exhaustively replace defined concept names with their definition (terminates due to acyclicity)

Acyclic TBoxes are always definitional:
- first expand, then set $A^\mathcal{I} := C^\mathcal{I}$ for all $A \equiv C \in \mathcal{T}$

Acyclic TBoxes II

For reasoning, acyclic TBox can be eliminated:
- to decide $C \sqsubseteq \mathcal{T} D$ with $\mathcal{T}$ acyclic,
  - expand $\mathcal{T}$
  - replace defined concept names in $C, D$ with their definition
  - decide $C \sqsubseteq D$
- analogously for satisfiability

May yield an exponential blow-up:

$$A_0 \equiv \forall r.A_1 \land \forall s.A_1$$
$$A_1 \equiv \forall r.A_2 \land \forall s.A_2$$

$$\ldots$$

$$A_{n-1} \equiv \forall r.A_n \land \forall s.A_n$$

General Concept Inclusions

View of TBox as set of constraints

**General TBox**: finite set of general concept implications (GCIs)

$$C \sqsubseteq D$$

with both $C$ and $D$ allowed to be complex

* e.g. $\text{Course} \sqsubseteq \forall \text{attended-by}\cdot\text{Sleeping} \sqsubseteq \text{Boring}$

Interpretation $\mathcal{I}$ is model of general TBox $\mathcal{T}$ if $C^\mathcal{I} \subseteq D^\mathcal{I}$ for all $C \sqsubseteq D \in \mathcal{T}$.

$C \equiv D$ is abbreviation for $C \sqsubseteq D, D \sqsubseteq C$

* e.g. $\text{Student} \sqsubseteq \exists\text{favourite}\cdot\text{Soccer Team} \equiv \text{Student} \sqsubseteq \exists\text{favourite}\cdot\text{Beer}$

Note: $C \sqsubseteq D$ equivalent to $\top \models C \rightarrow D$
**ABoxes**

ABoxes describe a snapshot of the world

An **ABox** is a finite set of assertions

\[
a : C \quad \text{[a individual name, C concept]}
\]

\[
(a, b) : R \quad \text{[(a, b) individual names, R role name]}
\]

E.g. \{peter : Student, (dl-course, uli) : taught-by\}

Interpretations \(\mathcal{I}\) map each individual name \(a\) to an element of \(\Delta^2\).

**\(\mathcal{I}\) satisfies** an assertion

\[
a : C \quad \text{iff} \quad a^2 \in C^2
\]

\[
(a, b) : R \quad \text{iff} \quad (a^2, b^2) \in R^2
\]

\(\mathcal{I}\) is a model for an ABox \(\mathcal{A}\) if \(\mathcal{I}\) satisfies all assertions in \(\mathcal{A}\).

---

**Reasoning with ABoxes**

**ABox consistency**

Given an ABox \(\mathcal{A}\) and a TBox \(\mathcal{T}\), do they have a common model?

**Instance checking**

Given an ABox \(\mathcal{A}\), a TBox \(\mathcal{T}\), an individual name \(a\), and a concept \(C\) does \(a^2 \in C^2\) hold in all models of \(\mathcal{A}\) and \(\mathcal{T}\)?

\[
(\text{written } \mathcal{A}, \mathcal{T} \models a : C)
\]

The two tasks are interreducible:

- \(\mathcal{A}\) consistent w.r.t. \(\mathcal{T}\) iff \(\mathcal{A}, \mathcal{T} \not\models a : \bot\)

- \(\mathcal{A}, \mathcal{T} \models a : C\) iff \(\mathcal{A} \cup \{a : \neg C\}\) is not consistent

---

**Example for ABox Reasoning**

**ABox**

<table>
<thead>
<tr>
<th>Name</th>
<th>Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>t14</td>
<td>Trunk</td>
</tr>
<tr>
<td>dumbbo</td>
<td>Mammal</td>
</tr>
</tbody>
</table>

**TBox**

\[
\begin{align*}
\text{dumbo} : \forall \text{color.Lightgrey} \\
\text{dumbo} : \forall \text{color.Lightgrey} \\
\text{dumbo : bodypart.Truk} \\
\text{dumbo : bodypart.Truk} \\
\text{dumbo : bodypart.Truk} \\
\text{dumbo : bodypart.Truk} \\
\text{dumbo : bodypart.Truk} \\
\text{dumbo : bodypart.Truk} \\
\end{align*}
\]

1. ABox is inconsistent w.r.t. TBox.
2. dumbbo is an instance of Elephant
2. Tableau algorithms for $\mathcal{ALC}$ and extensions

We see a tableau algorithm for $\mathcal{ALC}$ and extend it with
1. general TBoxes and
2. inverse roles

Goal: Design sound and complete decision procedures for satisfiability (and subsumption) of DLs which are well-suited for implementation purposes

---

A tableau algorithm for the satisfiability of $\mathcal{ALC}$ concepts

Goal: design an algorithm which takes an $\mathcal{ALC}$ concept $C_0$ and
1. returns “satisfiable” iff $C_0$ is satisfiable and
2. terminates, on every input,
i.e., which decides satisfiability of $\mathcal{ALC}$ concepts.

Recall: such an algorithm cannot exist for FOL since satisfiability of FOL is undecidable.

Idea: our algorithm
- is tableau-based and
- tries to construct a model of $C_0$
- by breaking $C_0$ down syntactically, thus
- inferring new constraints on such a model.

---

Preliminaries: Negation Normal Form

To make our life easier, we transform each concept $C_0$ into an equivalent $C_1$ in NNF

Equivalent: $C_0 \sqsubseteq C_1$ and $C_1 \sqsubseteq C_0$

NNF: negation occurs only in front of concept names

How? By pushing negation inwards (de Morgan et. al):

\[
\begin{align*}
\neg(C \cap D) & \leadsto \neg C \cup \neg D \\
\neg(C \cup D) & \leadsto \neg C \cap \neg D \\
\neg \neg C & \leadsto C \\
\neg \forall R.C & \leadsto \exists R.\neg C \\
\neg \exists R.C & \leadsto \forall R.\neg C
\end{align*}
\]

From now on: concepts are in NNF and
- $\text{sub}(C)$ denotes the set of all sub-concepts of $C$

---

More intuition

Find out whether $A \sqcap \exists R.B \sqcap \forall R.\neg B$ is satisfiable...

\[
A \sqcap \exists R.B \sqcap \forall R.\neg B \sqcap \exists S.E
\]

Our tableau algorithm works on a completion tree which
- represents a model $\mathcal{I}$: nodes represent elements of $\Delta^2$
  - each node $x$ is labelled with concepts $\mathcal{L}(x) \subseteq \text{sub}(C_0)$
  - $C \in \mathcal{L}(x)$ is read as “$x$ should be an instance of $C$”
- edges represent role successorship
  - each edge $\langle x, y \rangle$ is labelled with a role-name from $C_0$
    - $R \in \mathcal{L}(\langle x, y \rangle)$ is read as “$(x, y)$ should be in $R^2$”
- is initialised with a single root node $x_0$ with $\mathcal{L}(x_0) = \{C_0\}$
- is expanded using completion rules
\[ \square \text{-rule: if } C_1 \cap C_2 \in \mathcal{L}(x) \text{ and } \{C_1, C_2\} \not\subseteq \mathcal{L}(x) \]
then set \( \mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\} \)

\[ \bigcup \text{-rule: if } C_1 \cup C_2 \in \mathcal{L}(x) \text{ and } \{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset \]
then set \( \mathcal{L}(x) = \mathcal{L}(x) \cup \{C\} \) for some \( C \in \{C_1, C_2\} \)

\[ \exists \text{-rule: if } \exists S, C \in \mathcal{L}(x) \text{ and } x \text{ has no } S\text{-successor } y \text{ with } C \in \mathcal{L}(y) \]
then create a new node \( y \) with \( \mathcal{L}(\langle x, y \rangle) = \{S\} \) and \( \mathcal{L}(y) = \{C\} \)

\[ \forall \text{-rule: if } \forall S, C \in \mathcal{L}(x) \text{ and there is an } S\text{-successor } y \text{ of } x \text{ with } C \not\subseteq \mathcal{L}(y) \]
then set \( \mathcal{L}(y) = \mathcal{L}(y) \cup \{C\} \)

\[ \exists \text{-rule: if } \exists S, C \in \mathcal{L}(x) \text{ and } x \text{ has no } S\text{-successor } y \text{ with } C \in \mathcal{L}(y) \]
then create a new node \( y \) with \( \mathcal{L}(\langle x, y \rangle) = \{S\} \) and \( \mathcal{L}(y) = \{C\} \)

\[ \forall \text{-rule: if } \forall S, C \in \mathcal{L}(x) \text{ and there is an } S\text{-successor } y \text{ of } x \text{ with } C \not\subseteq \mathcal{L}(y) \]
then set \( \mathcal{L}(y) = \mathcal{L}(y) \cup \{C\} \)

We only apply rules if their application does “something new”

\[ \square \text{-rule: if } C_1 \cap C_2 \in \mathcal{L}(x) \text{ and } \{C_1, C_2\} \not\subseteq \mathcal{L}(x) \]
then set \( \mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\} \)

\[ \bigcup \text{-rule: if } C_1 \cup C_2 \in \mathcal{L}(x) \text{ and } \{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset \]
then set \( \mathcal{L}(x) = \mathcal{L}(x) \cup \{C\} \) for some \( C \in \{C_1, C_2\} \)

\[ \exists \text{-rule: if } \exists S, C \in \mathcal{L}(x) \text{ and } x \text{ has no } S\text{-successor } y \text{ with } C \in \mathcal{L}(y) \]
then create a new node \( y \) with \( \mathcal{L}(\langle x, y \rangle) = \{S\} \) and \( \mathcal{L}(y) = \{C\} \)

\[ \forall \text{-rule: if } \forall S, C \in \mathcal{L}(x) \text{ and there is an } S\text{-successor } y \text{ of } x \text{ with } C \not\subseteq \mathcal{L}(y) \]
then set \( \mathcal{L}(y) = \mathcal{L}(y) \cup \{C\} \)

Properties of the completion rules for ALC

The \( \bigcup \) rule is non-deterministic:

\[ \square \text{-rule: if } C_1 \cap C_2 \in \mathcal{L}(x) \text{ and } \{C_1, C_2\} \not\subseteq \mathcal{L}(x) \]
then set \( \mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\} \)

\[ \bigcup \text{-rule: if } C_1 \cup C_2 \in \mathcal{L}(x) \text{ and } \{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset \]
then set \( \mathcal{L}(x) = \mathcal{L}(x) \cup \{C\} \) for some \( C \in \{C_1, C_2\} \)

\[ \exists \text{-rule: if } \exists S, C \in \mathcal{L}(x) \text{ and } x \text{ has no } S\text{-successor } y \text{ with } C \in \mathcal{L}(y) \]
then create a new node \( y \) with \( \mathcal{L}(\langle x, y \rangle) = \{S\} \) and \( \mathcal{L}(y) = \{C\} \)

\[ \forall \text{-rule: if } \forall S, C \in \mathcal{L}(x) \text{ and there is an } S\text{-successor } y \text{ of } x \text{ with } C \not\subseteq \mathcal{L}(y) \]
then set \( \mathcal{L}(y) = \mathcal{L}(y) \cup \{C\} \)

Last details on tableau algorithm for ALC

Clash: a c-tree contains a clash if it has a node \( x \) with \( \bot \in \mathcal{L}(x) \) or \( \{A, \neg A\} \subseteq \mathcal{L}(x) \) — otherwise, it is clash-free

Complete: a c-tree is complete if none of the completion rules can be applied to it

Answer behaviour: when started for \( C_0 \) (in NNF!), the tableau algorithm
\bullet is initialised with a single root node \( x_0 \) with \( \mathcal{L}(x_0) = \{C_0\} \)
\bullet repeatedly applies the completion rules (in whatever order it likes)
\bullet answer “\( C_0 \) is satisfiable” iff the completion rules can be applied in such a way that it results in a complete and clash-free c-tree (careful: this is non-deterministic)

...go back to examples
Properties of our tableau algorithm

**Lemma:** Let \( C_0 \) an \( \mathcal{ALC} \)-concept in NNF. Then
1. the algorithm terminates when applied to \( C_0 \) and
2. the rules can be applied such that they generate a
clash-free and complete completion tree iff \( C_0 \) is satisfiable.

**Corollary:**
1. Our tableau algorithm decides satisfiability and subsumption of \( \mathcal{ALC} \).
2. Satisfiability (and subsumption) in \( \mathcal{ALC} \) is decidable in PSpace.
3. \( \mathcal{ALC} \) has the finite model property
   i.e., every satisfiable concept has a finite model.
4. \( \mathcal{ALC} \) has the tree model property
   i.e., every satisfiable concept has a tree model.
5. \( \mathcal{ALC} \) has the finite tree model property
   i.e., every satisfiable concept has a finite tree model.

Extend tableau algorithm to \( \mathcal{ALC} \) with general TBoxes: Preliminaries

We extend our tableau algorithm by adding a new completion rule:
- remember that nodes represent elements of \( \Delta^T \) and
- if \( C \sqsubseteq D \in \mathcal{T} \), then for each element \( x \) in a model \( I \) of \( \mathcal{T} \)
  if \( x \in C^I \), then \( x \in D^I \)
  hence \( x \in (\neg C)^I \) or \( x \in D^I \)
  \( x \in (\neg C \sqcup D)^I \)
  \( x \in (\text{NNF}(\neg C \sqcup D))^I \)
for NNF(\( E \)) the negation normal form of \( E \)

Extend tableau algorithm to \( \mathcal{ALC} \) with general TBoxes

Recall:
- Concept inclusion: of the form \( C \sqsubseteq D \) for \( C, D \) (complex) concepts
- (General) TBox: a finite set of concept inclusions
- \( I \) satisfies \( C \sqsubseteq D \) iff \( C^I \subseteq D^I \)
- \( I \) is a model of TBox \( \mathcal{T} \) iff \( I \) satisfies each concept equation in \( \mathcal{T} \)
- \( C_0 \) is satisfiable w.r.t. \( \mathcal{T} \) iff there is a model \( I \) of \( \mathcal{T} \) with \( C_0^I \neq \emptyset \)

Goal – Lemma: Let \( C_0 \) an \( \mathcal{ALC} \)-concept and \( \mathcal{T} \) be a \( \mathcal{ALC} \)-TBox. Then
1. the algorithm terminates when applied to \( \mathcal{T} \) and \( C_0 \) and
2. the rules can be applied such that they generate a clash-free
and complete completion tree iff \( C_0 \) is satisfiable w.r.t. \( \mathcal{T} \).

Completion rules for \( \mathcal{ALC} \) with TBoxes

\[ \square \text{-rule: if } C_1 \sqcap C_2 \in \mathcal{L}(x) \text{ and } \{ C_1, C_2 \} \not\subseteq \mathcal{L}(x) \]
\[
\text{then set } \mathcal{L}(x) = \mathcal{L}(x) \cup \{ C_1, C_2 \}
\]
\[ \sqcup \text{-rule: if } C_1 \sqcup C_2 \in \mathcal{L}(x) \text{ and } \{ C_1, C_2 \} \cap \mathcal{L}(x) = \emptyset \]
\[
\text{then set } \mathcal{L}(x) = \mathcal{L}(x) \cup \{ C \} \text{ for some } C \in \{ C_1, C_2 \}
\]
\[ \exists \text{-rule: if } \exists S.C \in \mathcal{L}(x) \text{ and } x \text{ has no } S \text{-successor } y \text{ with } C \in \mathcal{L}(y), \]
\[
\text{then create a new node } y \text{ with } \mathcal{L}((x, y)) = \{ S \} \text{ and } \mathcal{L}(y) = \{ C \}
\]
\[ \forall \text{-rule: if } \forall S.C \in \mathcal{L}(x) \text{ and there is an } S \text{-successor } y \text{ of } x \text{ with } C \notin \mathcal{L}(y) \]
\[
\text{then set } \mathcal{L}(y) = \mathcal{L}(y) \cup \{ C \}
\]
\[ \mathcal{T} \text{-rule: if } C_1 \sqsubseteq C_2 \in \mathcal{T} \text{ and } \text{NNF}(\neg C_1 \sqcup C_2) \not\subseteq \mathcal{L}(x) \]
\[
\text{then set } \mathcal{L}(x) = \mathcal{L}(x) \cup \{ \text{NNF}(\neg C_1 \sqcup C_2) \} \]
A tableau algorithm for \( \mathcal{ALC} \) with general TBoxes

Example: Consider satisfiability of \( C \) w.r.t. \( \{ C \sqsubseteq \exists R.C \} \)

Tableau algorithm no longer terminates!

Reason: size of concepts no longer decreases along paths in a completion tree

Observation: most nodes on this path look the same and we keep repeating ourselves

Regain termination with a “cycle-detection” technique called blocking

Intuitively, whenever we find a situation where \( y \) has to satisfy stronger constraints than \( x \), we freeze \( x \), i.e., block rules from being applied to \( x \)

\[ L(x) \subseteq L(y) \]

A tableau algorithm for \( \mathcal{ALC} \) with general TBoxes: Blocking

- \( x \) is directly blocked if it has an ancestor \( y \) with \( L(x) \subseteq L(y) \)
- in this case and if \( y \) is the “closest” such node to \( x \), we say that \( x \) is blocked by \( y \)
- a node is blocked if it is directly blocked or one of its ancestors is blocked

\( \oplus \) restrict the application of all rules to nodes which are not blocked

\( \lnot \)-rule: if \( x \) is not blocked then set \( L(x) = L(x) \cup \{ \lnot C \} \)

\( \forall \)-rule: if \( \forall S.C \in L(x) \), there is an \( S \)-successor \( y \) with \( C \notin L(y) \). Then set \( L(y) = L(y) \cup \{ C \} \)

\( \exists \)-rule: if \( \exists S.C \in L(x) \), then create a new node \( y \) with \( L((x, y)) = \{ S \} \) and \( L(y) = \{ C \} \)

\( \sqcap \)-rule: if \( x \) is not blocked then set \( L(x) = L(x) \cup \{ C \} \)

\( \sqcup \)-rule: if \( x \) is not blocked then set \( L(x) = L(x) \cup \{ C \} \)

\( \exists \)-rule: if \( \exists S.C \in L(x) \), then create a new node \( y \) with \( L((x, y)) = \{ S \} \) and \( L(y) = \{ C \} \)

\( \forall \)-rule: if \( \forall S.C \in L(x) \), there is an \( S \)-successor \( y \) with \( C \notin L(y) \). Then set \( L(y) = L(y) \cup \{ C \} \)

\( T \)-rule: if \( \sqsubseteq C \sqsubseteq C_2 \in T \), NNF(\( \lnot C_1 \sqcup C_2 \)) \( \notin L(x) \) and \( x \) is not blocked then set \( L(x) = L(x) \cup \{ \text{NNF}(\lnot C_1 \sqcup C_2) \} \)

\( \text{Tableaux Rules for } \mathcal{ALC} \)

<table>
<thead>
<tr>
<th>Rule</th>
<th>前提</th>
<th>归约</th>
<th>前提</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x \in { C_1 \sqcap C_2, \ldots } )</td>
<td>( \lnot )</td>
<td>( x \in { C_1 \sqcap C_2, C_1, C_2, \ldots } )</td>
<td></td>
</tr>
<tr>
<td>( x \in { C_1 \sqcup C_2, \ldots } )</td>
<td>( \sqcap )</td>
<td>( x \in { C_1 \sqcup C_2, C, \ldots } ) for ( C \in { C_1, C_2 } )</td>
<td></td>
</tr>
<tr>
<td>( x \in { \exists R.C, \ldots } )</td>
<td>( \sqcup )</td>
<td>( x \in { \exists R.C, \ldots } )</td>
<td></td>
</tr>
<tr>
<td>( R )</td>
<td>( y )</td>
<td>( y \in { C } )</td>
<td></td>
</tr>
<tr>
<td>( x \in { \forall R.C, \ldots } )</td>
<td>( \forall )</td>
<td>( x \in { \forall R.C, \ldots } )</td>
<td></td>
</tr>
<tr>
<td>( R )</td>
<td>( y )</td>
<td>( y \in { C, \ldots } )</td>
<td></td>
</tr>
</tbody>
</table>
Tableaux Rule for Transitive Roles

\[
\begin{array}{c|c}
 x \cdot \{\forall R.C, \ldots\} & x \cdot \{\forall R.C, \ldots\} \\
 R & R \\
y \cdot \{\ldots\} & y \cdot \{\forall R.C, \ldots\}
\end{array}
\]

Where \( R \) is a transitive role (i.e., \( (R^+)^+ = R^+ \))

☞ No longer naturally terminating (e.g., if \( C = \exists R \top \))

☞ Need blocking
  * Simple blocking suffices for \( \mathcal{ALC} \) plus transitive roles
  * I.e., do not expand node label if ancestor has superset label
  * More expressive logics (e.g., with inverse roles) need more sophisticated blocking strategies

Tableaux Algorithm — Example

Test satisfiability of \( \exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C) \) where \( R \) is a transitive role

\[
L(w) = \{\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}
\]
Tableaux Algorithm — Example

Test satisfiability of $\exists S.C \land \forall S.(\neg C \lor \neg D) \land \exists R.C \land \forall R.(\exists R.C')$ where $R$ is a transitive role

$L(w) = \{\exists S.C, \forall S.(\neg C \lor \neg D), \exists R.C, \forall R.(\exists R.C')\}$
Tableaux Algorithm — Example

Test satisfiability of $\exists S.C \land \forall S.\lnot (C \lor \lnot D) \land \exists R.C \land \forall R. (\exists R.C)$ where $R$ is a transitive role

$L(w) = \{\exists S.C, \forall S.\lnot (C \lor \lnot D), \exists R.C, \forall R. (\exists R.C)\}

L(x) = \{C, \lnot C \lor \lnot D\}

S

clash

Tableaux Algorithm — Example

Test satisfiability of $\exists S.C \land \forall S.\lnot (C \lor \lnot D) \land \exists R.C \land \forall R. (\exists R.C)$ where $R$ is a transitive role

$L(w) = \{\exists S.C, \forall S.\lnot (C \lor \lnot D), \exists R.C, \forall R. (\exists R.C)\}

L(x) = \{C, \lnot C \lor \lnot D\}, \lnot C\}

S

clash
Tableaux Algorithm — Example

Test satisfiability of $\exists S.C \land \forall S. (\neg C \sqcup \neg D) \land \exists R.C \land \forall R. (\exists R.C')$ where $R$ is a transitive role

$L(w) = \{ \exists S.C, \forall S. (\neg C \sqcup \neg D), \exists R.C, \forall R. (\exists R.C') \}$

$L(x) = \{ C, \neg C \sqcup \neg D \}$

Tableaux Algorithm — Example

Test satisfiability of $\exists S.C \land \forall S. (\neg C \sqcup \neg D) \land \exists R.C \land \forall R. (\exists R.C')$ where $R$ is a transitive role

$L(w) = \{ \exists S.C, \forall S. (\neg C \sqcup \neg D), \exists R.C, \forall R. (\exists R.C') \}$

$L(x) = \{ C, (\neg C \sqcup \neg D), \neg D \}$

Tableaux Algorithm — Example

Test satisfiability of $\exists S.C \land \forall S. (\neg C \sqcup \neg D) \land \exists R.C \land \forall R. (\exists R.C')$ where $R$ is a transitive role

$L(w) = \{ \exists S.C, \forall S. (\neg C \sqcup \neg D), \exists R.C, \forall R. (\exists R.C') \}$

$L(x) = \{ C, (\neg C \sqcup \neg D), \neg D \}$

$L(y) = \{ C \}$
Tableaux Algorithm — Example

Test satisfiability of \( \exists S. C \land \forall S. (\neg C \lor \neg D) \land \exists R.C \land \forall R. (\exists R.C) \) where \( R \) is a transitive role

\[ \mathcal{L}(w) = \{ \exists S. C, \forall S. (\neg C \lor \neg D), \exists R.C, \forall R. (\exists R.C) \} \]

\[ \mathcal{L}(x) = \{ C, (\neg C \lor \neg D), \neg D \} \]  
\[ \mathcal{L}(y) = \{ C \} \]
Test satisfiability of $\exists S.C \land \forall S. (\neg C \cup \neg D) \land \exists R.C \land \forall R. (\exists R.C)$ where $R$ is a transitive role

L(w) = \{\exists S.C, \forall S.(-C \cup -D),\exists R.C, \forall R.(\exists R.C)\}

L(x) = \{C, (\neg C \cup \neg D), \neg D\}

L(y) = \{C, \exists R.C, \forall R.(\exists R.C)\}

L(z) = \{C\}

Concept is satisfiable: T corresponds to model.
Tableaux Algorithm — Example

Test satisfiability of $\exists S.C \land \forall S.(\neg C \sqcup \neg D) \land \exists R.C \land \forall R.(\exists R.C)$ where $R$ is a transitive role

$\mathcal{L}(w) = \{\exists S.C, \forall S.(\neg C \sqcup \neg D), \exists R.C, \forall R.(\exists R.C)\}$

$\mathcal{L}(x) = \{C, (\neg C \sqcup \neg D), \neg D\}$

$\mathcal{L}(y) = \{C, \exists R.C, \forall R.(\exists R.C)\}$

Concept is satisfiable: $\mathcal{T}$ corresponds to model

Properties of our tableau algorithm for $\mathcal{ALC}$ with TBoxes

Lemma: Let $\mathcal{T}$ be a general $\mathcal{ALC}$-Tbox and $C_0$ an $\mathcal{ALC}$-concept. Then
1. the algorithm terminates when applied to $\mathcal{T}$ and $C_0$ and
2. the rules can be applied such that they generate a clash-free and complete completion tree iff $C_0$ is satisfiable w.r.t. $\mathcal{T}$.

Corollary: 1. Satisfiability of $\mathcal{ALC}$-concept w.r.t. TBoxes is decidable
2. $\mathcal{ALC}$ with TBoxes has the finite model property
3. $\mathcal{ALC}$ with TBoxes has the tree model property

A tableau algorithm for $\mathcal{ALC}$ with general TBoxes: Summary

The tableau algorithm presented here
- decides satisfiability of $\mathcal{ALC}$-concepts w.r.t. TBoxes, and thus also
- decides subsumption of $\mathcal{ALC}$-concepts w.r.t. TBoxes
- uses blocking to ensure termination, and
- is non-deterministic due to the $\rightarrow_{\text{LT}}$-rule
- in the worst case, it builds a tree of depth exponential in the size of the input, and thus of double exponential size. Hence it runs in (worst case) $2\text{NExpTime}$,
- can be implemented in various ways,
  - order/priorities of rules
  - data structure
  - etc.
- is amenable to optimisations – more on this next week

Challenges

☞ Increased expressive power
- Existing DL systems implement (at most) $\mathcal{SHIQ}$
- OWL extends $\mathcal{SHIQ}$ with datatypes and nominals

☞ Scalability
- Very large KBs
- Reasoning with (very large numbers of) individuals

☞ Other reasoning tasks
- Querying
- Matching
- Least common subsumer
- ...

☞ Tools and Infrastructure
- Support for large scale ontological engineering and deployment
Summary

☞ **Description Logics** are family of logical KR formalisms

☞ **Applications** of DLs include DataBases and **Semantic Web**
  - Ontologies will provide vocabulary for semantic markup
  - OWL web ontology language based on $SHIQ$ DL
  - Set to become W3C standard (OWL) & already widely adopted
  - Use of DL provides formal foundations and reasoning support

☞ **DL Reasoning** based on tableau algorithms

☞ **Highly Optimised** implementations used in DL systems

☞ **Challenges** remain
  - Reasoning with full OWL language
  - (Convincing) demonstration(s) of scalability
  - New reasoning tasks
  - Development of (high quality) tools and infrastructure

Resources

Slides from this talk

FaCT system (open source)
  - [http://www.cs.man.ac.uk/FaCT/](http://www.cs.man.ac.uk/FaCT/)

OilEd (open source)
  - [http://oiled.man.ac.uk/](http://oiled.man.ac.uk/)

OIL
  - [http://www.ontoknowledge.org/oil/](http://www.ontoknowledge.org/oil/)

W3C Web-Ontology (WebOnt) working group (OWL)
  - [http://www.w3.org/2001/sw/WebOnt/](http://www.w3.org/2001/sw/WebOnt/)

**DL Handbook**, Cambridge University Press
  - [http://books.cambridge.org/0521781760.htm](http://books.cambridge.org/0521781760.htm)