F. Description Logics – Part 2

This section is based on material from:
- Ian Horrocks: http://www.cs.man.ac.uk/~horrocks/Teaching/cs646/

Syntax für DLs (ohne concrete domains)

<table>
<thead>
<tr>
<th>Concepts</th>
<th>Atomic</th>
<th>Not</th>
<th>And</th>
<th>Or</th>
<th>Exists</th>
<th>For all</th>
<th>At least</th>
<th>At most</th>
<th>Nominal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A, B</td>
<td>¬C</td>
<td>C ∧ D</td>
<td>C ∨ D</td>
<td>∃R.C</td>
<td>∀R.C</td>
<td>2n R.C</td>
<td>5n R.C</td>
<td>(l₁, ..., lₙ)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Roles</th>
<th>Atomic</th>
<th>Inverse</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R</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, ..., (unary predicates)
role names R, S, ..., (binary predicates)

Constructors:
- ¬C                   (negation)
- C ∧ D                 (conjunction)
- C ∨ D                 (disjunction)
- ∃R.C                 (existential restriction)
- ∀R.C                 (value restriction)

Abbreviations:
- C → D = ¬C ∨ D        (implication)
- C ↔ D = C → D        (bi-implication)
- T = (A ∨ ¬A)         (top concept)
- ⊥ = A ∧ ¬A           (bottom concept)

Examples

- Person ⊓ Female
- Person ⊓ ∃Attends.Course
- Person ⊓ ∀Attends.(Course → ¬Easy)
- Person ⊓ ∃teaches.(Course ⊓ ∀attended-by.(Bored ⊓ Sleeping))
Interpretations

Semantics based on interpretations $(\Delta^\mathcal{I}, \cdot^\mathcal{I})$, where

- $\Delta^\mathcal{I}$ is a non-empty set (the domain)
- $\cdot^\mathcal{I}$ is the interpretation function mapping
each concept name $A$ to a subset $A^\mathcal{I}$ of $\Delta^\mathcal{I}$ and
each role name $R$ to a binary relation $R^\mathcal{I}$ over $\Delta^\mathcal{I}$.

Intuition: interpretation is complete description of the world

Technically: interpretation is first-order structure

with only unary and binary predicates

Semantics of Complex Concepts

$(\neg C)^\mathcal{I} = \Delta^\mathcal{I}\setminus C^\mathcal{I}$
$(C \cap D)^\mathcal{I} = C^\mathcal{I} \cap D^\mathcal{I}$
$(C \cup D)^\mathcal{I} = C^\mathcal{I} \cup D^\mathcal{I}$
$(\exists R.C)^\mathcal{I} = \{d \mid \text{there is an } e \in \Delta^\mathcal{I} \text{ with } (d,e) \in R^\mathcal{I} \text{ and } e \in C^\mathcal{I}\}$
$(\forall R.C)^\mathcal{I} = \{d \mid \text{for all } e \in \Delta^\mathcal{I}, (d,e) \in R^\mathcal{I} \text{ implies } e \in C^\mathcal{I}\}$

Example

TBoxes

Capture an application’s terminology means defining concepts

TBoxes are used to store concept definitions:

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

Semantics:
interpretation $\mathcal{I}$ satisfies $A \equiv C$ 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 $\equiv$ Person $\sqcap \exists$teaches.Course

Yields two kinds of concept names: defined and primitive
**TBox: Example**

TBoxes are used as ontologies:

- Woman ≡ Person ⊓ Female
- Man ≡ Person ⊓ ¬Woman
- Lecturer ≡ Person ⊓ ∃teaches.Course
- Student ≡ Person ⊓ ∃attends.Course
- BadLecturer ≡ Person ⊓ ∀teaches.(Course → Boring)

**Reasoning Tasks — Subsumption**

\[ C \subseteq_T D \] (written \( C \subseteq_T D \))

iff

\[ C^\downarrow \subseteq D^\downarrow \] holds for all models \( \mathcal{I} \) of \( T \)

**Intuition:** If \( C \subseteq_T D \), then \( D \) is more general than \( C \).

**Example:**

- Lecturer ≡ Person ⊓ ∃teaches.Course
- Student ≡ Person ⊓ ∃attends.Course
- Then

  Lecturer ⊓ ∃attends.Course \( \subseteq_T \) Student

**TBox: Example II**

A TBox restricts the set of admissible interpretations.

- Lecturer ≡ Person ⊓ ∃teaches.Course
- Student ≡ Person ⊓ ∃attends.Course

**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

\[ C \text{ is satisfiable w.r.t. } \mathcal{T} \iff \mathcal{T} \text{ has a model with } C \neq \emptyset \]

Intuition: If unsatisfiable, the concept contains a contradiction.

Example: \( \text{Woman} \sqsubseteq \text{Person} \sqcap \text{Female} \)
\( \text{Man} \sqsubseteq \text{Person} \sqcap \neg \text{Woman} \)

Then \( \exists \text{Sibling}. \text{Man} \sqcap \neg \exists \text{Sibling}. \text{Woman} \) is unsatisfiable w.r.t. \( \mathcal{T} \)

Subsumption can be reduced to (un)satisfiability and vice versa:

- \( C \sqsubseteq_T D \) iff \( C \sqcap \neg D \) is not satisfiable w.r.t. \( \mathcal{T} \)
- \( C \) is satisfiable w.r.t. \( \mathcal{T} \) if not \( C \sqsubseteq_T \bot \).

Many reasoners decide satisfiability rather than subsumption.

Definitorial TBoxes

A primitive interpretation for TBox \( \mathcal{T} \) interpretes

- the primitive concept names in \( \mathcal{T} \)
- all role names

A TBox is called definitorial 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 definitorial

\( \text{Person} \sqsubseteq \exists \text{parent}. \text{Person} \)

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

A TBox $T$ is acyclic if there are no definitional cycles:

$$\text{Lecturer} \sqsubseteq \text{Person} \sqcap \exists \text{Teaches.Course}$$
$$\text{Course} \sqsubseteq \exists \text{Has-title.Title} \sqcap \exists \text{Tought-by.Lecturer}$$

Expansion of acyclic TBox $T$:

exhaustively replace defined concept names with their definition
(terminates due to acyclicity)

Acyclic TBoxes are always definitional:

first expand, then set $A^I := C^I$ for all $A \sqsubseteq C \in T$

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. $\text{Course} \sqcap \exists \text{Attended-by.Sleeping} \sqsubseteq \exists \text{Boring}$

Interpretation $I$ is model of general TBox $T$ if

$$C^I \sqsubseteq D^I \text{ for all } C \sqsubseteq D \in T.$$  

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

e.g. $\exists \text{Has-favourite.SoccerTeam} \sqsubseteq \exists \text{Has-favourite.Beer}$

Note: $C \sqsubseteq D$ equivalent to $T \models C \rightarrow D$

Acyclic TBoxes II

For reasoning, acyclic TBox can be eliminated

- to decide $C \sqsubseteq D$ with $T$ acyclic,
  - expand $T$
  - replace defined concept names in $C, D$ with their definition
  - decide $C \sqsubseteq D$
- analogously for satisfiability

May yield an exponential blow-up:

$$A_0 \sqsubseteq \forall r.A_1 \sqcap \forall s.A_1$$
$$A_1 \sqsubseteq \forall r.A_2 \sqcap \forall s.A_2$$
$$\ldots$$
$$A_{n-1} \sqsubseteq \forall r.A_n \sqcap \forall s.A_n$$

ABoxes

ABoxes describe a snapshot of the world

An ABox is a finite set of assertions

$$a : C \quad (a \text{ individual name}, C \text{ concept})$$
$$(a, b) : R \quad (a, b \text{ individual names}, R \text{ role name})$$

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

Interpretations $I$ map each individual name $a$ to an element of $\Delta^I$.

$I$ satisfies an assertion

$$a : C \quad \text{iff} \quad a^I \in C^I$$
$$(a, b) : R \quad \text{iff} \quad (a^I, b^I) \in R^I$$

$I$ is a model for an ABox $\mathcal{A}$ if $I$ satisfies all assertions in $\mathcal{A}$. 
Note:
- interpretations describe the state if the world in a complete way
- ABoxes describe the state if the world in an incomplete way

(uli, dl-course) : taught-by uli : Female
does not imply
dl-course : ∀tought-by.Female

An ABox has many models!
An ABox constraints the set of admissible models similar to a TBox

### 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 $\alpha$, and a concept $C$
does $\alpha^2 \in C^2$ hold in all models of $\mathcal{A}$ and $\mathcal{T}$?

(written $\mathcal{A}, \mathcal{T} \models \alpha : C$)

The two tasks are interreducible:
- $\mathcal{A}$ consistent w.r.t. $\mathcal{T}$ iff $\mathcal{A}, \mathcal{T} \not\models \alpha : \bot$
- $\mathcal{A}, \mathcal{T} \models \alpha : C$ iff $\mathcal{A} \cup \{\alpha : \neg C\}$ is not consistent

### Example for ABox Reasoning

**ABox**
- dumbo : Mammal
t14 : Trunk
g23 : Darkgrey
(dumbo, t14) : bodypart
dumbo : color
(dumbo, g23) : color
dumbo : ∀color.Lightgrey

**TBox**
- Elephant $\sqsubset$ Mammal $\sqcap \exists$bodypart.Trunk $\sqcap \forall$color.Grey
- Grey $\sqsubseteq$ Lightgrey $\sqcup$ Darkgrey
- $\bot \equiv$ Lightgrey $\sqcap$ Darkgrey

1. ABox is inconsistent w.r.t. TBox.
2. dumbo 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 ALC concepts

**Goal:** design an algorithm which takes an 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 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:** $C_0 \subseteq C_1$ and $C_1 \subseteq C_0$

**NNF:** negation occurs only in front of concept names

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

$\neg(C \cap D) \iff \neg C \cup \neg D$
$\neg(C \cup D) \iff \neg C \cap \neg D$
$\neg\neg C \iff C$
$\neg\exists R.C \iff \exists R.\neg C$
$\neg\forall R.C \iff \forall R.\neg C$

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

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

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

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

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

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

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

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

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

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

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

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.
Proof of the Lemma: Termination

(1) **Termination** is an immediate consequence of these observations:

1. the c-tree is constructed in a *monotonic* way,
   each rule either adds nodes or extends node labels, nothing is removed
2. node labels are restricted to subsets of \( \text{sub}(C_0) \) and \( \# \text{sub}(C_0) \leq |C_0| \),
   at each position in \( C_0 \), at most one sub-concepts starts
3. the c-tree is of **bounded breadth** \( \leq |C_0| \),
   at most 1 successor for each \( \exists R.C \in \text{sub}(C_0) \)
4. the c-tree is of **bounded depth** \( \leq |C_0| \),
   the maximal depth of concepts in node labels decreases from a node to its successor,
   i.e., for \( y \) a successor of \( x \):
   \[ \max\{|C| | C \in \mathcal{L}(y)\} < \max\{|C| | C \in \mathcal{L}(x)\} \]

Proof of the Lemma: Soundness

(2) Let the algorithm stop with a complete and clash-free c-tree.

   From this, define an interpretation \( \mathcal{I} \) as follows:
   \[
   \Delta^\mathcal{I} := \{ x \mid x \text{ is a node in c-tree} \}
   \]
   \[
   A^\mathcal{I} := \{ x \mid A \in \mathcal{L}(x) \} \text{ for concept names } A
   \]
   \[
   R^\mathcal{I} := \{ (x, y) \mid y \text{ is an } R\text{-successor of } x \text{ in c-tree} \}
   \]
   and show, by induction on structure of concepts, for all \( x \in \Delta^\mathcal{I} \), \( D \in \text{sub}(C_0, \mathcal{I}) \):
   \[
   D \in \mathcal{L}(x) \text{ implies } x \in D^\mathcal{I}
   \]
   \[
   \text{concept names } D: \text{ by definition of } \mathcal{I}
   \]
   \[
   \text{for negated concept names } D: \text{ due to clash-freeness and induction}
   \]
   \[
   \text{for conjunctions/disjunctions/existential restrictions/universal restrictions } D:
   \text{ due to completeness and by induction}
   \]
   \[
   \text{since } C_0 \text{ is in label of root node, } \mathcal{I} \text{ is a model of } C_0
   \]

Proof of the Lemma: Completeness

(3) Let \( C_0 \) be satisfiable, and let \( \mathcal{I} \) be a model of it with \( a_0 \in C_0^\mathcal{I} \).

   Use \( \mathcal{I} \) to steer the application of the (only non-deterministic) \( \sqcup \)-rule:

   Inductively define a total mapping \( \pi \):
   start with \( \pi(x_0) = a_0 \), and show that each rule can be applied such that \((*)\) is preserved
   \[
   \text{(*) if } C \in \mathcal{L}(x), \text{ then } \pi(x) \in C^\mathcal{I}
   \]
   \[
   \text{if } y \text{ is an } R\text{-succ. of } x, \text{ then } \langle \pi(x), \pi(y) \rangle \in R^\mathcal{I}
   \]
   \[
   \text{easy for } \sqcap\text{- and } \forall\text{-rule,}
   \]
   \[
   \text{for } \exists\text{-rule, we need to extend } \pi \text{ to the newly created } R\text{-successor}
   \]
   \[
   \text{for } \sqcup\text{-rule, if } C_1 \sqcup C_2 \in \mathcal{L}(x), \text{ (*) implies that } \pi(x) \in (C_1 \sqcup C_2)^\mathcal{I}
   \]
   \[
   \text{we can choose } C_i \text{ with } \pi(x) \in C_i^\mathcal{I} \text{ to add to } \mathcal{L}(x) \text{ and thus preserve (*)}
   \]
   \[
   \text{easy to see: (*) implies that c-tree is clash-free}
   \]
Proof of the Lemma: Harvest

Look again at the model $I$ constructed for a clash-free, complete c-tree:

- $I$ is finite because c-tree has finitely many nodes
- $I$ is a tree because c-tree is a tree

Hence we get Corollary (3) – (5) for free from our proof:

$C_0$ is satisfiable
  $\rightsquigarrow$ tableau algorithm stops with clash-free, complete c-tree
  $\rightsquigarrow$ $C_0$ 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 $\Delta^I$ and
- if $C \subseteq D \in T$, then for each element $x$ in a model $I$ of $T$
  - 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 $E$

Completion rules for $\mathcal{ALC}$ with TBoxes

- $\forall$-rule: if $C_1 \cap C_2 \subseteq L(x)$ and $\{C_1, C_2\} \not\subseteq L(x)$
  then set $L(x) = L(x) \cup \{C_1, C_2\}$

- $\exists$-rule: if $\exists S.C \subseteq L(x)$ and $x$ has no $S$-successor $y$ with $C \in L(y)$,
  then create a new node $y$ with $L((x, y)) = \{S\}$ and $L(y) = \{C\}$

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

- $T$-rule: if $C_1 \subseteq C_2 \in T$ and $\text{NNF}(\neg C_1 \sqcup C_2) \not\subseteq L(x)$
  then set $L(x) = L(x) \cup \{\text{NNF}(\neg C_1 \sqcup C_2)\}$
A tableau algorithm for \(\mathcal{ALC}\) with general TBoxes

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

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 \(x\)

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

\(\rightarrow\) completion rules for \(\mathcal{ALC}\) w.r.t. TBoxes

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

\[ x \cdot \{C_1 \sqcap C_2, \ldots\} \rightarrow_\sqcap x \cdot \{C_1 \sqcap C_2, C_1, C_2, \ldots\} \]

\[ x \cdot \{C_1 \sqcup C_2, \ldots\} \rightarrow_\sqcup x \cdot \{C_1 \sqcup C_2, C, \ldots\} \quad \text{for } C \in \{C_1, C_2\} \]

\[ x \cdot \{\exists R. C, \ldots\} \rightarrow_\exists x \cdot \{\exists R. C, \ldots\} \]

\[ x \cdot \forall R. C, \ldots \rightarrow_\forall x \cdot \forall R. C, \ldots \]

\[ R \downarrow x \cdot \{\ldots\} \quad y \cdot \{C\} \]

\[ R \downarrow y \cdot \{\ldots\} \quad x \cdot \{\forall R. C, \ldots\} \]
### Tableaux Rule for Transitive Roles

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

- 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 $\exists S.C \land \forall S. (\neg C \cup \neg D) \land \exists R.C \land \forall R. (\exists R.C)$} where $R$ is a transitive role

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

Tableaux Algorithm — Example

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

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

$L(x) = \{C\}$

Tableaux Algorithm — Example

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

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

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

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

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

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

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

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

\[\text{Tableaux Algorithm — Example}\]

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

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

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

\[\text{clash}\]

\[\text{Tableaux Algorithm — Example}\]

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

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

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

\[\text{clash}\]
Tableaux Algorithm — Example

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

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

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

Tableaux Algorithm — Example

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

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

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

Tableaux Algorithm — Example

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

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

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

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

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

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

\[ \mathcal{L}(x) = \{ C, (-C \lor \neg D), \neg D \} \]

\[ \mathcal{L}(y) = \{ C \} \]

Tableaux Algorithm — Example

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

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

\[ \mathcal{L}(x) = \{ C, (-C \lor \neg D), \neg D \} \]

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

Reasoning with Expressive Description Logics – p. 7/27

61

62

63

64
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

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

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

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

\[
L(z) = \{ C \}
\]

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

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

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

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

\[
L(z) = \{ C, \exists R.C, \forall R.(\exists R.C) \}
\]

Concept is satisfiable: \( T \) corresponds to model
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

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

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

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

Concept is satisfiable: $T$ corresponds to model

Proof of the Lemma: Termination

(1) termination is, again, due to the following properties: let $n = |C_0| + |T|$ and $\text{sub}(C_0, T) = \text{sub}(C_0) \cup \bigcup_{C \subseteq D \in T} \text{sub}(C) \cup \text{sub}(D)$

1. the c-tree is built in a monotonic way:
   each rule either extends node labels or adds a node (with a label)
2. node labels are restricted to subsets of $\text{sub}(C_0, T)$ and $\# \text{sub}(C_0, T) \leq n$
3. the breadth of the c-tree is bounded by $n$:
   at most 1 successor per $\exists R.C \in \text{sub}(C_0, T)$
4. the depth of the c-tree is bounded:
   on a path of length $2^n$, blocking occurs, and thus it does not get longer

Important: in the presence of TBoxes, c-tree can be of exponential depth whereas without TBoxes, depth was linearly bounded

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

Lemma: Let $T$ be a general $\mathcal{ALC}$-Tbox and $C_0$ an $\mathcal{ALC}$-concept. Then
1. the algorithm terminates when applied to $T$ 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. $T$.

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

Proof of the Lemma: Soundness

(2) let the algorithm stop with a complete and clash-free c-tree.
Again, from this, we define an interpretation:

$\Delta^T := \{x \mid x$ is a node in $T$, $x$ is not blocked\}$

$A^T := \{x \in \Delta^T \mid A \in \mathcal{L}(x)\}$ for concept names $A$

$R^T := \{(x, y) \in \Delta^T^2 \mid y$ is an $R$-succ of $x$ in c-tree or $y$ blocks an $R$-succ of $x$ in c-tree\}$

and show, by induction on the structure of concepts, for all $x \in \Delta^T$, $D \in \text{sub}(C_0, T)$:

$D \in \mathcal{L}(x)$ implies $x \in D^T$.

This implies that $\mathcal{I}$ is indeed a model of $C_0$ and $T$ because
(a) $C_0$ is in the label of the root node which cannot be blocked (!) and
(b) $\neg C \lor D$ is in the label of each node, for each $C \subseteq D \in T$
Proof of the Lemma: Completeness

(3) Let $C_0$ be satisfiable w.r.t. $T$ and $I$ a model of them with $a_0 \in C^I_0$.
Use $I$ to steer the application of the (only non-deterministic) $\sqcup$-rule:

Inductively define a total mapping $\pi: \text{nodes of completion tree} \to \Delta^I$,
start with $\pi(x_0) = a_0$, and show that

each rule can be applied in such a way that $(*)$ is preserved

if $C \in \mathcal{L}(x)$, then $\pi(x) \in C^I$ $(*)$
if $y$ is an $R$-succ. of $x$, then $(\pi(x), \pi(y)) \in R^I$

• easy for $\forall$-, $\exists$-, and $\forall$-rule,
• for $\exists$-rule, we need to extend $\pi$ to the newly created $R$-successor
• for $\sqcup$-rule, if $C_1 \sqcup C_2 \in \mathcal{L}(x)$, $(*)$ implies that $\pi(x) \in (C_1 \sqcup C_2)^I$
  $\leadsto$ we can choose $C_i$ with $\pi(x) \in C_i^I$ to add to $\mathcal{L}(x)$ and thus preserve $(*)$
  $\leadsto$ easy to see: $(*)$ implies that c-tree is clash-free

Proof of the Lemma: Harvest

Look again at the model $I$ constructed for a clash-free, complete c-tree:

$I$ is
- finite because c-tree has finitely many nodes
- but it is not a tree if blocking occurs

Hence we get Corollary (2) for free from our proof:

$C_0$ is satisfiable
$\leadsto$ tableau algorithm stops with clash-free, complete c-tree
  $\leadsto$ $C_0$ has a finite model.

To obtain Corollary (3), the tree model property, we must work a bit more:

$\leadsto$ build the model in a different way, “unravel” the c-tree into an infinite tree
  intuitively, instead of going to a blocked node, go to a copy of its blocking node

What next?

Next, we could
- discuss implementation issues for our tableau algorithms, e.g.,
  - datastructures,
  - more efficient (i.e., less strict) blocking conditions,
  - a good strategy for the order of rule applications,
  - how to “determinise” our non-deterministic algorithm: e.g., backtracking
  - etc.
- discuss other reasoning techniques for DLs
- analyse computational complexity of DLs
- further extend our tableau algorithm for more expressive DLs
  with one more expressive means

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 $\rightarrow_L$-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) 2NExpTime,
- can be implemented in various ways,
  - order/priorities of rules
  - data structure
  - etc.
- is amenable to optimisations – more on this next week
Naive Implementations

Problems include:
☞ Space usage
  • Storage required for tableaux datastructures
  • Rarely a serious problem in practice
  • But problems can arise with inverse roles and cyclical KBs
☞ Time usage
  • Search required due to non-deterministic expansion
  • Serious problem in practice
  • Mitigated by:
    – Careful choice of algorithm
    – Highly optimised implementation

Careful Choice of Algorithm

☞ Transitive roles instead of transitive closure
  • Deterministic expansion of $\exists R.C$, even when $R \in R_+$
  • (Relatively) simple blocking conditions
  • Cycles always represent (part of) valid cyclical models
☞ Direct algorithm/implementation instead of encodings
  • GCI axioms can be used to “encode” additional operators/axioms
  • Powerful technique, particularly when used with FL closure
  • Can encode cardinality constraints, inverse roles, range/domain,
    ...
    – E.g., (domain $R.C$) $\equiv \exists R.T \subseteq C$
  • (FL) encodings introduce (large numbers of) axioms
  • BUT even simple domain encoding is disastrous with large numbers of roles

Dependency Directed Backtracking

☞ Allows rapid recovery from bad branching choices
☞ Most commonly used technique is backjumping
  • Tag concepts introduced at branch points (e.g., when expanding disjunctions)
  • Expansion rules combine and propagate tags
  • On discovering a clash, identify most recently introduced concepts involved
  • Jump back to relevant branch points without exploring alternative branches
  • Effect is to prune away part of the search space
☞ Highly effective — essential for usable system
  • E.g., GALEN KB, 30s (with) → months++ (without)

Backjumping

E.g., if $\exists R.\neg A \sqcap \forall R.(A \sqcap B) \sqcap (C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n) \subseteq L(x)$
Backjumping

E.g., if $\exists R. \neg A \land \forall R. (A \land B) \land (C_1 \cup D_1) \land \ldots \land (C_n \cup D_n) \subseteq \mathcal{L}(x)$
E.g., if \( \exists R. \neg A \sqcap \forall R. (A \sqcap B) \sqcap (C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n) \subseteq L(x) \)

\[
\begin{align*}
L(x) \cup \{C_1\} & \quad \downarrow \\
L(x) \cup \{C_{n-1}\} & \quad \downarrow \\
L(x) \cup \{C_n\} & \quad \downarrow \\
L(y) & = \{(A \sqcap B), \neg A, A, B\} \\
\end{align*}
\]

\( \text{clash} \)

E.g., if \( \exists R. \neg A \sqcap \forall R. (A \sqcap B) \sqcap (C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n) \subseteq L(x) \)

\[
\begin{align*}
L(x) \cup \{C_1\} & \quad \downarrow \\
L(x) \cup \{C_{n-1}\} & \quad \downarrow \\
L(x) \cup \{C_n\} & \quad \downarrow \\
L(y) & = \{(A \sqcap B), \neg A, A, B\} \\
\end{align*}
\]

\( \text{clash} \)

E.g., if \( \exists R. \neg A \sqcap \forall R. (A \sqcap B) \sqcap (C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n) \subseteq L(x) \)

\[
\begin{align*}
L(x) \cup \{C_1\} & \quad \downarrow \\
L(x) \cup \{C_{n-1}\} & \quad \downarrow \\
L(x) \cup \{C_n\} & \quad \downarrow \\
L(y) & = \{(A \sqcap B), \neg A, A, B\} \\
\end{align*}
\]

\( \text{clash} \)

E.g., if \( \exists R. \neg A \sqcap \forall R. (A \sqcap B) \sqcap (C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n) \subseteq L(x) \)

\[
\begin{align*}
L(x) \cup \{C_1\} & \quad \downarrow \\
L(x) \cup \{C_{n-1}\} & \quad \downarrow \\
L(x) \cup \{C_n\} & \quad \downarrow \\
L(y) & = \{(A \sqcap B), \neg A, A, B\} \\
\end{align*}
\]

\( \text{clash} \)

\( \ldots \ldots \)

\[
\begin{align*}
\text{Pruning} & \quad \text{Backjump}
\end{align*}
\]
Inverse Roles

Consider the following TBox

\[
\begin{align*}
\text{Control-rod} & \sqsubseteq \text{Device} \sqcap \exists \text{part-of}.\text{Reactor-core} \\
\text{Reactor-core} & \sqsubseteq \text{Device} \sqcap \exists \text{has-part}.\text{Control-rod} \sqcap \\
& \text{\exists \text{part-of}.\text{N-reactor}},
\end{align*}
\]

\(\text{Reactor-core} \sqcap \exists \text{has-part}.\text{Faulty} \sqsubseteq \text{Dangerous},\)

Now, w.r.t. such a TBox, we find that

\(\text{Control-rod} \sqcap \text{Faulty} \) should be subsumed by \(\exists \text{part-of}.\text{Dangerous}\)

But this is not true: no interaction between part-of and has-part!

\(\sim\) also allow for \(\exists R^- . C \) and \(\forall R^- . C, \) where \((R^-) = \{\langle y, x \rangle \mid \langle x, y \rangle \in R^2\}\)

A tableau algorithm for \(\mathcal{ALCI}\) with general TBoxes

\(\mathcal{ALCI}\) is the extension of \(\mathcal{ALC}\) with inverse roles \(R^-\) in the place of role names:

\((R^-) = \{\langle y, x \rangle \mid \langle x, y \rangle \in R^2\}\)

Example: does \(\forall \text{parent}.\forall \text{child}.\text{Blond} \sqsubseteq \text{Blond}\) w.r.t. \{\(T \sqsubseteq \exists \text{parent}.T\)?

does \(\forall \text{parent}.\forall \text{parent}^- . \text{Blond} \sqsubseteq \text{Blond}\) w.r.t. \{\(T \sqsubseteq \exists \text{parent}.T\)?

Example: is \(C_0 = \exists R . \exists S . \exists T. A\) satisf. w.r.t. \{\(C \sqsubseteq \exists R.C \sqcap \forall R.B\)

\(T \sqsubseteq \forall T^- . \forall S^- . \forall R^- . C\)?

Clear: inverse roles \(\sim\) tableau algorithm must reason \textit{up and down edges}

A tableau algorithm for \(\mathcal{ALCI}\) with general TBoxes

\begin{itemize}
\item \(\square\)-rule: if \(C_1 \sqcap C_2 \in L(x), \{C_1, C_2\} \not\subseteq L(x), \) and \(x\) is not blocked
\then \(L(x) = L(x) \cup \{C_1, C_2\}\)
\item \(\square\)-rule: if \(C_1 \sqcup C_2 \in L(x), \{C_1, C_2\} \cap L(x) = \emptyset, \) and \(x\) is not blocked
\then \(L(x) = L(x) \cup \{C\} \) for some \(C \in \{C_1, C_2\}\)
\item \(\exists\)-rule: if \(\exists S . C \in L(x), \) \(x\) has no \(S\)-neighbour \(y\) with \(C \subseteq L(y),\)
\and \(x\) is not blocked
\then create a new node \(y\) with \(L((x, y)) = \{S\}\) and \(L(y) = \{C\}\)
\item \(\forall\)-rule: if \(\forall S . C \in L(x), \) there is an \(S\)-neighbour \(y\) of \(x\) with \(C \not\subseteq L(y),\)
\and \(x\) is not indirectly blocked
\then \(L(y) = L(y) \cup \{C\}\)
\item \(T\)-rule: if \(C_1 \sqsubseteq C_2 \in T, \) \(\text{NNF}(\neg C_1 \sqcup C_2) \not\subseteq L(x),\)
\and \(x\) is not blocked
\then \(L(x) = L(x) \cup \{\text{NNF}(\neg C_1 \sqcup C_2)\}\)
\end{itemize}
A tableau algorithm for ALCI with general TBoxes

Example: is $A$ satisfiable w.r.t. $\{ A \sqsubseteq \exists R^- A \sqcap \forall R.(\neg A \sqcup \exists S.B) \}$?

Example: is $\exists R.B$ satisfiable w.r.t. $\{ B \sqsubseteq \exists R.B \sqcap \forall R^- \forall R^- \bot \}$?

Problem: algorithm returns “satisfiable” for unsatisfiable input $\Rightarrow$ incorrect!

Reason: blocking condition $L(y') \subseteq L(y)$ is too loose: universal value restrictions from blocking node may be violated

Solution: tighten blocking condition to $L(y') = L(y)$

---

Proof of the Lemma: Soundness

(2) let the algorithm stop with a complete and clash-free c-tree. Again, from this, we define an interpretation:

$\Delta^T := \{ x | x \text{ is a node in } T, x \text{ is not blocked} \}$

$A^T := \{ x \in \Delta^T | A \in L(x) \}$ for concept names $A$

$R^T := \{ (x, y) \in \Delta^T | y \text{ is an } R\text{-succ of } x \text{ or } y \text{ blocks an } R\text{-succ of } x \text{ or } x \text{ is an } R^-\text{-succ of } y \text{ or } x \text{ blocks an } R^-\text{-succ of } y \}$

and show, by induction on the structure of concepts, for all $x \in \Delta^T$, $D \in \text{sub}(C_0, T)$:

$D \in L(x) \Rightarrow x \in D^T$.

As for $\mathcal{ALC}$, this implies that $I$ is indeed a model of $C_0$ and $T$

---

Proof of the Lemma: Completeness

(3) completely identical to the $\mathcal{ALC}$ case...

That’s it!

I hope you got an idea of how we can

• build tableau algorithms for description logics and
• see that they do indeed what we want them to do,
  i.e., decide satisfiability
Research Challenges

Challenges

☞ Increased expressive power
  • Existing DL systems implement (at most) $SHIQ$
  • OWL extends $SHIQ$ 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

 Increased Expressive Power: Datatypes

☞ OWL has simple form of datatypes
  • Unary predicates plus disjoint object-class/datatype domains

☞ Well understood theoretically
  • Existing work on concrete domains [Baader & Hanschke, Lutz]
  • Algorithm already known for $SHOQ(D)$ [Horrocks & Sattler]
  • Can use hybrid reasoning (DL reasoner + datatype “oracle”)

☞ May be practically challenging
  • All XMLS datatypes supported (?)

☞ Already seeing some (partial) implementations
  • Cerebra system (Network Inference), Racer system (Hamburg)

Increased Expressive Power: Nominals

☞ OWL oneOf constructor equivalent to hybrid logic nominals
  • Extensionally defined concepts, e.g., EU $\equiv \{\text{France, Italy, \ldots}\}$

☞ Theoretically very challenging
  • Resulting logic has known high complexity (NExpTime)
  • No known “practical” algorithm
  • Not obvious how to extend tableaux techniques in this direction
    – Loss of tree model property
    – Spy-points: $\top \subseteq \exists R.\{Spy\}$
    – Finite domains: $\{Spy\} \subseteq \leq nR^-$

☞ Standard solution is weaker semantics for nominals
  • Treat nominals as (disjoint) primitive classes
  • Loss of completeness/soundness
Increased Expressive Power: Extensions

- **OWL not expressive enough** for all applications
- Extensions **wish list** includes:
  - Feature chain (path) agreement, e.g., output of component of composite process equals input of subsequent process
  - Complex roles/role inclusions, e.g., a city located in part of a country is located in that country
  - Rules—proposal(s) already exist for “datalog/LP style rules”
  - Temporal and spatial reasoning
- ...  
- May be impossible/undesirable to resist such extensions
- Extended language sure to be **undecidable**
- How can extensions best be **integrated** with OWL?
- How can reasoners be developed/adapted for extended languages
  - Some existing work on language **fusions** and **hybrid** reasoners

Performance Solutions (Maybe)

- Excessive **memory usage**
  - Problem exacerbated by over-cautious double blocking condition (e.g., root node can never block)
  - Promising results from more precise blocking condition [Sattler & Horrocks]
- **Qualified number restrictions**
  - Problem exacerbated by naive expansion rules
  - Promising results from optimised expansion using Algebraic Methods [Haarslev & Möller]
- **Caching** and merging
  - Can still work in some situations (work in progress)
- Reasoning with **very large KBs**
  - DL systems shown to work with ≈100k concept KB [Haarslev & Möller]
  - But KB only exploited small part of DL language

Scalability

- Reasoning **hard** (ExpTime) even without nominals (i.e., $\mathcal{SHIQ}$)
- Web ontologies may grow **very large**
- Good **empirical evidence** of scalability/tractability for DL systems
  - E.g., 5,000 (complex) classes; 100,000+ (simple) classes
  - But evidence mostly w.r.t. $\mathcal{SHF}$ (no inverse)
- **Problems** can arise when $\mathcal{SHF}$ extended to $\mathcal{SHIQ}$
  - Important **optimisations** no longer (fully) work
- Reasoning with **individuals**
  - Deployment of web ontologies will mean reasoning with (possibly very large numbers of) individuals/tuples
  - Unlikely that standard **Abox** techniques will be able to cope

Other Reasoning Tasks

- **Querying**
  - Retrieval and instantiation wont be sufficient
  - Minimum requirement will be **DB style query language**
  - May also need “what can I say about $x$?” style of query
- **Explanation**
  - To support ontology design
  - Justifications and proofs (e.g., of query results)
- **“Non-Standard Inferences”**, e.g., LCS, matching
  - To support ontology integration
  - To support “bottom up” design of ontologies
**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**

- 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/)

**Select Bibliography**


