Description Logic Reasoning
Basic Inference Problems
Basic Inference Problems

☞ **Subsumption** — check knowledge is correct

- $C \subseteq_K D \ ? \ C^I \subseteq D^I$ in all models $I$ of $K$

Reasoning with Expressive Description Logics – p. 2/27
Basic Inference Problems

☞ **Subsumption** — check knowledge is correct
  
  • \( C \subseteq_{\mathcal{K}} D \) ? \( C^\mathcal{I} \subseteq D^\mathcal{I} \) in all models \( \mathcal{I} \) of \( \mathcal{K} \)

☞ **Equivalence** — check knowledge is minimally redundant
  
  • \( C \equiv_{\mathcal{K}} D \) ? \( C^\mathcal{I} = D^\mathcal{I} \) in all models \( \mathcal{I} \) of \( \mathcal{K} \)
Basic Inference Problems

- **Subsumption** — check knowledge is correct
  - $C \subseteq_{\mathcal{K}} D \ ? \ C^I \subseteq D^I$ in all models $\mathcal{I}$ of $\mathcal{K}$

- **Equivalence** — check knowledge is minimally redundant
  - $C \equiv_{\mathcal{K}} D \ ? \ C^I = D^I$ in all models $\mathcal{I}$ of $\mathcal{K}$

- **Consistency** — check knowledge is meaningful
  - $C \equiv \bot \ C^I \neq \emptyset$ in some model $\mathcal{I}$ of $\mathcal{K}$
Basic Inference Problems

☞ **Subsumption** — check knowledge is correct
  - \( C \subseteq_{\mathcal{K}} D \) ? \( C^I \subseteq D^I \) in all models \( \mathcal{I} \) of \( \mathcal{K} \)

☞ **Equivalence** — check knowledge is minimally redundant
  - \( C \equiv_{\mathcal{K}} D \) ? \( C^I = D^I \) in all models \( \mathcal{I} \) of \( \mathcal{K} \)

☞ **Consistency** — check knowledge is meaningful
  - \( C \equiv \bot \) \( C^I \neq \emptyset \) in some model \( \mathcal{I} \) of \( \mathcal{K} \)

☞ **Instantiation** — check if individual \( i \) instance of class \( C \)
  - \( i \in_{\mathcal{K}} C \) ? \( i \in C^I \) in all models \( \mathcal{I} \) of \( \mathcal{K} \)
Basic Inference Problems

- **Subsumption** — check knowledge is correct
  - \( C \subseteq_K D \) ? \( C^I \subseteq D^I \) in all models \( \mathcal{I} \) of \( \mathcal{K} \)

- **Equivalence** — check knowledge is minimally redundant
  - \( C \equiv_K D \) ? \( C^I = D^I \) in all models \( \mathcal{I} \) of \( \mathcal{K} \)

- **Consistency** — check knowledge is meaningful
  - \( C \equiv \bot \) \( C^I \neq \emptyset \) in some model \( \mathcal{I} \) of \( \mathcal{K} \)

- **Instantiation** — check if individual \( i \) instance of class \( C \)
  - \( i \in_K C \) ? \( i \in C^I \) in all models \( \mathcal{I} \) of \( \mathcal{K} \)

Problems all **reducible** to KB consistency (satisfiability):
- e.g., \( C \subseteq_K D \) iff \( C \cap \neg D \) not consistent w.r.t. \( \mathcal{K} \)
Basic Inference Problems

- **Subsumption** — check knowledge is correct
  - $C \sqsubseteq_K D$? $C^\mathcal{I} \subseteq D^\mathcal{I}$ in all models $\mathcal{I}$ of $\mathcal{K}$

- **Equivalence** — check knowledge is minimally redundant
  - $C \equiv_K D$? $C^\mathcal{I} = D^\mathcal{I}$ in all models $\mathcal{I}$ of $\mathcal{K}$

- **Consistency** — check knowledge is meaningful
  - $C \equiv \bot$ $C^\mathcal{I} \neq \emptyset$ in some model $\mathcal{I}$ of $\mathcal{K}$

- **Instantiation** — check if individual $i$ instance of class $C$
  - $i \in_K C$? $i \in C^\mathcal{I}$ in all models $\mathcal{I}$ of $\mathcal{K}$

Problems all **reducible** to KB consistency (satisfiability):
- e.g., $C \sqsubseteq_K D$ iff $C \sqcap \neg D$ not consistent w.r.t. $\mathcal{K}$

KB consistency **reducible** to concept consistency via **internalisation**
  - For logics supporting, e.g., a transitive “top” role
Tableaux Algorithms — Basics

Tableaux algorithms used to test satisfiability. Try to build a tree-like model of the input concept. Work on concepts in negation normal form, pushing in negation using de Morgan's laws. Break down concepts syntactically, inferring constraints on elements of the input. Decomposition uses tableau rules corresponding to constructors in the logic (e.g., \( \land, \lor \)). Some rules are nondeterministic (e.g., \( \neg, \cdot \)). In practice, this means search. Stop when a clash occurs or when no rules are applicable. Blocking (cycle check) is used to guarantee termination. Return "C is consistent" if \( C \) is consistent. Tree model property.

Reasoning with Expressive Description Logics – p. 3/27
Tableaux algorithms used to test *satisfiability*
Tableaux Algorithms — Basics

- Tableaux algorithms used to test **satisfiability**
- Try to build **tree-like model** $\mathcal{I}$ of input concept $C$
Tableaux Algorithms — Basics

- Tableaux algorithms used to test **satisfiability**
- Try to build **tree-like model** $\mathcal{I}$ of input concept $C$
- Work on concepts in **negation normal form**
  - Push in negation using de Morgan’s, $\neg \exists R. C \iff \forall R. \neg C$ etc.
Tableaux Algorithms — Basics

- Tableaux algorithms used to test **satisfiability**
- Try to build **tree-like model** \( \mathcal{I} \) of input concept \( C \)
- Work on concepts in **negation normal form**
  - Push in negation using de Morgan’s, \( \neg \exists R. C \iff \forall R. \neg C \) etc.
- Break down \( C \) **syntactically**, inferring constraints on elements of \( \mathcal{I} \)
Tableaux Algorithms — Basics

- Tableaux algorithms used to test **satisfiability**
- Try to build **tree-like model** $\mathcal{I}$ of input concept $C$
- Work on concepts in **negation normal form**
  - Push in negation using de Morgan’s, $\neg \exists R.C \iff \forall R.\neg C$ etc.
- Break down $C$ **syntactically**, inferring constraints on elements of $\mathcal{I}$
- Decomposition uses **tableau rules** corresponding to constructors in logic (e.g., $\sqcap$, $\exists$)
  - Some rules are **nondeterministic** (e.g., $\sqcup$, $\leq$)
  - In practice, this means **search**
Tableaux Algorithms — Basics

- Tableaux algorithms used to test satisfiability
- Try to build tree-like model \( \mathcal{I} \) of input concept \( C \)
- Work on concepts in negation normal form
  - Push in negation using de Morgan’s, \( \neg \exists R.C \leftrightarrow \forall R.\neg C \) etc.
- Break down \( C \) syntactically, inferring constraints on elements of \( \mathcal{I} \)
- Decomposition uses tableau rules corresponding to constructors in logic (e.g., \( \sqcap \), \( \exists \))
  - Some rules are nondeterministic (e.g., \( \sqcup \), \( \leq \))
  - In practice, this means search
- Stop when clash occurs or when no rules are applicable
Tableaux Algorithms — Basics

- Tableaux algorithms used to test **satisfiability**
- Try to build **tree-like model** $\mathcal{I}$ of input concept $C$
- Work on concepts in **negation normal form**
  - Push in negation using de Morgan’s, $\neg \exists R. C \iff \forall R. \neg C$ etc.
- Break down $C$ **syntactically**, inferring constraints on elements of $\mathcal{I}$
- Decomposition uses **tableau rules** corresponding to constructors in logic (e.g., $\sqcap$, $\exists$)
  - Some rules are **nondeterministic** (e.g., $\sqcup$, $\leq$)
  - In practice, this means **search**
- Stop when **clash** occurs or when no rules are applicable
- **Blocking** (cycle check) used to guarantee **termination**
Tableaux Algorithms — Basics

- Tableaux algorithms used to test **satisfiability**
- Try to build **tree-like model** $\mathcal{I}$ of input concept $C$
- Work on concepts in **negation normal form**
  - Push in negation using de Morgan’s, $\neg\exists R.C \iff \forall R.\neg C$ etc.
- Break down $C$ **syntactically**, inferring constraints on elements of $\mathcal{I}$
- Decomposition uses **tableau rules** corresponding to constructors in logic (e.g., $\sqcap$, $\exists$)
  - Some rules are **nondeterministic** (e.g., $\sqcup$, $\leq$)
  - In practice, this means **search**
- Stop when **clash** occurs or when no rules are applicable
- **Blocking** (cycle check) used to guarantee **termination**
- Return “$C$ is consistent” **iff** $C$ is consistent
  - Tree model property
Tableaux Algorithms — Details

Tableaux are algorithms for reasoning in description logics. They involve constructing a tree (T) that represents a model of a concept (C). Nodes in the tree represent elements of the model (I), and are labeled with subconcepts of the concept. Edges in the tree represent role-successorships between elements of the model.

The tree T is initialized with a single root node labeled f_C g, where f and g are functions that map concepts to their intended interpretations.

Tableaux rules are repeatedly applied to the node labels. These rules can extend labels or extend/modify the structure of T. Some rules may be blocked, for example, if a predecessor node has a superset label.

Nondeterministic rules are searched for possible extensions of T.

A clash occurs in T if there is an obvious contradiction in some node label. For example, f_A g L(x) for some concept A and node x.

T is fully expanded if no rules are applicable.

The concept C is satisfiable if T is fully expanded and clash-free, meaning that a trivial correspondence exists between the tableau and a model of the concept.
Work on tree $T$ representing model $\mathcal{I}$ of concept $C$:
- Nodes represent elements of $\Delta^\mathcal{I}$; labeled with subconcepts of $C$.
- Edges represent role-successorships between elements of $\Delta^\mathcal{I}$.

$T$ is initialized with a single root node labeled $f_C$.

Tableau rules are repeatedly applied to node labels:
- Extend labels or extend/modify $T$ structure.
- Rules can be blocked, e.g., if the predecessor has a superset label.

Nondeterministic rules allow searching possible extensions.

$T$ contains a clash if an obvious contradiction occurs in some node label.
- E.g., $f_A; g_L(x)$ for some concept $A$ and node $x$.

$T$ is fully expanded if no rules are applicable.

$C$ is satisfiable if $T$ is fully expanded and clash free.

Trivial correspondence between such a $T$ and a model of $C$. 

Reasoning with Expressive Description Logics – p. 4/27
Tableaux Algorithms — Details

- Work on tree $T$ representing model $\mathcal{I}$ of concept $C$
  - Nodes represent elements of $\Delta^\mathcal{I}$; labeled with subconcepts of $C$
  - Edges represent role-successorships between elements of $\Delta^\mathcal{I}$

- $T$ initialised with single root node labeled $\{C\}$
Tableaux Algorithms — Details

- Work on tree $T$ representing model $\mathcal{I}$ of concept $C$
  - Nodes represent elements of $\Delta^\mathcal{I}$; labeled with subconcepts of $C$
  - Edges represent role-successorships between elements of $\Delta^\mathcal{I}$
- $T$ initialised with single root node labeled $\{C\}$
- Tableau rules repeatedly applied to node labels
  - Extend labels or extend/modify $T$ structure
  - Rules can be blocked, e.g., if predecessor has superset label
  - Nondeterministic rules $\rightarrow$ search possible extensions

Reasoning with Expressive Description Logics – p. 4/27
Tableaux Algorithms — Details

- Work on tree $T$ representing model $\mathcal{I}$ of concept $C$
  - Nodes represent elements of $\Delta^\mathcal{I}$; labeled with subconcepts of $C$
  - Edges represent role-successorships between elements of $\Delta^\mathcal{I}$
- $T$ initialised with single root node labeled $\{C\}$
- Tableau rules repeatedly applied to node labels
  - Extend labels or extend/modify $T$ structure
  - Rules can be blocked, e.g., if predecessor has superset label
  - Nondeterministic rules $\rightarrow$ search possible extensions
- $T$ contains Clash if obvious contradiction in some node label
  - E.g., $\{A, \neg A\} \subseteq \mathcal{L}(x)$ for some concept $A$ and node $x$
Tableaux Algorithms — Details

- Work on tree $T$ representing model $\mathcal{I}$ of concept $C$
  - Nodes represent elements of $\Delta^\mathcal{I}$; labeled with subconcepts of $C$
  - Edges represent role-successorships between elements of $\Delta^\mathcal{I}$
- $T$ initialised with single root node labeled $\{C\}$
- Tableau rules repeatedly applied to node labels
  - Extend labels or extend/modify $T$ structure
  - Rules can be blocked, e.g., if predecessor has superset label
  - Nondeterministic rules $\rightarrow$ search possible extensions
- $T$ contains Clash if obvious contradiction in some node label
  - E.g., $\{A, \neg A\} \subseteq \mathcal{L}(x)$ for some concept $A$ and node $x$
- $T$ fully expanded if no rules are applicable
Tableaux Algorithms — Details

- Work on tree $T$ representing model $I$ of concept $C$
  - Nodes represent elements of $\Delta^I$; labeled with subconcepts of $C$
  - Edges represent role-successorships between elements of $\Delta^I$
- $T$ initialised with single root node labeled $\{C\}$
- Tableau rules repeatedly applied to node labels
  - Extend labels or extend/modify $T$ structure
  - Rules can be blocked, e.g., if predecessor has superset label
  - Nondeterministic rules $\rightarrow$ search possible extensions
- $T$ contains Clash if obvious contradiction in some node label
  - E.g., $\{A, \neg A\} \subseteq \mathcal{L}(x)$ for some concept $A$ and node $x$
- $T$ fully expanded if no rules are applicable
- $C$ satisfiable iff fully expanded clash free $T$ found
  - Trivial correspondence between such a $T$ and a model of $C$
Tableaux Rules for $\mathcal{ALC}$
### Tableaux Rules for $\mathcal{ALC}$

<table>
<thead>
<tr>
<th>Rule</th>
<th>Left-hand side</th>
<th>Right-hand side</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $x \cdot {C_1 \sqcap C_2, \ldots}$</td>
<td>$\rightarrow \sqcap$</td>
<td>$x \cdot {C_1 \sqcap C_2, C_1, C_2, \ldots}$</td>
</tr>
<tr>
<td>2. $x \cdot {C_1 \sqcup C_2, \ldots}$</td>
<td>$\rightarrow \sqcup$</td>
<td>$x \cdot {C_1 \sqcap C_2, C, \ldots}$ for $C \in {C_1, C_2}$</td>
</tr>
</tbody>
</table>
| 3. $x \cdot \{\exists R.C, \ldots\}$ | $\rightarrow \exists$ | $x \cdot \{\exists R.C, \ldots\}$
| | | $\overbrace{\exists R.C, \ldots}$
| | | $R$
| | | $y \cdot \{C\}$ |
| 4. $x \cdot \{\forall R.C, \ldots\}$
| $R$
| $y \cdot \{\ldots\}$ | $\rightarrow \forall$ | $x \cdot \{\forall R.C, \ldots\}$
| | | $R$
| | | $y \cdot \{C, \ldots\}$ |
Tableaux Rule for Transitive Roles

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

\[
( R I )^+ = R I
\]

No longer naturally terminating (e.g., if \( C := R: > \))

Need blocking

Simple blocking suffices for 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 Rule for Transitive Roles

Where $R$ is a transitive role (i.e., $(R^\uparrow)^+ = R^\uparrow$)
Tableaux Rule for Transitive Roles

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

No longer naturally terminating (e.g., if $C = \exists R. \top$)
Tableaux Rule for Transitive Roles

\[
x \bullet \{ \forall R.C, \ldots \} \\
R \\
y \bullet \{ \ldots \}
\rightarrow_{\forall^+}
\]

\[
x \bullet \{ \forall R.C, \ldots \} \\
R \\
y \bullet \{ \forall R.C, \ldots \}
\]

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

Reasoning with Expressive Description Logics – p. 6/27
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
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

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

Reasoning with Expressive Description Logics – p. 7/27
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

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

Reasoning with Expressive Description Logics – p. 7/27
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

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

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

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

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

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

Reasoning with Expressive Description Logics – p. 7/27
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

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

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

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

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

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

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

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

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

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

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

$x$ 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

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

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

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

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

Reasoning with Expressive Description Logics – p. 7/27
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.
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

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

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

\[
\mathcal{L}(y) = \{C\}
\]
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 $\exists S. C \land \forall S. (\neg C \sqcup \neg D) \land \exists R. C \land \forall R. (\exists R. C)$ where $R$ is a transitive role

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

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

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

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

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

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

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

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

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

$L(z) = \{C\}$
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.

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

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

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

$\mathcal{L}(z) = \{C\}$
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.

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

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

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

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

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

Reasoning with Expressive Description Logics – p. 7/27
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

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: \( T \) corresponds to model
More Advanced Techniques

For each axiom $CvD^2T$, add $CtD$ to every node label.

More expressive DLs
- Basic technique can be extended to deal with role inclusion axioms (role hierarchy)
- Number restrictions
- Inverse roles
- Concrete domains and datatypes
- Aboxes
- etc.

Extend expansion rules and use more sophisticated blocking strategy.

Forest instead of Tree (for Aboxes)
- Root nodes correspond to individuals in Abox.
More Advanced Techniques

Satisfiability w.r.t. a Terminology

For each axiom $C \sqsubseteq D \in \mathcal{T}$, add $\neg C \sqsubseteq D$ to every node label
More Advanced Techniques

**Satisfiability w.r.t. a Terminology**

For each axiom $C \sqsubseteq D \in T$, add $\neg C \sqsubseteq D$ to every node label

More expressive DLs
More Advanced Techniques

Satisfiability w.r.t. a Terminology
☞ For each axiom $C \sqsubseteq D \in \mathcal{T}$, add $\neg C \sqsubseteq D$ to every node label

More expressive DLs
☞ Basic technique can be extended to deal with
  ● Role inclusion axioms (role hierarchy)
  ● Number restrictions
  ● Inverse roles
  ● Concrete domains and datatypes
  ● Aboxes
  ● etc.
More Advanced Techniques

Satisfiability w.r.t. a Terminology
☞ For each axiom $C \sqsubseteq D \in T$, add $\neg C \sqsubseteq D$ to every node label

More expressive DLs
☞ Basic technique can be extended to deal with
  • Role inclusion axioms (role hierarchy)
  • Number restrictions
  • Inverse roles
  • Concrete domains and datatypes
  • Aboxes
  • etc.
☞ Extend expansion rules and use more sophisticated blocking strategy
More Advanced Techniques

Satisfiability w.r.t. a Terminology
☞ For each axiom $C \sqsubseteq D \in \mathcal{T}$, add $\neg C \sqcup D$ to every node label

More expressive DLs
☞ Basic technique can be extended to deal with
  ● Role inclusion axioms (role hierarchy)
  ● Number restrictions
  ● Inverse roles
  ● Concrete domains and datatypes
  ● Aboxes
  ● etc.
☞ Extend expansion rules and use more sophisticated blocking strategy
☞ Forest instead of Tree (for Aboxes)
  ● Root nodes correspond to individuals in Abox
Implementing DL Systems
Naive Implementations

Problems include:

- Space usage: Storage required for tableaux datastructures is 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. This is a serious problem in practice. Mitigated by:
  - Careful choice of algorithm
  - Highly optimised implementation
Naive Implementations

Problems include:

☞ Space usage
Naive Implementations

Problems include:

☞ **Space** usage
  - Storage required for tableaux datastructures
**Naive Implementations**

Problems include:

♂ *Space* usage
  - Storage required for tableaux datastructures
  - Rarely a serious problem in practice
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
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
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
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
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 \( R:C \), even when \( R \rightarrow R + \) (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., \((\text{domain } R:C \rightarrow v C)\)

(FL) encodings introduce (large numbers of) axioms

BUT even simple domain encoding is disastrous with large numbers of roles

Reasoning with Expressive Description Logics – p. 11/27
Careful Choice of Algorithm

Transitive roles instead of transitive closure
Careful Choice of Algorithm

Transitive roles instead of transitive closure

- Deterministic expansion of $\exists R. C$, even when $R \in \mathbb{R}_+$
Careful Choice of Algorithm

Transitive roles instead of transitive closure

- Deterministic expansion of $\exists R.C$, even when $R \in \mathbb{R}_+$
- (Relatively) simple blocking conditions
Careful Choice of Algorithm

Transitive roles instead of transitive closure
- Deterministic expansion of $\exists R.C$, even when $R \in \mathbb{R}_+$
- (Relatively) simple blocking conditions
- Cycles always represent (part of) valid cyclical models
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
Careful Choice of Algorithm

- **Transitive roles** instead of transitive closure
  - Deterministic expansion of $\exists R.C$, even when $R \in \mathbb{R}_+$
  - (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

Reasoning with Expressive Description Logics – p. 11/27
Careful Choice of Algorithm

Transitive roles instead of transitive closure
- Deterministic expansion of $\exists R.C$, even when $R \in \mathbb{R}_+$
- (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
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,

...
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 $R.C$) $\equiv \exists R.T \sqsubseteq C$
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., $(\text{domain } R.C) \equiv \exists R. \top \sqsubseteq C$

- (FL) encodings introduce (large numbers of) axioms
**Careful Choice of Algorithm**

- **Transitive roles** instead of transitive closure
  - Deterministic expansion of $\exists R.C$, even when $R \in \mathbb{R}_+$
  - (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., $(\text{domain } R.C) \equiv \exists R. \top \sqsubseteq C$
  - (FL) encodings introduce (large numbers of) axioms
  - **BUT** even simple domain encoding is **disastrous** with large numbers of roles
Highly Optimised Implementation

- Naive implementation results in effective non-termination.

Modern systems include MANY optimisations:

Optimised classification:
- Compute partial ordering
- Use enhanced traversal (exploit information from previous tests)
- Use structural information to select classification order

Optimised subsumption testing:
- Search for models
- Normalisation and simplification of concepts
- Absorption (rewriting) of general axioms
- Davis-Putnam style semantic branching search
- Dependency directed backtracking
- Caching of satisfiability results and (partial) models
- Heuristic ordering of propositional and modal expansion
Highly Optimised Implementation

Naive implementation $\rightarrow$ effective non-termination
Highly Optimised Implementation

- Naive implementation → effective non-termination
- Modern systems include MANY optimisations
Highly Optimised Implementation

- Naive implementation $\rightarrow$ effective non-termination
- Modern systems include **MANY** optimisations

Optimised **classification** (compute partial ordering)
- Use enhanced traversal (exploit information from previous tests)
- Use structural information to select classification order
Highly Optimised Implementation

- Naive implementation → effective non-termination
- Modern systems include **MANY** optimisations
- Optimised **classification** (compute partial ordering)
  - Use enhanced traversal (exploit information from previous tests)
  - Use structural information to select classification order
- Optimised **subsumption** testing (search for models)
  - Normalisation and simplification of concepts
  - Absorption (rewriting) of general axioms
  - Davis-Putnam style semantic branching search
  - Dependency directed backtracking
  - Caching of satisfiability results and (partial) models
  - Heuristic ordering of propositional and modal expansion
  - ...
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)
Dependency Directed Backtracking

- Allows rapid recovery from bad branching choices
Dependency Directed Backtracking

- Allows **rapid recovery** from bad branching choices
- Most commonly used technique is **backjumping**
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)
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**
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
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
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

Reasoning with Expressive Description Logics – p. 13/27
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 \( \exists R. \neg A \sqcap \forall R. (A \sqcap B) \sqcap (C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n) \subseteq \mathcal{L}(x) \)
Backjumping

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

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

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

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

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

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

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

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

- Increased expressive power
  - Existing DL systems implement (at most) $SHI\!Q$
  - OWL extends $SHI\!Q$ with datatypes and nominals
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
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
  - ...
Challenges

- **Increased expressive power**
  - Existing DL systems implement (at most) \textit{SHIQ}
  - OWL extends \textit{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
**Increased Expressive Power: Datatypes**

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

- Reasoning with Expressive Description Logics – p. 17/27
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”)
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 (?)
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

Extensionally defined concepts, e.g., EU, France, Italy, ... Resulting logic has known high complexity. No known "practical" algorithm. Not obvious how to extend tableaux techniques in this direction – Loss of tree model property – Spy-points: > v 9 R: f Spy g – Finite domains: f Spy g v 6 nR

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

- OWL `oneOf` constructor equivalent to hybrid logic `nominals`
  - Extensionally defined concepts, e.g., \( \text{EU} \equiv \{ \text{France}, \text{Italy}, \ldots \} \)
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: $\{Spy\} \subseteq \leq n R^-$
Increased Expressive Power: Nominals

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

- 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
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
- ...
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
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**
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?
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
Scalability

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. SHF (no inverse). Problems can arise when SHF extended to SHIQ. Important optimisations no longer (fully) work.

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.
Scalability

Reasoning hard (ExpTime) even without nominals (i.e., $SHIQ$)
Scalability

- Reasoning **hard** (ExpTime) even without nominals (i.e., $SHIQ$)
- Web ontologies may grow **very large**
Scalability

- Reasoning hard (ExpTime) even without nominals (i.e., $SHIQ$)
- 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
Scalability

- Reasoning **hard** (ExpTime) even without nominals (i.e., $SHIQ$)
- 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. $SHF$ (no inverse)
Scalability

- Reasoning **hard** (ExpTime) even without nominals (i.e., 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. SHF (no inverse)
- **Problems** can arise when SHF extended to SHIQ
  - Important **optimisations** no longer (fully) work
Scalability

- Reasoning **hard** (ExpTime) even without nominals (i.e., $SHIQ$)
- 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. $SHF$ (no inverse)
- **Problems** can arise when $SHF$ extended to $SHIQ$
  - 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
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
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
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]

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
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
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]
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
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)
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**
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 $\approx 100k$ concept KB [Haarslev & Möller]
  - But KB only exploited small part of DL language
Other Reasoning Tasks

Querying

Retrieval and instantiation won't 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 and justifications and proofs (e.g., of query results).

"Non-Standard Inferences," e.g., LCS, matching

To support ontology integration and "bottom-up" design of ontologies.
Other Reasoning Tasks

Querying

- Retrieval and instantiation won't be sufficient
- Minimum requirement will be DB style query language
- May also need “what can I say about $x$?” style of query
Other Reasoning Tasks

Querying
- Retrieval and instantiation won't 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)
Other Reasoning Tasks

- **Querying**
  - Retrieval and instantiation won't 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 a family of logical KR formalisms. Applications of DLs include Data Bases and the Semantic Web. Ontologies will provide a vocabulary for semantic markup. OWL, the web ontology language, is based on the SHIQ DL and is set to become a W3C standard. OWL is widely adopted and its use provides formal foundations and reasoning support.

DL Reasoning is based on tableau algorithms. Highly optimised implementations are used in DL systems. Challenges remain, including reasoning with the full OWL language, convincing demonstrations of scalability, new reasoning tasks, and the development of high-quality tools and infrastructure.

Reasoning with Expressive Description Logics – p. 23/27
Description Logics are family of logical KR formalisms
**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
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
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
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
Acknowledgements
Acknowledgements

Members of the OIL, DAML+OIL and OWL development teams, in particular Dieter Fensel (DERI), Frank van Harmelen (Amsterdam) and Peter Patel-Schneider (Bell Labs)
Acknowledgements

- Members of the OIL, DAML+OIL and OWL development teams, in particular Dieter Fensel (DERI), Frank van Harmelen (Amsterdam) and Peter Patel-Schneider (Bell Labs)
- Franz Baader and Stefan Tobies (Dresden)
Acknowledgements

- Members of the OIL, DAML+OIL and OWL development teams, in particular Dieter Fensel (DERI), Frank van Harmelen (Amsterdam) and Peter Patel-Schneider (Bell Labs)

- Franz Baader and Stefan Tobies (Dresden)

- Uli Sattler, Carole Goble and other Members of the Information Management, Medical Informatics and Formal Methods Groups at the University of Manchester
Resources

Slides from this talk

http://www.cs.man.ac.uk/~horrocks/Slides/Innsbruck-tutorial/

FaCT system (open source)

http://www.cs.man.ac.uk/FaCT/

OilEd (open source)

http://oiled.man.ac.uk/

OIL

http://www.ontoknowledge.org/oil/

W3C Web-Ontology (WebOnt) working group (OWL)

http://www.w3.org/2001/sw/WebOnt/

DL Handbook, Cambridge University Press

http://books.cambridge.org/0521781760.htm


