Description Logic Reasoning
Basic Inference Problems
**Basic Inference Problems**

- **Subsumption** — check knowledge is correct
  - \( K \models C \subseteq D \) ? \( C^I \subseteq D^I \) in all models \( I \) of \( K \)
Basic Inference Problems

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

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

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

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

☞ **Consistency** (satisfiability) — check knowledge is meaningful
  
  • $\mathcal{K} \not\models C \equiv \perp$? $C^\mathcal{I} \neq \emptyset$ in some model $\mathcal{I}$ of $\mathcal{K}$
Basic Inference Problems

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

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

☞ **Consistency** (satisfiability) — check knowledge is meaningful
   - $\mathcal{K} \not\models C \equiv \bot \ ? \ C^\mathcal{I} \neq \emptyset$ in some model $\mathcal{I}$ of $\mathcal{K}$

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

☞ **Subsumption** — check knowledge is correct

- $\mathcal{K} \models C \subseteq D$? $C^\mathcal{I} \subseteq D^\mathcal{I}$ in all models $\mathcal{I}$ of $\mathcal{K}$

☞ **Equivalence** — check knowledge is minimally redundant

- $\mathcal{K} \models C \equiv D$? $C^\mathcal{I} = D^\mathcal{I}$ in all models $\mathcal{I}$ of $\mathcal{K}$

☞ **Consistency** (satisfiability) — check knowledge is meaningful

- $\mathcal{K} \not\models C \equiv \bot$? $C^\mathcal{I} \neq \emptyset$ in some model $\mathcal{I}$ of $\mathcal{K}$

☞ **Instantiation** — check if individual $i$ instance of class $C$

- $\mathcal{K} \models i : C$? $i^\mathcal{I} \in C^\mathcal{I}$ in all models $\mathcal{I}$ of $\mathcal{K}$

These problems are all reducible to **KB satisfiability**

- $\langle \mathcal{T}, \mathcal{A} \rangle \models C \subseteq D$ iff $\langle \mathcal{T}, \mathcal{A} \cup a : (C \cap \neg D) \rangle$ is unsatisfiable

- $\langle \mathcal{T}, \mathcal{A} \rangle \models i : C$ iff $\langle \mathcal{T}, \mathcal{A} \cup i : \neg C \rangle$ is unsatisfiable
Concept Satisfiability Problem
Concept Satisfiability Problem

Algorithm research has focused on concept satisfiability
Concept Satisfiability Problem

Algorithm research has focused on **concept satisfiability**

- **Tbox reasoning** (i.e., $\mathcal{A} = \emptyset$) sufficient for many applications
  - e.g., ontology (schema) design
Algorithm research has focused on **concept satisfiability**

- **Tbox reasoning** (i.e., $A = \emptyset$) sufficient for many applications
  - e.g., ontology (schema) design
- **Satisfiability** of $\langle T, \{i : C\} \rangle$ equivalent to satisfiability of $C$ w.r.t. $T$
Algorithm research has focused on concept satisfiability

- **Tbox reasoning** (i.e., $A = \emptyset$) sufficient for many applications
  - e.g., ontology (schema) design

- **Satisfiability** of $\langle T, \{i : C\} \rangle$ equivalent to satisfiability of $C$ w.r.t. $T$

- Satisfiability **w.r.t. a Tbox** often reducible to concept satisfiability
Algorithm research has focused on **concept satisfiability**

- **Tbox reasoning** (i.e., $A = \emptyset$) sufficient for many applications
  - e.g., ontology (schema) design
- **Satisfiability** of $\langle T, \{i : C\} \rangle$ equivalent to satisfiability of $C$ w.r.t. $T$
- Satisfiability **w.r.t. a Tbox** often reducible to concept satisfiability
  - Unique, acyclic definition axioms can be **unfolded**
    - e.g., $A$ satisfiable w.r.t. $\{A \sqsubseteq C\}$ iff $A \cap C$ is satisfiable
Concept Satisfiability Problem

Algorithm research has focused on **concept satisfiability**

- **Tbox reasoning** (i.e., $A = \emptyset$) sufficient for many applications
  - e.g., ontology (schema) design

- **Satisfiability** of $\langle T, \{i : C\} \rangle$ equivalent to satisfiability of $C$ w.r.t. $T$

- Satisfiability **w.r.t. a Tbox** often reducible to concept satisfiability
  - Unique, acyclic definition axioms can be **unfolded**
    - e.g., $A$ satisfiable w.r.t. $\{A \sqsubseteq C\}$ iff $A \sqcap C$ is satisfiable
  - For more expressive logics, Tbox axioms can be **internalised**
    - $C \sqsubseteq D$ equivalent to $\top \sqsubseteq D \sqcup \neg C$
    - $\{\top \sqsubseteq C_1, \ldots, \top \sqsubseteq C_n\}$ equivalent to $\{\top \sqsubseteq C_1 \sqcap \ldots \sqcap C_n\}$
    - $C$ satisfiable w.r.t. $\{\top \sqsubseteq G\}$ iff $C \sqcap G \sqcap \forall S.G$ not satisfiable
      where $S$ is transitive “top” role
Algorithm research has focused on **concept satisfiability**

- **Tbox reasoning** (i.e., $A = \emptyset$) sufficient for many applications
  - e.g., ontology (schema) design
- **Satisfiability** of $\langle T, \{i : C\} \rangle$ equivalent to satisfiability of $C$ w.r.t. $T$
- Satisfiability w.r.t. a Tbox often reducible to concept satisfiability
  - Unique, acyclic definition axioms can be **unfolded**
    - e.g., $A$ satisfiable w.r.t. $\{A \sqsubseteq C\}$ iff $A \sqcap C$ is satisfiable
  - For more expressive logics, Tbox axioms can be **internalised**
    - $C \sqsubseteq D$ equivalent to $\top \sqsubseteq D \sqcup \neg C$
    - $\{\top \sqsubseteq C_1, \ldots, \top \sqsubseteq C_n\}$ equivalent to $\{\top \sqsubseteq C_1 \sqcap \ldots \sqcap C_n\}$
    - $C$ satisfiable w.r.t. $\{\top \sqsubseteq G\}$ iff $C \sqcap G \sqcap \forall S.G$ not satisfiable
      where $S$ is transitive “top” role
- **Tableaux algorithms** often used to decide concept satisfiability
  - Can easily be extended to deal with Tbox and/or Abox
Tableaux Algorithms — Basics
Try to build tree-like model $\mathcal{I}$ of input concept $C$
Tableaux Algorithms — Basics

- 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

- 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 $\mathcal{I}$
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., $\sqcup$, $\leq$)
Tableaux Algorithms — Basics

- 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, $\neg \exists R. C \rightsquigarrow \forall R. \neg C$ etc.
- Break down $C$ **syntactically**, inferring constraints on elements of $\mathcal{I}$
- Decomposition uses **tableau rules** corresponding to constructors in logic (e.g., $\cap$, $\exists$)
  - Some rules are **nondeterministic** (e.g., $\sqcup$, $\leq$)
- **Stop** when
  - conflicting constraints are derived, or
  - no rules are applicable (syntax fully decomposed)
Tableaux Algorithms — Basics

- 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., $\sqcup$, $\leq$)
- **Stop** when
  - conflicting constraints are derived, or
  - no rules are applicable (syntax fully decomposed)
- May be **worst-case** optimal (w.r.t. complexity of problem)
  - but focus is usually on good **typical-case** performance
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 $\Delta^\mathcal{I}$
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 $\{C\}$
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
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 $x$
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 $\mathcal{I}$ 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
- $C$ 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 $\text{ALC}$

<table>
<thead>
<tr>
<th>$x \bullet {C_1 \cap C_2, \ldots}$</th>
<th>$\rightarrow \cap$</th>
<th>$x \bullet {C_1 \cap C_2, C_1, C_2, \ldots}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \bullet {C_1 \sqcup C_2, \ldots}$</td>
<td>$\rightarrow \sqcup$</td>
<td>$x \bullet {C_1 \cap C_2, C, \ldots}$ for $C \in {C_1, C_2}$</td>
</tr>
<tr>
<td>$x \bullet {\exists R.C, \ldots}$</td>
<td>$\rightarrow \exists$</td>
<td>$x \bullet {\exists R.C, \ldots}$</td>
</tr>
<tr>
<td>$R$</td>
<td>$\rightarrow$</td>
<td>$R$</td>
</tr>
<tr>
<td>$y \bullet {\ldots}$</td>
<td>$\rightarrow$</td>
<td>$y \bullet {C}$</td>
</tr>
<tr>
<td>$y \bullet {\ldots}$</td>
<td>$\rightarrow$</td>
<td>$y \bullet {C, \ldots}$</td>
</tr>
</tbody>
</table>
Algorithm is a Decision Procedure
Lemma

Let $C_0$ be an $\mathcal{ALC}$ concept and $T$ a tree obtained by applying the tableau rules to $C_0$. Then

1. the rule application terminates,
2. if $T$ is consistent and $\rightarrow$ is applicable to $T$, then $\rightarrow$ can be applied such that it yields consistent $T'$,
3. if $T$ contains a clash, then $T$ has no model, and
4. if $T$ is fully expanded (no more rules applicable) and clash free, then $T$ defines a (canonical) model for $C_0$. 
Algorithm is a Decision Procedure II

Proof of the Lemma
Proof of the Lemma

1. (Termination) The algorithm “monotonically” constructs a tree whose
   \textbf{depth} is linear in $|C_0|$: quantifier depth decreases from node to succs.
   \textbf{breadth} is linear in $|C_0|$: upper bound on $\exists$-rule appns. to a label.
Algorithm is a Decision Procedure II

Proof of the Lemma

1. (Termination) The algorithm “monotonically” constructs a tree whose
   - **depth** is linear in $|C_0|$: quantifier depth decreases from node to succs.
   - **breadth** is linear in $|C_0|$: upper bound on $\exists$-rule appns. to a label

2. (Local Consistency) Easy to prove (by defn. of the semantics) that
   if $I$ is a model of $T$, then $\rightarrow$ can be applied to $T$ such that
   $I$ is a model of $T' := \rightarrow(T)$
Algorithm is a Decision Procedure II

Proof of the Lemma

1. (Termination) The algorithm “monotonically” constructs a tree whose
   \textbf{depth} is linear in $|C_0|$: quantifier depth decreases from node to succs.
   \textbf{breadth} is linear in $|C_0|$: upper bound on $\exists$-rule appns. to a label

2. (Local Consistency) Easy to prove (by defn. of the semantics) that
   if $\mathcal{I}$ is a model of $T$, then $\rightarrow$ can be applied to $T$ such that
   $\mathcal{I}$ is a model of $T' := \rightarrow(T)$

3. Obvious: $T$ with a clash has no model—recall definition of a clash:
   \begin{align*}
   \{A, \neg A\} & \subseteq \mathcal{L}(x)
   \end{align*}
Algorithm is a Decision Procedure II

Proof of the Lemma

1. (Termination) The algorithm “monotonically” constructs a tree whose
   - **depth** is linear in $|C_0|$: quantifier depth decreases from node to succs.
   - **breadth** is linear in $|C_0|$: upper bound on $\exists$-rule appns. to a label

2. (Local Consistency) Easy to prove (by defn. of the semantics) that
   - if $\mathcal{I}$ is a model of $\mathcal{T}$, then $\rightarrow$ can be applied to $\mathcal{T}$ such that
     $$\mathcal{I} \text{ is a model of } \mathcal{T}' := \rightarrow(\mathcal{T})$$

3. Obvious: $\mathcal{T}$ with a clash has no model—recall definition of a clash:
   $$\{A, \neg A\} \subseteq \mathcal{L}(x)$$

4. (Canonical model) “Complete” tree $\mathcal{T}$ defines a (tree) model $\mathcal{I}$:
   - nodes correspond to elements of $\Delta^\mathcal{I}$
   - edges define role-relationship
   - $x \in A^\mathcal{I}$ iff $A \in \mathcal{L}(x)$ for concept names $A$
Tableaux Rule for Number Restrictions
**Tableaux Rule for Number Restrictions**

- **$x \bullet \{(\geq n \ R), \ldots\}$**
- **$x$ has no $R$-succ.**

- **$\rightarrow_{\geq}$**
  - **$x \bullet \{(\geq n \ R), \ldots\}$**
  - **$R$**
  - **$y \bullet \{}$**

- **$\rightarrow_{\leq}$**
  - **$x \bullet \{(\leq n \ R), \ldots\}$**
  - **$R$**
  - **$\ldots > n$**

- **merge two $R$-succs.**
Tableaux Rule for Transitive Roles
Tableaux Rule for Transitive Roles

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

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

\begin{itemize}
  \item No longer naturally terminating (e.g., if $C = \exists R. \top$)
\end{itemize}
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
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

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

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

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

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

$\square$
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

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

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

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

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

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

Diagram:

- Node $w$ with $L(w)$
- Node $x$ with $L(x)$
- Edge from $S$ to $x$
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 $\exists S. C \sqcap \forall S. (\neg C \sqcup \neg D) \sqcap \exists R. C \sqcap \forall R. (\exists R. C)$ where $R$ is a transitive role

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

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

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

$L(x) = \{C, (\neg C \lor \neg D), \neg C\}$
Tableaux Algorithm — Example

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

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

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

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

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

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

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

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

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

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

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

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

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

DL Reasoning – p. 11/12
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

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

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

\[
\mathcal{L}(y) = \{ C \}
\]
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 \textit{transitive} role.

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

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

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

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

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

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

$\mathcal{L}(y) = \{C, \exists R.C, \forall R.(\exists R.C')\}$
Tableaux Algorithm — Example

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

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

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

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

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

Concept is satisfiable: $\top$ 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

Concept is satisfiable: $T$ corresponds to model
More Advanced Techniques
More Advanced Techniques

Satisfiability w.r.t. a general Tbox

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 general Tbox

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

More expressive DLs
More Advanced Techniques

Satisfiability w.r.t. a general Tbox

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)
- Nominals
- Inverse roles
- Qualified number restrictions
- Concrete domains and datatypes
- Aboxes
- etc.
More Advanced Techniques

Satisfiability w.r.t. a general Tbox

For each axiom $C \sqsubseteq D \in T$, 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)
- Nominals
- Inverse roles
- Qualified number restrictions
- Concrete domains and datatypes
- Aboxes
- etc.

Extend expansion rules and use more sophisticated blocking strategy
More Advanced Techniques

Satisfiability w.r.t. a general Tbox
☞ For each axiom $C \sqsubseteq D \in T$, 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)
  - Nominals
  - Inverse roles
  - Qualified number restrictions
  - Concrete domains and datatypes
  - Aboxes
  - etc.
☞ Extend expansion rules and use more sophisticated blocking strategy
☞ Forest instead of Tree (for Aboxes/Nominals)
  - Root nodes correspond to individuals in Abox