Description Logic Reasoning

Basic Inference Problems

Basic Inference Problems

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

Basic Inference Problems

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

Basic Inference Problems

- ☞ **Subsumption** — check knowledge is correct
  - $C \sqsubseteq_{\mathcal{K}} D$? $\quad C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ in all models $\mathcal{I}$ of $\mathcal{K}$
- ☞ **Equivalence** — check knowledge is minimally redundant
  - $C \equiv_{\mathcal{K}} D$? $\quad C^{\mathcal{I}} = D^{\mathcal{I}}$ in all models $\mathcal{I}$ of $\mathcal{K}$
- ☞ **Consistency** — check knowledge is meaningful
  - $C \equiv \bot \quad C^{\mathcal{I}} \neq \emptyset$ in some model $\mathcal{I}$ of $\mathcal{K}$

Basic Inference Problems

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

Basic Inference Problems

- **Subsumption** — check knowledge is correct
  - $C \sqsubseteq_{\mathcal{K}} D$? $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ 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 $\mathcal{K}$
- **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
  - $i \in_{\mathcal{K}} C$? $i \in C^{\mathcal{I}}$ in all models $\mathcal{I}$ of $\mathcal{K}$
- Problems all **reducible** to KB consistency (satisfiability):
  - e.g., $C \sqsubseteq_{\mathcal{K}} D$ iff $C \sqcap \neg D$ not consistent w.r.t. $\mathcal{K}$

Basic Inference Problems

- **Subsumption** — check knowledge is correct
  - $C \sqsubseteq_{\mathcal{K}} D$? $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ 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 $\mathcal{K}$
- **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
  - $i \in_{\mathcal{K}} C$? $i \in C^{\mathcal{I}}$ in all models $\mathcal{I}$ of $\mathcal{K}$
- Problems all **reducible** to KB consistency (satisfiability):
  - e.g., $C \sqsubseteq_{\mathcal{K}} D$ 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 — Basics

- 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, $\neg\exists R.C \rightsquigarrow \forall R.\neg C$ 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, $\neg\exists R.C \rightsquigarrow \forall R.\neg C$ 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, $\neg\exists R.C \rightsquigarrow \forall R.\neg C$ etc.
- Break down C **syntactically**, inferring constraints on elements of $\mathcal{I}$
- Decomposition uses **tableau rules** corresponding to constructors in logic (e.g., $\sqcap$, $\exists$)
  - Some rules are **nondeterministic** (e.g., $\sqcup$, $\leqslant$)
  - 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, $\neg\exists R.C \leadsto \forall R.\neg C$ 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., $\sqcup$, $\leqslant$)
  - 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, $\neg\exists R.C \leadsto \forall R.\neg C$ 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., $\sqcup$, $\leqslant$)
  - 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, $\neg\exists R.C \rightsquigarrow \forall R.\neg C$ etc.
- Break down C **syntactically**, inferring constraints on elements of $\mathcal{I}$
- Decomposition uses **tableau rules** corresponding to constructors in logic (e.g., $\sqcap$, $\exists$)
  - Some rules are **nondeterministic** (e.g., $\sqcup$, $\leqslant$)
  - 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 Algorithms — Details

☞ Work on **tree** $\mathbf{T}$ 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** $\mathbf{T}$ 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 $\Delta^{\mathcal{I}}$

☞ $\mathbf{T}$ initialised with single **root node** labeled $\{C\}$

Tableaux Algorithms — Details

- ☞ Work on **tree** $\mathbf{T}$ 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 $\Delta^{\mathcal{I}}$
- ☞ $\mathbf{T}$ initialised with single **root node** labeled $\{C\}$
- ☞ **Tableau rules** repeatedly applied to node labels
  - Extend labels or extend/modify $\mathbf{T}$ structure
  - Rules can be **blocked**, e.g, if predecessor has **superset** label
  - Nondeterministic rules $\longrightarrow$ **search** possible extensions

Tableaux Algorithms — Details

- Work on **tree** $\mathbf{T}$ 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 $\Delta^{\mathcal{I}}$
- $\mathbf{T}$ initialised with single **root node** labeled $\{C\}$
- **Tableau rules** repeatedly applied to node labels
  - Extend labels or extend/modify $\mathbf{T}$ structure
  - Rules can be **blocked**, e.g, if predecessor has **superset** label
  - Nondeterministic rules $\longrightarrow$ **search** possible extensions
- $\mathbf{T}$ 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** $\mathbf{T}$ 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 $\Delta^{\mathcal{I}}$
- ☞ $\mathbf{T}$ initialised with single **root node** labeled $\{C\}$
- ☞ **Tableau rules** repeatedly applied to node labels
  - Extend labels or extend/modify $\mathbf{T}$ structure
  - Rules can be **blocked**, e.g, if predecessor has **superset** label
  - Nondeterministic rules $\longrightarrow$ **search** possible extensions
- ☞ $\mathbf{T}$ contains **Clash** if obvious contradiction in some node label
  - E.g., $\{A, \neg A\} \subseteq \mathfrak{L}(x)$ for some concept A and node x
- ☞ $\mathbf{T}$ **fully expanded** if no rules are applicable

Tableaux Algorithms — Details

- Work on **tree** $\mathbf{T}$ 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 $\Delta^{\mathcal{I}}$
- $\mathbf{T}$ initialised with single **root node** labeled $\{C\}$
- **Tableau rules** repeatedly applied to node labels
  - Extend labels or extend/modify $\mathbf{T}$ structure
  - Rules can be **blocked**, e.g, if predecessor has **superset** label
  - Nondeterministic rules $\longrightarrow$ **search** possible extensions
- $\mathbf{T}$ 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
- $\mathbf{T}$ **fully expanded** if no rules are applicable
- C satisfiable iff fully expanded clash free $\mathbf{T}$ found
  - Trivial correspondence between such a $\mathbf{T}$ and a model of C

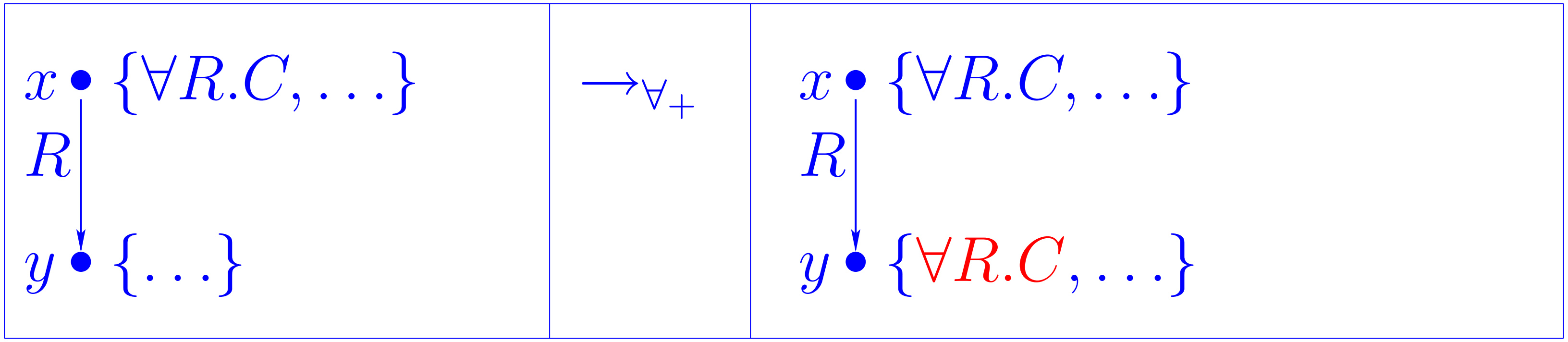
Tableaux Rules for $\mathcal{ALC}$

Tableaux Rules for $\mathcal{ALC}$

$x \bullet \{C_1 \sqcap C_2, \ldots\}$	$\rightarrow_\sqcap$	$x \bullet \{C_1 \sqcap C_2, C_1, C_2, \ldots\}$
$x \bullet \{C_1 \sqcup C_2, \ldots\}$	$\rightarrow_\sqcup$	$x \bullet \{C_1 \sqcup C_2, C, \ldots\}$ for $C \in \{C_1, C_2\}$
$x \bullet \{\exists R.C, \ldots\}$	$\rightarrow_\exists$	$x \bullet \{\exists R.C, \ldots\}$ $R \downarrow$ $y \bullet \{C\}$
$x \bullet \{\forall R.C, \ldots\}$ $R \downarrow$ $y \bullet \{\ldots\}$	$\rightarrow_\forall$	$x \bullet \{\forall R.C, \ldots\}$ $R \downarrow$ $y \bullet \{C, \ldots\}$

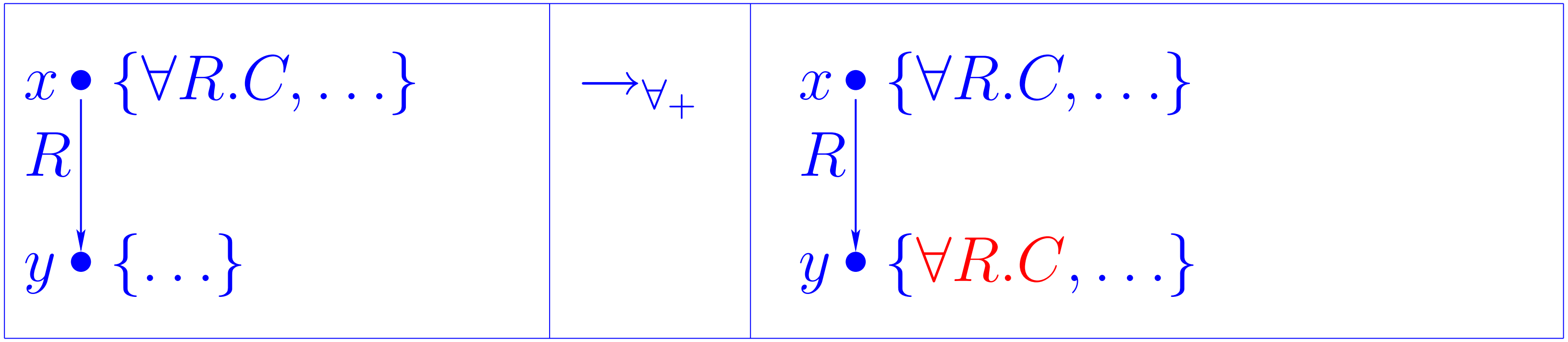
Tableaux Rule for Transitive Roles

Tableaux Rule for Transitive Roles



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

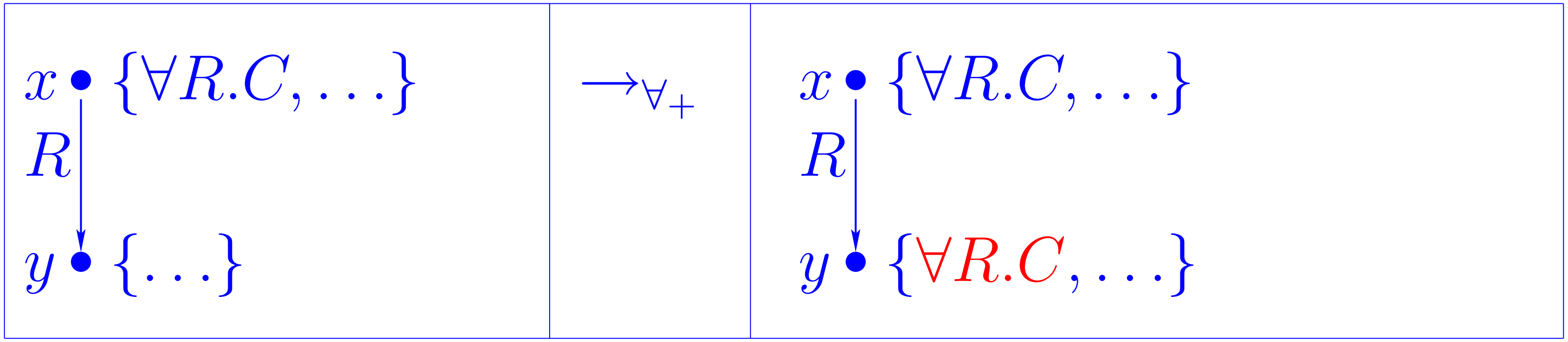
Tableaux Rule for Transitive Roles



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

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

Tableaux Rule for Transitive Roles



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

- ☞ 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 $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role

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

w

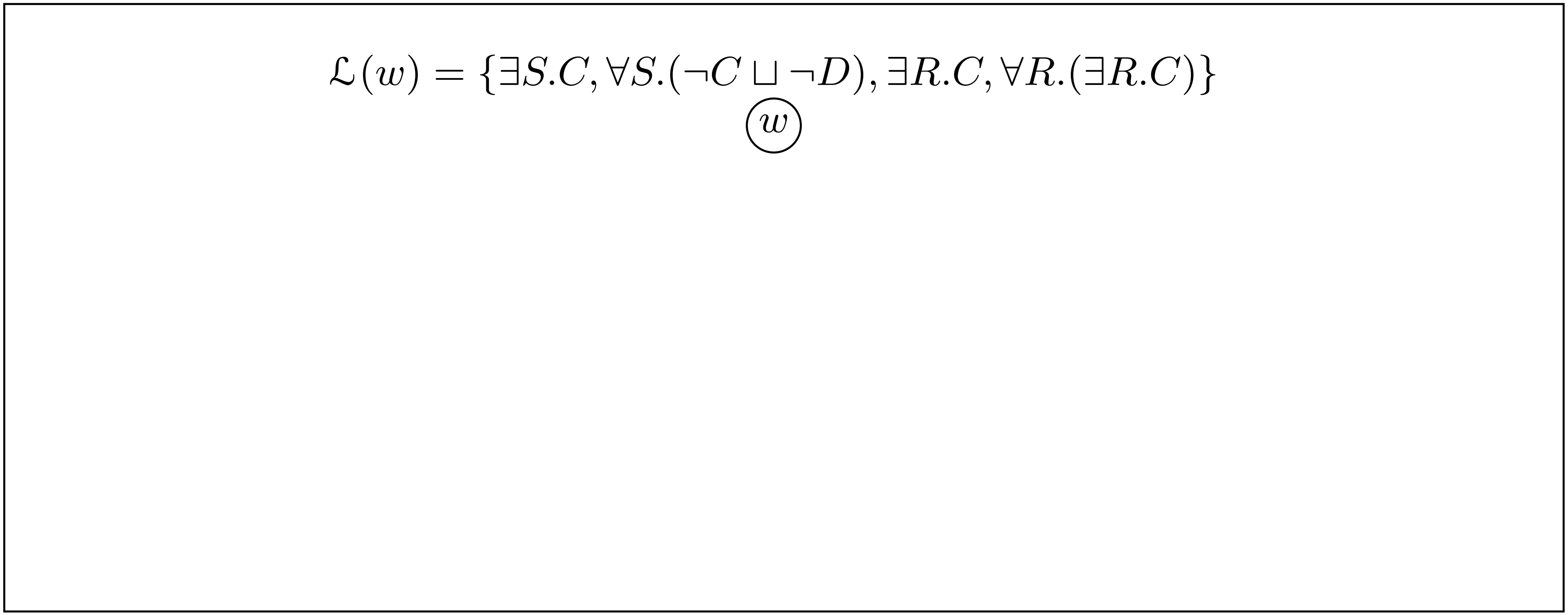
Tableaux Algorithm — Example

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 $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



Tableaux Algorithm — Example

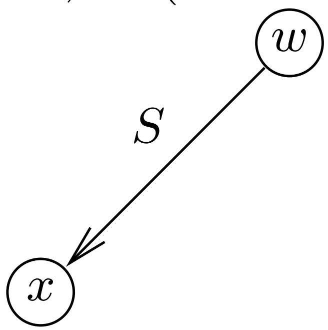
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role

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

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

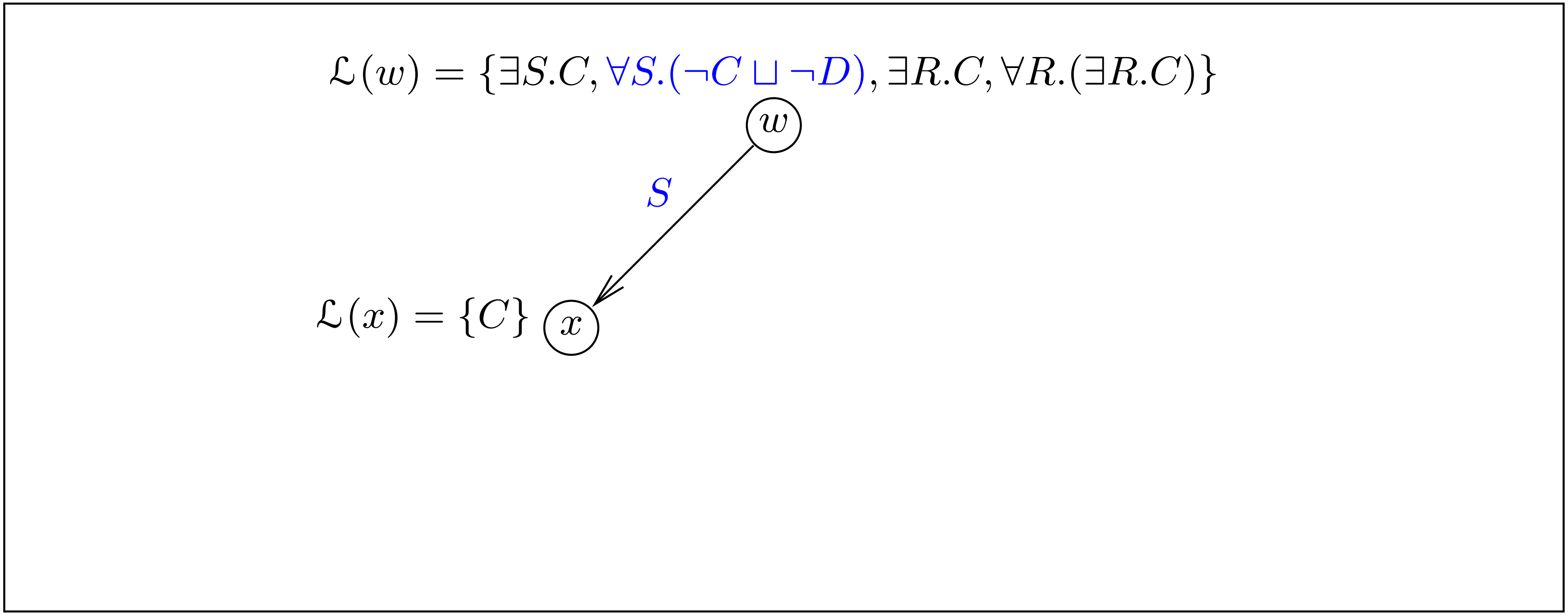
$$\mathcal{L}(w) = \{\exists S.C, \forall S.(\neg C \sqcup \neg D), \exists R.C, \forall R.(\exists R.C)\}$$



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

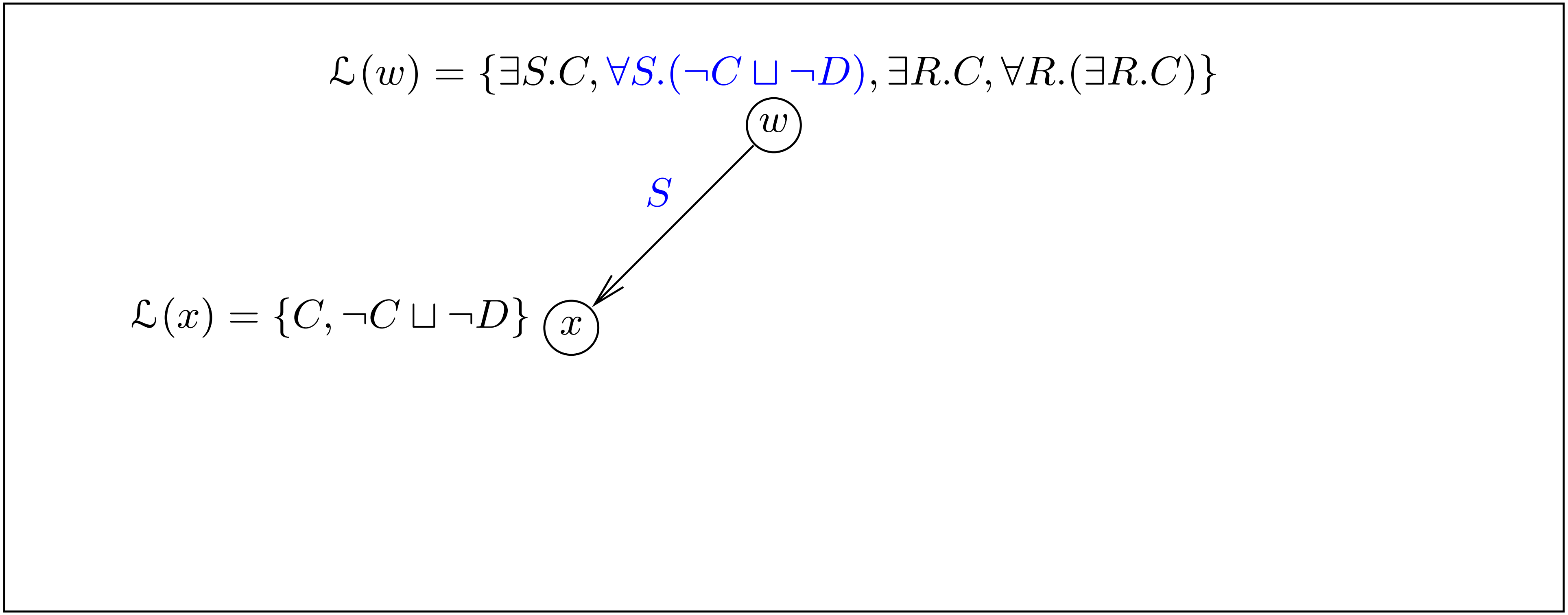
Tableaux Algorithm — Example

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



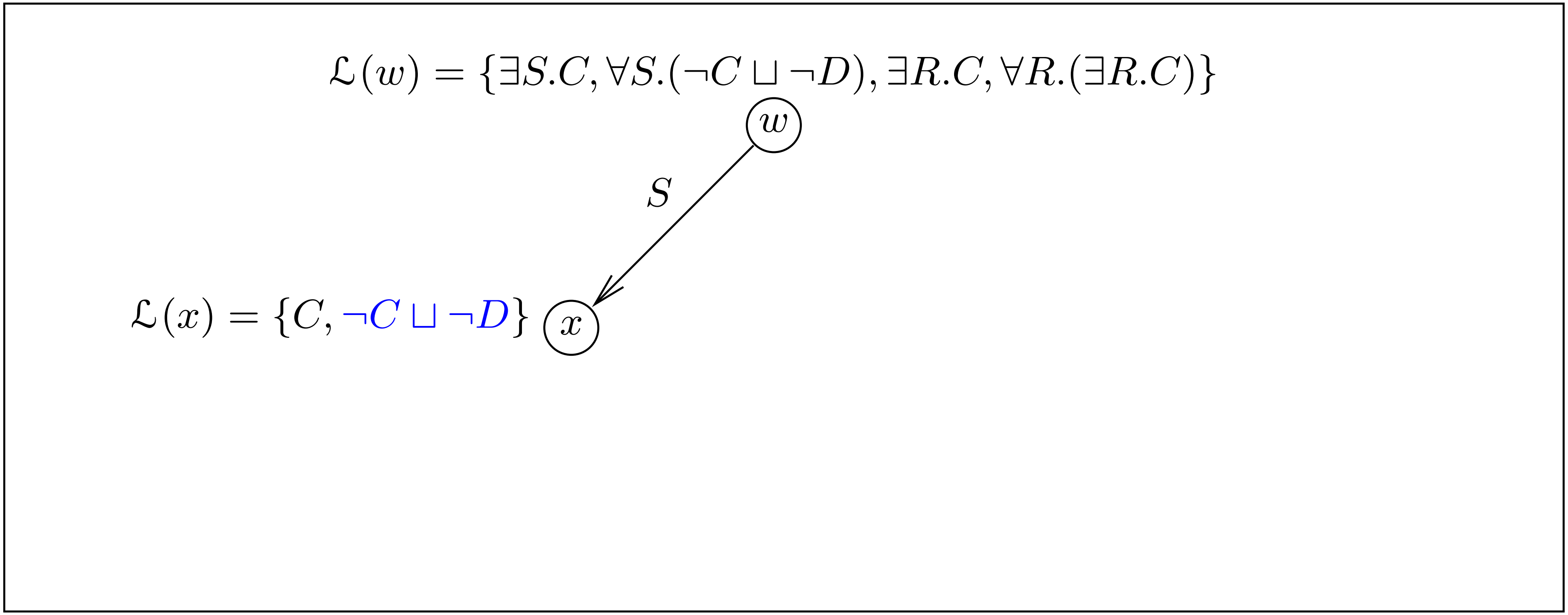
Tableaux Algorithm — Example

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



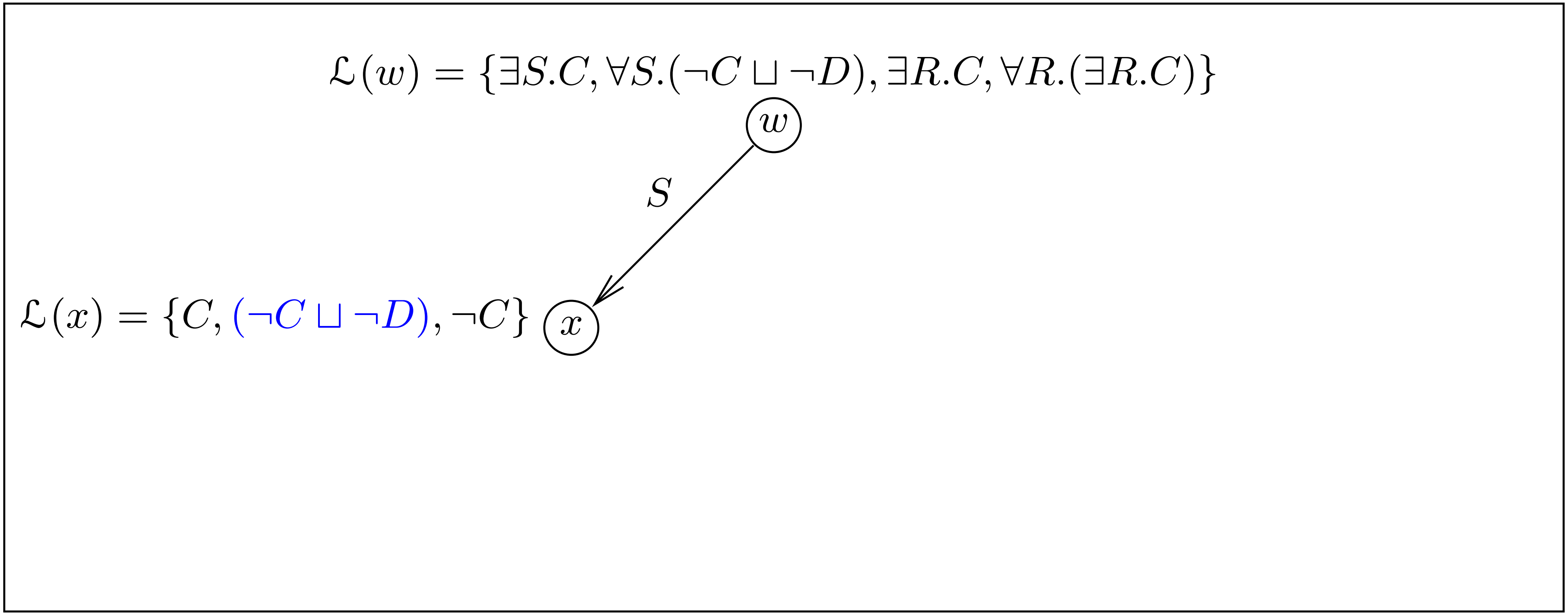
Tableaux Algorithm — Example

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



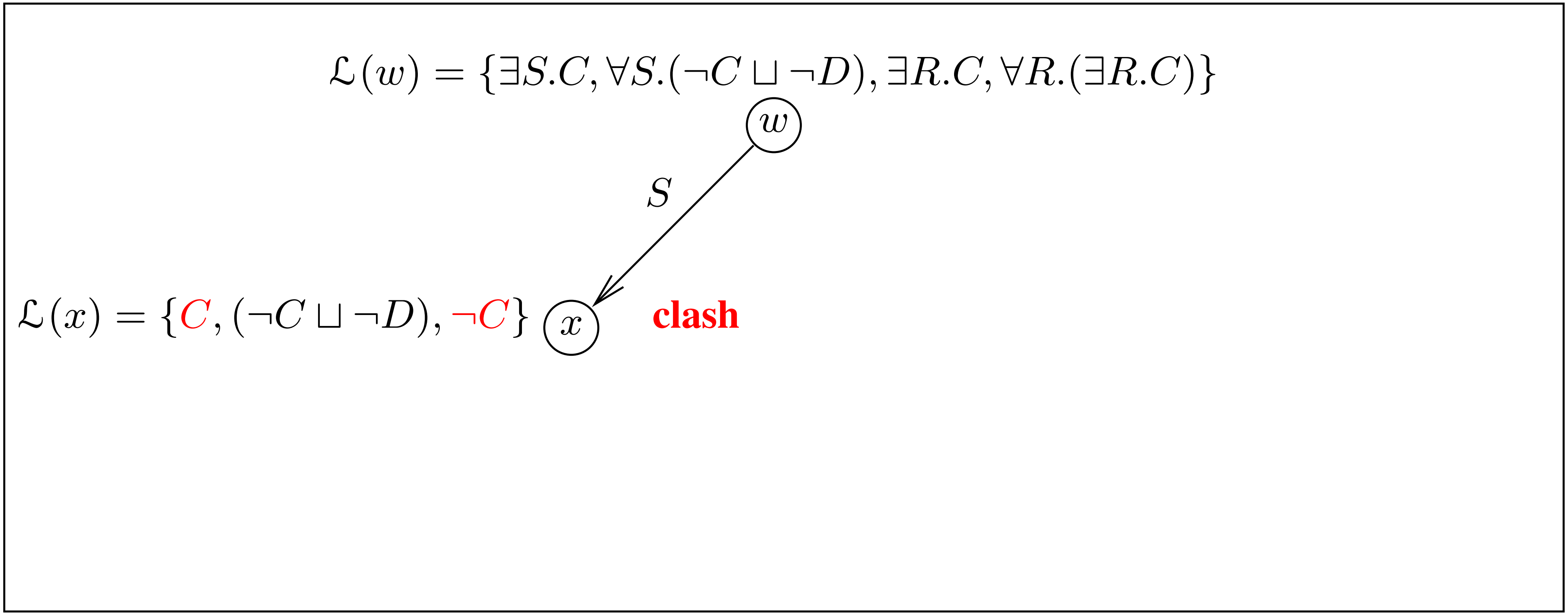
Tableaux Algorithm — Example

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



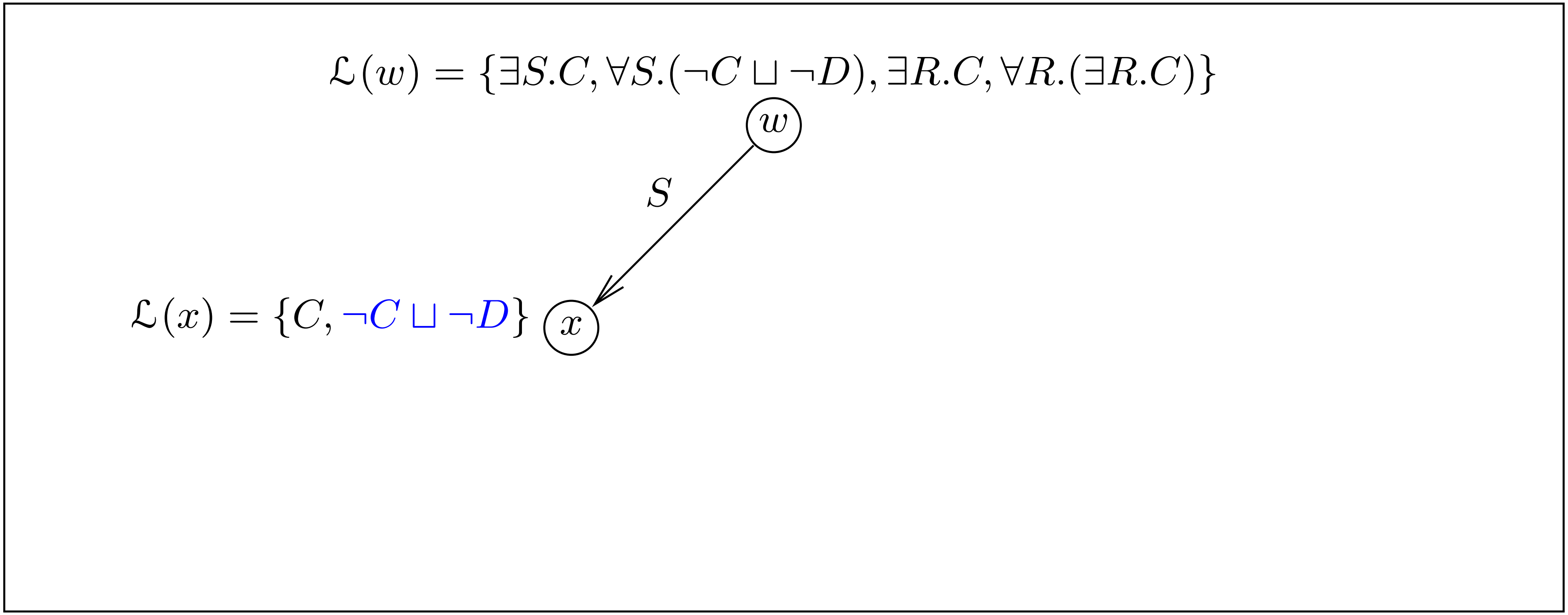
Tableaux Algorithm — Example

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



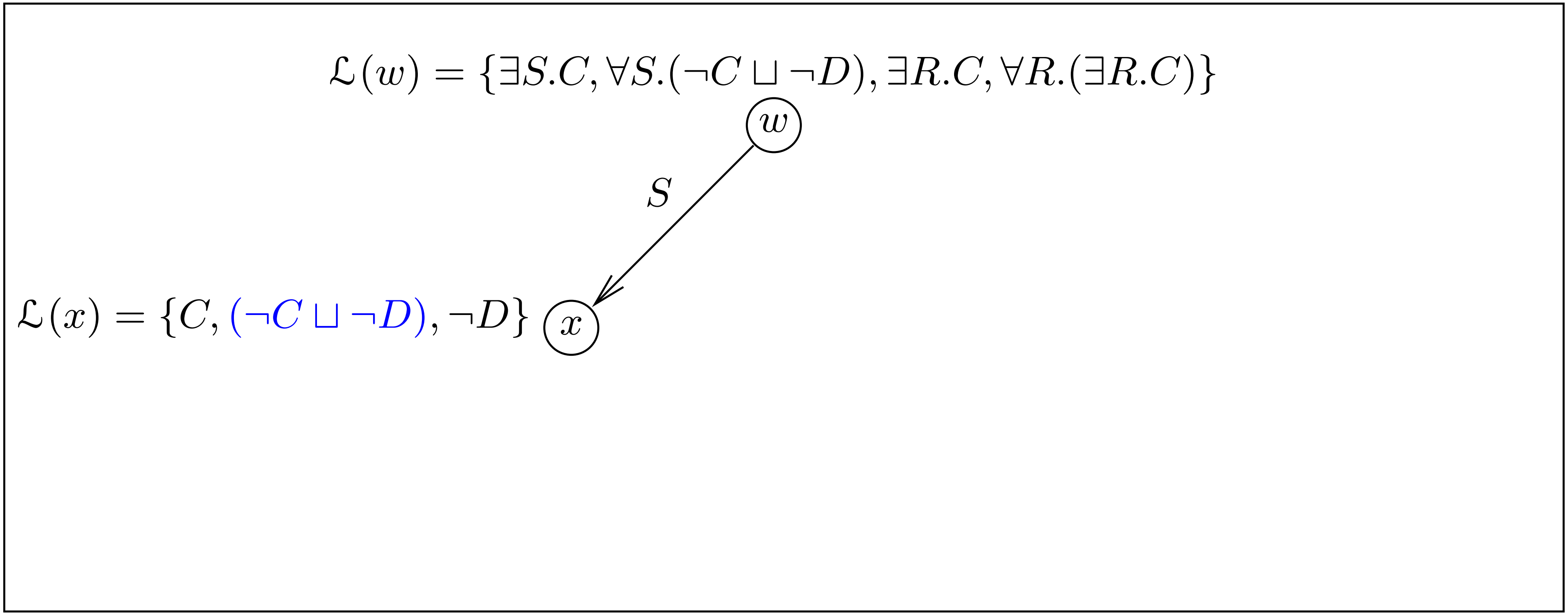
Tableaux Algorithm — Example

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



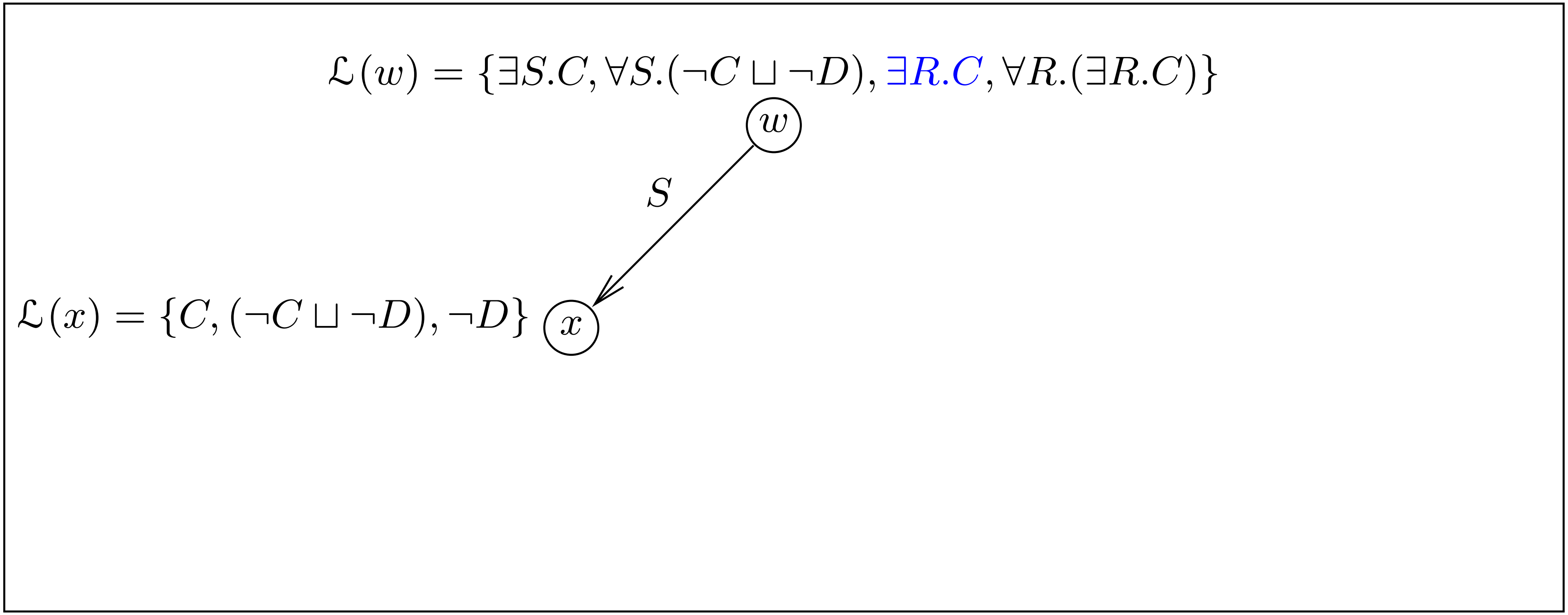
Tableaux Algorithm — Example

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



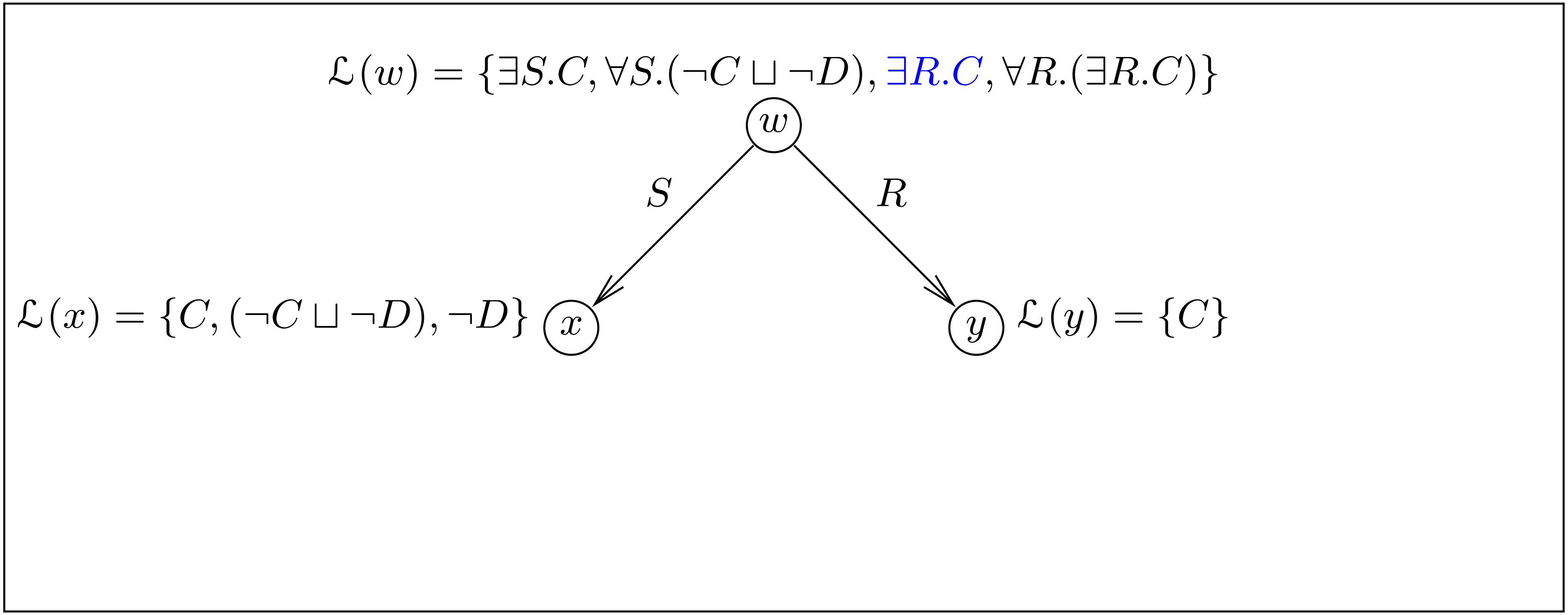
Tableaux Algorithm — Example

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



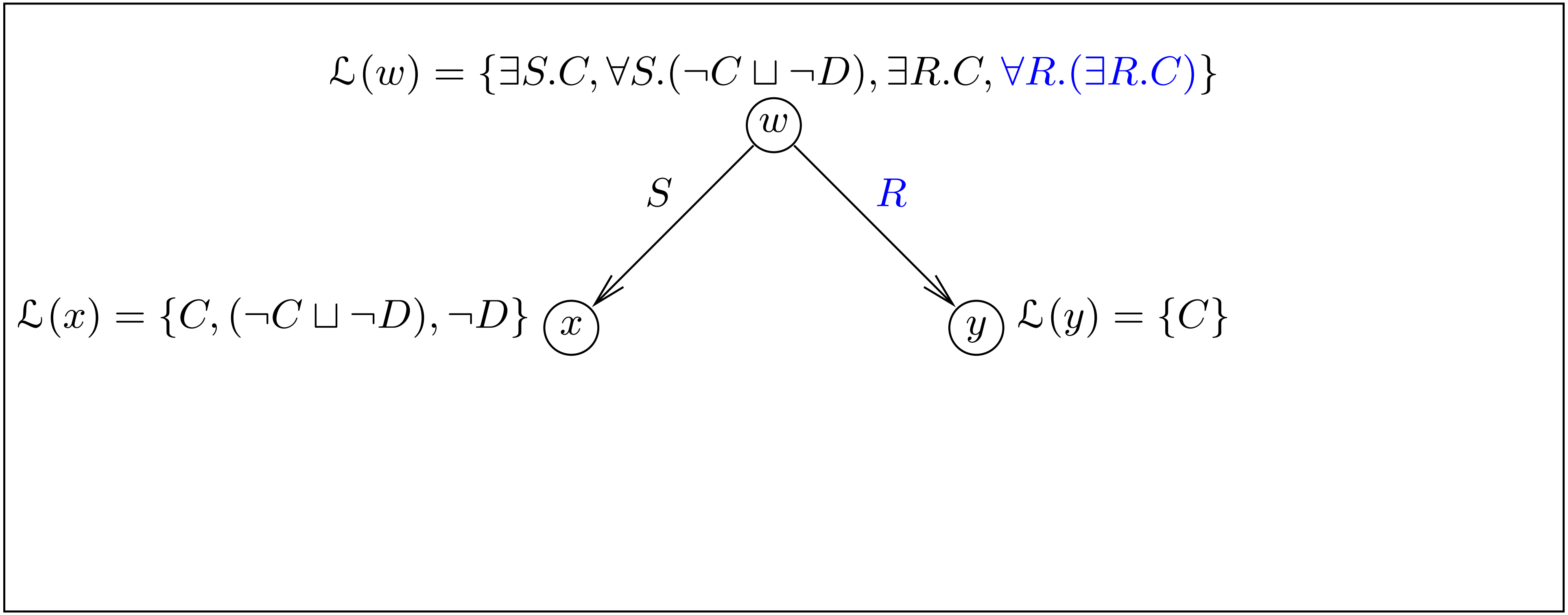
Tableaux Algorithm — Example

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



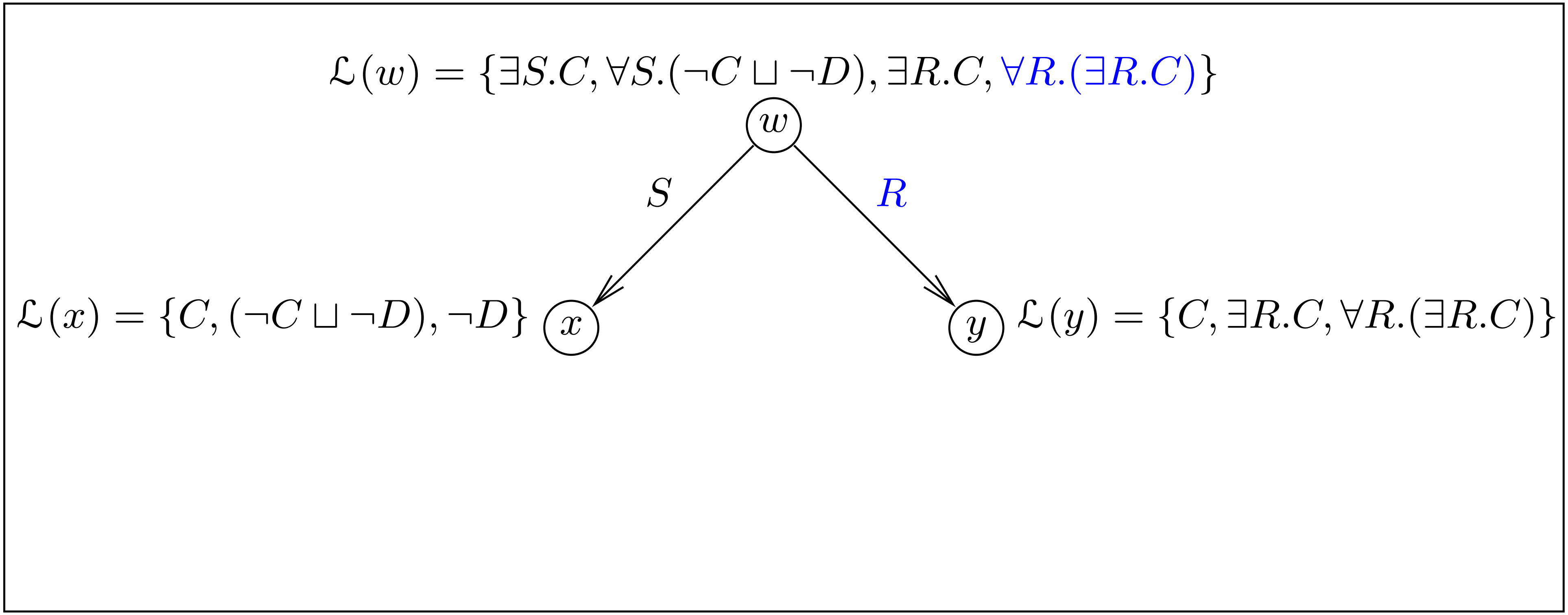
Tableaux Algorithm — Example

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



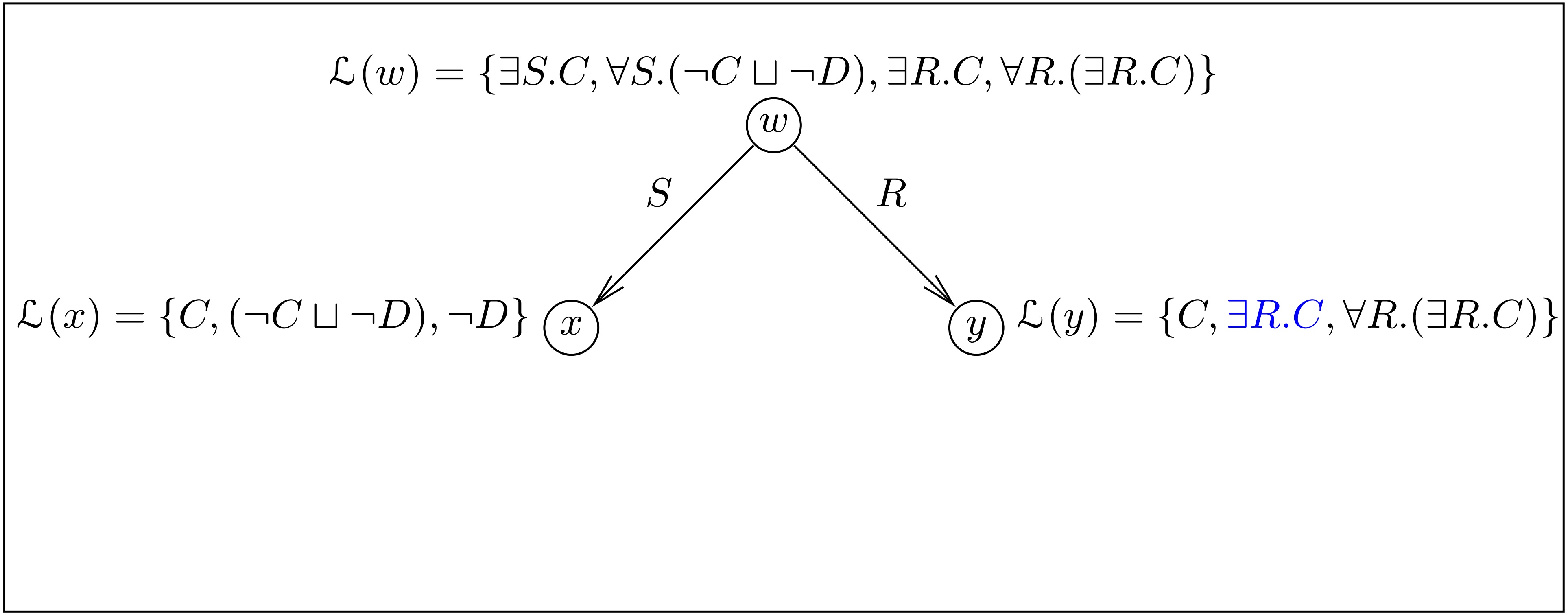
Tableaux Algorithm — Example

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



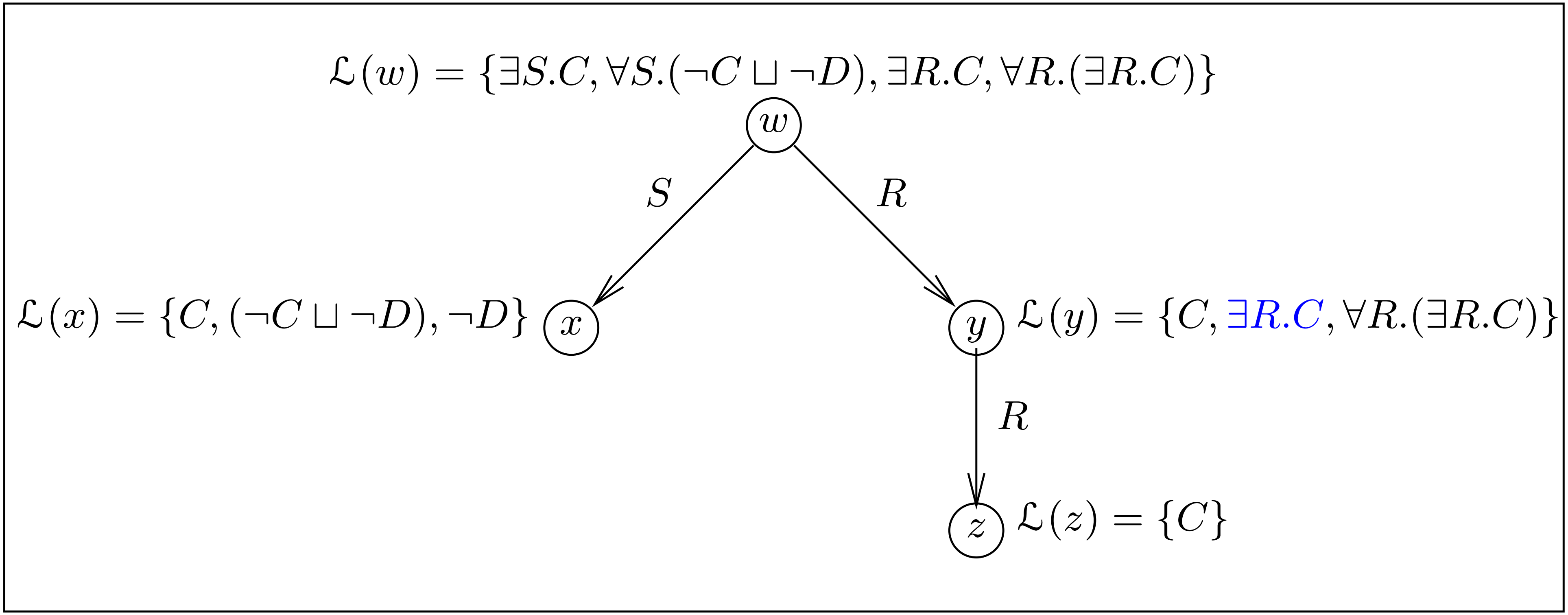
Tableaux Algorithm — Example

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



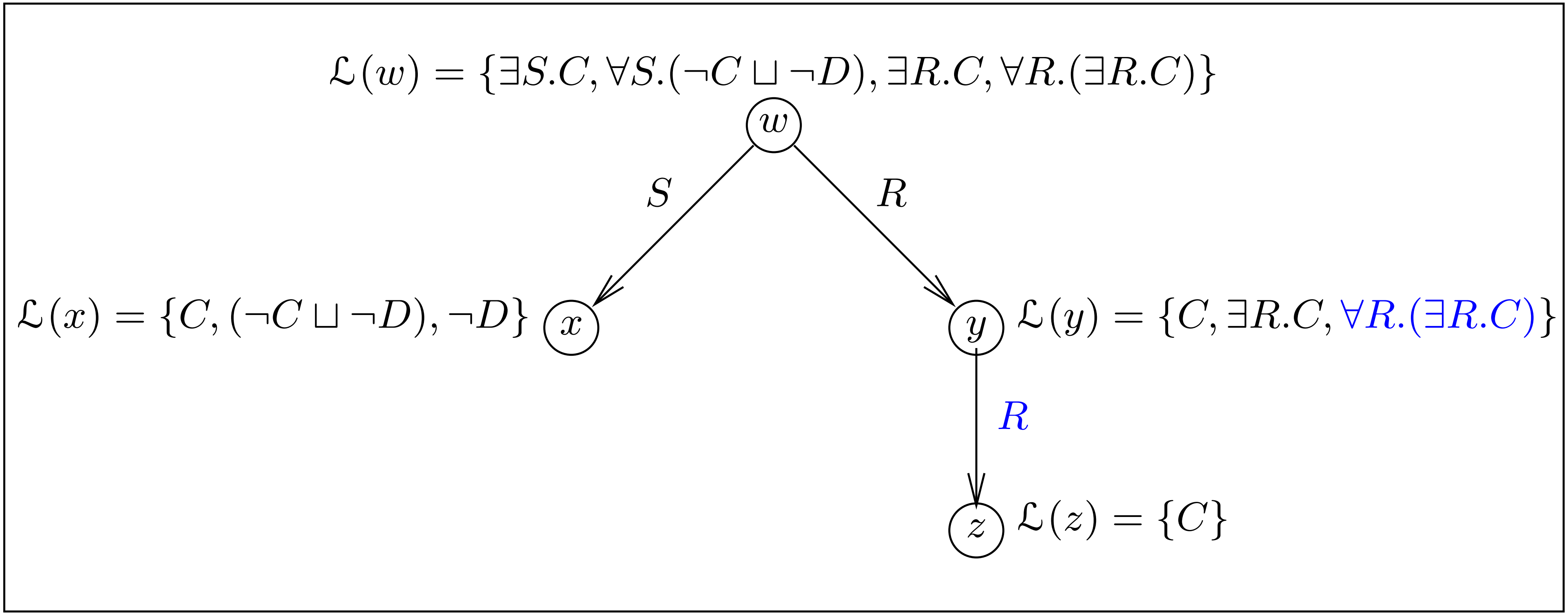
Tableaux Algorithm — Example

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



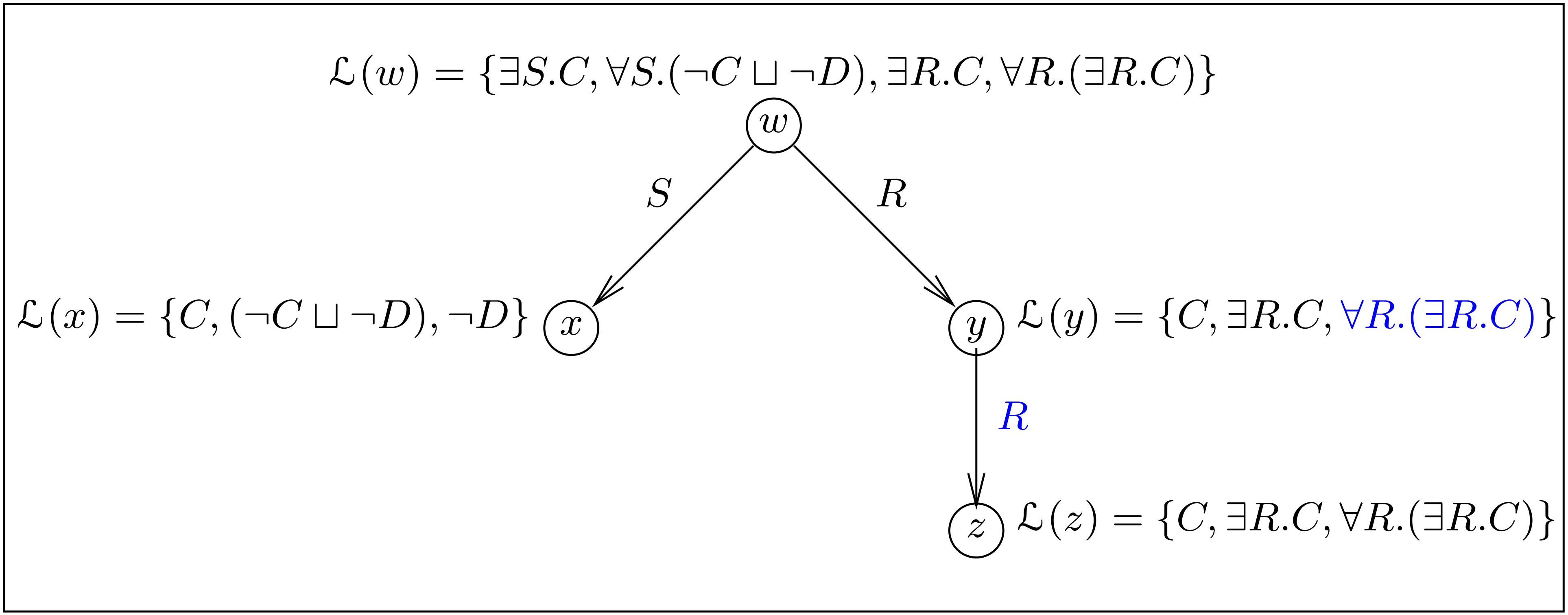
Tableaux Algorithm — Example

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



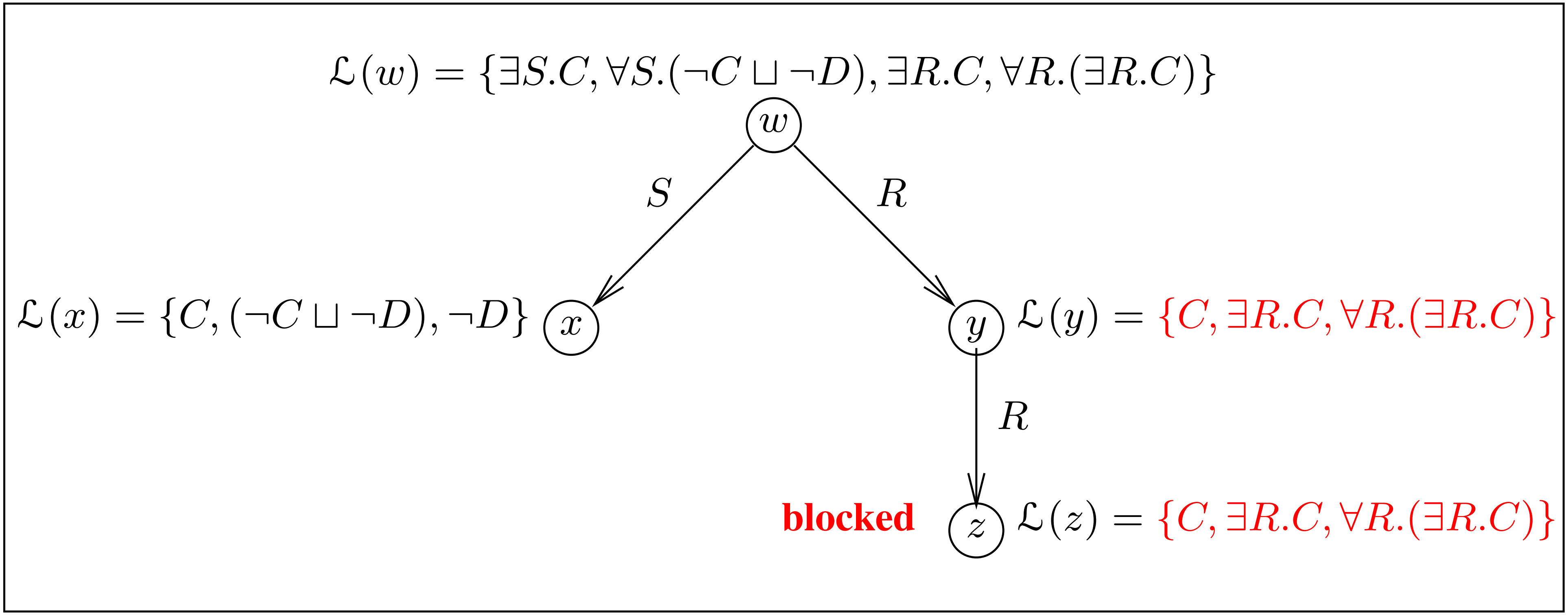
Tableaux Algorithm — Example

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



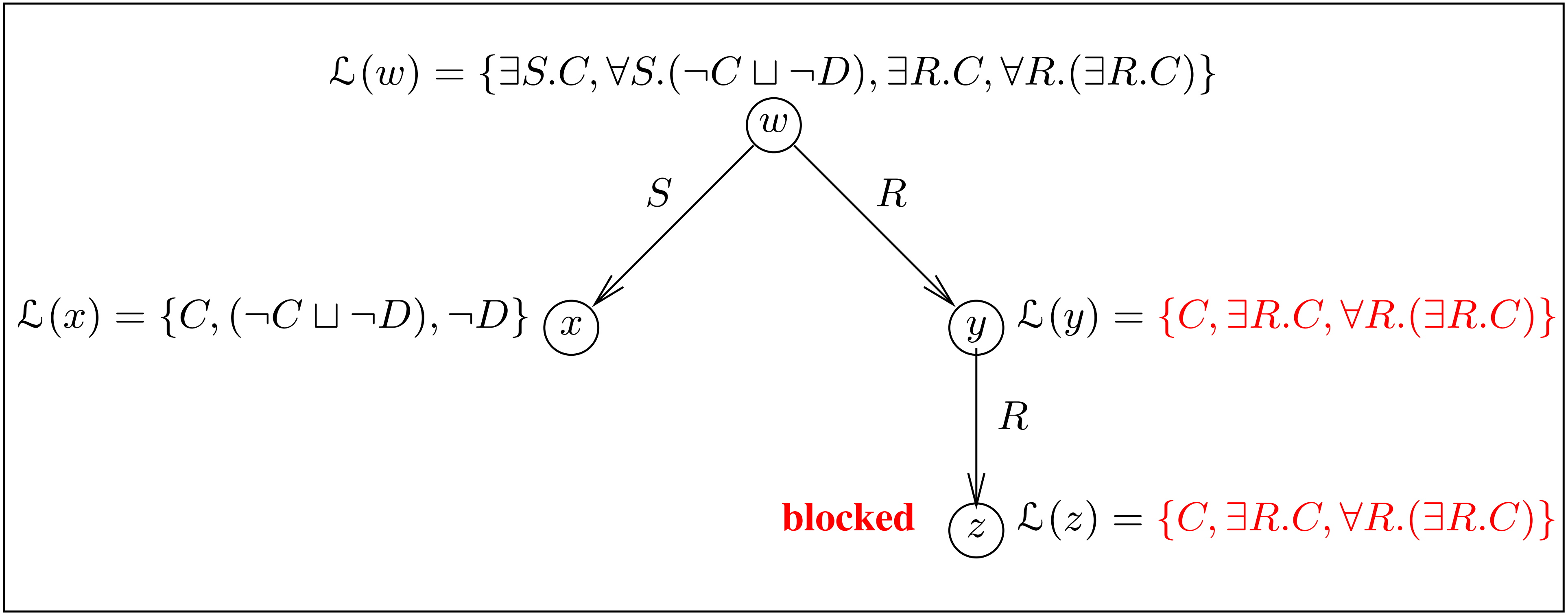
Tableaux Algorithm — Example

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



Tableaux Algorithm — Example

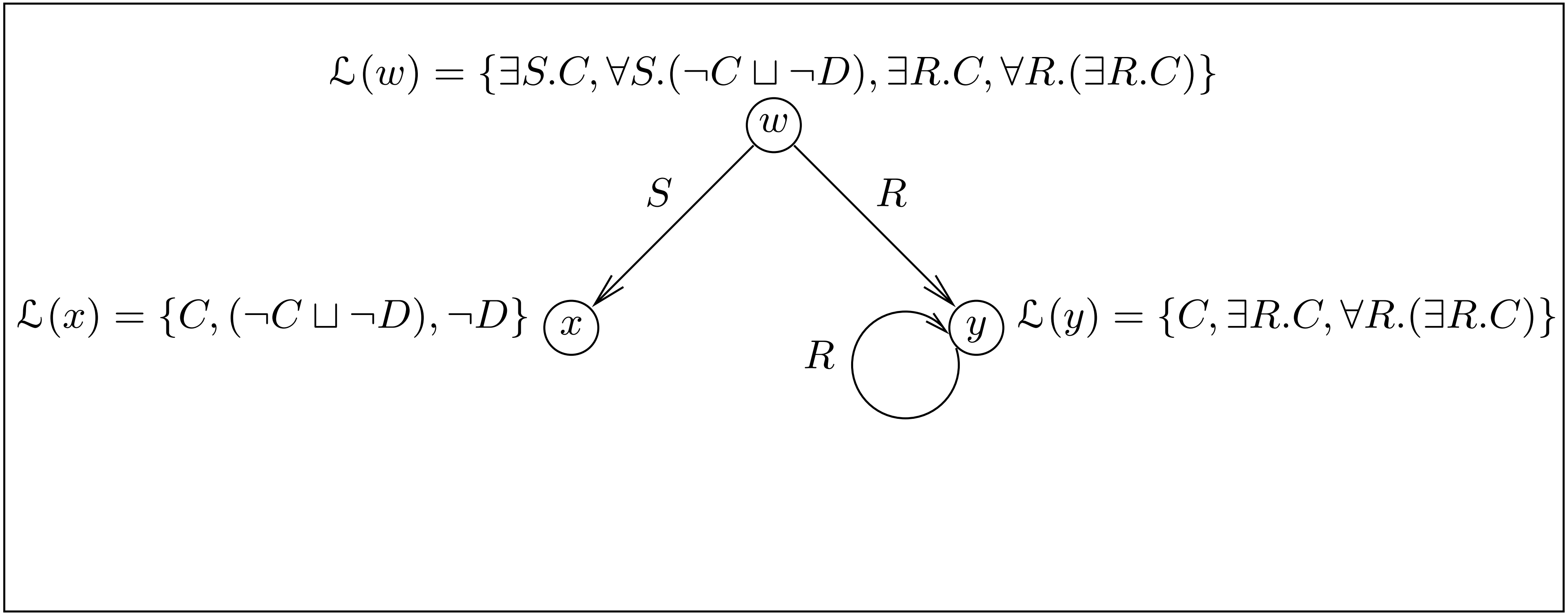
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



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

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



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

More Advanced Techniques

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 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

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.

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

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:

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

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 $R \in \mathbf{R}_+$

Careful Choice of Algorithm

- ☞ **Transitive roles** instead of transitive closure
  - Deterministic expansion of $\exists R.C$, even when $R \in \mathbf{R}_+$
  - (Relatively) simple blocking conditions

Careful Choice of Algorithm

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

Careful Choice of Algorithm

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

Careful Choice of Algorithm

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

Highly Optimised Implementation

Highly Optimised Implementation

- ☞ Naive implementation $\longrightarrow$ effective non-termination

Highly Optimised Implementation

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

Highly Optimised Implementation

- ☞ Naive implementation $\longrightarrow$ 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 $\longrightarrow$ 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

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

Dependency Directed Backtracking

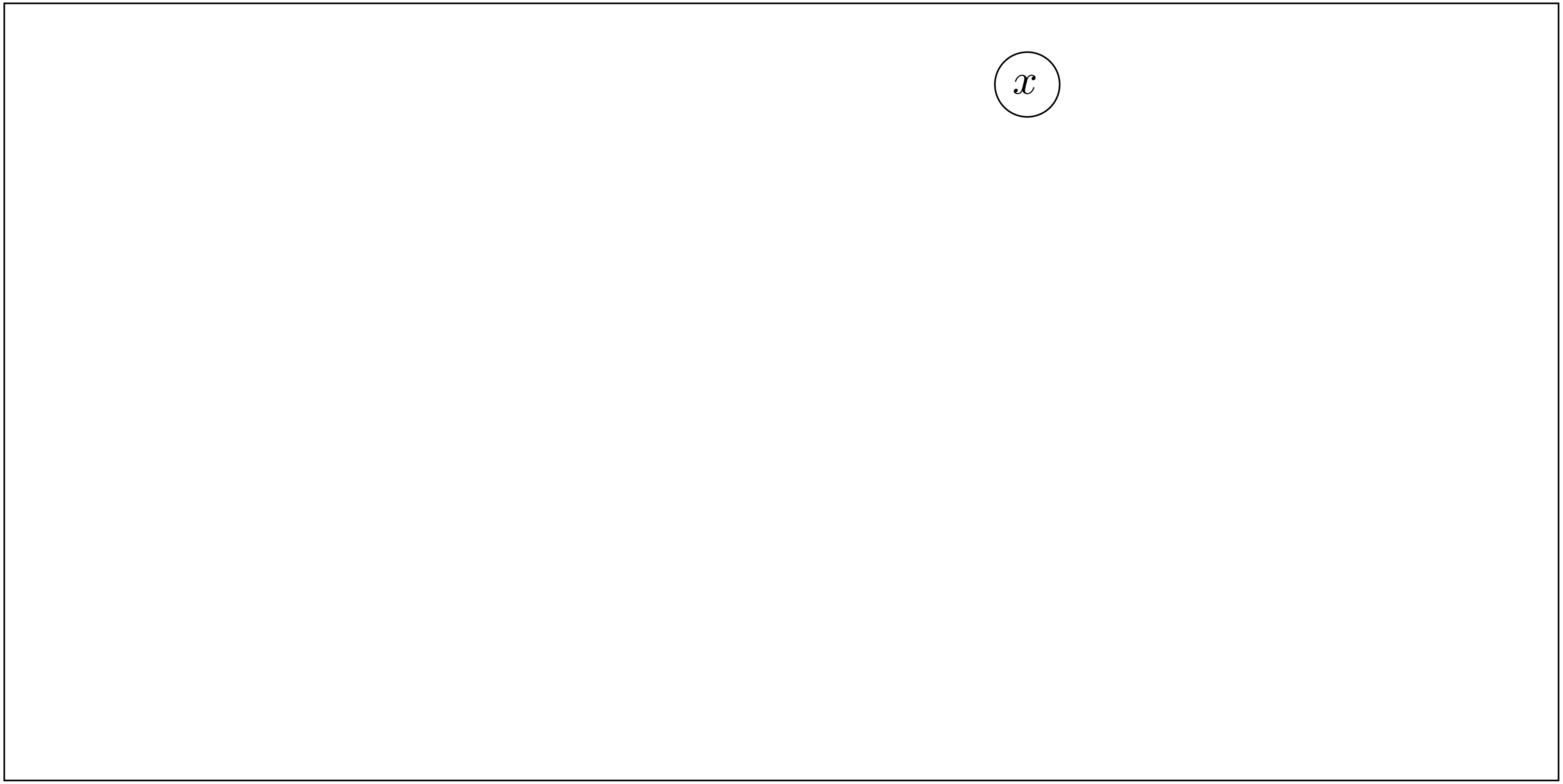
- Allows **rapid recovery** from bad branching choices
- Most commonly used technique is **backjumping**
  - Tag concepts introduced at **branch points** (e.g., when expanding disjunctions)
  - Expansion rules combine and **propagate tags**
  - On discovering a clash, **identify** most recently introduced 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) $\longrightarrow$ months++ (without)

Backjumping

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

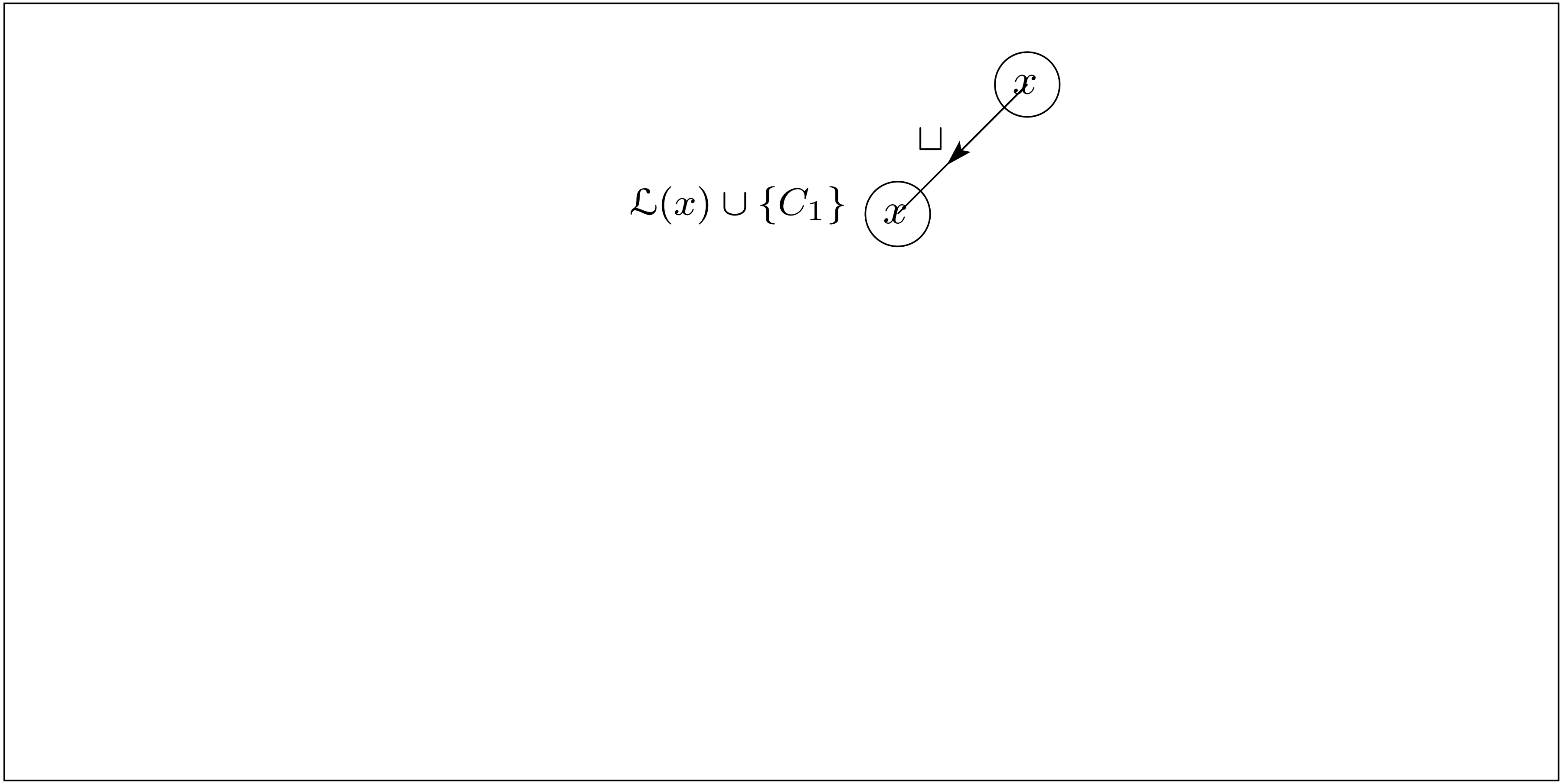
Backjumping

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



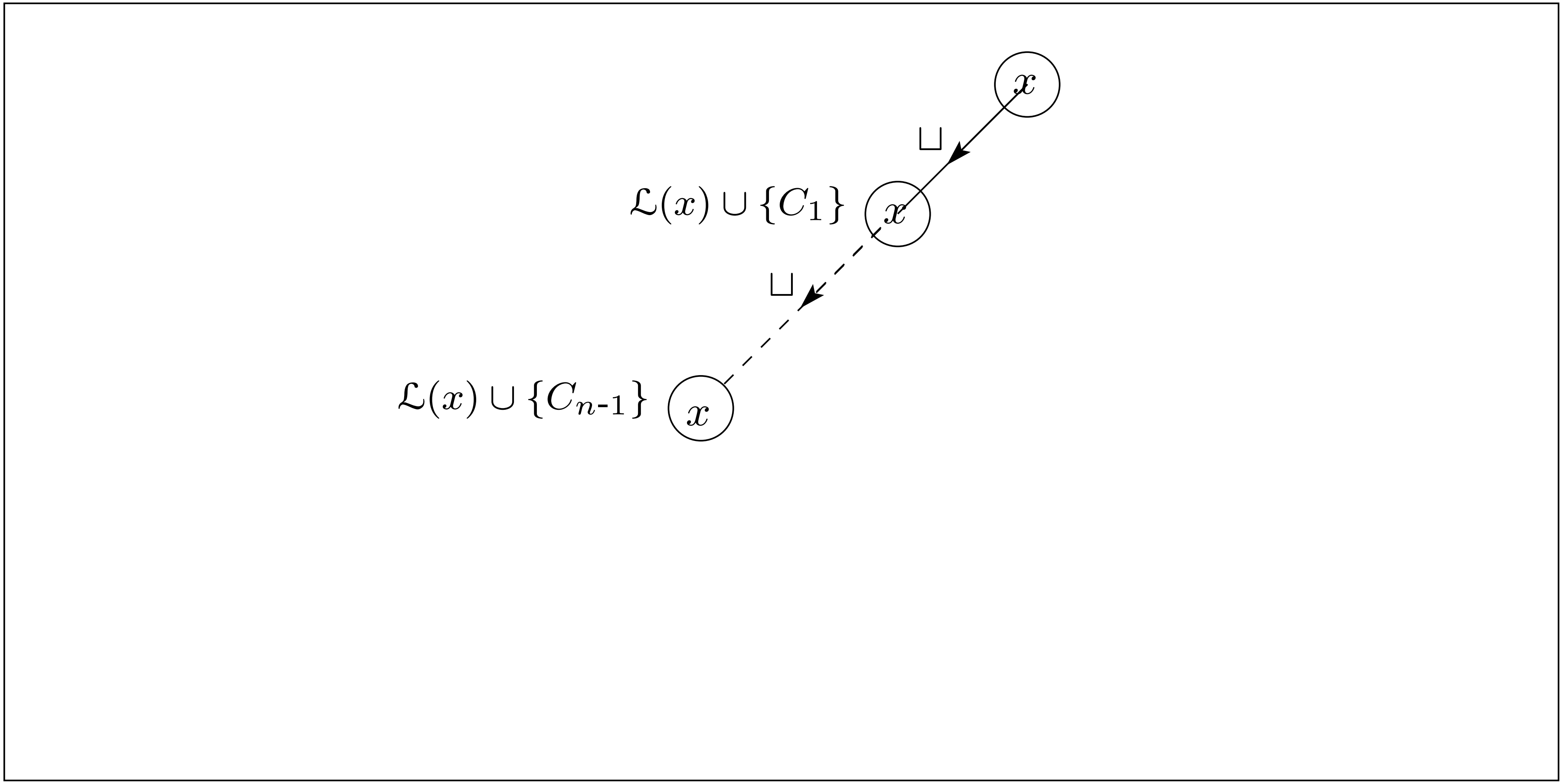
Backjumping

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



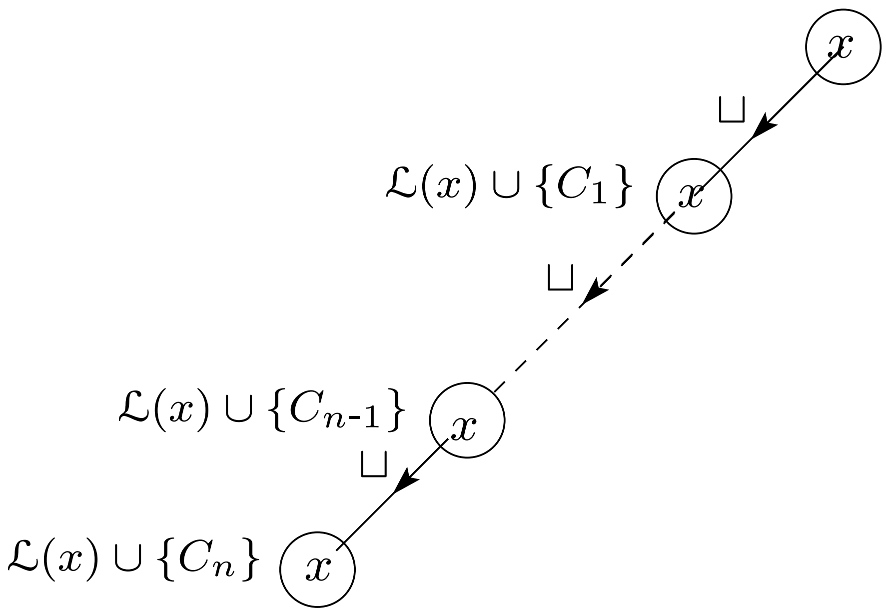
Backjumping

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



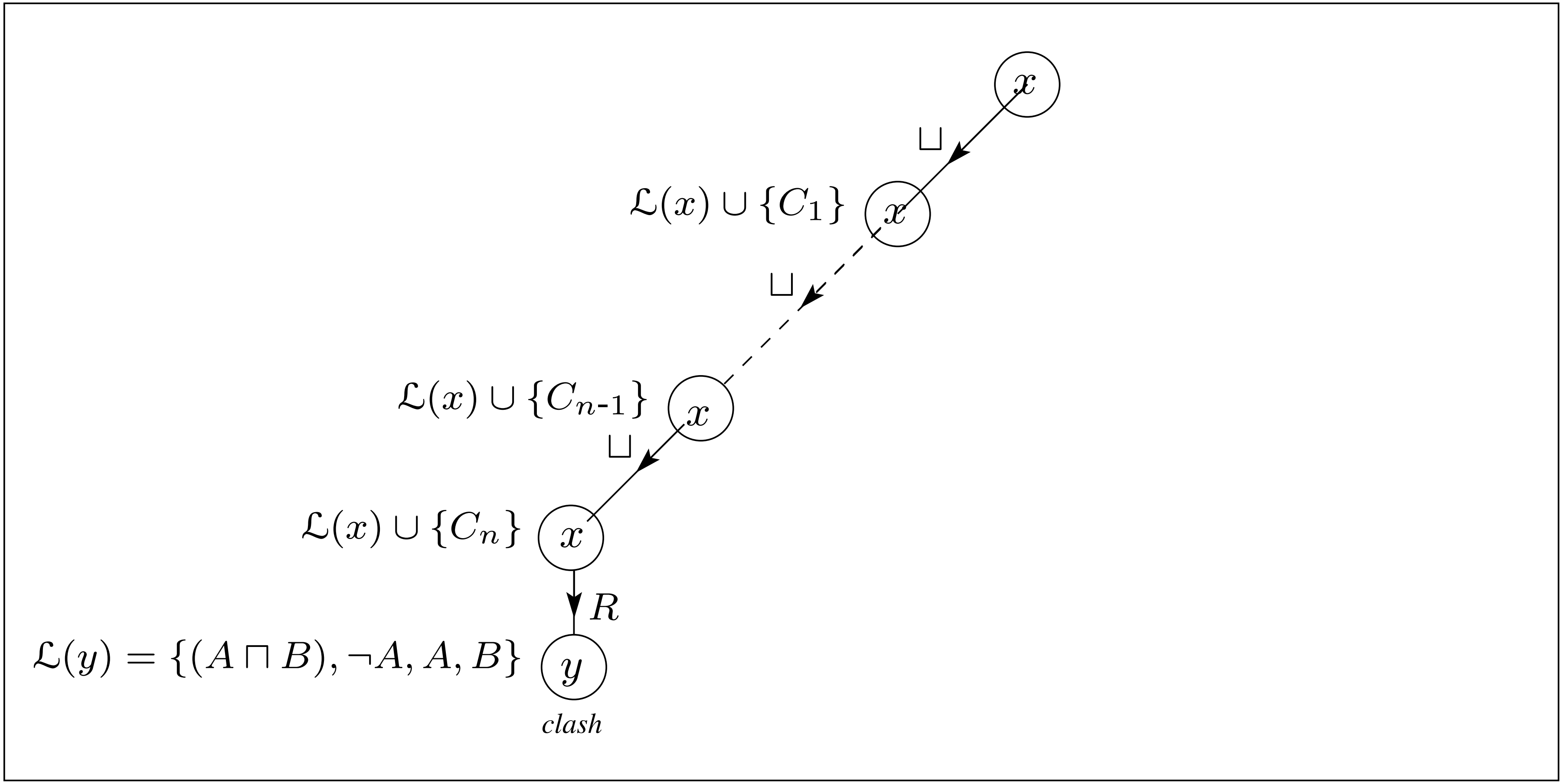
Backjumping

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



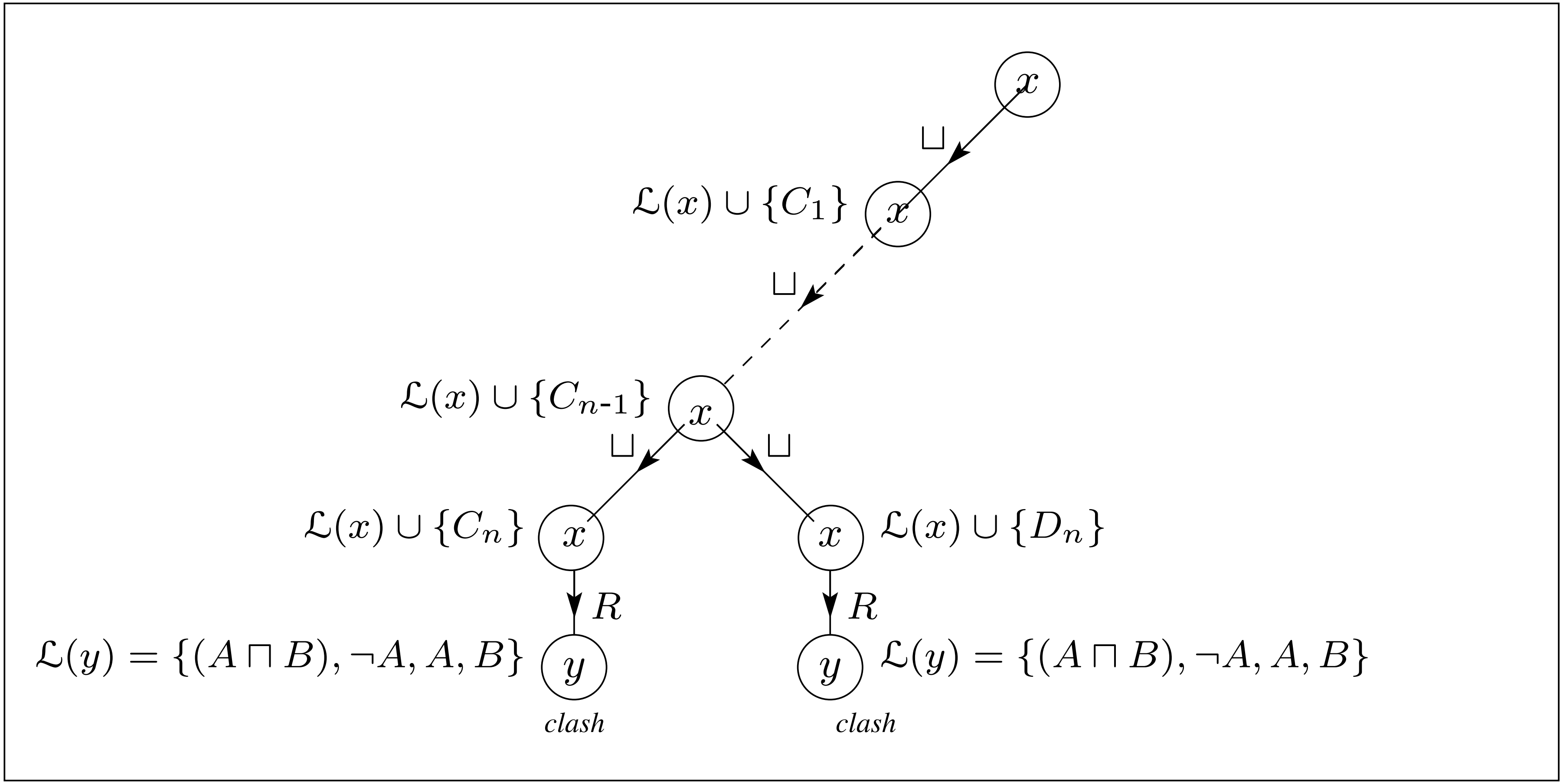
Backjumping

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



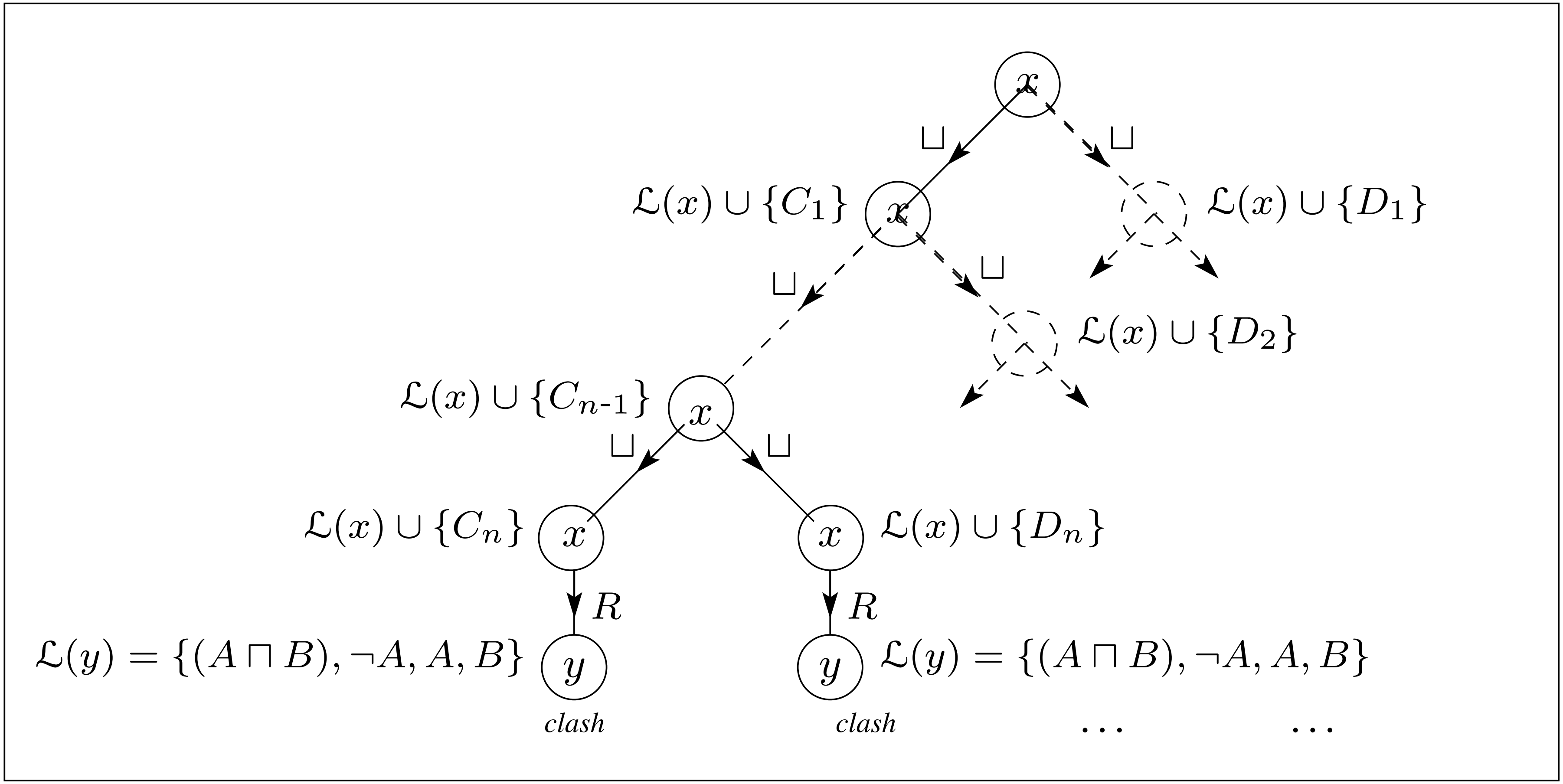
Backjumping

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



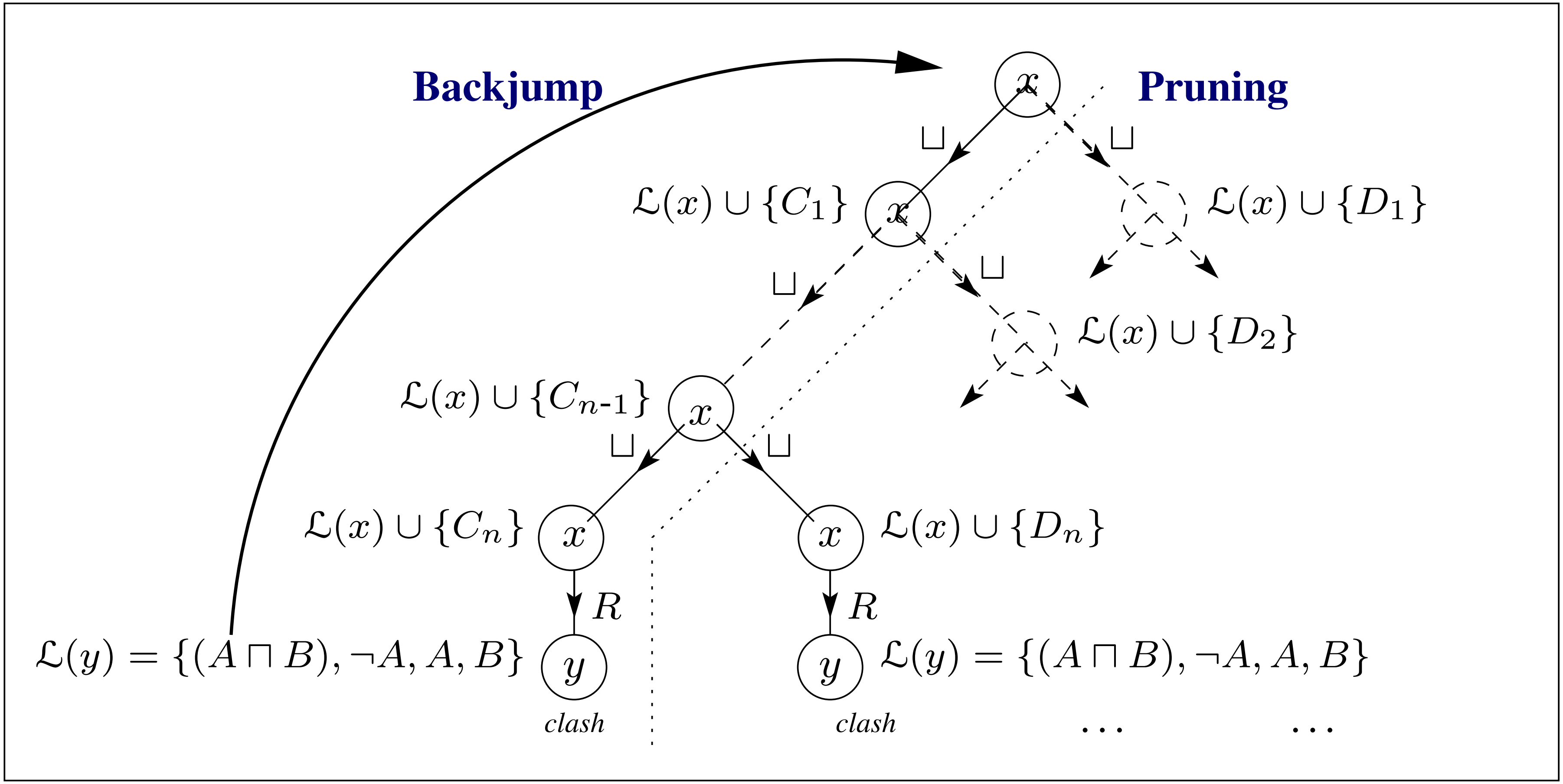
Backjumping

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



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

Challenges

☞ **Increased expressive power**

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

Challenges

- **Increased expressive power**
  - Existing DL systems implement (at most) $\mathcal{SHIQ}$
  - OWL extends $\mathcal{SHIQ}$ with datatypes and nominals
- **Scalability**
  - Very large KBs
  - Reasoning with (very large numbers of) individuals

Challenges

- **Increased expressive power**
  - Existing DL systems implement (at most) $\mathcal{SHIQ}$
  - OWL extends $\mathcal{SHIQ}$ with datatypes and nominals
- **Scalability**
  - Very large KBs
  - Reasoning with (very large numbers of) individuals
- **Other reasoning tasks**
  - Querying
  - Matching
  - Least common subsumer
  - . . .

Challenges

- **Increased expressive power**
  - Existing DL systems implement (at most) $\mathcal{SHIQ}$
  - OWL extends $\mathcal{SHIQ}$ with datatypes and nominals
- **Scalability**
  - Very large KBs
  - Reasoning with (very large numbers of) individuals
- **Other reasoning tasks**
  - Querying
  - Matching
  - Least common subsumer
  - . . .
- **Tools and Infrastructure**
  - Support for large scale ontological engineering and deployment

Increased Expressive Power: Datatypes

Increased Expressive Power: Datatypes

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

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 $\mathcal{SHOQ}(\mathbf{D})$ [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 $\mathcal{SHOQ}(\mathbf{D})$ [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 $\mathcal{SHOQ}(\mathbf{D})$ [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

Increased Expressive Power: Nominals

☞ OWL **oneOf** constructor equivalent to hybrid logic **nominals**

- Extensionally defined concepts, e.g., 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 $\equiv \{\text{France}, \text{Italy}, \ldots\}$
- 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: $\top \sqsubseteq \exists R.\{Spy\}$
    - Finite domains: $\{Spy\} \sqsubseteq \leqslant n R^-$

Increased Expressive Power: Nominals

- OWL **oneOf** constructor equivalent to hybrid logic **nominals**
  - Extensionally defined concepts, e.g., EU $\equiv \{\text{France}, \text{Italy}, \ldots\}$
- 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: $\top \sqsubseteq \exists R.\{Spy\}$
    - Finite domains: $\{Spy\} \sqsubseteq \leqslant n R^{-}$
- **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

Scalability

- Reasoning **hard** (ExpTime) even without nominals (i.e., $\mathcal{SHIQ}$)

Scalability

- ☞ Reasoning **hard** (ExpTime) even without nominals (i.e., $\mathcal{SHIQ}$)
- ☞ Web ontologies may grow **very large**

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

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)

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

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

Performance Solutions (Maybe)

Performance Solutions (Maybe)

- Excessive **memory usage**

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]

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 ≈100k concept KB [Haarslev & Möller]
- But KB only exploited small part of DL language

Other Reasoning Tasks

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

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)

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

Summary

- ☞ **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 $\mathcal{SHIQ}$ DL
  - Set to become W3C standard (OWL) & already widely adopted
  - Use of DL provides formal foundations and reasoning support

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 $\mathcal{SHIQ}$ 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

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 $\mathcal{SHIQ}$ 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 $\mathcal{SHIQ}$ 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)

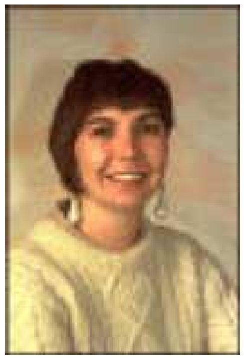


Acknowledgements

- 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

Select Bibliography

I. Horrocks. DAML+OIL: a reason-able web ontology language. In *Proc. of EDBT 2002*, number 2287 in Lecture Notes in Computer Science, pages 2–13. Springer-Verlag, Mar. 2002.

I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. Reviewing the design of DAML+OIL: An ontology language for the semantic web. In *Proc. of AAAI 2002*, 2002. To appear.

I. Horrocks and S. Tessaris. Querying the semantic web: a formal approach. In I. Horrocks and J. Hendler, editors, *Proc. of the 2002 International Semantic Web Conference (ISWC 2002)*, number 2342 in Lecture Notes in Computer Science. Springer-Verlag, 2002.

C. Lutz. *The Complexity of Reasoning with Concrete Domains*. PhD thesis, Teaching and Research Area for Theoretical Computer Science, RWTH Aachen, 2001.

Select Bibliography

I. Horrocks and U. Sattler. Ontology reasoning in the $\mathcal{SHOQ}(\mathbf{D})$ description logic. In B. Nebel, editor, *Proc. of IJCAI-01*, pages 199–204. Morgan Kaufmann, 2001.

F. Baader, S. Brandt, and R. Küsters. Matching under side conditions in description logics. In B. Nebel, editor, *Proc. of IJCAI-01*, pages 213–218, Seattle, Washington, 2001. Morgan Kaufmann.

A. Borgida, E. Franconi, and I. Horrocks. Explaining $\mathcal{ALC}$ subsumption. In *Proc. of ECAI 2000*, pages 209–213. IOS Press, 2000.

D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. A principled approach to data integration and reconciliation in data warehousing. In *Proceedings of the International Workshop on Design and Management of Data Warehouses (DWDM'99)*, 1999.