Reasoning with Expressive Description Logics: Theory and Practice

Ian Horrocks

horrocks@cs.man.ac.uk

University of Manchester
Manchester, UK

Talk Outline

Talk Outline

Introduction to Description Logics (DLs)

Talk Outline

- **Introduction to Description Logics (DLs)**
- **Reasoning techniques**

Talk Outline

- Introduction to Description Logics (DLs)
- Reasoning techniques
- Implementing DL systems

Talk Outline

- Introduction to Description Logics (DLs)
- Reasoning techniques
- Implementing DL systems
- DL applications

Talk Outline

- **Introduction to Description Logics (DLs)**
- **Reasoning techniques**
- **Implementing DL systems**
- **DL applications**
  - **Including demos** (time permitting)

Introduction to DLs

What are Description Logics?

What are Description Logics?

A family of logic based Knowledge Representation formalisms

What are Description Logics?

A family of logic based Knowledge Representation formalisms

☞ Based on **concepts** (classes) and **roles**

- Concepts (classes) are interpreted as sets of objects
- Roles are interpreted as binary relations on objects

What are Description Logics?

A family of logic based Knowledge Representation formalisms

- ☞ Based on **concepts** (classes) and **roles**
  - Concepts (classes) are interpreted as sets of objects
  - Roles are interpreted as binary relations on objects
- ☞ Descendants of semantic networks, frame based systems and KL-ONE

What are Description Logics?

A family of logic based Knowledge Representation formalisms

☞ Based on **concepts** (classes) and **roles**
  - Concepts (classes) are interpreted as sets of objects
  - Roles are interpreted as binary relations on objects

☞ Descendants of semantic networks, frame based systems and KL-ONE

☞ Decidable fragments of FOL
  - Many DLs are fragments of L2, C2 or the **Guarded Fragment**

What are Description Logics?

A family of logic based Knowledge Representation formalisms

- ☞ Based on **concepts** (classes) and **roles**
  - Concepts (classes) are interpreted as sets of objects
  - Roles are interpreted as binary relations on objects
- ☞ Descendants of semantic networks, frame based systems and KL-ONE
- ☞ Decidable fragments of FOL
  - Many DLs are fragments of L2, C2 or the **Guarded Fragment**
- ☞ Closely related to **propositional modal logics**

What are Description Logics?

A family of logic based Knowledge Representation formalisms

- ☞ Based on **concepts** (classes) and **roles**
  - Concepts (classes) are interpreted as sets of objects
  - Roles are interpreted as binary relations on objects
- ☞ Descendants of semantic networks, frame based systems and KL-ONE
- ☞ Decidable fragments of FOL
  - Many DLs are fragments of L2, C2 or the **Guarded Fragment**
- ☞ Closely related to **propositional modal logics**
- ☞ Also known as terminological logics, concept languages, etc.

What are Description Logics?

A family of logic based Knowledge Representation formalisms

- ☞ Based on **concepts** (classes) and **roles**
  - Concepts (classes) are interpreted as sets of objects
  - Roles are interpreted as binary relations on objects
- ☞ Descendants of semantic networks, frame based systems and KL-ONE
- ☞ Decidable fragments of FOL
  - Many DLs are fragments of L2, C2 or the **Guarded Fragment**
- ☞ Closely related to **propositional modal logics**
- ☞ Also known as terminological logics, concept languages, etc.
- ☞ Key features of DLs are
  - Well defined semantics (they are logics)
  - Provision of inference services

DL Applications

DLs have may applications including:

DL Applications

DLs have may applications including:

Terminological KR (including **Ontologies**)

- Medical terminology/controlled vocabulary (**Galen**)
- Bio-ontologies (**Tambis**, **GO**)
- Web based ontology languages (**OIL**, **DAML+OIL**)

DL Applications

DLs have may applications including:

Terminological KR (including **Ontologies**)

- ☞ Medical terminology/controlled vocabulary (**Galen**)
- ☞ Bio-ontologies (**Tambis**, **GO**)
- ☞ Web based ontology languages (**OIL**, **DAML+OIL**)

Configuration

- ☞ Classic system used to configure telecom equipment

DL Applications

DLs have may applications including:

Terminological KR (including **Ontologies**)

- ☞ Medical terminology/controlled vocabulary (**Galen**)
- ☞ Bio-ontologies (**Tambis**, **GO**)
- ☞ Web based ontology languages (**OIL**, **DAML+OIL**)

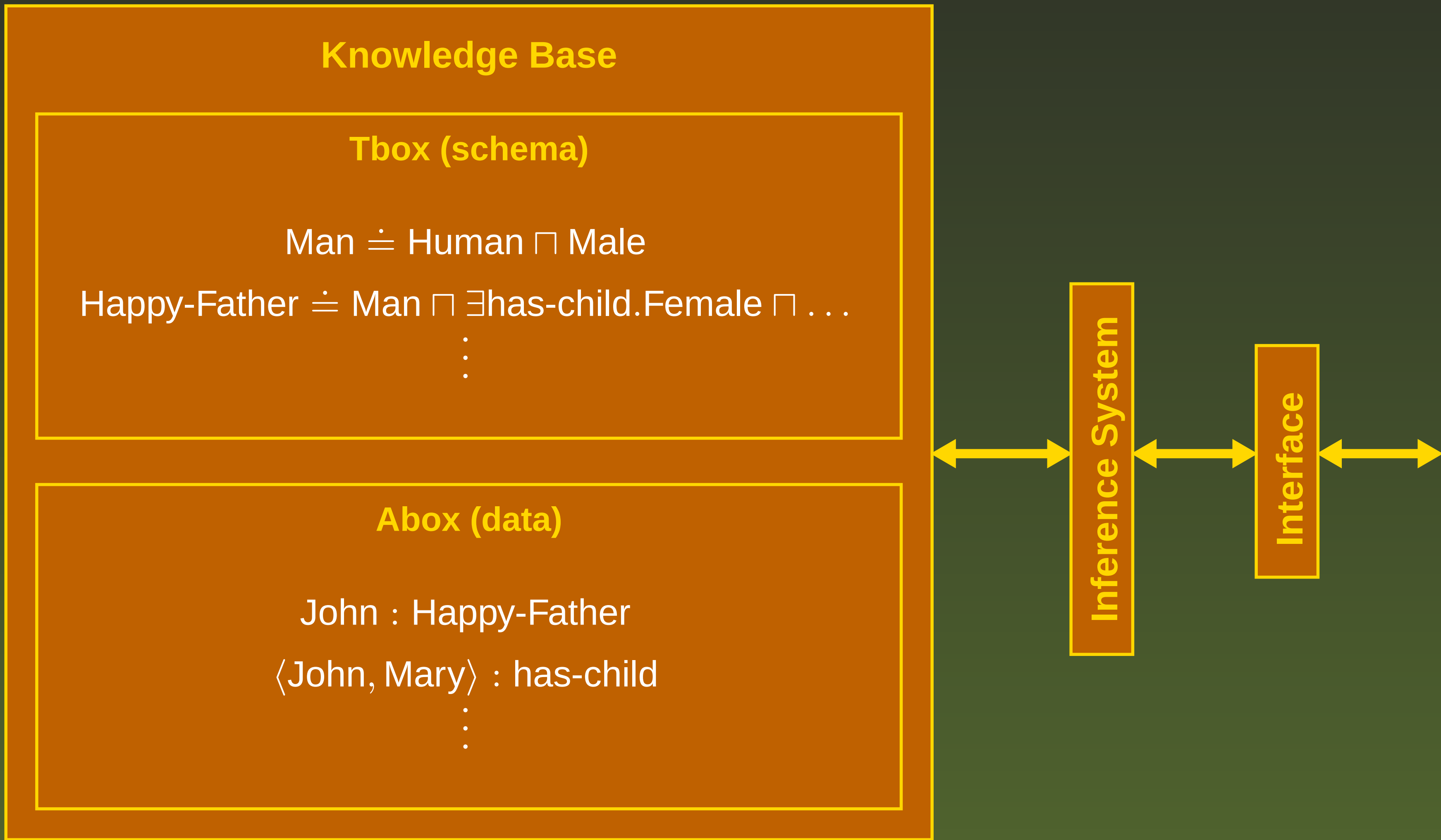
Configuration

- ☞ Classic system used to configure telecom equipment

Database schema and query reasoning

- ☞ Schema design and query optimisation
- ☞ Interoperability and federation
- ☞ Query containment (w.r.t. schema)

DL System Architecture



DL Constructors

Particular DLs characterised by **set of constructors** provided for building complex concepts and roles from simpler ones

DL Constructors

Particular DLs characterised by **set of constructors** provided for building complex concepts and roles from simpler ones

- Usually include at least:
  - Conjunction ($\sqcap$), disjunction ($\sqcup$), negation ($\neg$)
  - Restricted (guarded) forms of quantification ($\exists$, $\forall$)

DL Constructors

Particular DLs characterised by **set of constructors** provided for building complex concepts and roles from simpler ones

- ☞ Usually include at least:
  - Conjunction ($\sqcap$), disjunction ($\sqcup$), negation ($\neg$)
  - Restricted (guarded) forms of quantification ($\exists$, $\forall$)
- ☞ This basic DL is known as $\mathcal{ALC}$

DL Constructors

Particular DLs characterised by **set of constructors** provided for building complex concepts and roles from simpler ones

- Usually include at least:
  - Conjunction ($\sqcap$), disjunction ($\sqcup$), negation ($\neg$)
  - Restricted (guarded) forms of quantification ($\exists$, $\forall$)
- This basic DL is known as $\mathcal{ALC}$

For example, concept **Happy Father** in $\mathcal{ALC}$:

$$\begin{array}{lll} \text{Man} & \sqcap & \exists\text{has-child.Male} \\ & \sqcap & \exists\text{has-child.Female} \\ & \sqcap & \forall\text{has-child.}(\text{Doctor} \sqcup \text{Lawyer}) \end{array}$$

DL Syntax and Semantics

Semantics given by **interpretation** $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$

DL Syntax and Semantics

Semantics given by **interpretation** $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$

Constructor	Syntax	Example	Semantics
atomic concept	A	Human	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
atomic role	R	has-child	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
and for C, D concepts and R a role name			
conjunction	$C \sqcap D$	Human $\sqcap$ Male	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
disjunction	$C \sqcup D$	Doctor $\sqcup$ Lawyer	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
negation	$\neg C$	$\neg$Male	$\Delta^{\mathcal{I}} \setminus C$
exists restr.	$\exists R.C$	$\exists$has-child.Male	$\{x \mid \exists y.\langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
value restr.	$\forall R.C$	$\forall$has-child.Doctor	$\{x \mid \forall y.\langle x, y\rangle \in R^{\mathcal{I}} \implies y \in C^{\mathcal{I}}\}$

Other DL Constructors

Many different DLs/DL constructors have been investigated, e.g.

Other DL Constructors

Many different DLs/DL constructors have been investigated, e.g.

Constructor	Syntax	Example	Semantics
number restr.	$\geqslant nR$	$\geqslant 3$ has-child	$\{x \mid \lvert\{y.\langle x,y\rangle \in R^{\mathcal{I}}\}\rvert \geqslant n\}$
	$\leqslant nR$	$\leqslant 1$ has-mother	$\{x \mid \lvert\{y.\langle x,y\rangle \in R^{\mathcal{I}}\}\rvert \leqslant n\}$
inverse role	R^-	has-child$^-$	$\{\langle x,y\rangle \mid \langle y,x\rangle \in R^{\mathcal{I}}\}$
trans. role	R^*	has-child*	$(R^{\mathcal{I}})^*$
concrete domain	$f_1,\ldots,f_n.P$	earns spends $<$	$\{x \mid P(f_1^{\mathcal{I}},\ldots,f_n^{\mathcal{I}})\}$

$\vdots$

DL Knowledge Base (Tbox)

Terminological part (**Tbox**) is set of axioms describing **structure** of domain

DL Knowledge Base (Tbox)

Terminological part (**Tbox**) is set of axioms describing **structure** of domain

- **Definition** axioms introduce macros/names for concepts
  - $A \doteq C$, $A \sqsubseteq C$
  - $\text{Father} \doteq \text{Man} \sqcap \exists\text{has-child.Human}$
  - $\text{Human} \sqsubseteq \text{Animal} \sqcap \text{Biped}$

DL Knowledge Base (Tbox)

Terminological part (**Tbox**) is set of axioms describing **structure** of domain

- **Definition** axioms introduce macros/names for concepts
  - $A \doteq C$, $A \sqsubseteq C$
  - $\mathsf{Father} \doteq \mathsf{Man} \sqcap \exists \mathsf{has\text{-}child}.\mathsf{Human}$
  - $\mathsf{Human} \sqsubseteq \mathsf{Animal} \sqcap \mathsf{Biped}$
- **Inclusion** (GCI) axioms assert subsumption relations
  - $C \sqsubseteq D$ (note $C \doteq D$ equivalent to $C \sqsubseteq D$ and $D \sqsubseteq C$)
  - $\exists \mathsf{has\text{-}degree}.\mathsf{Masters} \sqsubseteq \exists \mathsf{has\text{-}degree}.\mathsf{Bachelors}$

DL Knowledge Base (Tbox)

Terminological part (**Tbox**) is set of axioms describing **structure** of domain

- **Definition** axioms introduce macros/names for concepts
  - $A \doteq C$, $A \sqsubseteq C$
  - Father $\doteq$ Man $\sqcap$ $\exists$has-child.Human
  - Human $\sqsubseteq$ Animal $\sqcap$ Biped
- **Inclusion** (GCI) axioms assert subsumption relations
  - $C \sqsubseteq D$ (note $C \doteq D$ equivalent to $C \sqsubseteq D$ and $D \sqsubseteq C$)
  - $\exists$has-degree.Masters $\sqsubseteq$ $\exists$has-degree.Bachelors

An interpretation $\mathcal{I}$ **satisfies**

- $C \doteq D$ iff $C^{\mathcal{I}} = D^{\mathcal{I}}$ $\quad$ $C \sqsubseteq D$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
- A **Tbox** $\mathcal{T}$ iff it satisfies every axiom in $\mathcal{T}$ ($\mathcal{I} \models \mathcal{T}$)

DL Knowledge Base (Abox)

Assertional part (**Abox**) is set of axioms describing **concrete situation**

DL Knowledge Base (Abox)

Assertional part (**Abox**) is set of axioms describing **concrete situation**

Concept assertions

$a : C$

John : Man $\sqcap$ $\exists$has-child.Female

DL Knowledge Base (Abox)

Assertional part (**Abox**) is set of axioms describing **concrete situation**

Concept assertions

$$a : C$$

$$\text{John} : \text{Man} \sqcap \exists \text{has-child.Female}$$

Role assertions

$$\langle a, b \rangle : R$$

$$\langle \text{John}, \text{Mary} \rangle : \text{has-child}$$

DL Knowledge Base (Abox)

Assertional part (**Abox**) is set of axioms describing **concrete situation**

Concept assertions

$a : C$

$\text{John} : \text{Man} \sqcap \exists\text{has-child}.\text{Female}$

Role assertions

$\langle a, b \rangle : R$

$\langle \text{John}, \text{Mary} \rangle : \text{has-child}$

An interpretation $\mathcal{I}$ **satisfies**

$a : C$ iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$ $\langle a, b \rangle : R$ iff $\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$

An **Abox** $\mathcal{A}$ iff it satisfies every axiom in $\mathcal{A}$ ($\mathcal{I} \models \mathcal{A}$)

A **KB** $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$ iff it satisfies both $\mathcal{T}$ and $\mathcal{A}$ ($\mathcal{I} \models \Sigma$)

Basic Inference Problems

Basic Inference Problems

Subsumption (structure knowledge, compute taxonomy)

$C \sqsubseteq D$?   Is $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ in all interpretations?

Basic Inference Problems

Subsumption (structure knowledge, compute taxonomy)

$C \sqsubseteq D$? Is $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ in all interpretations?

Subsumption w.r.t. Tbox $\mathcal{T}$

$C \sqsubseteq_{\mathcal{T}} D$? Is $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ in all models of $\mathcal{T}$?

Basic Inference Problems

Subsumption (structure knowledge, compute taxonomy)

$C \sqsubseteq D$? Is $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ in all interpretations?

Subsumption w.r.t. Tbox $\mathcal{T}$

$C \sqsubseteq_{\mathcal{T}} D$? Is $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ in all models of $\mathcal{T}$?

Consistency

Is C consistent w.r.t. $\mathcal{T}$? Is there a model $\mathcal{I}$ of $\mathcal{T}$ s.t. $C^{\mathcal{I}} \neq \emptyset$?

Basic Inference Problems

Subsumption (structure knowledge, compute taxonomy)

$C \sqsubseteq D$? Is $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ in all interpretations?

Subsumption w.r.t. Tbox $\mathcal{T}$

$C \sqsubseteq_{\mathcal{T}} D$? Is $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ in all models of $\mathcal{T}$?

Consistency

Is C consistent w.r.t. $\mathcal{T}$? Is there a model $\mathcal{I}$ of $\mathcal{T}$ s.t. $C^{\mathcal{I}} \neq \emptyset$?

KB Consistency

Is $\langle \mathcal{T}, \mathcal{A} \rangle$ consistent? Is there a model $\mathcal{I}$ of $\langle \mathcal{T}, \mathcal{A} \rangle$?

Basic Inference Problems

Subsumption (structure knowledge, compute taxonomy)

$C \sqsubseteq D$? Is $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ in all interpretations?

Subsumption w.r.t. Tbox $\mathcal{T}$

$C \sqsubseteq_{\mathcal{T}} D$? Is $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ in all models of $\mathcal{T}$?

Consistency

Is C consistent w.r.t. $\mathcal{T}$? Is there a model $\mathcal{I}$ of $\mathcal{T}$ s.t. $C^{\mathcal{I}} \neq \emptyset$?

KB Consistency

Is $\langle \mathcal{T}, \mathcal{A} \rangle$ consistent? Is there a model $\mathcal{I}$ of $\langle \mathcal{T}, \mathcal{A} \rangle$?

Problems are **closely related**:

$C \sqsubseteq_{\mathcal{T}} D$	iff	$C \sqcap \neg D$ is inconsistent w.r.t. $\mathcal{T}$
C is consistent w.r.t. $\mathcal{T}$	iff	$C \not\sqsubseteq_{\mathcal{T}} A \sqcap \neg A$

Reasoning Techniques

Subsumption and Satisfiability

Subsumption and Satisfiability

☞ Subsumption transformed into satisfiability

Subsumption and Satisfiability

- Subsumption transformed into satisfiability
- Tableaux algorithm used to test satisfiability

Subsumption and Satisfiability

- ☞ Subsumption transformed into satisfiability
- ☞ Tableaux algorithm used to test satisfiability
  - Try to build model of concept C

Subsumption and Satisfiability

- ☞ Subsumption transformed into satisfiability
- ☞ Tableaux algorithm used to test satisfiability
  - Try to build model of concept C
  - Model represented by tree $\mathbf{T}$
    - ➔ Nodes in $\mathbf{T}$ correspond to individuals in model
    - ➔ Nodes labeled with sets of subconcepts of C
    - ➔ Edges labeled with role names in C

Subsumption and Satisfiability

- Subsumption transformed into satisfiability
- Tableaux algorithm used to test satisfiability
  - Try to build model of concept C
  - Model represented by tree $\mathbf{T}$
    - Nodes in $\mathbf{T}$ correspond to individuals in model
    - Nodes labeled with sets of subconcepts of C
    - Edges labeled with role names in C
  - Start from root node labeled $\{C\}$

Subsumption and Satisfiability

- Subsumption transformed into satisfiability
- Tableaux algorithm used to test satisfiability
  - Try to build model of concept C
  - Model represented by tree $\mathbf{T}$
    - Nodes in $\mathbf{T}$ correspond to individuals in model
    - Nodes labeled with sets of subconcepts of C
    - Edges labeled with role names in C
  - Start from root node labeled $\{C\}$
  - Apply expansion rules to node labels until
    - Expansion completed (tree represents valid model)
    - Contradictions prove there is no model

Subsumption and Satisfiability

- Subsumption transformed into satisfiability
- Tableaux algorithm used to test satisfiability
  - Try to build model of concept C
  - Model represented by tree $\mathbf{T}$
    - Nodes in $\mathbf{T}$ correspond to individuals in model
    - Nodes labeled with sets of subconcepts of C
    - Edges labeled with role names in C
  - Start from root node labeled $\{C\}$
  - Apply expansion rules to node labels until
    - Expansion completed (tree represents valid model)
    - Contradictions prove there is no model
  - Non-deterministic expansion $\longrightarrow$ search (e.g., $C \sqcup D$)

Subsumption and Satisfiability

- Subsumption transformed into satisfiability
- Tableaux algorithm used to test satisfiability
  - Try to build model of concept C
  - Model represented by tree $\mathbf{T}$
    - Nodes in $\mathbf{T}$ correspond to individuals in model
    - Nodes labeled with sets of subconcepts of C
    - Edges labeled with role names in C
  - Start from root node labeled $\{C\}$
  - Apply expansion rules to node labels until
    - Expansion completed (tree represents valid model)
    - Contradictions prove there is no model
  - Non-deterministic expansion $\longrightarrow$ search (e.g., $C \sqcup D$)
  - **Blocking** ensures termination (with expressive DLs)

Tableaux Expansion

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 Expansion

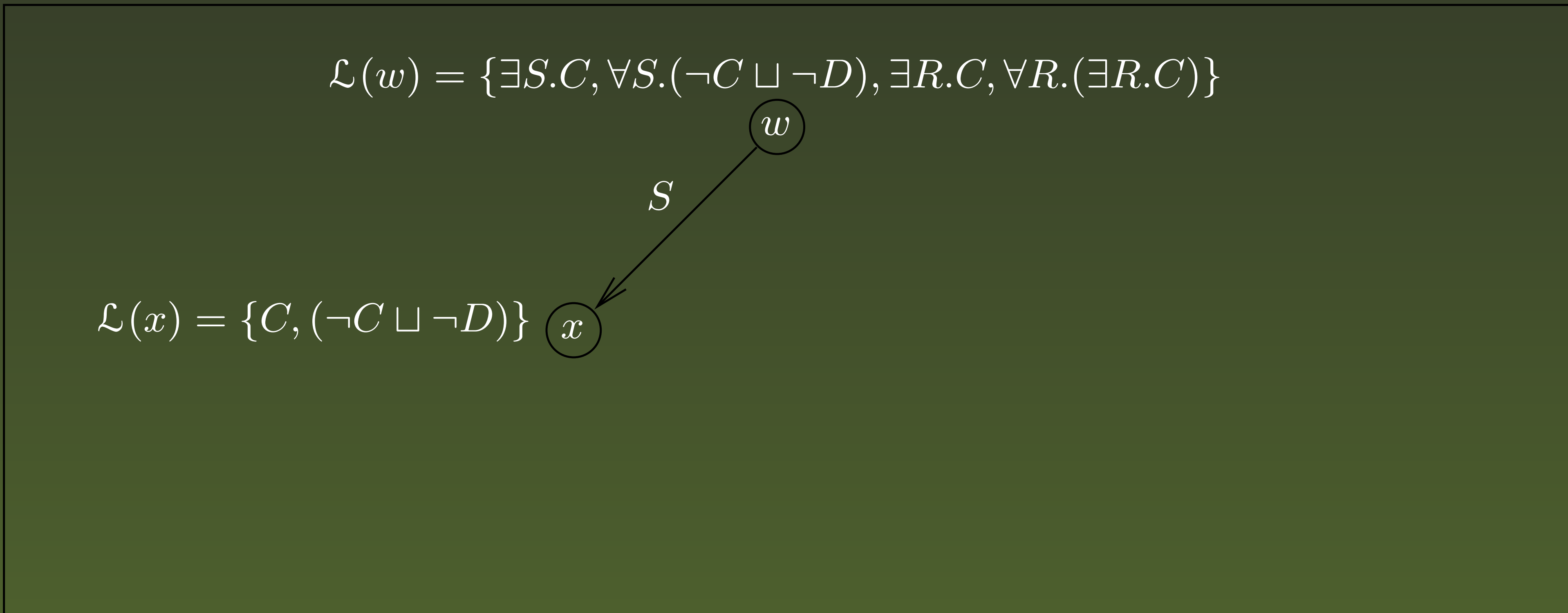
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role

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

w

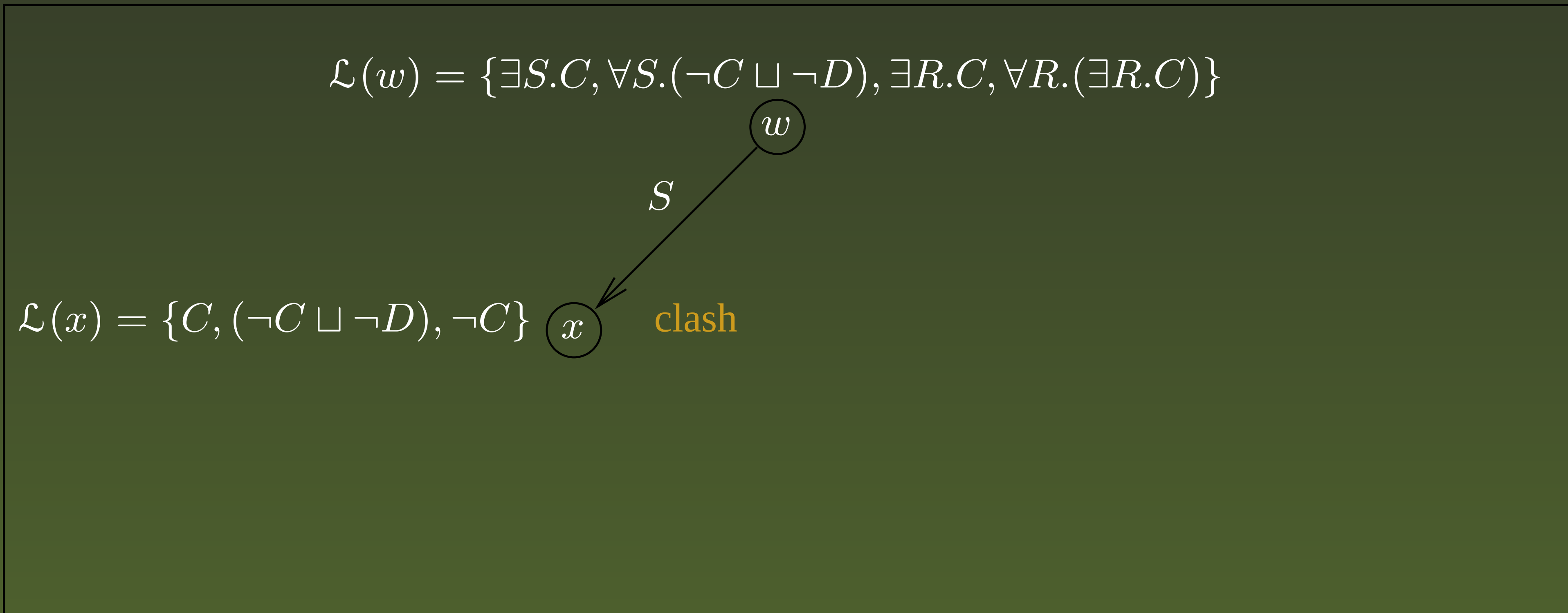
Tableaux Expansion

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



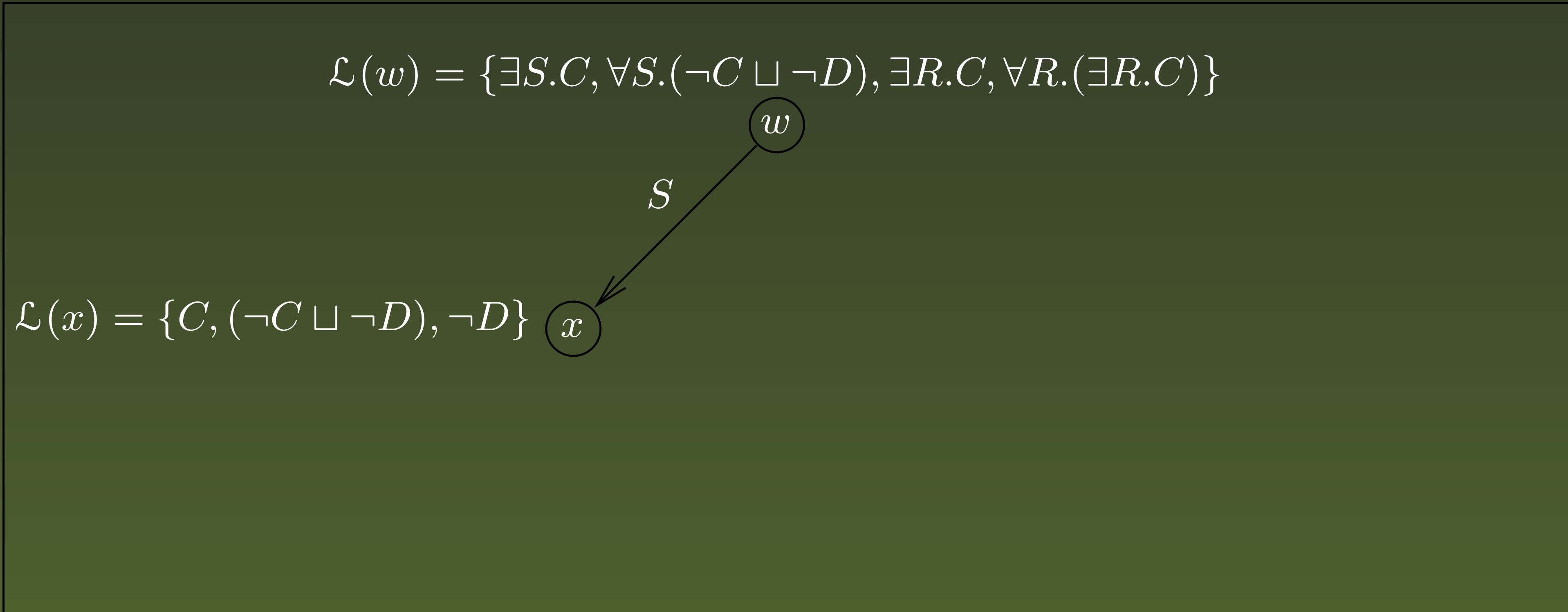
Tableaux Expansion

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



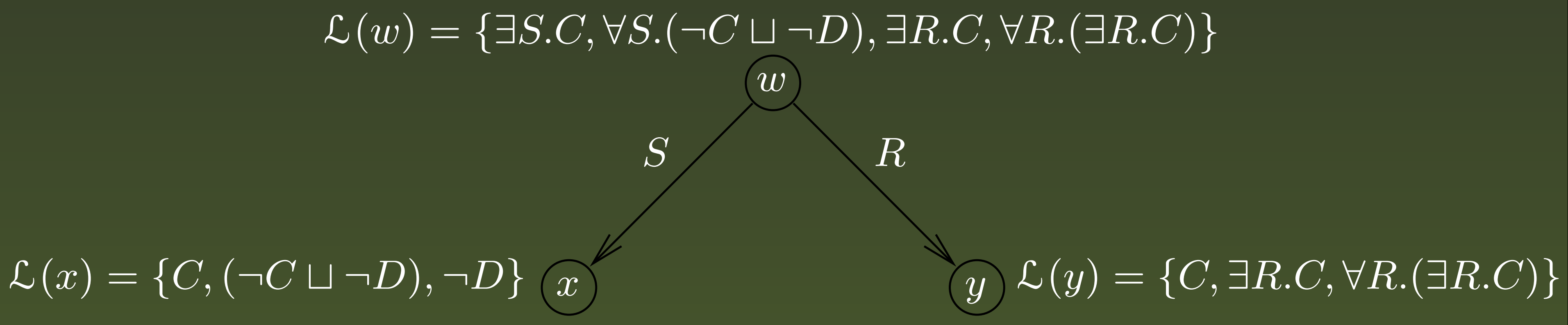
Tableaux Expansion

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



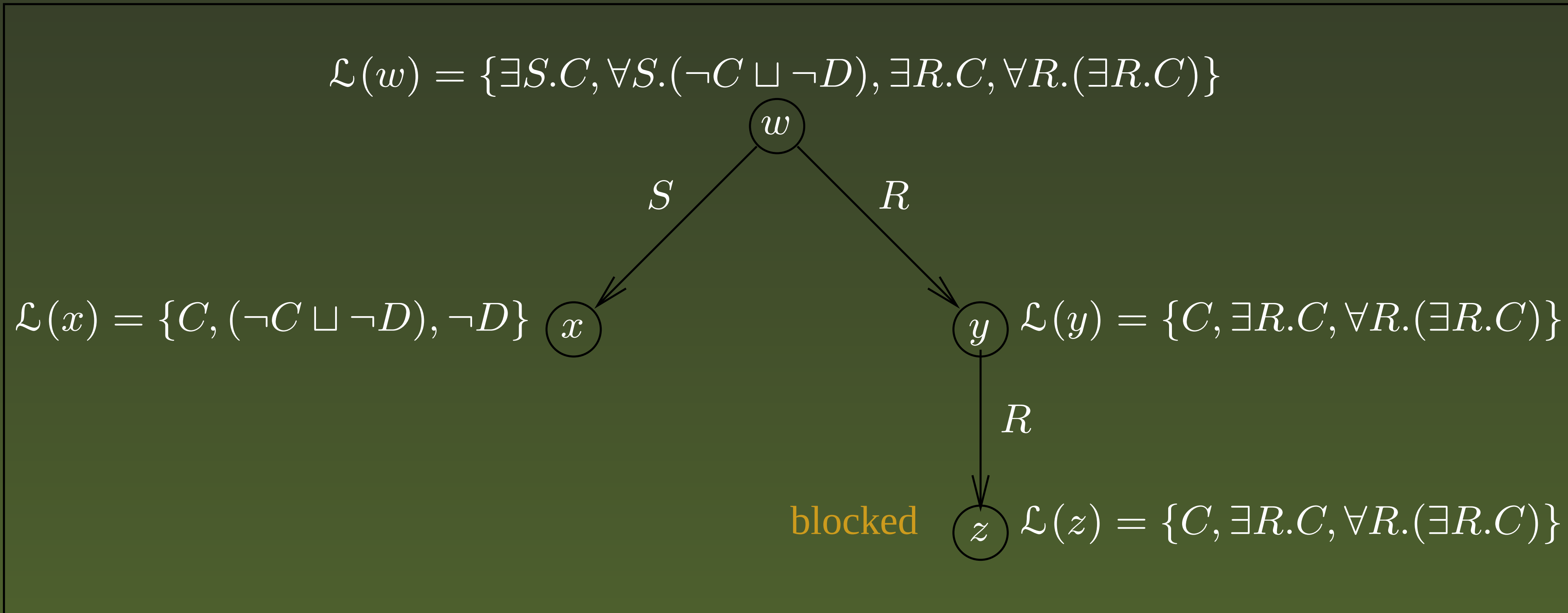
Tableaux Expansion

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



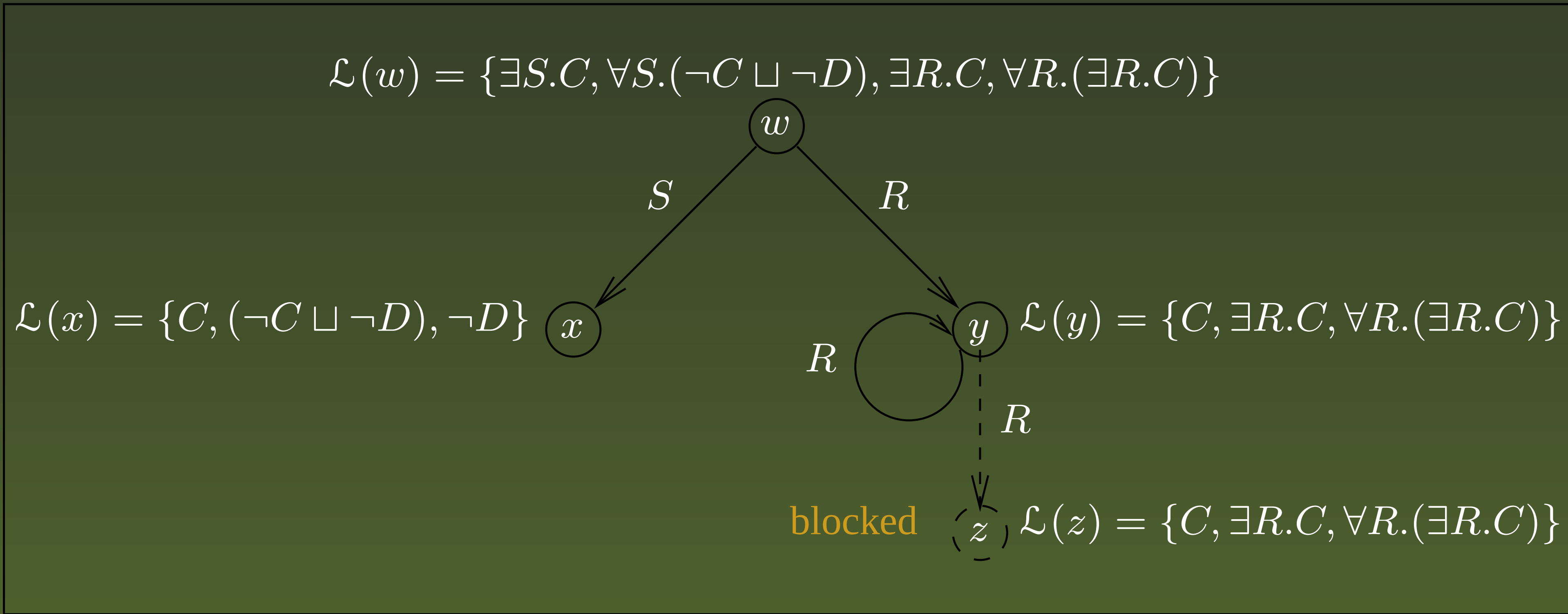
Tableaux Expansion

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 Expansion

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



More Advanced Techniques

More Advanced Techniques

Satisfiability w.r.t. a Terminology

- ☞ For each GCI $C \sqsubseteq D \in \mathcal{T}$, add $\neg C \sqcup D$ to **every** node label

More Advanced Techniques

Satisfiability w.r.t. a Terminology

- ☞ For each GCI $C \sqsubseteq D \in \mathcal{T}$, add $\neg C \sqcup D$ to **every** node label

More expressive DLs

More Advanced Techniques

Satisfiability w.r.t. a Terminology

☞ For each GCI $C \sqsubseteq D \in \mathcal{T}$, 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
- Aboxes
- etc.

More Advanced Techniques

Satisfiability w.r.t. a Terminology

- For each GCI $C \sqsubseteq D \in \mathcal{T}$, 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
  - Aboxes
  - etc.
- Extend **expansion rules** and use more sophisticated **blocking** strategy

More Advanced Techniques

Satisfiability w.r.t. a Terminology

- ☞ For each GCI $C \sqsubseteq D \in \mathcal{T}$, 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
  - Aboxes
  - etc.
- ☞ Extend **expansion rules** and use more sophisticated **blocking** strategy
- ☞ Forest instead of Tree (for Aboxes)

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
- Time usage

Naive Implementations

Problems include:

- ☞ Space usage
  - Storage required for tableaux datastructures
  - Rarely a serious problem in practice
- ☞ 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
- ☞ 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
- ☞ Time usage
  - Search required due to non-deterministic expansion
  - **Serious** problem in practice
  - Mitigated by:

Naive Implementations

Problems include:

- ☞ Space usage
  - Storage required for tableaux datastructures
  - Rarely a serious problem in practice
- ☞ Time usage
  - Search required due to non-deterministic expansion
  - **Serious** problem in practice
  - Mitigated by:
    - → Careful **choice of algorithm**

Naive Implementations

Problems include:

- ☞ Space usage
  - Storage required for tableaux datastructures
  - Rarely a serious problem in practice
- ☞ 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., $(\text{domain } R.C) \equiv \exists R.\top \sqsubseteq C$

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., $($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

Optimisation performed at 2 levels

Highly Optimised Implementation

Optimisation performed at 2 levels

- ☞ Computing **classification** (partial ordering) of concepts

Highly Optimised Implementation

Optimisation performed at 2 levels

- ☞ Computing **classification** (partial ordering) of concepts
  - Objective is to minimise number of subsumption tests

Highly Optimised Implementation

Optimisation performed at 2 levels

- ☞ Computing **classification** (partial ordering) of concepts
  - Objective is to minimise number of subsumption tests
  - Can use standard order-theoretic techniques

Highly Optimised Implementation

Optimisation performed at 2 levels

- Computing **classification** (partial ordering) of concepts
  - Objective is to minimise number of subsumption tests
  - Can use standard order-theoretic techniques
    - E.g., use **enhanced traversal** that exploits information from previous tests

Highly Optimised Implementation

Optimisation performed at 2 levels

- ☞ Computing **classification** (partial ordering) of concepts
  - Objective is to minimise number of subsumption tests
  - Can use standard order-theoretic techniques
    - ➜ E.g., use **enhanced traversal** that exploits information from previous tests
  - Also use structural information from KB

Highly Optimised Implementation

Optimisation performed at 2 levels

- ☞ Computing **classification** (partial ordering) of concepts
  - Objective is to minimise number of subsumption tests
  - Can use standard order-theoretic techniques
    - ➔ E.g., use **enhanced traversal** that exploits information from previous tests
  - Also use structural information from KB
    - ➔ E.g., to select order in which to classify concepts

Highly Optimised Implementation

Optimisation performed at 2 levels

- ☞ Computing **classification** (partial ordering) of concepts
  - Objective is to minimise number of subsumption tests
  - Can use standard order-theoretic techniques
    - ➔ E.g., use **enhanced traversal** that exploits information from previous tests
  - Also use structural information from KB
    - ➔ E.g., to select order in which to classify concepts
- ☞ Computing **subsumption** between concepts

Highly Optimised Implementation

Optimisation performed at 2 levels

- Computing **classification** (partial ordering) of concepts
  - Objective is to minimise number of subsumption tests
  - Can use standard order-theoretic techniques
    - E.g., use **enhanced traversal** that exploits information from previous tests
  - Also use structural information from KB
    - E.g., to select order in which to classify concepts
- Computing **subsumption** between concepts
  - Objective is to minimise cost of single subsumption tests

Highly Optimised Implementation

Optimisation performed at 2 levels

- Computing **classification** (partial ordering) of concepts
  - Objective is to minimise number of subsumption tests
  - Can use standard order-theoretic techniques
    - E.g., use **enhanced traversal** that exploits information from previous tests
  - Also use structural information from KB
    - E.g., to select order in which to classify concepts
- Computing **subsumption** between concepts
  - Objective is to minimise cost of single subsumption tests
  - Small number of hard tests can dominate classification time

Highly Optimised Implementation

Optimisation performed at 2 levels

- Computing **classification** (partial ordering) of concepts
  - Objective is to minimise number of subsumption tests
  - Can use standard order-theoretic techniques
    - E.g., use **enhanced traversal** that exploits information from previous tests
  - Also use structural information from KB
    - E.g., to select order in which to classify concepts
- Computing **subsumption** between concepts
  - Objective is to minimise cost of single subsumption tests
  - Small number of hard tests can dominate classification time
  - Recent DL research has addressed this problem (with considerable success)

Optimising Subsumption Testing

Optimisation techniques broadly fall into 2 categories

Optimising Subsumption Testing

Optimisation techniques broadly fall into 2 categories

- ☞ Pre-processing optimisations

Optimising Subsumption Testing

Optimisation techniques broadly fall into 2 categories

☞ Pre-processing optimisations

- Aim is to **simplify KB** and facilitate subsumption testing

Optimising Subsumption Testing

Optimisation techniques broadly fall into 2 categories

☞ Pre-processing optimisations

- Aim is to **simplify KB** and facilitate subsumption testing
- Largely algorithm independent

Optimising Subsumption Testing

Optimisation techniques broadly fall into 2 categories

- ☞ Pre-processing optimisations
  - Aim is to **simplify KB** and facilitate subsumption testing
  - Largely algorithm independent
  - Particularly important when KB contains GCI axioms

Optimising Subsumption Testing

Optimisation techniques broadly fall into 2 categories

- ☞ Pre-processing optimisations
  - Aim is to **simplify KB** and facilitate subsumption testing
  - Largely algorithm independent
  - Particularly important when KB contains GCI axioms
- ☞ Algorithmic optimisations

Optimising Subsumption Testing

Optimisation techniques broadly fall into 2 categories

- ☞ Pre-processing optimisations
  - Aim is to **simplify KB** and facilitate subsumption testing
  - Largely algorithm independent
  - Particularly important when KB contains GCI axioms
- ☞ Algorithmic optimisations
  - Main aim is to **reduce search space** due to non-determinism

Optimising Subsumption Testing

Optimisation techniques broadly fall into 2 categories

- ☞ Pre-processing optimisations
  - Aim is to **simplify KB** and facilitate subsumption testing
  - Largely algorithm independent
  - Particularly important when KB contains GCI axioms
- ☞ Algorithmic optimisations
  - Main aim is to **reduce search space** due to non-determinism
  - Integral part of implementation

Optimising Subsumption Testing

Optimisation techniques broadly fall into 2 categories

- Pre-processing optimisations
  - Aim is to **simplify KB** and facilitate subsumption testing
  - Largely algorithm independent
  - Particularly important when KB contains GCI axioms
- Algorithmic optimisations
  - Main aim is to **reduce search space** due to non-determinism
  - Integral part of implementation
  - But often generally applicable to search based algorithms

Pre-processing Optimisations

Useful techniques include

Pre-processing Optimisations

Useful techniques include

- ☞ Normalisation and simplification of concepts

Pre-processing Optimisations

Useful techniques include

- ☞ Normalisation and simplification of concepts
  - Refinement of technique first used in $\mathcal{KRIS}$ system

Pre-processing Optimisations

Useful techniques include

- ☞ Normalisation and simplification of concepts
  - Refinement of technique first used in $\mathcal{KRIS}$ system
  - Lexically normalise and simplify all concepts in KB

Pre-processing Optimisations

Useful techniques include

- Normalisation and simplification of concepts
  - Refinement of technique first used in $\mathcal{KRIS}$ system
  - Lexically normalise and simplify all concepts in KB
  - Combine with lazy unfolding in tableaux algorithm

Pre-processing Optimisations

Useful techniques include

- Normalisation and simplification of concepts
  - Refinement of technique first used in $\mathcal{KRIS}$ system
  - Lexically normalise and simplify all concepts in KB
  - Combine with lazy unfolding in tableaux algorithm
  - Facilitates early detection of inconsistencies (clashes)

Pre-processing Optimisations

Useful techniques include

☞ Normalisation and simplification of concepts

- Refinement of technique first used in $\mathcal{KRIS}$ system
- Lexically normalise and simplify all concepts in KB
- Combine with lazy unfolding in tableaux algorithm
- Facilitates early detection of inconsistencies (clashes)

☞ Absorption (simplification) of general axioms

Pre-processing Optimisations

Useful techniques include

- Normalisation and simplification of concepts
  - Refinement of technique first used in $\mathcal{KRIS}$ system
  - Lexically normalise and simplify all concepts in KB
  - Combine with lazy unfolding in tableaux algorithm
  - Facilitates early detection of inconsistencies (clashes)
- Absorption (simplification) of general axioms
  - Eliminate GCIs by absorbing into “definition” axioms

Pre-processing Optimisations

Useful techniques include

- ☞ Normalisation and simplification of concepts
  - Refinement of technique first used in $\mathcal{KRIS}$ system
  - Lexically normalise and simplify all concepts in KB
  - Combine with lazy unfolding in tableaux algorithm
  - Facilitates early detection of inconsistencies (clashes)
- ☞ Absorption (simplification) of general axioms
  - Eliminate GCIs by absorbing into "definition" axioms
  - Definition axioms efficiently dealt with by lazy expansion

Pre-processing Optimisations

Useful techniques include

- Normalisation and simplification of concepts
  - Refinement of technique first used in $\mathcal{KRIS}$ system
  - Lexically normalise and simplify all concepts in KB
  - Combine with lazy unfolding in tableaux algorithm
  - Facilitates early detection of inconsistencies (clashes)
- Absorption (simplification) of general axioms
  - Eliminate GCIs by absorbing into “definition” axioms
  - Definition axioms efficiently dealt with by lazy expansion
- Avoidance of potentially costly reasoning whenever possible

Pre-processing Optimisations

Useful techniques include

- Normalisation and simplification of concepts
  - Refinement of technique first used in $\mathcal{KRIS}$ system
  - Lexically normalise and simplify all concepts in KB
  - Combine with lazy unfolding in tableaux algorithm
  - Facilitates early detection of inconsistencies (clashes)
- Absorption (simplification) of general axioms
  - Eliminate GCIs by absorbing into "definition" axioms
  - Definition axioms efficiently dealt with by lazy expansion
- Avoidance of potentially costly reasoning whenever possible
  - Normalisation can discover "obvious" (un)satisfiability

Pre-processing Optimisations

Useful techniques include

- Normalisation and simplification of concepts
  - Refinement of technique first used in $\mathcal{KRIS}$ system
  - Lexically normalise and simplify all concepts in KB
  - Combine with lazy unfolding in tableaux algorithm
  - Facilitates early detection of inconsistencies (clashes)
- Absorption (simplification) of general axioms
  - Eliminate GCIs by absorbing into “definition” axioms
  - Definition axioms efficiently dealt with by lazy expansion
- Avoidance of potentially costly reasoning whenever possible
  - Normalisation can discover “obvious” (un)satisfiability
  - Structural analysis can discover “obvious” subsumption

Normalisation and Simplification

Normalisation and Simplification

☞ Normalise concepts to standard form, e.g.:

Normalisation and Simplification

- Normalise concepts to standard form, e.g.:
  - $\exists R.C \longrightarrow \neg\forall R.\neg C$

Normalisation and Simplification

- Normalise concepts to standard form, e.g.:
  - $\exists R.C \longrightarrow \neg\forall R.\neg C$
  - $C \sqcup D \longrightarrow \neg(\neg C \sqcap \neg D)$

Normalisation and Simplification

- Normalise concepts to standard form, e.g.:
  - $\exists R.C \longrightarrow \neg\forall R.\neg C$
  - $C \sqcup D \longrightarrow \neg(\neg C \sqcap \neg D)$
- Simplify concepts, e.g.:

Normalisation and Simplification

- Normalise concepts to standard form, e.g.:
  - $\exists R.C \longrightarrow \neg\forall R.\neg C$
  - $C \sqcup D \longrightarrow \neg(\neg C \sqcap \neg D)$
- Simplify concepts, e.g.:
  - $(D \sqcap C) \sqcap (A \sqcap D) \longrightarrow A \sqcap C \sqcap D$

Normalisation and Simplification

- Normalise concepts to standard form, e.g.:
  - $\exists R.C \longrightarrow \neg\forall R.\neg C$
  - $C \sqcup D \longrightarrow \neg(\neg C \sqcap \neg D)$
- Simplify concepts, e.g.:
  - $(D \sqcap C) \sqcap (A \sqcap D) \longrightarrow A \sqcap C \sqcap D$
  - $\forall R.\top \longrightarrow \top$

Normalisation and Simplification

- Normalise concepts to standard form, e.g.:
  - $\exists R.C \longrightarrow \neg\forall R.\neg C$
  - $C \sqcup D \longrightarrow \neg(\neg C \sqcap \neg D)$
- Simplify concepts, e.g.:
  - $(D \sqcap C) \sqcap (A \sqcap D) \longrightarrow A \sqcap C \sqcap D$
  - $\forall R.\top \longrightarrow \top$
  - $\ldots \sqcap C \sqcap \ldots \sqcap \neg C \sqcap \ldots \longrightarrow \bot$

Normalisation and Simplification

- Normalise concepts to standard form, e.g.:
  - $\exists R.C \longrightarrow \neg\forall R.\neg C$
  - $C \sqcup D \longrightarrow \neg(\neg C \sqcap \neg D)$
- Simplify concepts, e.g.:
  - $(D \sqcap C) \sqcap (A \sqcap D) \longrightarrow A \sqcap C \sqcap D$
  - $\forall R.\top \longrightarrow \top$
  - $\ldots \sqcap C \sqcap \ldots \sqcap \neg C \sqcap \ldots \longrightarrow \bot$
- Lazily unfold concepts in tableaux algorithm

Normalisation and Simplification

- Normalise concepts to standard form, e.g.:
  - $\exists R.C \longrightarrow \neg\forall R.\neg C$
  - $C \sqcup D \longrightarrow \neg(\neg C \sqcap \neg D)$
- Simplify concepts, e.g.:
  - $(D \sqcap C) \sqcap (A \sqcap D) \longrightarrow A \sqcap C \sqcap D$
  - $\forall R.\top \longrightarrow \top$
  - $\ldots \sqcap C \sqcap \ldots \sqcap \neg C \sqcap \ldots \longrightarrow \bot$
- Lazily unfold concepts in tableaux algorithm
  - Use names/pointers to refer to complex concepts

Normalisation and Simplification

- Normalise concepts to standard form, e.g.:
  - $\exists R.C \longrightarrow \neg\forall R.\neg C$
  - $C \sqcup D \longrightarrow \neg(\neg C \sqcap \neg D)$
- Simplify concepts, e.g.:
  - $(D \sqcap C) \sqcap (A \sqcap D) \longrightarrow A \sqcap C \sqcap D$
  - $\forall R.\top \longrightarrow \top$
  - $\ldots \sqcap C \sqcap \ldots \sqcap \neg C \sqcap \ldots \longrightarrow \bot$
- Lazily unfold concepts in tableaux algorithm
  - Use names/pointers to refer to complex concepts
  - Only add structure as required by progress of algorithm

Normalisation and Simplification

- Normalise concepts to standard form, e.g.:
  - $\exists R.C \longrightarrow \neg\forall R.\neg C$
  - $C \sqcup D \longrightarrow \neg(\neg C \sqcap \neg D)$
- Simplify concepts, e.g.:
  - $(D \sqcap C) \sqcap (A \sqcap D) \longrightarrow A \sqcap C \sqcap D$
  - $\forall R.\top \longrightarrow \top$
  - $\ldots \sqcap C \sqcap \ldots \sqcap \neg C \sqcap \ldots \longrightarrow \bot$
- Lazily unfold concepts in tableaux algorithm
  - Use names/pointers to refer to complex concepts
  - Only add structure as required by progress of algorithm
  - Detect clashes between lexically equivalent concepts

Normalisation and Simplification

- Normalise concepts to standard form, e.g.:
  - $\exists R.C \longrightarrow \neg\forall R.\neg C$
  - $C \sqcup D \longrightarrow \neg(\neg C \sqcap \neg D)$
- Simplify concepts, e.g.:
  - $(D \sqcap C) \sqcap (A \sqcap D) \longrightarrow A \sqcap C \sqcap D$
  - $\forall R.\top \longrightarrow \top$
  - $\ldots \sqcap C \sqcap \ldots \sqcap \neg C \sqcap \ldots \longrightarrow \bot$
- Lazily unfold concepts in tableaux algorithm
  - Use names/pointers to refer to complex concepts
  - Only add structure as required by progress of algorithm
  - Detect clashes between lexically equivalent concepts

E.g.:

$\{\text{HappyFather}, \neg\text{HappyFather}\} \longrightarrow$ **clash**

$\{\forall\text{has-child}.(\text{Doctor} \sqcup \text{Lawyer}), \exists\text{has-child}.(\neg\text{Doctor} \sqcap \neg\text{Lawyer})\} \longrightarrow$ **search**

Absorption I

Absorption I

- Reasoning w.r.t. set of GCI axioms can be very costly

Absorption I

☞ Reasoning w.r.t. set of GCI axioms can be very costly

- GCI $C \sqsubseteq D$ adds $D \sqcup \neg C$ to **every** node label

Absorption I

☞ Reasoning w.r.t. set of GCI axioms can be very costly

- GCI $C \sqsubseteq D$ adds $D \sqcup \neg C$ to **every** node label
- Expansion of disjunctions leads to search

Absorption I

☞ Reasoning w.r.t. set of GCI axioms can be very costly

- GCI $C \sqsubseteq D$ adds $D \sqcup \neg C$ to **every** node label
- Expansion of disjunctions leads to search
- With 10 axioms and 10 nodes search space already 2^{100}

Absorption I

- Reasoning w.r.t. set of GCI axioms can be very costly
  - GCI $C \sqsubseteq D$ adds $D \sqcup \neg C$ to **every** node label
  - Expansion of disjunctions leads to search
  - With 10 axioms and 10 nodes search space already 2^{100}
  - GALEN (medical terminology) KB contains **hundreds** of axioms

Absorption I

- ☞ Reasoning w.r.t. set of GCI axioms can be very costly
  - GCI $C \sqsubseteq D$ adds $D \sqcup \neg C$ to **every** node label
  - Expansion of disjunctions leads to search
  - With 10 axioms and 10 nodes search space already 2^{100}
  - GALEN (medical terminology) KB contains **hundreds** of axioms
- ☞ Reasoning w.r.t. "primitive definition" axioms is relatively efficient

Absorption I

- Reasoning w.r.t. set of GCI axioms can be very costly
  - GCI $C \sqsubseteq D$ adds $D \sqcup \neg C$ to **every** node label
  - Expansion of disjunctions leads to search
  - With 10 axioms and 10 nodes search space already 2^{100}
  - GALEN (medical terminology) KB contains **hundreds** of axioms
- Reasoning w.r.t. "primitive definition" axioms is relatively efficient
  - For CN $\sqsubseteq D$, add D **only** to node labels containing CN

Absorption I

- ☞ Reasoning w.r.t. set of GCI axioms can be very costly
  - GCI $C \sqsubseteq D$ adds $D \sqcup \neg C$ to **every** node label
  - Expansion of disjunctions leads to search
  - With 10 axioms and 10 nodes search space already 2^{100}
  - GALEN (medical terminology) KB contains **hundreds** of axioms
- ☞ Reasoning w.r.t. "primitive definition" axioms is relatively efficient
  - For CN $\sqsubseteq D$, add D **only** to node labels containing CN
  - For CN $\sqsupseteq D$, add $\neg D$ **only** to node labels containing $\neg$CN

Absorption I

- Reasoning w.r.t. set of GCI axioms can be very costly
  - GCI $C \sqsubseteq D$ adds $D \sqcup \neg C$ to **every** node label
  - Expansion of disjunctions leads to search
  - With 10 axioms and 10 nodes search space already 2^{100}
  - GALEN (medical terminology) KB contains **hundreds** of axioms
- Reasoning w.r.t. "primitive definition" axioms is relatively efficient
  - For CN $\sqsubseteq D$, add D **only** to node labels containing CN
  - For CN $\sqsupseteq D$, add $\neg D$ **only** to node labels containing $\neg$CN
  - Can expand definitions lazily

Absorption I

- Reasoning w.r.t. set of GCI axioms can be very costly
  - GCI $C \sqsubseteq D$ adds $D \sqcup \neg C$ to **every** node label
  - Expansion of disjunctions leads to search
  - With 10 axioms and 10 nodes search space already 2^{100}
  - GALEN (medical terminology) KB contains **hundreds** of axioms
- Reasoning w.r.t. "primitive definition" axioms is relatively efficient
  - For CN $\sqsubseteq D$, add D **only** to node labels containing CN
  - For CN $\sqsupseteq D$, add $\neg D$ **only** to node labels containing $\neg$CN
  - Can expand definitions lazily
    - → Only add definitions **after** other local (propositional) expansion

Absorption I

- Reasoning w.r.t. set of GCI axioms can be very costly
  - GCI $C \sqsubseteq D$ adds $D \sqcup \neg C$ to **every** node label
  - Expansion of disjunctions leads to search
  - With 10 axioms and 10 nodes search space already 2^{100}
  - GALEN (medical terminology) KB contains **hundreds** of axioms
- Reasoning w.r.t. "primitive definition" axioms is relatively efficient
  - For CN $\sqsubseteq D$, add D **only** to node labels containing CN
  - For CN $\sqsupseteq D$, add $\neg D$ **only** to node labels containing $\neg$CN
  - Can expand definitions lazily
    - → Only add definitions **after** other local (propositional) expansion
    - → Only add definitions one step at a time

Absorption II

Absorption II

☞ Transform GCIs into primitive definitions, e.g.

Absorption II

- Transform GCIs into primitive definitions, e.g.
  - $\mathsf{CN} \sqcap C \sqsubseteq D \longrightarrow \mathsf{CN} \sqsubseteq D \sqcup \neg C$

Absorption II

☞ Transform GCIs into primitive definitions, e.g.

- $\mathsf{CN} \sqcap C \sqsubseteq D \longrightarrow \mathsf{CN} \sqsubseteq D \sqcup \neg C$
- $\mathsf{CN} \sqcup C \sqsupseteq D \longrightarrow \mathsf{CN} \sqsupseteq D \sqcap \neg C$

Absorption II

- Transform GCIs into primitive definitions, e.g.
  - $\mathsf{CN} \sqcap C \sqsubseteq D \longrightarrow \mathsf{CN} \sqsubseteq D \sqcup \neg C$
  - $\mathsf{CN} \sqcup C \sqsupseteq D \longrightarrow \mathsf{CN} \sqsupseteq D \sqcap \neg C$
- Absorb into existing primitive definitions, e.g.

Absorption II

- Transform GCIs into primitive definitions, e.g.
  - $\mathsf{CN} \sqcap C \sqsubseteq D \longrightarrow \mathsf{CN} \sqsubseteq D \sqcup \neg C$
  - $\mathsf{CN} \sqcup C \sqsupseteq D \longrightarrow \mathsf{CN} \sqsupseteq D \sqcap \neg C$
- Absorb into existing primitive definitions, e.g.
  - $\mathsf{CN} \sqsubseteq A$, $\mathsf{CN} \sqsubseteq D \sqcup \neg C \longrightarrow \mathsf{CN} \sqsubseteq A \sqcap (D \sqcup \neg C)$

Absorption II

- Transform GCIs into primitive definitions, e.g.
  - CN $\sqcap C \sqsubseteq D \longrightarrow$ CN $\sqsubseteq D \sqcup \neg C$
  - CN $\sqcup C \sqsupseteq D \longrightarrow$ CN $\sqsupseteq D \sqcap \neg C$
- Absorb into existing primitive definitions, e.g.
  - CN $\sqsubseteq A$, CN $\sqsubseteq D \sqcup \neg C \longrightarrow$ CN $\sqsubseteq A \sqcap (D \sqcup \neg C)$
  - CN $\sqsupseteq A$, CN $\sqsupseteq D \sqcap \neg C \longrightarrow$ CN $\sqsupseteq A \sqcup (D \sqcap \neg C)$

Absorption II

- Transform GCIs into primitive definitions, e.g.
  - CN $\sqcap C \sqsubseteq D \longrightarrow$ CN $\sqsubseteq D \sqcup \neg C$
  - CN $\sqcup C \sqsupseteq D \longrightarrow$ CN $\sqsupseteq D \sqcap \neg C$
- Absorb into existing primitive definitions, e.g.
  - CN $\sqsubseteq A$, CN $\sqsubseteq D \sqcup \neg C \longrightarrow$ CN $\sqsubseteq A \sqcap (D \sqcup \neg C)$
  - CN $\sqsupseteq A$, CN $\sqsupseteq D \sqcap \neg C \longrightarrow$ CN $\sqsupseteq A \sqcup (D \sqcap \neg C)$
- Use lazy expansion technique with primitive definitions

Absorption II

- Transform GCIs into primitive definitions, e.g.
  - CN $\sqcap C \sqsubseteq D \longrightarrow$ CN $\sqsubseteq D \sqcup \neg C$
  - CN $\sqcup C \sqsupseteq D \longrightarrow$ CN $\sqsupseteq D \sqcap \neg C$
- Absorb into existing primitive definitions, e.g.
  - CN $\sqsubseteq A$, CN $\sqsubseteq D \sqcup \neg C \longrightarrow$ CN $\sqsubseteq A \sqcap (D \sqcup \neg C)$
  - CN $\sqsupseteq A$, CN $\sqsupseteq D \sqcap \neg C \longrightarrow$ CN $\sqsupseteq A \sqcup (D \sqcap \neg C)$
- Use lazy expansion technique with primitive definitions
  - Disjunctions only added to “relevant” node labels

Absorption II

- Transform GCIs into primitive definitions, e.g.
  - $\mathsf{CN} \sqcap C \sqsubseteq D \longrightarrow \mathsf{CN} \sqsubseteq D \sqcup \neg C$
  - $\mathsf{CN} \sqcup C \sqsupseteq D \longrightarrow \mathsf{CN} \sqsupseteq D \sqcap \neg C$
- Absorb into existing primitive definitions, e.g.
  - $\mathsf{CN} \sqsubseteq A$, $\mathsf{CN} \sqsubseteq D \sqcup \neg C \longrightarrow \mathsf{CN} \sqsubseteq A \sqcap (D \sqcup \neg C)$
  - $\mathsf{CN} \sqsupseteq A$, $\mathsf{CN} \sqsupseteq D \sqcap \neg C \longrightarrow \mathsf{CN} \sqsupseteq A \sqcup (D \sqcap \neg C)$
- Use lazy expansion technique with primitive definitions
  - Disjunctions only added to “relevant” node labels
- Performance improvements often too large to measure

Absorption II

- Transform GCIs into primitive definitions, e.g.
  - CN $\sqcap C \sqsubseteq D \longrightarrow$ CN $\sqsubseteq D \sqcup \neg C$
  - CN $\sqcup C \sqsupseteq D \longrightarrow$ CN $\sqsupseteq D \sqcap \neg C$
- Absorb into existing primitive definitions, e.g.
  - CN $\sqsubseteq A$, CN $\sqsubseteq D \sqcup \neg C \longrightarrow$ CN $\sqsubseteq A \sqcap (D \sqcup \neg C)$
  - CN $\sqsupseteq A$, CN $\sqsupseteq D \sqcap \neg C \longrightarrow$ CN $\sqsupseteq A \sqcup (D \sqcap \neg C)$
- Use lazy expansion technique with primitive definitions
  - Disjunctions only added to “relevant” node labels
- Performance improvements often too large to measure
  - At least **four orders of magnitude** with GALEN KB

Algorithmic Optimisations

Useful techniques include

Algorithmic Optimisations

Useful techniques include

- ☞ Avoiding redundancy in search branches

Algorithmic Optimisations

Useful techniques include

- ☞ Avoiding redundancy in search branches
  - Davis-Putnam style semantic branching search

Algorithmic Optimisations

Useful techniques include

☞ Avoiding redundancy in search branches

- Davis-Putnam style semantic branching search
- Syntactic branching with no-good list

Algorithmic Optimisations

Useful techniques include

- ☞ Avoiding redundancy in search branches
  - Davis-Putnam style semantic branching search
  - Syntactic branching with no-good list
- ☞ Dependency directed backtracking

Algorithmic Optimisations

Useful techniques include

- ☞ Avoiding redundancy in search branches
  - Davis-Putnam style semantic branching search
  - Syntactic branching with no-good list
- ☞ Dependency directed backtracking
  - Backjumping

Algorithmic Optimisations

Useful techniques include

- ☞ Avoiding redundancy in search branches
  - Davis-Putnam style semantic branching search
  - Syntactic branching with no-good list
- ☞ Dependency directed backtracking
  - Backjumping
  - Dynamic backtracking

Algorithmic Optimisations

Useful techniques include

- ☞ Avoiding redundancy in search branches
  - Davis-Putnam style semantic branching search
  - Syntactic branching with no-good list
- ☞ Dependency directed backtracking
  - Backjumping
  - Dynamic backtracking
- ☞ Caching

Algorithmic Optimisations

Useful techniques include

- ☞ Avoiding redundancy in search branches
  - Davis-Putnam style semantic branching search
  - Syntactic branching with no-good list
- ☞ Dependency directed backtracking
  - Backjumping
  - Dynamic backtracking
- ☞ Caching
  - Cache partial models

Algorithmic Optimisations

Useful techniques include

- ☞ Avoiding redundancy in search branches
  - Davis-Putnam style semantic branching search
  - Syntactic branching with no-good list
- ☞ Dependency directed backtracking
  - Backjumping
  - Dynamic backtracking
- ☞ Caching
  - Cache partial models
  - Cache satisfiability status (of labels)

Algorithmic Optimisations

Useful techniques include

- ☞ Avoiding redundancy in search branches
  - Davis-Putnam style semantic branching search
  - Syntactic branching with no-good list
- ☞ Dependency directed backtracking
  - Backjumping
  - Dynamic backtracking
- ☞ Caching
  - Cache partial models
  - Cache satisfiability status (of labels)
- ☞ Heuristic ordering of propositional and modal expansion

Algorithmic Optimisations

Useful techniques include

- ☞ Avoiding redundancy in search branches
  - Davis-Putnam style semantic branching search
  - Syntactic branching with no-good list
- ☞ Dependency directed backtracking
  - Backjumping
  - Dynamic backtracking
- ☞ Caching
  - Cache partial models
  - Cache satisfiability status (of labels)
- ☞ Heuristic ordering of propositional and modal expansion
  - Min/maximise constrainedness (e.g., MOMS)

Algorithmic Optimisations

Useful techniques include

- Avoiding redundancy in search branches
  - Davis-Putnam style semantic branching search
  - Syntactic branching with no-good list
- Dependency directed backtracking
  - Backjumping
  - Dynamic backtracking
- Caching
  - Cache partial models
  - Cache satisfiability status (of labels)
- Heuristic ordering of propositional and modal expansion
  - Min/maximise constrainedness (e.g., MOMS)
  - Maximise backtracking (e.g., oldest first)

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

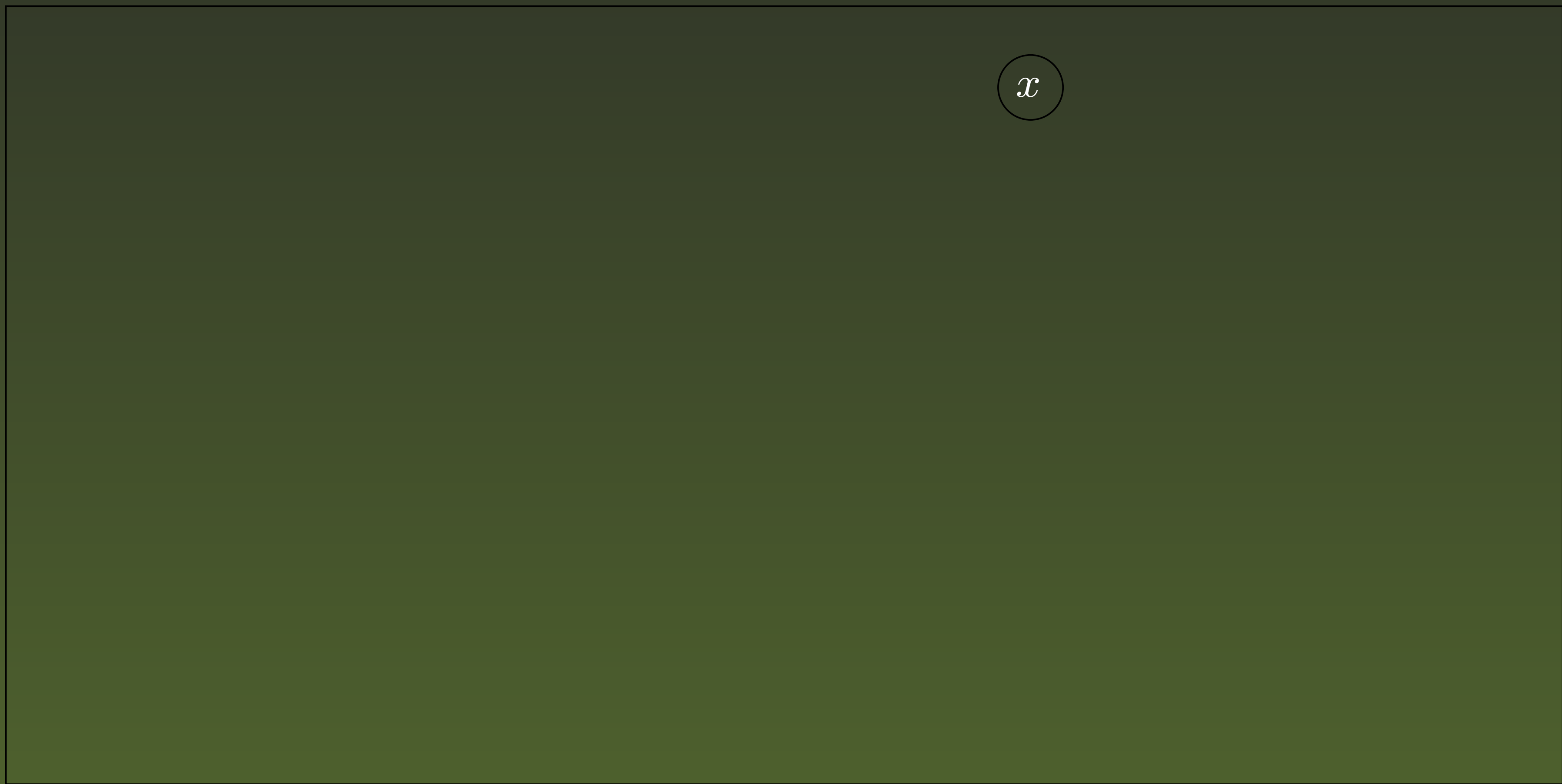
- Allows rapid recovery from bad branching choices
- Most commonly used technique is **backjumping**
  - Tag concepts introduced at branch points (e.g., when expanding disjunctions)
  - Expansion rules combine and propagate tags
  - On discovering a clash, identify most recently introduced concepts involved
  - Jump back to relevant branch points **without exploring** alternative branches
  - Effect is to prune away part of the search space
  - Performance improvements with GALEN KB again **too large to measure**

Backjumping

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

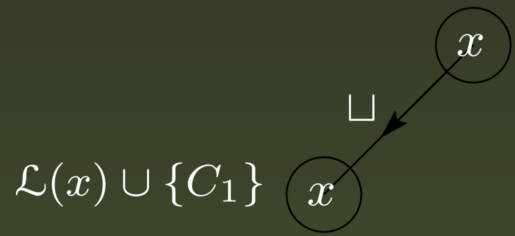
Backjumping

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



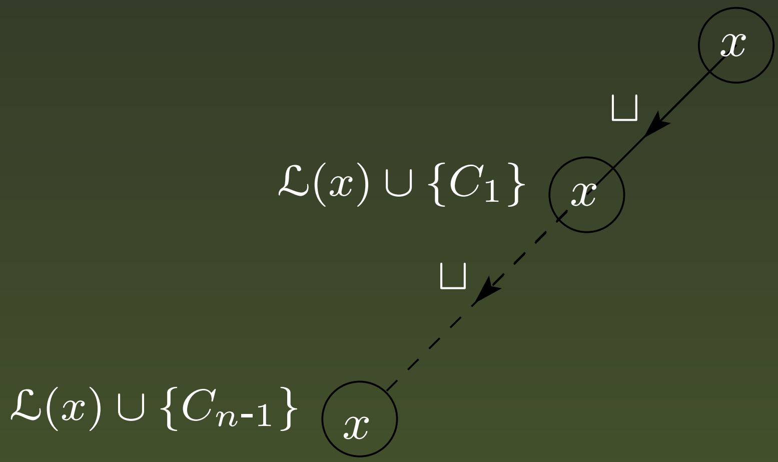
Backjumping

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



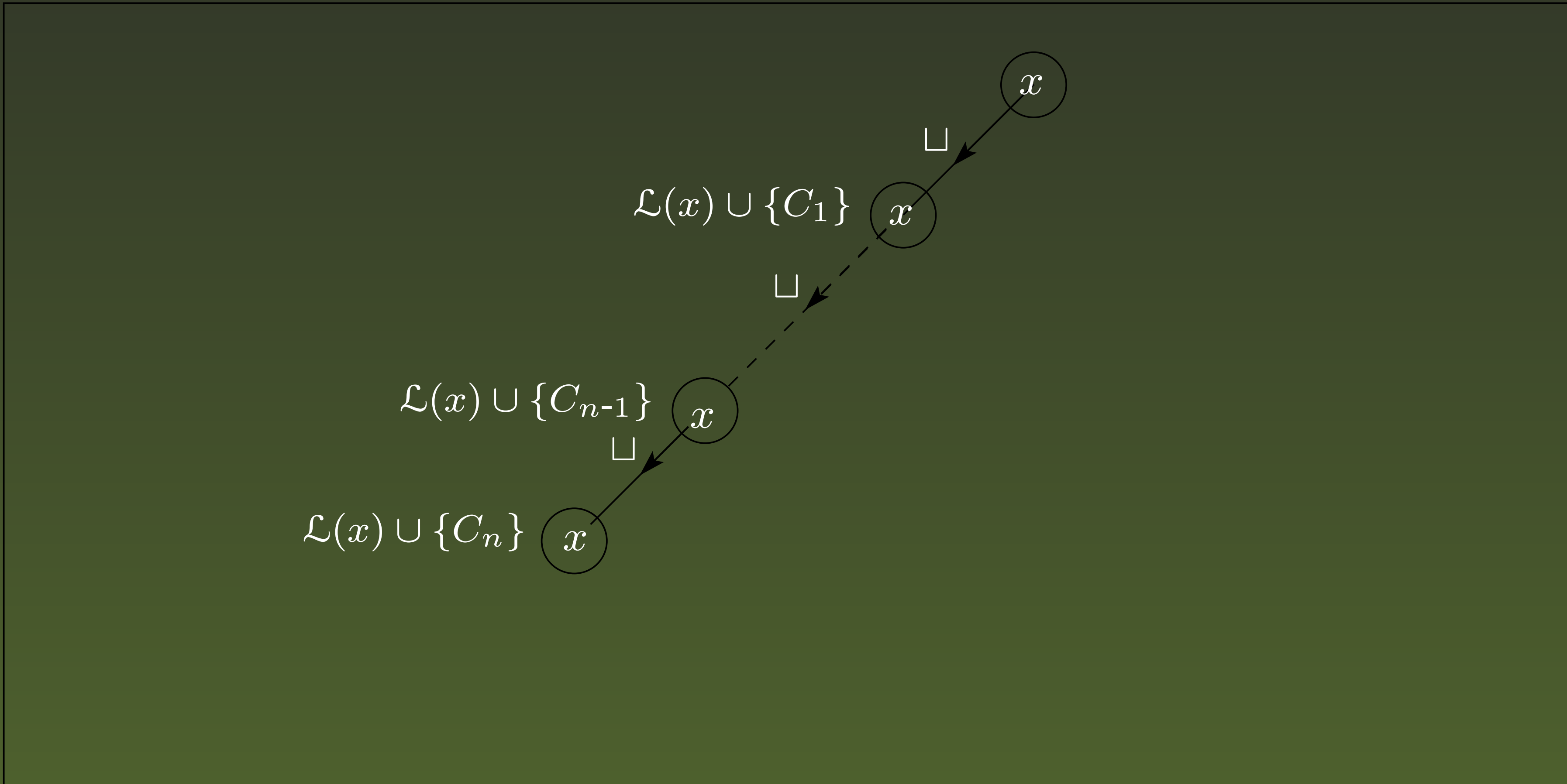
Backjumping

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



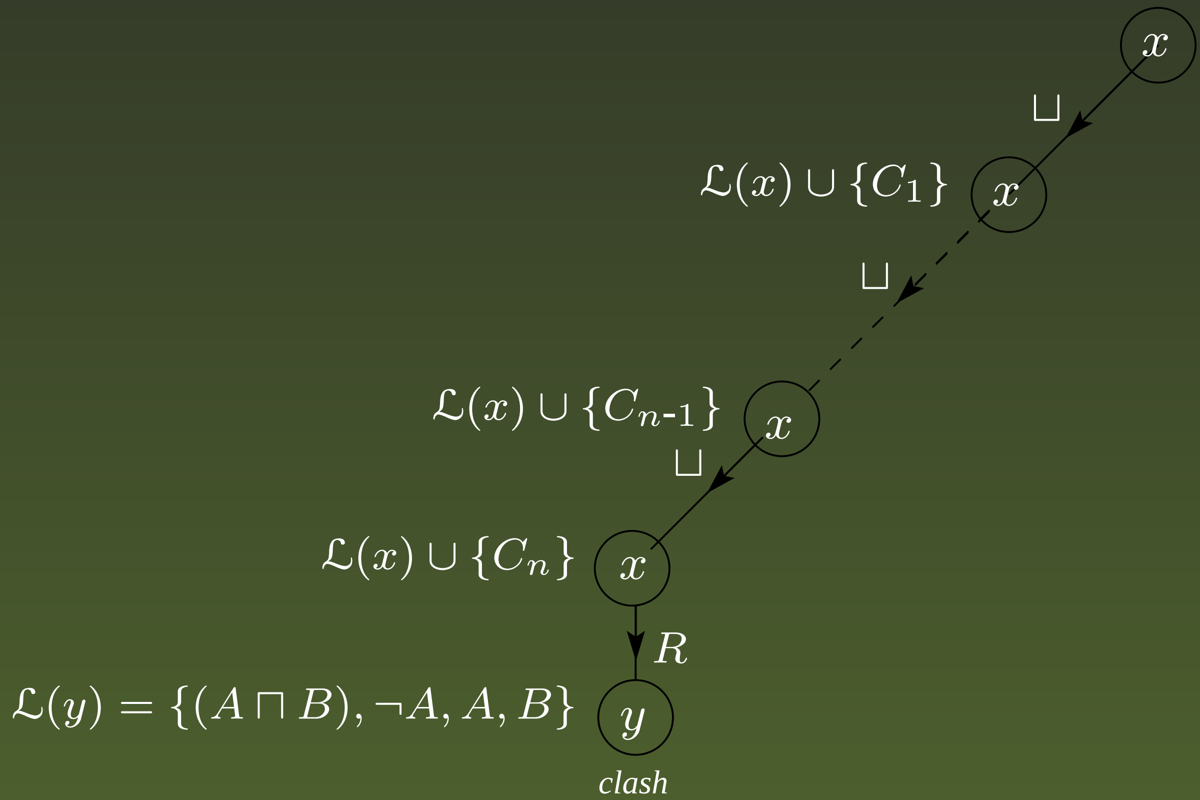
Backjumping

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



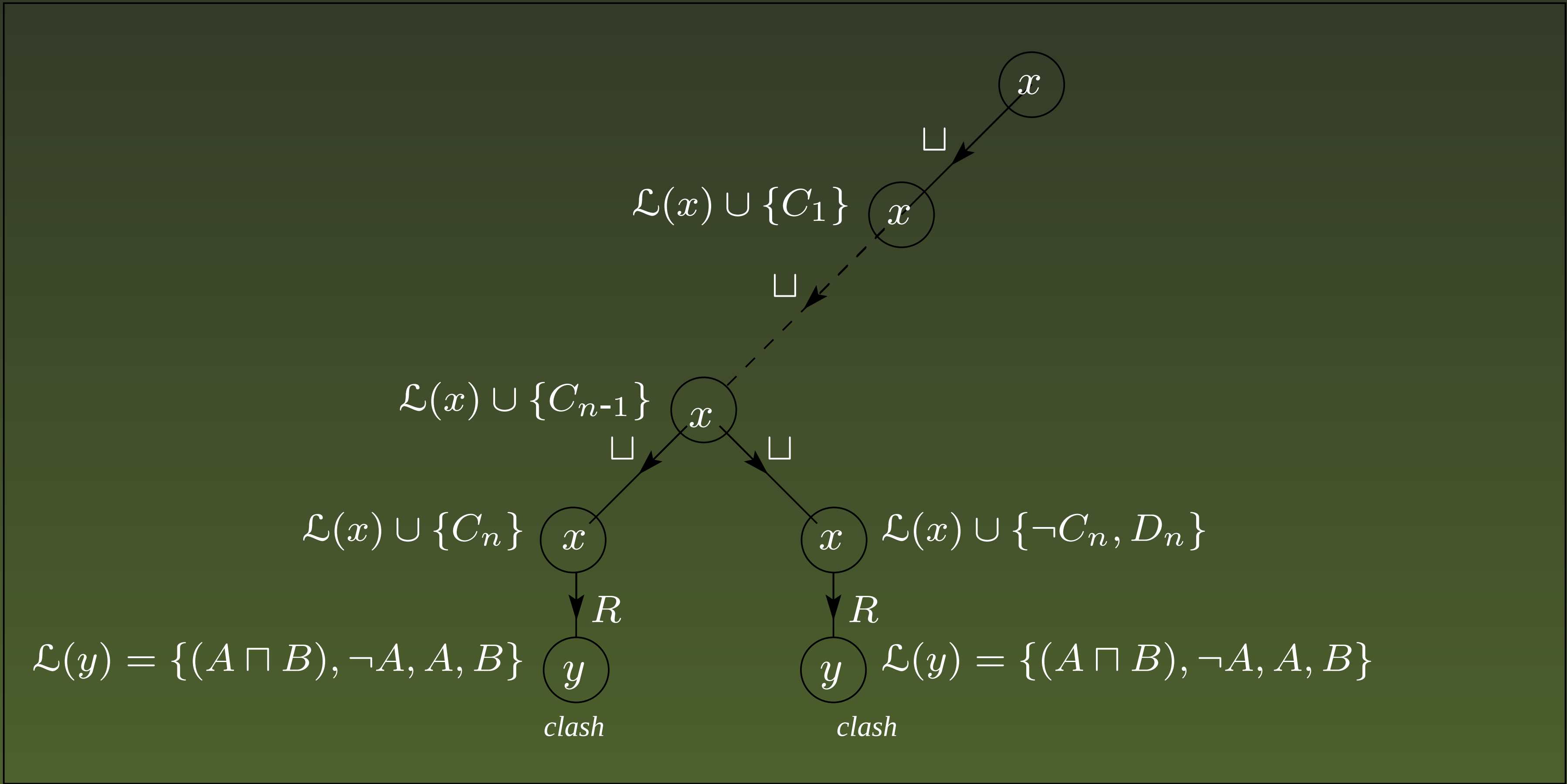
Backjumping

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



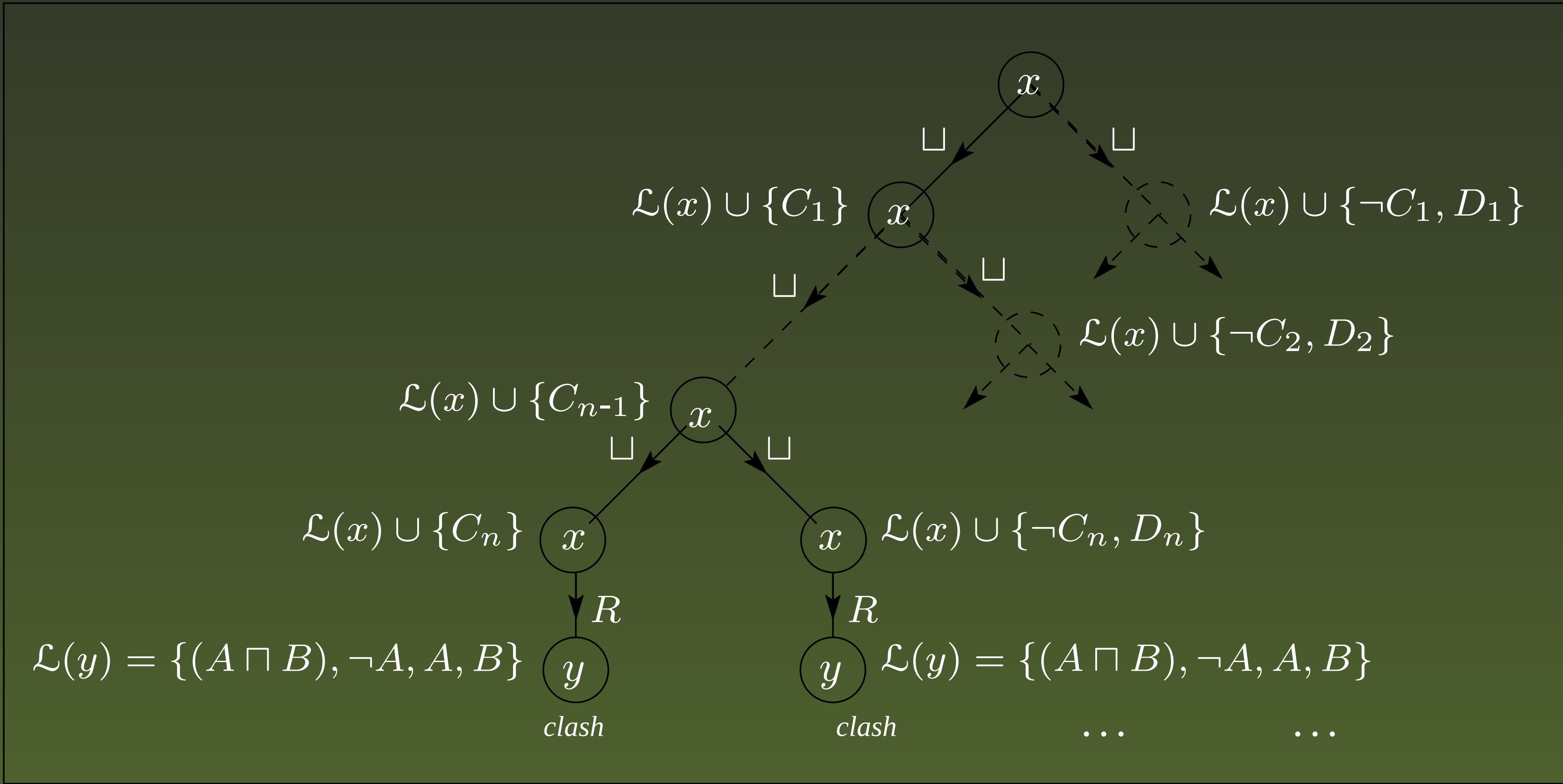
Backjumping

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



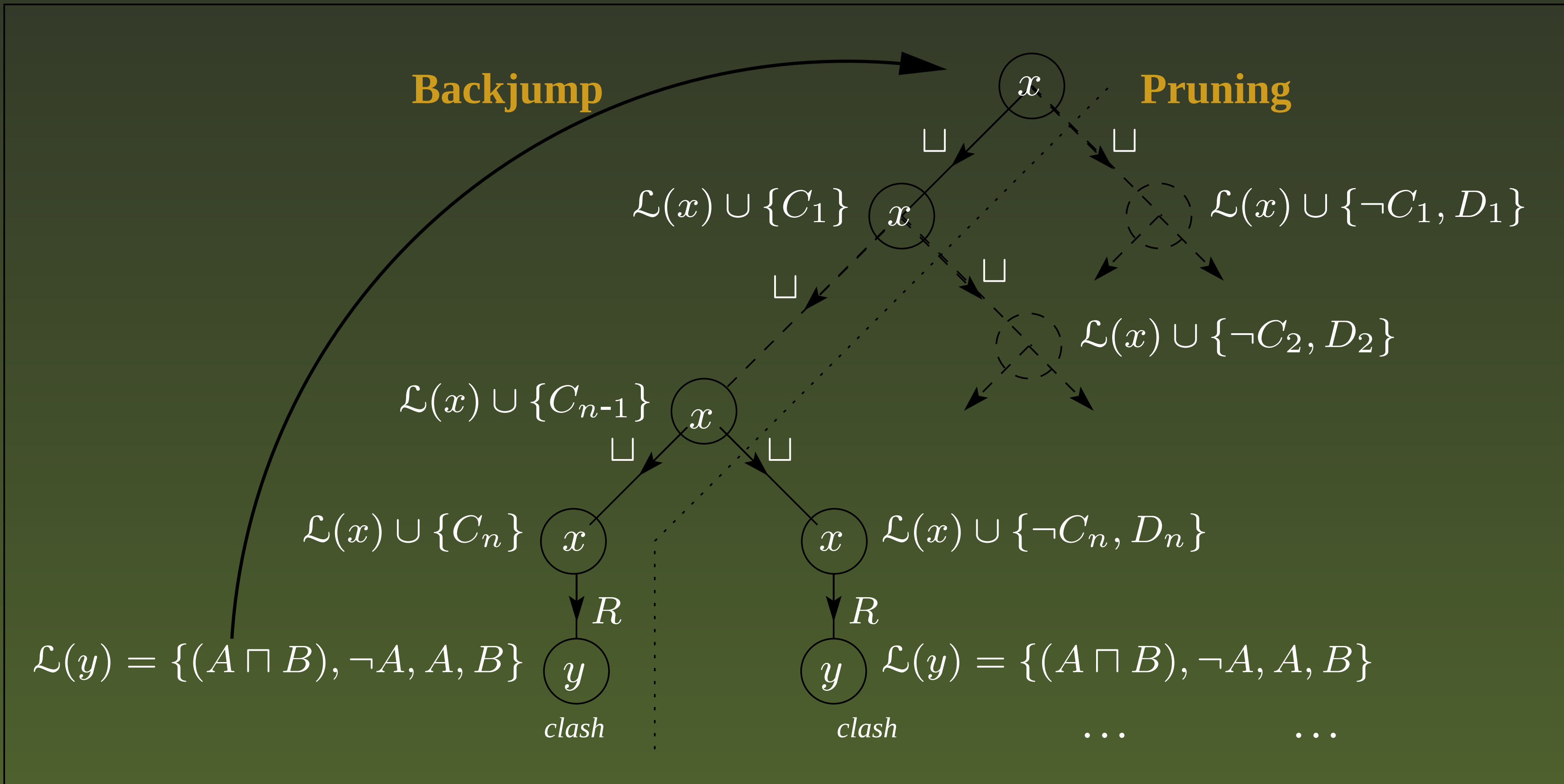
Backjumping

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 $\exists R.\neg A \sqcap \forall R.(A \sqcap B) \sqcap (C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n) \subseteq \mathfrak{L}(x)$



Caching

Caching

- Cache the satisfiability status of a node label

Caching

- ☞ Cache the satisfiability status of a node label
    - Identical node labels often recur during expansion

Caching

- ☞ Cache the satisfiability status of a node label
  - Identical node labels often recur during expansion
  - Avoid re-solving problems by caching satisfiability status

Caching

- Cache the satisfiability status of a node label
  - Identical node labels often recur during expansion
  - Avoid re-solving problems by caching satisfiability status
    - When $\mathcal{L}(x)$ initialised, look in cache

Caching

- Cache the satisfiability status of a node label
  - Identical node labels often recur during expansion
  - Avoid re-solving problems by caching satisfiability status
    - When $\mathcal{L}(x)$ initialised, look in cache
    - Use result, or add status once it has been computed

Caching

- ☞ Cache the satisfiability status of a node label
  - Identical node labels often recur during expansion
  - Avoid re-solving problems by caching satisfiability status
    - ➜ When $\mathcal{L}(x)$ initialised, look in cache
    - ➜ Use result, or add status once it has been computed
  - Can use sub/super set caching to deal with similar labels

Caching

- ☞ Cache the satisfiability status of a node label
  - Identical node labels often recur during expansion
  - Avoid re-solving problems by caching satisfiability status
    - ➜ When $\mathcal{L}(x)$ initialised, look in cache
    - ➜ Use result, or add status once it has been computed
  - Can use sub/super set caching to deal with similar labels
  - Care required when used with blocking or inverse roles

Caching

- Cache the satisfiability status of a node label
  - Identical node labels often recur during expansion
  - Avoid re-solving problems by caching satisfiability status
    - When $\mathcal{L}(x)$ initialised, look in cache
    - Use result, or add status once it has been computed
  - Can use sub/super set caching to deal with similar labels
  - Care required when used with blocking or inverse roles
  - Significant performance gains with some kinds of problem

Caching

- Cache the satisfiability status of a node label
  - Identical node labels often recur during expansion
  - Avoid re-solving problems by caching satisfiability status
    - When $\mathcal{L}(x)$ initialised, look in cache
    - Use result, or add status once it has been computed
  - Can use sub/super set caching to deal with similar labels
  - Care required when used with blocking or inverse roles
  - Significant performance gains with some kinds of problem
- Cache (partial) models of concepts

Caching

- ☞ Cache the satisfiability status of a node label
  - Identical node labels often recur during expansion
  - Avoid re-solving problems by caching satisfiability status
    - ➔ When $\mathcal{L}(x)$ initialised, look in cache
    - ➔ Use result, or add status once it has been computed
  - Can use sub/super set caching to deal with similar labels
  - Care required when used with blocking or inverse roles
  - Significant performance gains with some kinds of problem
- ☞ Cache (partial) models of concepts
  - Use to detect "obvious" non-subsumption

Caching

- Cache the satisfiability status of a node label
  - Identical node labels often recur during expansion
  - Avoid re-solving problems by caching satisfiability status
    - When $\mathcal{L}(x)$ initialised, look in cache
    - Use result, or add status once it has been computed
  - Can use sub/super set caching to deal with similar labels
  - Care required when used with blocking or inverse roles
  - Significant performance gains with some kinds of problem
- Cache (partial) models of concepts
  - Use to detect "obvious" non-subsumption
  - $C \not\sqsubseteq D$ if $C \sqcap \neg D$ is satisfiable

Caching

- Cache the satisfiability status of a node label
  - Identical node labels often recur during expansion
  - Avoid re-solving problems by caching satisfiability status
    - When $\mathcal{L}(x)$ initialised, look in cache
    - Use result, or add status once it has been computed
  - Can use sub/super set caching to deal with similar labels
  - Care required when used with blocking or inverse roles
  - Significant performance gains with some kinds of problem
- Cache (partial) models of concepts
  - Use to detect "obvious" non-subsumption
  - $C \not\sqsubseteq D$ if $C \sqcap \neg D$ is satisfiable
  - $C \sqcap \neg D$ satisfiable if models of C and $\neg D$ can be merged

Caching

- Cache the satisfiability status of a node label
  - Identical node labels often recur during expansion
  - Avoid re-solving problems by caching satisfiability status
    - When $\mathcal{L}(x)$ initialised, look in cache
    - Use result, or add status once it has been computed
  - Can use sub/super set caching to deal with similar labels
  - Care required when used with blocking or inverse roles
  - Significant performance gains with some kinds of problem
- Cache (partial) models of concepts
  - Use to detect "obvious" non-subsumption
  - $C \not\sqsubseteq D$ if $C \sqcap \neg D$ is satisfiable
  - $C \sqcap \neg D$ satisfiable if models of C and $\neg D$ can be merged
  - If not, continue with standard subsumption test

Caching

- Cache the satisfiability status of a node label
  - Identical node labels often recur during expansion
  - Avoid re-solving problems by caching satisfiability status
    - When $\mathcal{L}(x)$ initialised, look in cache
    - Use result, or add status once it has been computed
  - Can use sub/super set caching to deal with similar labels
  - Care required when used with blocking or inverse roles
  - Significant performance gains with some kinds of problem
- Cache (partial) models of concepts
  - Use to detect "obvious" non-subsumption
  - $C \not\sqsubseteq D$ if $C \sqcap \neg D$ is satisfiable
  - $C \sqcap \neg D$ satisfiable if models of C and $\neg D$ can be merged
  - If not, continue with standard subsumption test
  - Can use same technique in sub-problems

DL applications

Terminological KR and Ontologies

Terminological KR and Ontologies

Initial motivation for work on **FaCT** system was **Galen** project

Terminological KR and Ontologies

Initial motivation for work on **FaCT** system was **Galen** project

☞ General requirement for medical terminologies

Terminological KR and Ontologies

Initial motivation for work on **FaCT** system was **Galen** project

- ☞ General requirement for medical terminologies
- ☞ Static lists/taxonomies difficult to build and maintain
  - Need to be very large and highly interconnected
  - Inevitably contain many errors and omissions

Terminological KR and Ontologies

Initial motivation for work on **FaCT** system was **Galen** project

- ☞ General requirement for medical terminologies
- ☞ Static lists/taxonomies difficult to build and maintain
  - Need to be very large and highly interconnected
  - Inevitably contain many errors and omissions
- ☞ Galen project aims to replace static hierarchy with DL
  - Describe concepts (e.g., spiral fracture of left femur)
  - Use DL classifier to build taxonomy

Terminological KR and Ontologies

Initial motivation for work on **FaCT** system was **Galen** project

- ☞ General requirement for medical terminologies
- ☞ Static lists/taxonomies difficult to build and maintain
  - Need to be very large and highly interconnected
  - Inevitably contain many errors and omissions
- ☞ Galen project aims to replace static hierarchy with DL
  - Describe concepts (e.g., spiral fracture of left femur)
  - Use DL classifier to build taxonomy
- ☞ Needed expressive DL **and** efficient reasoning
  - Descriptions use transitive roles, inverses, GCIs etc.
  - Even prototype KB is very large ($\approx$3,000 concepts)
  - Existing (incomplete) classifier took $\approx$24 hours to classify KB
  - FaCT system (sound and complete) takes $\approx$60s

The Semantic Web

The Semantic Web

US **DAML** programme aims to develop so-called **Semantic Web**

The Semantic Web

US **DAML** programme aims to develop so-called **Semantic Web**

☞ Most existing Web resources only human understandable

The Semantic Web

US **DAML** programme aims to develop so-called **Semantic Web**

- ☞ Most existing Web resources only human understandable
  - Markup (HTML) provides rendering information

The Semantic Web

US **DAML** programme aims to develop so-called **Semantic Web**

☞ Most existing Web resources only human understandable

- Markup (HTML) provides rendering information
- Textual/graphical information for human consumption

The Semantic Web

US **DAML** programme aims to develop so-called **Semantic Web**

- ☞ Most existing Web resources only human understandable
  - Markup (HTML) provides rendering information
  - Textual/graphical information for human consumption
- ☞ Semantic Web aims at machine understandability

The Semantic Web

US **DAML** programme aims to develop so-called **Semantic Web**

- ☞ Most existing Web resources only human understandable
  - Markup (HTML) provides rendering information
  - Textual/graphical information for human consumption
- ☞ Semantic Web aims at machine understandability
  - **Semantic** markup will be added to web resources

The Semantic Web

US **DAML** programme aims to develop so-called **Semantic Web**

- ☞ Most existing Web resources only human understandable
  - Markup (HTML) provides rendering information
  - Textual/graphical information for human consumption
- ☞ Semantic Web aims at machine understandability
  - **Semantic** markup will be added to web resources
  - Markup will use **Ontologies** for shared understanding

The Semantic Web

US **DAML** programme aims to develop so-called **Semantic Web**

- Most existing Web resources only human understandable
  - Markup (HTML) provides rendering information
  - Textual/graphical information for human consumption
- Semantic Web aims at machine understandability
  - **Semantic** markup will be added to web resources
  - Markup will use **Ontologies** for shared understanding
  - Requirement for DAML ontology language

The Semantic Web

US **DAML** programme aims to develop so-called **Semantic Web**

- ☞ Most existing Web resources only human understandable
  - Markup (HTML) provides rendering information
  - Textual/graphical information for human consumption
- ☞ Semantic Web aims at machine understandability
  - **Semantic** markup will be added to web resources
  - Markup will use **Ontologies** for shared understanding
  - Requirement for DAML ontology language
  - Should extend existing Web standards (XML, RDF, RDFS)

OIL and DAML+OIL

OIL and DAML+OIL

OIL language already developed to meet similar requirements

OIL and DAML+OIL

OIL language already developed to meet similar requirements

- ☞ Intuitive (frame) syntax plus high expressive power

OIL and DAML+OIL

OIL language already developed to meet similar requirements

- ☞ Intuitive (frame) syntax plus high expressive power
- ☞ Well defined semantics via mapping to $\mathcal{SHIQ}$ DL

OIL and DAML+OIL

OIL language already developed to meet similar requirements

- ☞ Intuitive (frame) syntax plus high expressive power
- ☞ Well defined semantics via mapping to $\mathcal{SHIQ}$ DL
- ☞ Can use FaCT system to reason with OIL ontologies

OIL and DAML+OIL

OIL language already developed to meet similar requirements

- ☞ Intuitive (frame) syntax plus high expressive power
- ☞ Well defined semantics via mapping to $\mathcal{SHIQ}$ DL
- ☞ Can use FaCT system to reason with OIL ontologies
- ☞ Extends existing Web standards (XML, RDF, RDFS)

OIL and DAML+OIL

OIL language already developed to meet similar requirements

- ☞ Intuitive (frame) syntax plus high expressive power
- ☞ Well defined semantics via mapping to $\mathcal{SHIQ}$ DL
- ☞ Can use FaCT system to reason with OIL ontologies
- ☞ Extends existing Web standards (XML, RDF, RDFS)

Two efforts merged to produce single language, **DAML+OIL**

OIL and DAML+OIL

OIL language already developed to meet similar requirements

- Intuitive (frame) syntax plus high expressive power
- Well defined semantics via mapping to $\mathcal{SHIQ}$ DL
- Can use FaCT system to reason with OIL ontologies
- Extends existing Web standards (XML, RDF, RDFS)

Two efforts merged to produce single language, **DAML+OIL**

- Effectively a DL with RDFS based syntax

OIL and DAML+OIL

OIL language already developed to meet similar requirements

- Intuitive (frame) syntax plus high expressive power
- Well defined semantics via mapping to $\mathcal{SHIQ}$ DL
- Can use FaCT system to reason with OIL ontologies
- Extends existing Web standards (XML, RDF, RDFS)

Two efforts merged to produce single language, **DAML+OIL**

- Effectively a DL with RDFS based syntax
- Can use DL reasoning with DAML+OIL

OIL and DAML+OIL

OIL language already developed to meet similar requirements

- ☞ Intuitive (frame) syntax plus high expressive power
- ☞ Well defined semantics via mapping to $\mathcal{SHIQ}$ DL
- ☞ Can use FaCT system to reason with OIL ontologies
- ☞ Extends existing Web standards (XML, RDF, RDFS)

Two efforts merged to produce single language, **DAML+OIL**

- ☞ Effectively a DL with RDFS based syntax
- ☞ Can use DL reasoning with DAML+OIL
- ☞ E.g., **OilEd** ontology editor
  - Frame based interface (e.g., Protegé, OntoEdit)
  - Extended to capture whole of OIL/DAML+OIL languages
  - Reasoning support from FaCT (via CORBA interface)

OilEd

E.g., DAML+OIL medical terminology ontology

OilEd

E.g., DAML+OIL medical terminology ontology

- ☞ Transitive roles capture transitive partonomy, causality, etc.

OilEd

E.g., DAML+OIL medical terminology ontology

- Transitive roles capture transitive partonomy, causality, etc.

  Smoking $\sqsubseteq$ $\exists$causes.Cancer plus Cancer $\sqsubseteq$ $\exists$causes.Death

  $\Rightarrow$ Cancer $\sqsubseteq$ FatalThing

OilEd

E.g., DAML+OIL medical terminology ontology

- Transitive roles capture transitive partonomy, causality, etc.

  Smoking $\sqsubseteq$ $\exists$causes.Cancer plus Cancer $\sqsubseteq$ $\exists$causes.Death

  $\Rightarrow$ Cancer $\sqsubseteq$ FatalThing

- GCIs represent additional non-definitional knowledge

OilEd

E.g., DAML+OIL medical terminology ontology

- ☞ Transitive roles capture transitive partonomy, causality, etc.

  $\mathsf{Smoking} \sqsubseteq \exists \mathsf{causes.Cancer}$ plus $\mathsf{Cancer} \sqsubseteq \exists \mathsf{causes.Death}$

  $\Rightarrow \mathsf{Cancer} \sqsubseteq \mathsf{FatalThing}$

- ☞ GCIs represent additional non-definitional knowledge

  $\mathsf{Stomach\text{-}Ulcer} \doteq \mathsf{Ulcer} \sqcap \exists \mathsf{hasLocation.Stomach}$ plus

  $\mathsf{Stomach\text{-}Ulcer} \sqsubseteq \exists \mathsf{hasLocation.Lining\text{-}Of\text{-}Stomach}$

  $\Rightarrow \mathsf{Ulcer} \sqcap \exists \mathsf{hasLocation.Stomach} \sqsubseteq \mathsf{OrganLiningLesion}$

OilEd

E.g., DAML+OIL medical terminology ontology

- Transitive roles capture transitive partonomy, causality, etc.

  $\text{Smoking} \sqsubseteq \exists\text{causes.Cancer}$ plus $\text{Cancer} \sqsubseteq \exists\text{causes.Death}$

  $\Rightarrow \text{Cancer} \sqsubseteq \text{FatalThing}$

- GCIs represent additional non-definitional knowledge

  $\text{Stomach-Ulcer} \doteq \text{Ulcer} \sqcap \exists\text{hasLocation.Stomach}$ plus

  $\text{Stomach-Ulcer} \sqsubseteq \exists\text{hasLocation.Lining-Of-Stomach}$

  $\Rightarrow \text{Ulcer} \sqcap \exists\text{hasLocation.Stomach} \sqsubseteq \text{OrganLiningLesion}$

- Inverse roles capture e.g. causes/causedBy relationship

OilEd

E.g., DAML+OIL medical terminology ontology

- Transitive roles capture transitive partonomy, causality, etc.

  $\mathsf{Smoking} \sqsubseteq \exists \mathsf{causes.Cancer}$ plus $\mathsf{Cancer} \sqsubseteq \exists \mathsf{causes.Death}$

  $\Rightarrow \mathsf{Cancer} \sqsubseteq \mathsf{FatalThing}$

- GCIs represent additional non-definitional knowledge

  $\mathsf{Stomach\text{-}Ulcer} \doteq \mathsf{Ulcer} \sqcap \exists \mathsf{hasLocation.Stomach}$ plus

  $\mathsf{Stomach\text{-}Ulcer} \sqsubseteq \exists \mathsf{hasLocation.Lining\text{-}Of\text{-}Stomach}$

  $\Rightarrow \mathsf{Ulcer} \sqcap \exists \mathsf{hasLocation.Stomach} \sqsubseteq \mathsf{OrganLiningLesion}$

- Inverse roles capture e.g. causes/causedBy relationship

  $\mathsf{Death} \sqcap \exists \mathsf{causedBy.Smoking} \sqsubseteq \mathsf{PrematureDeath}$

  $\Rightarrow \mathsf{Smoking} \sqsubseteq \mathsf{CauseOfPrematureDeath}$

OilEd

E.g., DAML+OIL medical terminology ontology

- ☞ Transitive roles capture transitive partonomy, causality, etc.

  $\mathsf{Smoking} \sqsubseteq \exists \mathsf{causes}.\mathsf{Cancer}$ plus $\mathsf{Cancer} \sqsubseteq \exists \mathsf{causes}.\mathsf{Death}$

  $\Rightarrow \mathsf{Cancer} \sqsubseteq \mathsf{FatalThing}$

- ☞ GCIs represent additional non-definitional knowledge

  $\mathsf{Stomach\text{-}Ulcer} \doteq \mathsf{Ulcer} \sqcap \exists \mathsf{hasLocation}.\mathsf{Stomach}$ plus

  $\mathsf{Stomach\text{-}Ulcer} \sqsubseteq \exists \mathsf{hasLocation}.\mathsf{Lining\text{-}Of\text{-}Stomach}$

  $\Rightarrow \mathsf{Ulcer} \sqcap \exists \mathsf{hasLocation}.\mathsf{Stomach} \sqsubseteq \mathsf{OrganLiningLesion}$

- ☞ Inverse roles capture e.g. causes/causedBy relationship

  $\mathsf{Death} \sqcap \exists \mathsf{causedBy}.\mathsf{Smoking} \sqsubseteq \mathsf{PrematureDeath}$

  $\Rightarrow \mathsf{Smoking} \sqsubseteq \mathsf{CauseOfPrematureDeath}$

- ☞ Cardinality restrictions add consistency constraints

OilEd

E.g., DAML+OIL medical terminology ontology

- Transitive roles capture transitive partonomy, causality, etc.

  $\mathsf{Smoking} \sqsubseteq \exists \mathsf{causes.Cancer}$ plus $\mathsf{Cancer} \sqsubseteq \exists \mathsf{causes.Death}$

  $\Rightarrow \mathsf{Cancer} \sqsubseteq \mathsf{FatalThing}$

- GCIs represent additional non-definitional knowledge

  $\mathsf{Stomach\text{-}Ulcer} \doteq \mathsf{Ulcer} \sqcap \exists \mathsf{hasLocation.Stomach}$ plus

  $\mathsf{Stomach\text{-}Ulcer} \sqsubseteq \exists \mathsf{hasLocation.Lining\text{-}Of\text{-}Stomach}$

  $\Rightarrow \mathsf{Ulcer} \sqcap \exists \mathsf{hasLocation.Stomach} \sqsubseteq \mathsf{OrganLiningLesion}$

- Inverse roles capture e.g. causes/causedBy relationship

  $\mathsf{Death} \sqcap \exists \mathsf{causedBy.Smoking} \sqsubseteq \mathsf{PrematureDeath}$

  $\Rightarrow \mathsf{Smoking} \sqsubseteq \mathsf{CauseOfPrematureDeath}$

- Cardinality restrictions add consistency constraints

  $\mathsf{BloodPressure} \sqsubseteq \exists \mathsf{hasValue.}(\mathsf{High} \sqcup \mathsf{Low}) \sqcap \leqslant 1 \mathsf{hasValue}$ plus

  $\mathsf{High} \sqsubseteq \neg \mathsf{Low} \Rightarrow \mathsf{HighLowBloodPressure} \sqsubseteq \bot$

Database Schema and Query Reasoning

Database Schema and Query Reasoning

- $\mathcal{DLR}$ (n-ary DL) can capture semantics of many datamodelling methodologies (e.g., EER)

Database Schema and Query Reasoning

- $\mathcal{DLR}$ (n-ary DL) can capture semantics of many datamodelling methodologies (e.g., EER)
- Satisfiability preserving mapping to $\mathcal{SHIQ}$ allows use of DL reasoners (e.g., FaCT, RACER)

Database Schema and Query Reasoning

- $\mathcal{DLR}$ (n-ary DL) can capture semantics of many datamodelling methodologies (e.g., EER)
- Satisfiability preserving mapping to $\mathcal{SHIQ}$ allows use of DL reasoners (e.g., FaCT, RACER)
- DL Abox can also capture semantics of conjunctive queries
  - Can reason about query containment w.r.t. schema

Database Schema and Query Reasoning

- $\mathcal{DLR}$ (n-ary DL) can capture semantics of many datamodelling methodologies (e.g., EER)
- Satisfiability preserving mapping to $\mathcal{SHIQ}$ allows use of DL reasoners (e.g., FaCT, RACER)
- DL Abox can also capture semantics of conjunctive queries
  - Can reason about query containment w.r.t. schema
- DL reasoning can be used to support, e.g.
  - Schema design and integration
  - Query optimisation
  - Interoperability and federation

Database Schema and Query Reasoning

- $\mathcal{DLR}$ (n-ary DL) can capture semantics of many datamodelling methodologies (e.g., EER)
- Satisfiability preserving mapping to $\mathcal{SHIQ}$ allows use of DL reasoners (e.g., FaCT, RACER)
- DL Abox can also capture semantics of conjunctive queries
  - Can reason about query containment w.r.t. schema
- DL reasoning can be used to support, e.g.
  - Schema design and integration
  - Query optimisation
  - Interoperability and federation
- E.g., **I.COM** Intelligent Conceptual Modelling tool (Enrico Franconi)
  - Uses FaCT system to provide reasoning support for EER

Summary

Summary

☞ DLs are **logic based** KR formalisms

Summary

- DLs are **logic based** KR formalisms
- DL systems provide **efficient inference services**

Summary

- DLs are **logic based** KR formalisms
- DL systems provide **efficient inference services**
    - Careful choice of logic/algorithm

Summary

- ☞ DLs are **logic based** KR formalisms
- ☞ DL systems provide **efficient inference services**
  - Careful choice of logic/algorithm
  - Highly optimised implementation

Summary

- DLs are **logic based** KR formalisms
- DL systems provide **efficient inference services**
  - Careful choice of logic/algorithm
  - Highly optimised implementation
- DLs have proved effective in a range of applications

Summary

- DLs are **logic based** KR formalisms
- DL systems provide **efficient inference services**
  - Careful choice of logic/algorithm
  - Highly optimised implementation
- DLs have proved effective in a range of applications
  - Terminologies/Ontologies

Summary

- DLs are **logic based** KR formalisms
- DL systems provide **efficient inference services**
  - Careful choice of logic/algorithm
  - Highly optimised implementation
- DLs have proved effective in a range of applications
  - Terminologies/Ontologies
  - Databases

Summary

- DLs are **logic based** KR formalisms
- DL systems provide **efficient inference services**
  - Careful choice of logic/algorithm
  - Highly optimised implementation
- DLs have proved effective in a range of applications
  - Terminologies/Ontologies
  - Databases
- DLs have been influential in development of Semantic Web

Summary

- ☞ DLs are **logic based** KR formalisms
- ☞ DL systems provide **efficient inference services**
  - Careful choice of logic/algorithm
  - Highly optimised implementation
- ☞ DLs have proved effective in a range of applications
  - Terminologies/Ontologies
  - Databases
- ☞ DLs have been influential in development of Semantic Web
  - Web standard ontology language will be DL based

Resources

Slides from this talk

`www.cs.man.ac.uk/~horrocks/Slides/leipzig-jun-01.pdf`

FaCT system

`www.cs.man.ac.uk/fact`

OIL

`www.ontoknowledge.org/oil/`

DAML+OIL

`www.daml.org/language/`

OilEd

`img.cs.man.ac.uk/oil`

I.COM

`www.cs.man.ac.uk/~franconi/icom/`

Select Bibliography

F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H.-J. Profitlich. An empirical analysis of optimization techniques for terminological representation systems or: Making KRIS get a move on. In B. Nebel, C. Rich, and W. Swartout, editors, *Proc. of KR'92*, pages 270–281. Morgan Kaufmann, 1992.

F. Giunchiglia and R. Sebastiani. A SAT-based decision procedure for $\mathcal{ALC}$. In *Proc. of KR'96*, pages 304–314. Morgan Kaufmann, 1996.

V. Haarslev and R. Möller. High performance reasoning with very large knowledge bases: A practical case study. In *Proc. of IJCAI 2001* (to appear).

B. Hollunder and W. Nutt. Subsumption algorithms for concept languages. In *Proc. of ECAI'90*, pages 348–353. John Wiley & Sons Ltd., 1990.

Select Bibliography

I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.

I. Horrocks and P. F. Patel-Schneider. Comparing subsumption optimizations. In *Proc. of DL'98*, pages 90–94. CEUR, 1998.

I. Horrocks and P. F. Patel-Schneider. Optimising description logic subsumption. *Journal of Logic and Computation*, 9(3):267–293, 1999.

I. Horrocks and S. Tobies. Reasoning with axioms: Theory and practice. In *Proc. of KR'00* pages 285–296. Morgan Kaufmann, 2000.

E. Franconi and G. Ng. The i.com tool for intelligent conceptual modelling. In *Proc. of (KRDB'00)*, August 2000.

D. Fensel, F. van Harmelen, I. Horrocks, D. McGuinness, and P. F. Patel-Schneider. OIL: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2):38–45, 2001.