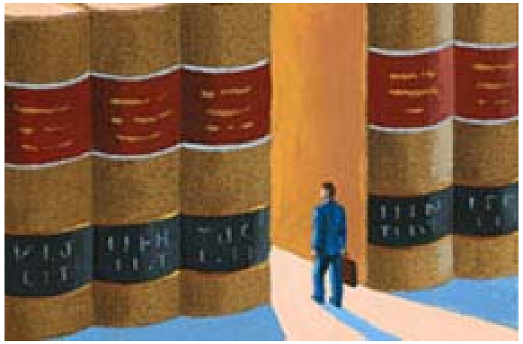
F. Description Logics – Part 2



This section is based on material from:

- Carsten Lutz, Uli Sattler: http://www.computational-logic.org/content/events/iccl-ss-2005/lectures/lutz/index.php?id=24
- Ian Horrocks: http://www.cs.man.ac.uk/~horrocks/Teaching/cs646/

Syntax für DLs (ohne concrete domains)

Hitzler & Sure, 2005

AIFB

	Concepts	
ALC	Atomic	A, B
ALC	Not	¬C
ALC	And	C ⊓ D
ALC	Or	C ⊔ D
ALC	Exists	∃R.C
ALC	For all	∀R.C
Q (N)	At least	≥n R.C (≥n R)
Q (N)	At most	≤n R.C (≤n R)
O	Nominal	$\{i_1,\ldots,i_n\}$

	Roles	
	Atomic	R
I	Inverse	R⁻

Ontology (=Knowledge Base)

Concept Axioms (TBox)	
Subclass	C ⊑ D
Equivalent	C ≡ D

	Role Axioms (RBox)	
H	Subrole	R ⊑ S
S	Transitivity	Trans(S)

Assertional Axioms (ABox)	
Instance	C(a)
Role	R(a,b)
Same	a = b
Different	a ≠ b

S = ALC + Transitivity

OWL DL = SHOIN(D) (D: concrete domain)

The Description Logic $\mathcal{ALC}$: Syntax

Atomic types: concept names $A, B, \ldots$ (unary predicates)

role names $R, S, \ldots$ (binary predicates)

Constructors:
- $\neg C$ (negation)
- $C \sqcap D$ (conjunction)
- $C \sqcup D$ (disjunction)
- $\exists R.C$ (existential restriction)
- $\forall R.C$ (value restriction)

Abbreviations:
- $C \rightarrow D = \neg C \sqcup D$ (implication)
- $C \leftrightarrow D = C \rightarrow D \sqcap D \rightarrow C$ (bi-implication)
- $\top = (A \sqcup \neg A)$ (top concept)
- $\bot = A \sqcap \neg A$ (bottom concept)

Examples

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

Interpretations

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

- $\Delta^{\mathcal{I}}$ is a non-empty set (the domain)
- $\cdot^{\mathcal{I}}$ is the interpretation function mapping

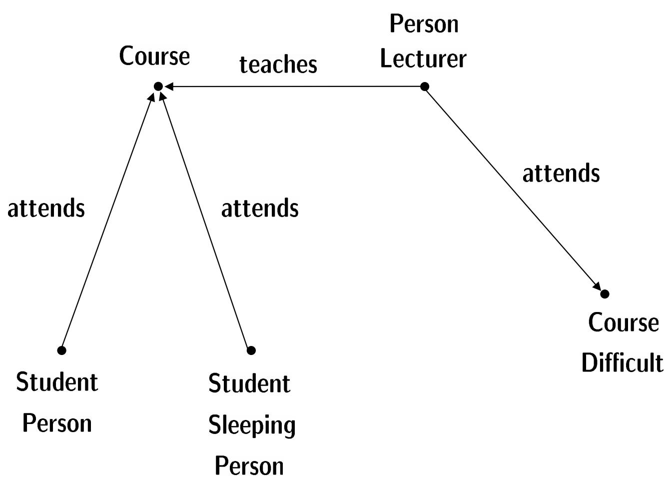
  each concept name A to a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$ and

  each role name R to a binary relation $R^{\mathcal{I}}$ over $\Delta^{\mathcal{I}}$.

Intuition: interpretation is complete description of the world

Technically: interpretation is first-order structure
with only unary and binary predicates

Example

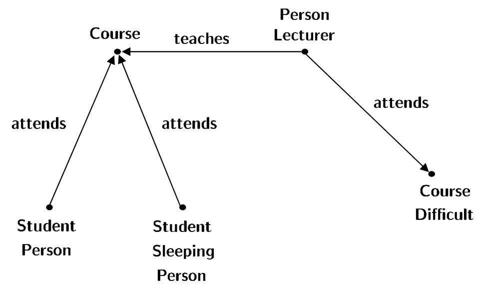


Semantics of Complex Concepts

$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$ $\quad (C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$ $\quad (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$

$(\exists R.C)^{\mathcal{I}} = \{d \mid \text{there is an } e \in \Delta^{\mathcal{I}} \text{ with } (d,e) \in R^{\mathcal{I}} \text{ and } e \in C^{\mathcal{I}}\}$

$(\forall R.C)^{\mathcal{I}} = \{d \mid \text{for all } e \in \Delta^{\mathcal{I}}, (d,e) \in R^{\mathcal{I}} \text{ implies } e \in C^{\mathcal{I}}\}$



Person $\sqcap$ $\exists$attends.Course

Person $\sqcap$ $\forall$attends.($\neg$Course $\sqcup$ Difficult)

TBoxes

Capture an application's terminology means defining concepts

TBoxes are used to store concept definitions:

Syntax:

finite set of concept equations $A \doteq C$

with A concept name and C concept

left-hand sides must be unique!

Semantics:

interpretation $\mathcal{I}$ satisfies $A \doteq C$ iff $A^{\mathcal{I}} = C^{\mathcal{I}}$

$\mathcal{I}$ is model of $\mathcal{T}$ if it satisfies all definitions in $\mathcal{T}$

E.g.: Lecturer $\doteq$ Person $\sqcap$ $\exists$teaches.Course

Yields two kinds of concept names: defined and primitive

TBox: Example

TBoxes are used as ontologies:

$$\text{Woman} \doteq \text{Person} \sqcap \text{Female}$$

$$\text{Man} \doteq \text{Person} \sqcap \neg\text{Woman}$$

$$\text{Lecturer} \doteq \text{Person} \sqcap \exists\text{teaches.Course}$$

$$\text{Student} \doteq \text{Person} \sqcap \exists\text{attends.Course}$$

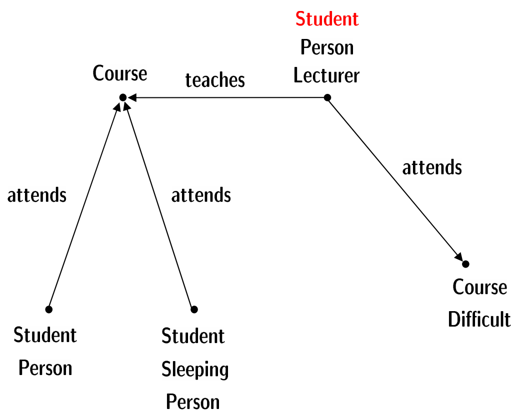
$$\text{BadLecturer} \doteq \text{Person} \sqcap \forall\text{teaches.}(\text{Course} \rightarrow \text{Boring})$$

TBox: Example II

A TBox restricts the set of admissible interpretations.

$$\text{Lecturer} \doteq \text{Person} \sqcap \exists\text{teaches.Course}$$

$$\text{Student} \doteq \text{Person} \sqcap \exists\text{attends.Course}$$



Reasoning Tasks — Subsumption

C subsumed by D w.r.t. $\mathcal{T}$ (written $C \sqsubseteq_{\mathcal{T}} D$)

iff

$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for all models $\mathcal{I}$ of $\mathcal{T}$

Intuition: If $C \sqsubseteq_{\mathcal{T}} D$, then D is more general than C

Example:

$$\text{Lecturer} \doteq \text{Person} \sqcap \exists\text{teaches.Course}$$

$$\text{Student} \doteq \text{Person} \sqcap \exists\text{attends.Course}$$

Then

$$\text{Lecturer} \sqcap \exists\text{attends.Course} \sqsubseteq_{\mathcal{T}} \text{Student}$$

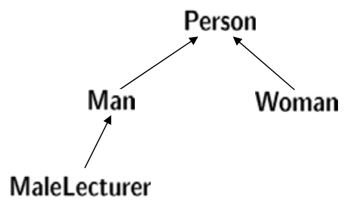
Reasoning Tasks — Classification

Classification: arrange all defined concepts from a TBox in a hierarchy w.r.t. generality

$$\text{Woman} \doteq \text{Person} \sqcap \text{Female}$$

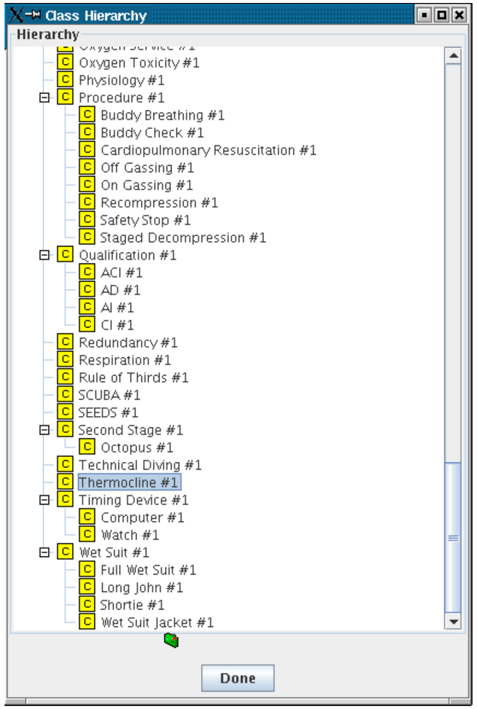
$$\text{Man} \doteq \text{Person} \sqcap \neg\text{Woman}$$

$$\text{MaleLecturer} \doteq \text{Man} \sqcap \exists\text{teaches.Course}$$



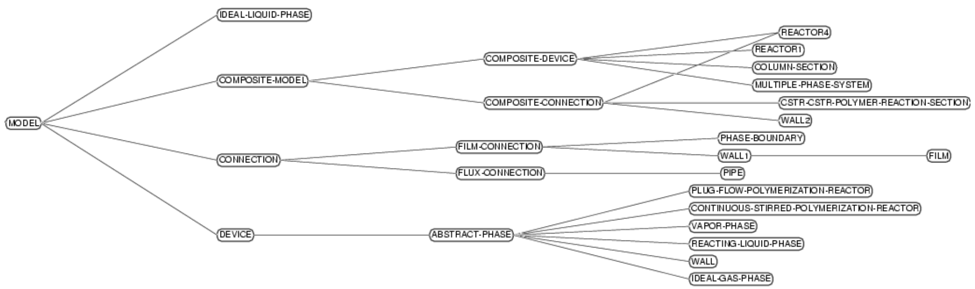
Can be computed using multiple subsumption tests

Provides a principled view on ontology for browsing, maintaining, etc.



A Concept Hierarchy

Excerpt from a process engineering ontology



Reasoning Tasks — Satisfiability

C is satisfiable w.r.t. $\mathcal{T}$ iff $\mathcal{T}$ has a model with $C^\mathcal{I} \neq \emptyset$

Intuition: If unsatisfiable, the concept contains a contradiction.

Example: Woman $\doteq$ Person $\sqcap$ Female

Man $\doteq$ Person $\sqcap$ $\neg$Woman

Then $\exists$sibling.Man $\sqcap$ $\forall$sibling.Woman is unsatisfiable w.r.t. $\mathcal{T}$

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

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

Many reasoners decide satisfiability rather than subsumption.

Definitorial TBoxes

A primitive interpretation for TBox $\mathcal{T}$ interpretes

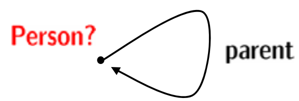
- the primitive concept names in $\mathcal{T}$
- all role names

A TBox is called definitorial if every primitive interpretation for $\mathcal{T}$ can be uniquely extended to a model of $\mathcal{T}$.

i.e.: primitive concepts (and roles) uniquely determine defined concepts

Not all TBoxes are definitorial:

Person $\doteq$ $\exists$parent.Person



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

Acyclic TBoxes

TBox $\mathcal{T}$ is acyclic if there are no definitorial cycles:

~~Lecturer $\doteq$ Person $\sqcap$ $\exists$teaches.Course~~

~~Course $\doteq$ $\exists$has-title.Title $\sqcap$ $\exists$tought-by.Lecturer~~

Expansion of acyclic TBox $\mathcal{T}$:

exhaustively replace defined concept names with their definition

(terminates due to acyclicity)

Acyclic TBoxes are always definitorial:

first expand, then set $A^{\mathcal{I}} := C^{\mathcal{I}}$ for all $A \doteq C \in \mathcal{T}$

Acyclic TBoxes II

For reasoning, acyclic TBox can be eliminated:

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

May yield an exponential blow-up:

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

General Concept Inclusions

View of TBox as set of constraints

General TBox: finite set of general concept implications (GCIs)

$$C \sqsubseteq D$$

with both C and D allowed to be complex

e.g. Course $\sqcap$ $\forall$attended-by.Sleeping $\sqsubseteq$ Boring

Interpretation $\mathcal{I}$ is model of general TBox $\mathcal{T}$ if

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

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

e.g. Student $\sqcap$ $\exists$has-favourite.SoccerTeam $\doteq$ Student $\sqcap$ $\exists$has-favourite.Beer

Note: $C \sqsubseteq D$ equivalent to $\top \doteq C \rightarrow D$

ABoxes

ABoxes describe a snapshot of the world

An ABox is a finite set of assertions

$a : C$	(a individual name, C concept)
$(a, b) : R$	(a, b individual names, R role name)

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

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

$\mathcal{I}$ satisfies an assertion

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

$\mathcal{I}$ is a model for an ABox $\mathcal{A}$ if $\mathcal{I}$ satisfies all assertions in $\mathcal{A}$.

ABoxes II

Note:

- interpretations describe the state if the world in a complete way
- ABoxes describe the state if the world in an incomplete way

(uli, dl-course) : tought-by   uli : Female

does not imply

dl-course : ∀tought-by.Female

An ABox has many models!

An ABox constraints the set of admissibile models similar to a TBox

Example for ABox Reasoning

ABox

dumbo : Mammal   t14 : Trunk

~~g23 : Darkgrey~~   (dumbo, t14) : bodypart

(dumbo, g23) : color

dumbo : ∀color.Lightgrey

TBox

$$\text{Elephant} \doteq \text{Mammal} \sqcap \exists\text{bodypart.Trunk} \sqcap \forall\text{color.Grey}$$

$$\text{Grey} \doteq \text{Lightgrey} \sqcup \text{Darkgrey}$$

$$\bot \doteq \text{Lightgrey} \sqcap \text{Darkgrey}$$

1. ABox is inconsistent w.r.t. TBox.
2. dumbo is an instance of Elephant

Reasoning with ABoxes

ABox consistency

Given an ABox $\mathcal{A}$ and a TBox $\mathcal{T}$, do they have a common model?

Instance checking

Given an ABox $\mathcal{A}$, a TBox $\mathcal{T}$, an individual name a, and a concept C does $a^\mathcal{I} \in C^\mathcal{I}$ hold in all models of $\mathcal{A}$ and $\mathcal{T}$?

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

The two tasks are interreducible:

- $\mathcal{A}$ consistent w.r.t. $\mathcal{T}$ iff $\mathcal{A}, \mathcal{T} \not\models a : \bot$
- $\mathcal{A}, \mathcal{T} \models a : C$ iff $\mathcal{A} \cup \{a : \neg C\}$ is not consistent

2. Tableau algorithms for $\mathcal{ALC}$ and extensions

We see a tableau algorithm for $\mathcal{ALC}$ and extend it with

① general TBoxes and

② inverse roles

Goal: Design sound and complete desicion procedures for satisfiability (and subsumption) of DLs which are well-suited for implementation purposes

A tableau algorithm for the satisfiability of $\mathcal{ALC}$ concepts

Goal: design an algorithm which takes an $\mathcal{ALC}$ concept C_0 and

1. returns *"satisfiable"* iff C_0 is satisfiable and
2. terminates, on every input,

i.e., which **decides** satisfiability of $\mathcal{ALC}$ concepts.

Recall: such an algorithm **cannot** exist for FOL since satisfiability of FOL is undecidable.

Idea: our algorithm

- is tableau-based and
- tries to construct a **model** of C_0
- by breaking C_0 down syntactically, thus
- inferring new constraints on such a model.

Preliminaries: Negation Normal Form

To make our life easier, we transform each concept C_0 into an **equivalent** C_1 in **NNF**

Equivalent: $C_0 \sqsubseteq C_1$ and $C_1 \sqsubseteq C_0$

NNF: negation occurs only in front of concept names

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

$$\begin{aligned}
\neg(C \sqcap D) &\rightsquigarrow \neg C \sqcup \neg D \\
\neg(C \sqcup D) &\rightsquigarrow \neg C \sqcap \neg D \\
\neg\neg C &\rightsquigarrow C \\
\neg\forall R.C &\rightsquigarrow \exists R.\neg C \\
\neg\exists R.C &\rightsquigarrow \forall R.\neg C
\end{aligned}$$

From now on: concepts are in NNF and
sub(C) denotes the set of all sub-concepts of C

More intuition

Find out whether $A \sqcap \exists R.B \sqcap \forall R.\neg B$ is satisfiable...
$A \sqcap \exists R.B \sqcap \forall R.(\neg B \sqcup \exists S.E)$

Our tableau algorithm works on a **completion tree** which

- represents a model $\mathcal{I}$: **nodes** represent elements of $\Delta^{\mathcal{I}}$
  $\rightsquigarrow$ each node x is labelled with concepts $\mathcal{L}(x) \subseteq$ **sub**(C_0)
  $C \in \mathcal{L}(x)$ is read as "x should be an instance of C"

  edges represent role successorship
  $\rightsquigarrow$ each edge $\langle x, y\rangle$ is labelled with a role-name from C_0
  $R \in \mathcal{L}(\langle x, y\rangle)$ is read as "(x, y) should be in $R^{\mathcal{I}}$"
- is initialised with a single root node x_0 with $\mathcal{L}(x_0) = \{C_0\}$
- is expanded using **completion rules**

Completion rules for $\mathcal{ALC}$

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

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

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

$\forall$-rule: if $\forall S.C \in \mathcal{L}(x)$ and there is an S-successor y of x with $C \notin \mathcal{L}(y)$
then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$

Properties of the completion rules for $\mathcal{ALC}$

We only apply rules if their application does **"something new"**

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

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

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

$\forall$-rule: if $\forall S.C \in \mathcal{L}(x)$ and there is an S-successor y of x with $C \notin \mathcal{L}(y)$
then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$

Properties of the completion rules for $\mathcal{ALC}$

The $\sqcup$-rule is **non-deterministic**:

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

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

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

$\forall$-rule: if $\forall S.C \in \mathcal{L}(x)$ and there is an S-successor y of x with $C \notin \mathcal{L}(y)$
then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$

Last details on tableau algorithm for $\mathcal{ALC}$

Clash: a c-tree contains a **clash** if it has a node x with $\bot \in \mathcal{L}(x)$ or $\{A, \neg A\} \subseteq \mathcal{L}(x)$ — otherwise, it is **clash-free**

Complete: a c-tree is **complete** if none of the completion rules can be applied to it

Answer behaviour: when started for C_0 (in NNF!), the tableau algorithm

- **is initialised** with a single root node x_0 with $\mathcal{L}(x_0) = \{C_0\}$
- repeatedly applies the **completion rules** (in whatever order it likes)
- **answer** "C_0 *is satisfiable*" iff the completion rules **can** be applied **in such a way that** it results in a complete and clash-free c-tree (careful: this is non-deterministic)

...go back to examples

Properties of our tableau algorithm

Lemma: Let C_0 an $\mathcal{ALC}$-concept in NNF. Then

1. the algorithm terminates when applied to C_0 and
2. the rules can be applied such that they generate a clash-free and complete completion tree iff C_0 is satisfiable.

Corollary:

1. Our tableau algorithm **decides satisfiability** and **subsumption** of $\mathcal{ALC}$.
2. Satisfiability (and subsumption) in $\mathcal{ALC}$ is decidable in **PSpace**.
3. $\mathcal{ALC}$ has the **finite model property**
   i.e., every satisfiable concept has a **finite** model.
4. $\mathcal{ALC}$ has the **tree model property**
   i.e., every satisfiable concept has a **tree** model.
5. $\mathcal{ALC}$ has the **finite tree model property**
   i.e., every satisfiable concept has a **finite tree** model.

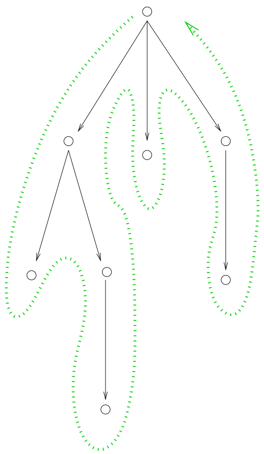
Proof of the Lemma: Termination

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

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

Proof of the Lemma: Termination

If we construct c-tree in depth-first manner and
re-use space for branches already visited,
mark $\exists R.C \in \mathcal{L}(x)$ with "todo" or "done"



we can run tableau algorithm in **polynomial space**:

- c-tree is of depth bounded by $|C_0|$, and
- we keep only a single branch in memory at any time.

$\rightsquigarrow$ (2) of our corollary: $\mathcal{ALC}$ is in PSpace

Proof of the Lemma: Soundness

(2) Let the algorithm stop with a complete and clash-free c-tree.
From this, define an interpretation $\mathcal{I}$ as follows:

$$\begin{aligned}
\Delta^{\mathcal{I}} &:= \{x \mid x \text{ is a node in c-tree}\}\\
A^{\mathcal{I}} &:= \{x \mid A \in \mathcal{L}(x)\} \text{ for concept names } A\\
R^{\mathcal{I}} &:= \{(x,y) \mid y \text{ is an } R\text{-successor of } x \text{ in c-tree}\}
\end{aligned}$$

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

$$D \in \mathcal{L}(x) \textbf{ implies } x \in D^{\mathcal{I}}$$

→ concept names D: by definition of $\mathcal{I}$

→ for negated concept names D: due to clash-freeness and induction

→ for conjunctions/disjunctions/existential restrictions/universal restrictions D:
due to completeness and by induction

$\rightsquigarrow$ since C_0 is in label of root node, $\mathcal{I}$ is a model of C_0

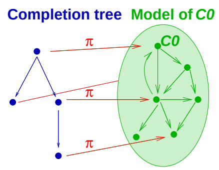
Proof of the Lemma: Completeness

(3) Let C_0 be satisfiable, and let $\mathcal{I}$ be a model of it with $a_0 \in C_0^{\mathcal{I}}$.
Use $\mathcal{I}$ to steer the application of the (only non-deterministic) $\sqcup$-rule:

Inductively define a total mapping π :

start with $\pi(x_0) = a_0$, and show that

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



$$\begin{aligned}
(*)\ &\text{if } C \in \mathcal{L}(x), \text{ then } \pi(x) \in C^{\mathcal{I}}\\
&\text{if } y \text{ is an } R\text{-succ. of } x, \text{ then } \langle \pi(x), \pi(y)\rangle \in R^{\mathcal{I}}
\end{aligned}$$

- easy for $\sqcap$- and $\forall$-rule,
- for $\exists$-rule, we need to extend π to the newly created R-successor
- for $\sqcup$-rule, if $C_1 \sqcup C_2 \in \mathcal{L}(x)$, $(*)$ implies that $\pi(x) \in (C_1 \sqcup C_2)^{\mathcal{I}}$
  $\rightsquigarrow$ we can choose C_i with $\pi(x) \in C_i^{\mathcal{I}}$ to add to $\mathcal{L}(x)$ and thus preserve $(*)$

$\rightsquigarrow$ easy to see: $(*)$ implies that c-tree is **clash-free**

Proof of the Lemma: Harvest

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

$\mathcal{I}$ is
- **finite** because c-tree has finitely many nodes
- a **tree** because c-tree is a tree

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

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

Extend tableau algorithm to $\mathcal{ALC}$ with general TBoxes

Recall:
- **Concept inclusion:** of the form $C \dot{\sqsubseteq} D$ for C, D (complex) concepts
- **(General) TBox:** a finite set of concept inclusions
- $\mathcal{I}$ **satisfies** $C \dot{\sqsubseteq} D$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
- $\mathcal{I}$ **is a model of TBox** $\mathcal{T}$ iff $\mathcal{I}$ satisfies each concept equation in $\mathcal{T}$
- C_0 **is satisfiable w.r.t.** $\mathcal{T}$ iff there is a model $\mathcal{I}$ of $\mathcal{T}$ with $C_0^{\mathcal{I}} \neq \emptyset$

Goal – Lemma: Let C_0 an $\mathcal{ALC}$-concept and $\mathcal{T}$ be a an $\mathcal{ALC}$-TBox. Then

1. the algorithm terminates when applied to $\mathcal{T}$ and C_0 and
2. the rules can be applied such that they generate a clash-free and complete completion tree iff C_0 is satisfiable w.r.t. $\mathcal{T}$.

Extend tableau algorithm to $\mathcal{ALC}$ with general TBoxes: Preliminaries

We extend our tableau algorithm by adding a **new completion rule**:

- remember that nodes represent elements of $\Delta^{\mathcal{I}}$ and
- if $C \dot{\sqsubseteq} D \in \mathcal{T}$, then for each element x in a model $\mathcal{I}$ of $\mathcal{T}$
  if $x \in C^{\mathcal{I}}$, then $x \in D^{\mathcal{I}}$
  hence $x \in (\neg C)^{\mathcal{I}}$ or $x \in D^{\mathcal{I}}$
  $x \in (\neg C \sqcup D)^{\mathcal{I}}$
  $x \in (\mathsf{NNF}(\neg C \sqcup D))^{\mathcal{I}}$

  for $\mathsf{NNF}(E)$ the negation normal form of E

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

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

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

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

$\forall$-rule: if $\forall S.C \in \mathcal{L}(x)$ and there is an S-successor y of x with $C \notin \mathcal{L}(y)$
then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$

$\mathcal{T}$-rule: if $C_1 \dot{\sqsubseteq} C_2 \in \mathcal{T}$ and $\mathsf{NNF}(\neg C_1 \sqcup C_2) \notin \mathcal{L}(x)$
then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{\mathsf{NNF}(\neg C_1 \sqcup C_2)\}$

A tableau algorithm for $\mathcal{ALC}$ with general TBoxes

Example: Consider satisfiability of C w.r.t. $\{C \dot\sqsubseteq \exists R.C\}$

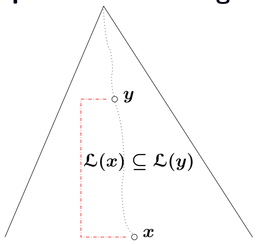
Tableau algorithm no longer terminates!

Reason: size of concepts no longer decreases along paths in a completion tree

Observation: most nodes on this path look the same and we keep repeating ourselves

Regain termination with a "cycle-detection" technique called blocking

Intuitively, whenever we find a situation where y has to satisfy *stronger* constraints than x, we *freeze* x, i.e., block rules from being applied to x



A tableau algorithm for $\mathcal{ALC}$ with general TBoxes: Blocking

- x is **directly blocked** if it has an ancestor y with $\mathcal{L}(x) \subseteq \mathcal{L}(y)$
- in this case and if y is the "closest" such node to x, we say that x **is blocked by** y
- a node is **blocked** if it is directly blocked or one of its ancestors is blocked

$\oplus$ restrict the application of all rules to nodes which are not blocked

$\rightsquigarrow$ **completion rules for $\mathcal{ALC}$ w.r.t. TBoxes**

A tableau algorithm for $\mathcal{ALC}$ with general TBoxes

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

$\sqcup$-rule: if $C_1 \sqcup C_2 \in \mathcal{L}(x)$, $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$, and x is not blocked
then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C\}$ for some $C \in \{C_1, C_2\}$

$\exists$-rule: if $\exists S.C \in \mathcal{L}(x)$, x has no S-successor y with $C \in \mathcal{L}(y)$, and x is not blocked
then create a new node y with $\mathcal{L}(\langle x, y\rangle) = \{S\}$ and $\mathcal{L}(y) = \{C\}$

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

$\mathcal{T}$-rule: if $C_1 \dot\sqsubseteq C_2 \in \mathcal{T}$, NNF$(\neg C_1 \sqcup C_2) \notin \mathcal{L}(x)$ and x is not blocked
then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{\text{NNF}(\neg C_1 \sqcup C_2)\}$

Tableaux Rules for $\mathcal{ALC}$

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

Tableaux Rule for Transitive Roles

$x \bullet \{\forall R.C, \ldots\}$ $R \downarrow$ $y \bullet \{\ldots\}$	$\rightarrow_{\forall_+}$	$x \bullet \{\forall R.C, \ldots\}$ $R \downarrow$ $y \bullet \{\forall R.C, \ldots\}$

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

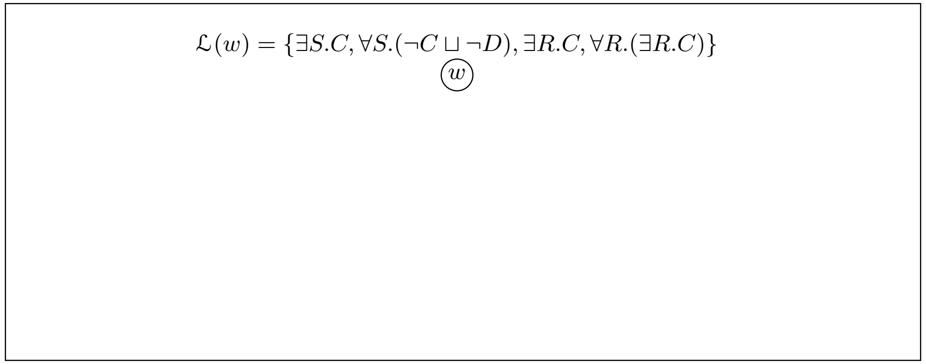
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role

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

w

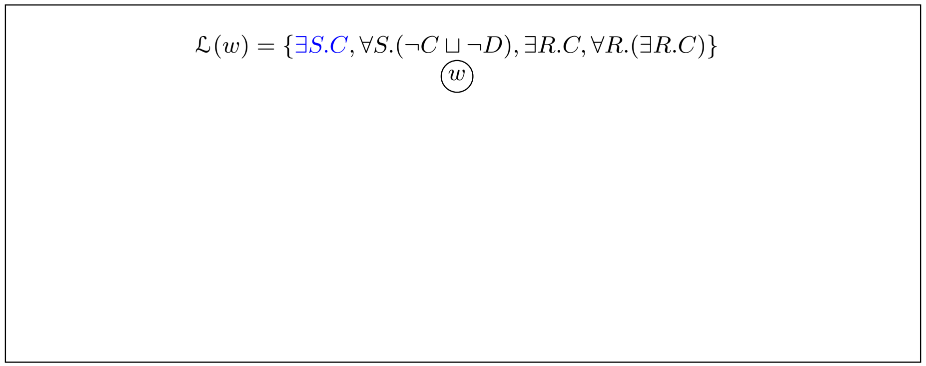
Tableaux Algorithm — Example

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



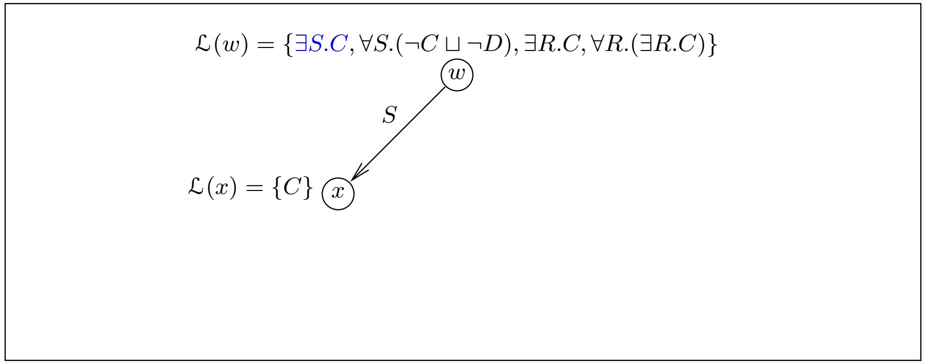
Tableaux Algorithm — Example

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



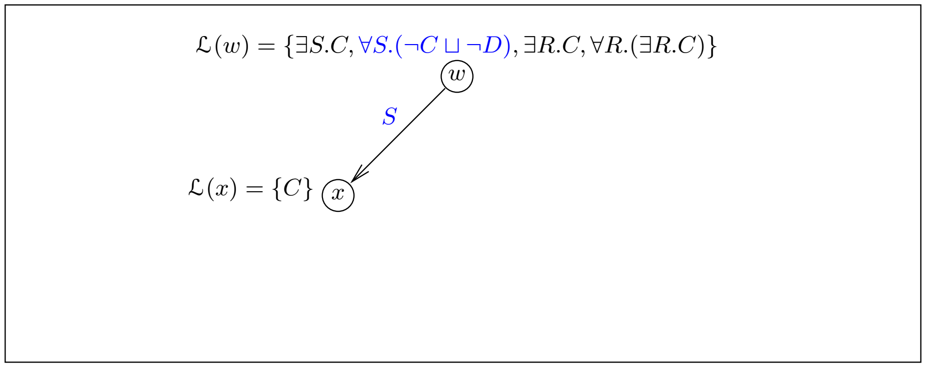
Tableaux Algorithm — Example

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



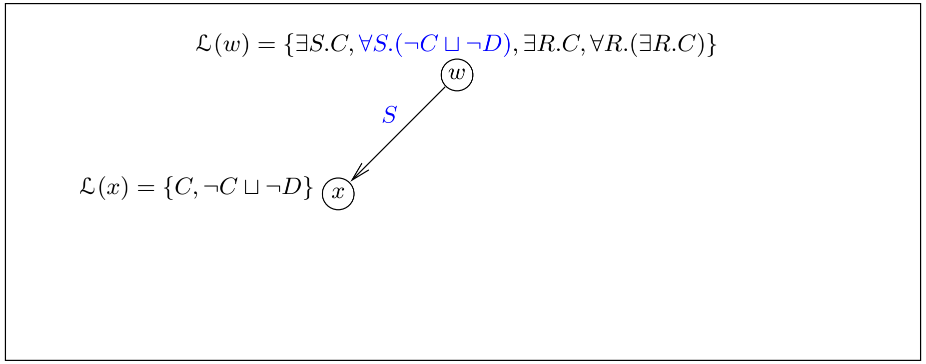
Tableaux Algorithm — Example

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



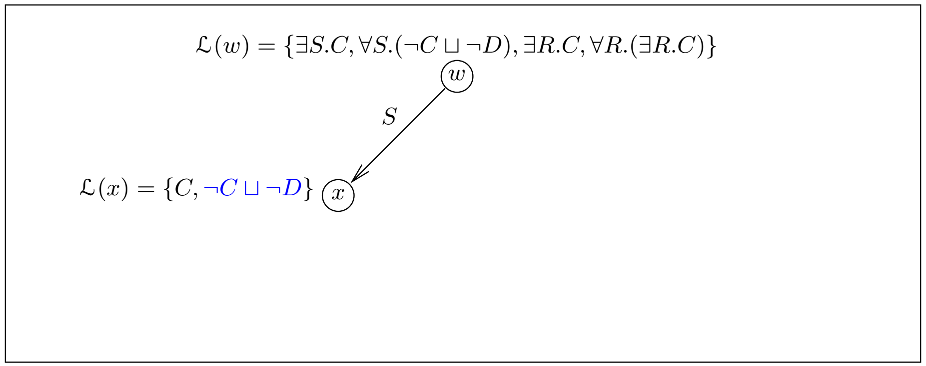
Tableaux Algorithm — Example

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



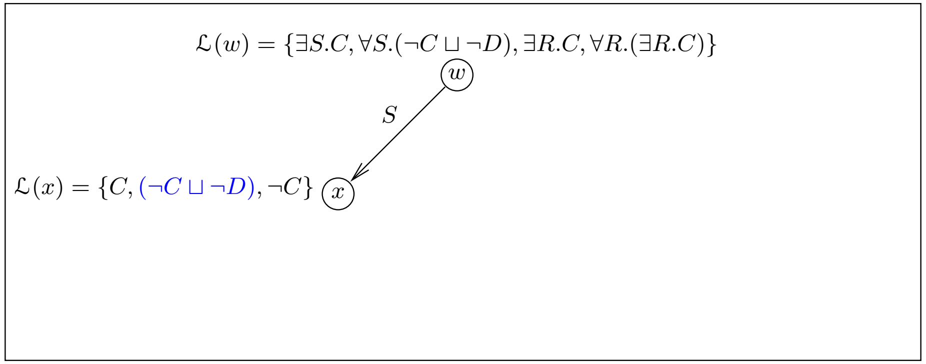
Tableaux Algorithm — Example

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



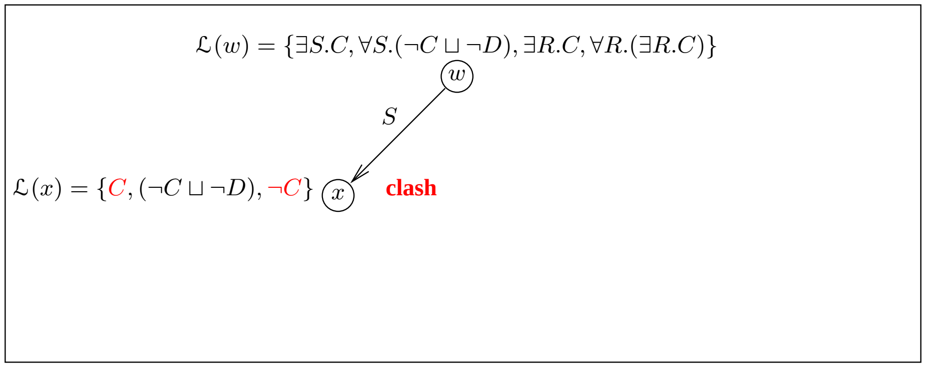
Tableaux Algorithm — Example

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



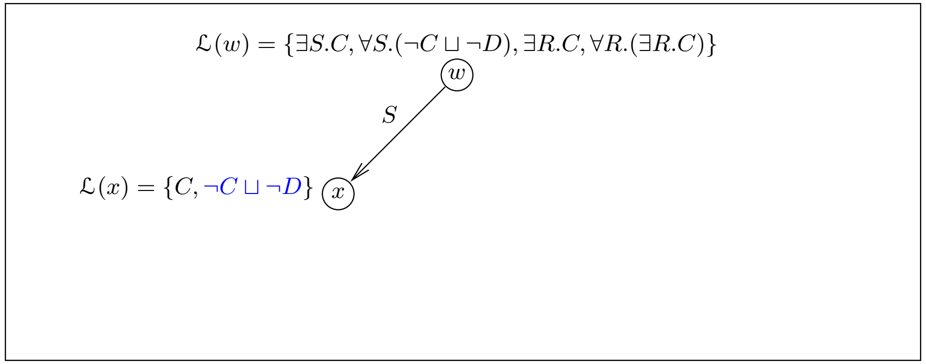
Tableaux Algorithm — Example

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



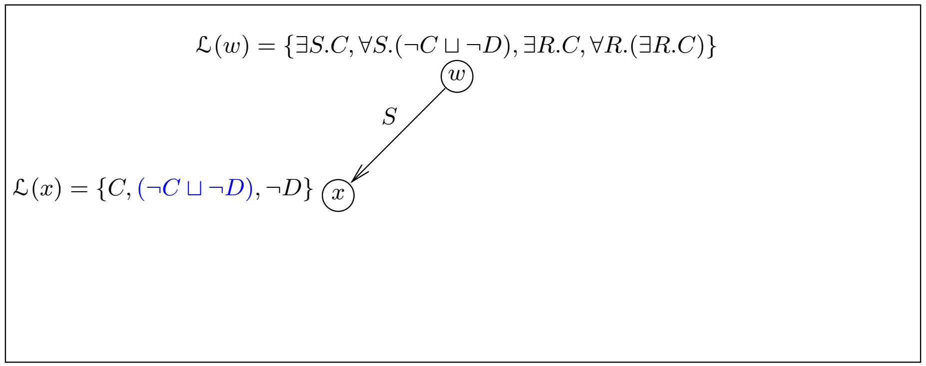
Tableaux Algorithm — Example

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



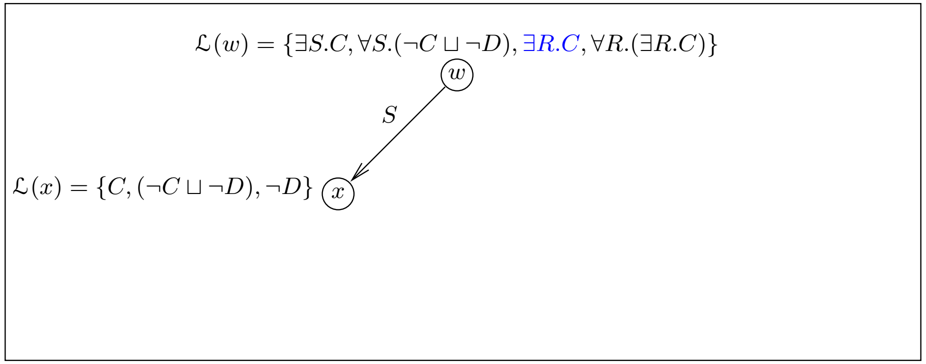
Tableaux Algorithm — Example

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



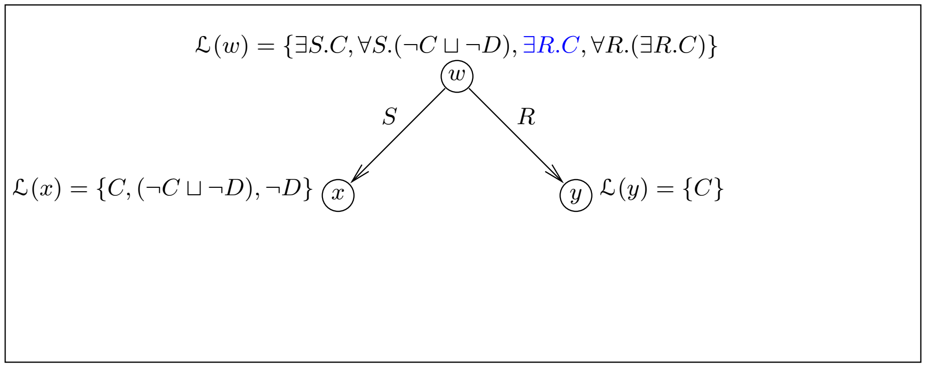
Tableaux Algorithm — Example

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



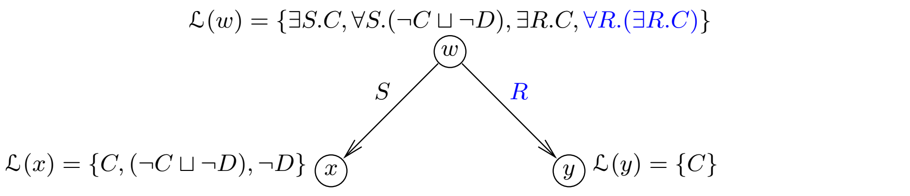
Tableaux Algorithm — Example

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



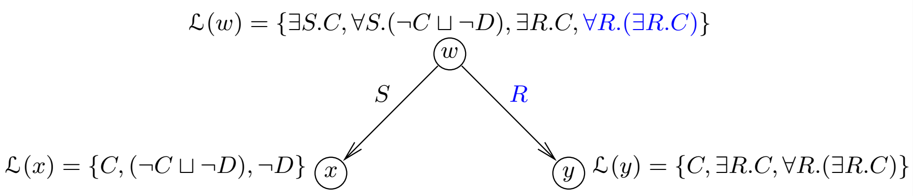
Tableaux Algorithm — Example

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



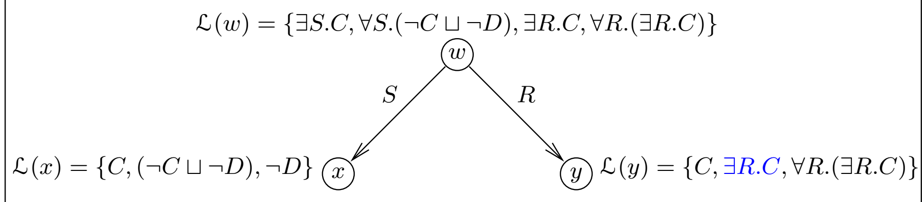
Tableaux Algorithm — Example

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



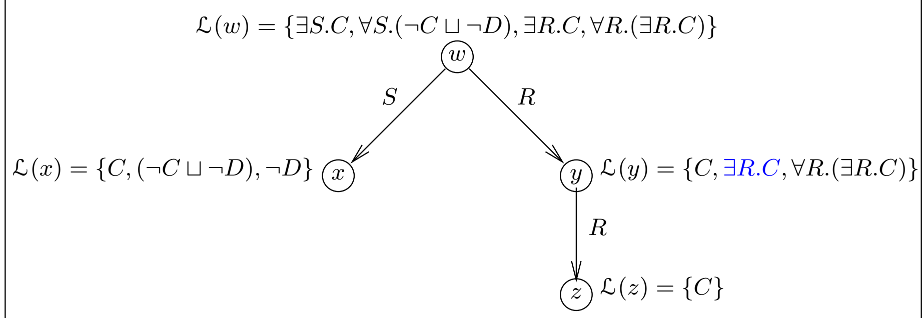
Tableaux Algorithm — Example

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



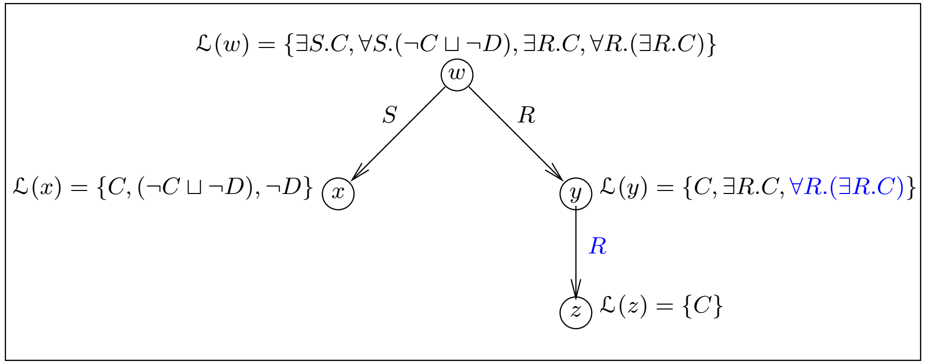
Tableaux Algorithm — Example

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



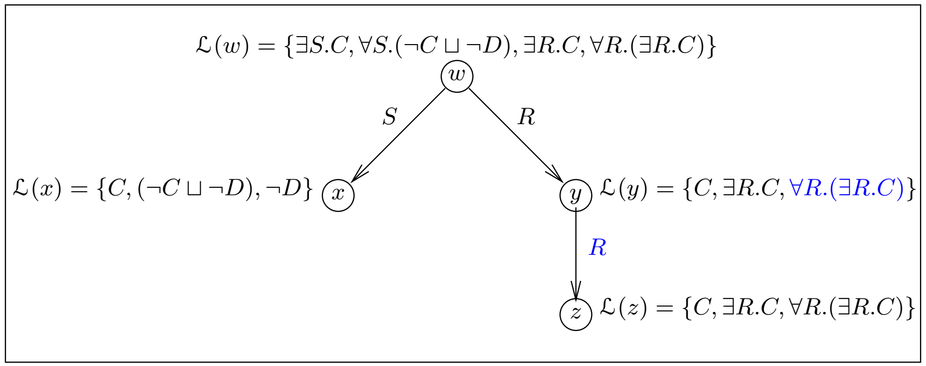
Tableaux Algorithm — Example

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



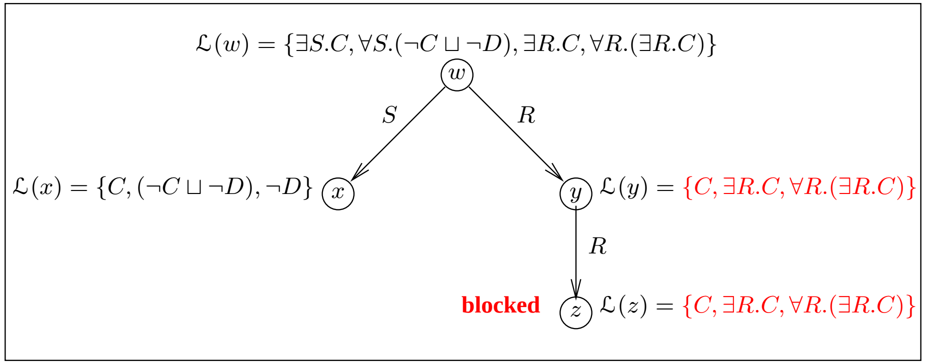
Tableaux Algorithm — Example

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



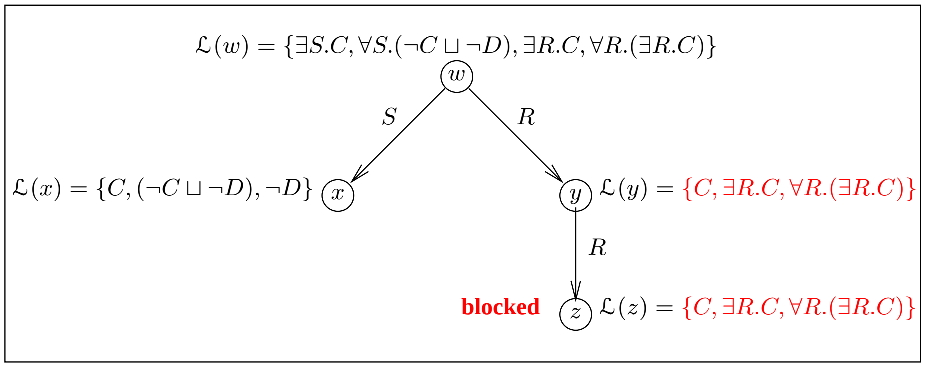
Tableaux Algorithm — Example

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

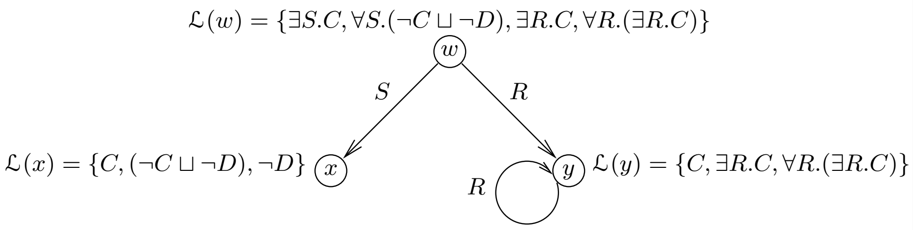
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



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**

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

Lemma: Let $\mathcal{T}$ be a general $\mathcal{ALC}$-Tbox and C_0 an $\mathcal{ALC}$-concept. Then

1. the algorithm terminates when applied to $\mathcal{T}$ and C_0 and
2. the rules can be applied such that they generate a clash-free and complete completion tree iff C_0 is satisfiable w.r.t. $\mathcal{T}$.

Corollary:
1. Satisfiability of $\mathcal{ALC}$-concept w.r.t. TBoxes is decidable
2. $\mathcal{ALC}$ with TBoxes has the finite model property
3. $\mathcal{ALC}$ with TBoxes has the tree model property

Proof of the Lemma: Termination

(1) termination is, again, due to the following properties: let $n = |C_0| + |C_\mathcal{T}|$ and

$$\mathbf{sub}(C_0, \mathcal{T}) = \mathbf{sub}(C_0) \cup \bigcup_{C \dot{\sqsubseteq} D \in \mathcal{T}} \mathbf{sub}(C) \cup \mathbf{sub}(D)$$

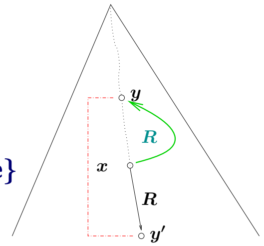
1. the c- tree is built in a **monotonic way**:
   each rule either extends node labels or adds a node (with a label)
2. node labels are restricted to subsets of $\mathbf{sub}(C_0, \mathcal{T})$ and $\# \mathbf{sub}(C_0, \mathcal{T}) \leq n$
3. the **breadth** of the c-tree is bounded by n:
   at most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

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

Proof of the Lemma: Soundness

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

$$\Delta^\mathcal{I} := \{x \mid x \text{ is a node in } \mathcal{T}, x \text{ is not blocked}\}$$
$$A^\mathcal{I} := \{x \in \Delta^\mathcal{I} \mid A \in \mathcal{L}(x)\} \text{ for concept names } A$$
$$R^\mathcal{I} := \{\langle x, y \rangle \in \Delta^{\mathcal{I}^2} \mid y \text{ is an } R\text{-succ of } x \text{ in c-tree or } y \text{ blocks an } R\text{-succ of } x \text{ in c-tree}\}$$



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

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

This implies that $\mathcal{I}$ **is indeed a model of** C_0 **and** $\mathcal{T}$ because
(a) C_0 is in the label of the root node which cannot be blocked (!) and
(b) $\neg C \sqcup D$ is in the label of each node, for each $C \dot{\sqsubseteq} D \in \mathcal{T}$

Proof of the Lemma: Completeness

(3) Let C_0 be satisfiable w.r.t. $\mathcal{T}$ and $\mathcal{I}$ a model of them with $a_0 \in C_0^{\mathcal{I}}$.

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

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

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

$$\begin{array}{ll} \text{if } C \in \mathcal{L}(x), \text{ then } \pi(x) \in C^{\mathcal{I}} & (*) \\ \text{if } y \text{ is an } R\text{-succ. of } x, \text{ then } \langle \pi(x), \pi(y) \rangle \in R^{\mathcal{I}} & \end{array}$$

- easy for $\sqcap$-, $\mathcal{T}$-, and $\forall$-rule,
- for $\exists$-rule, we need to extend π to the newly created R-successor
- for $\sqcup$-rule, if $C_1 \sqcup C_2 \in \mathcal{L}(x)$, $(*)$ implies that $\pi(x) \in (C_1 \sqcup C_2)^{\mathcal{I}}$
  $\rightsquigarrow$ we can choose C_i with $\pi(x) \in C_i^{\mathcal{I}}$ to add to $\mathcal{L}(x)$ and thus preserve $(*)$

$\rightsquigarrow$ easy to see: $(*)$ **implies that c-tree is clash-free**

Proof of the Lemma: Harvest

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

$\mathcal{I}$ is
- **finite** because c-tree has finitely many nodes
- but it is **not a tree** if blocking occurs

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

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

To obtain Corollary (3), the tree model property, we must work a bit more:
$\rightsquigarrow$ build the model in a different way, "unravel" the c-tree into an infinite tree intuitively, instead of going to a blocked node, go to a copy of its blocking node

A tableau algorithm for $\mathcal{ALC}$ with general TBoxes: Summary

The tableau algorithm presented here

- ➜ **decides** satisfiability of $\mathcal{ALC}$-concepts w.r.t. TBoxes, and thus also
- ➜ **decides** subsumption of $\mathcal{ALC}$-concepts w.r.t. TBoxes
- ➜ uses **blocking** to ensure termination, and
- ➜ is **non-deterministic** due to the $\longrightarrow_{\sqcup}$-rule
- ➜ in the worst case, it builds a tree of depth exponential in the size of the input, and thus of double exponential size. Hence it runs in (worst case) 2NExpTime,
- ➜ can be implemented in various ways,
  - order/priorities of rules
  - data structure
  - etc.
- ➜ is amenable to optimisations – more on this next week

What next?

Next, we could

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

Naive Implementations

Problems include:

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

Careful Choice of Algorithm

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

Dependency Directed Backtracking

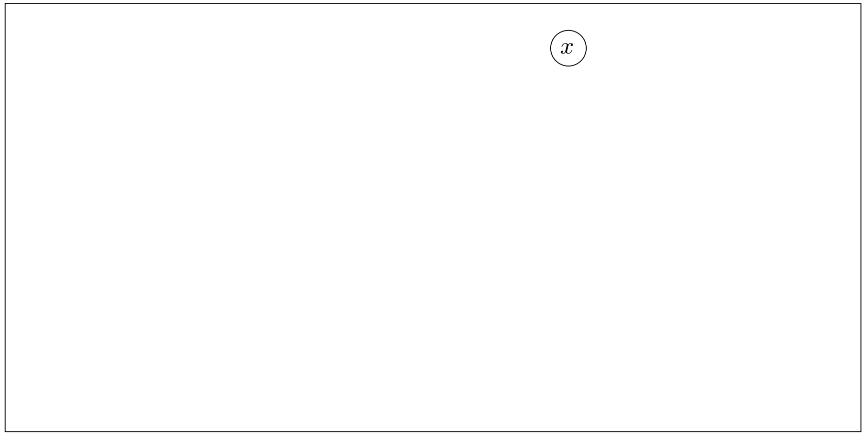
- ☞ Allows **rapid recovery** from bad branching choices
- ☞ Most commonly used technique is **backjumping**
  - Tag concepts introduced at **branch points** (e.g., when expanding disjunctions)
  - Expansion rules combine and **propagate tags**
  - On discovering a clash, **identify** most recently introduced 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 $\exists R.\neg A \sqcap \forall R.(A \sqcap B) \sqcap (C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n) \subseteq \mathcal{L}(x)$

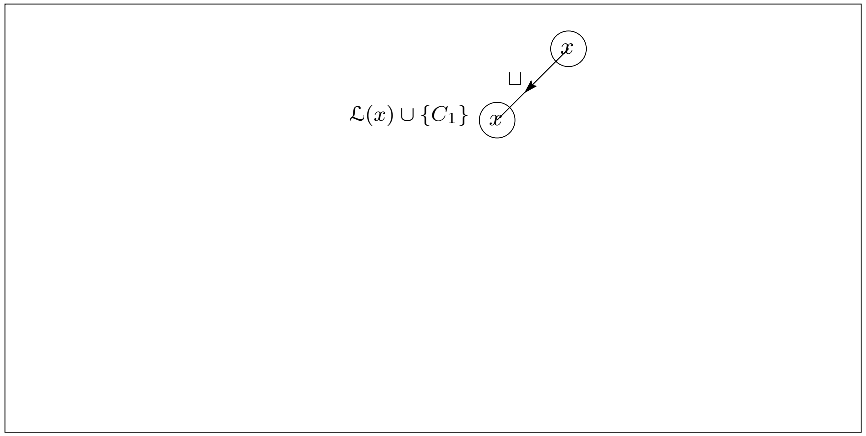
Backjumping

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



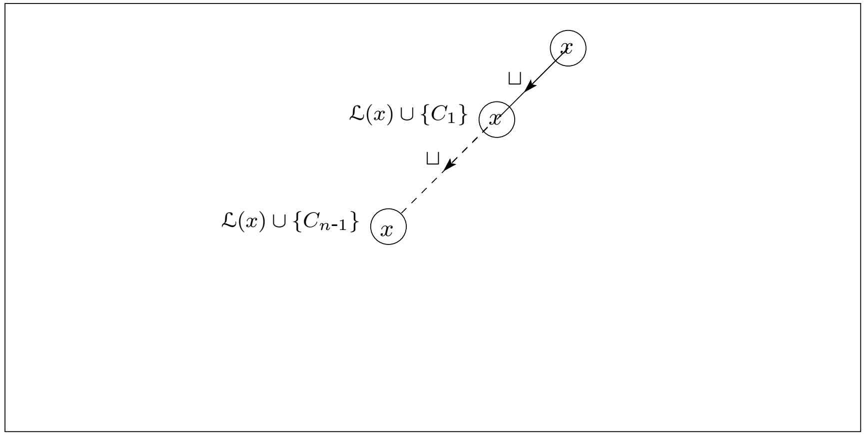
Backjumping

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



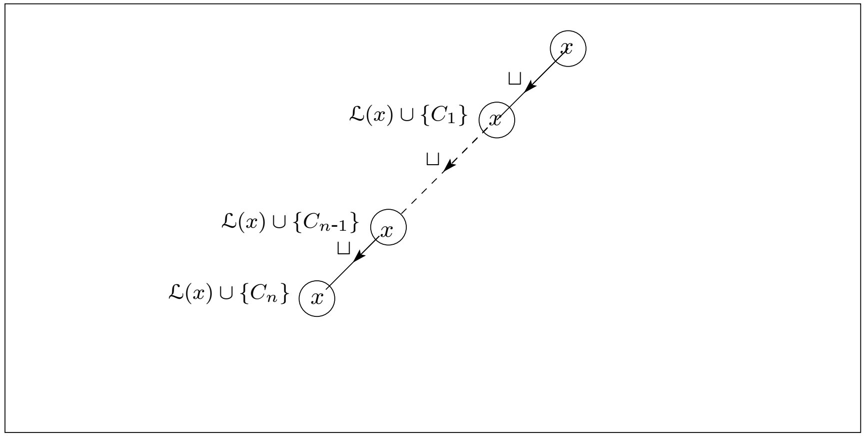
Backjumping

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



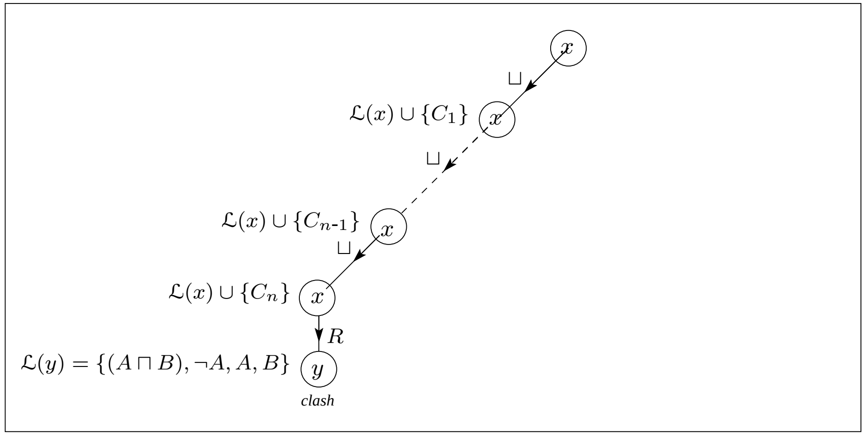
Backjumping

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



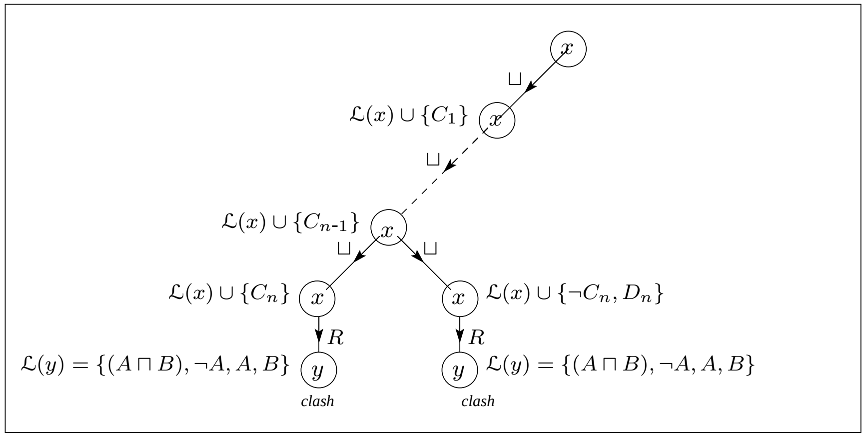
Backjumping

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



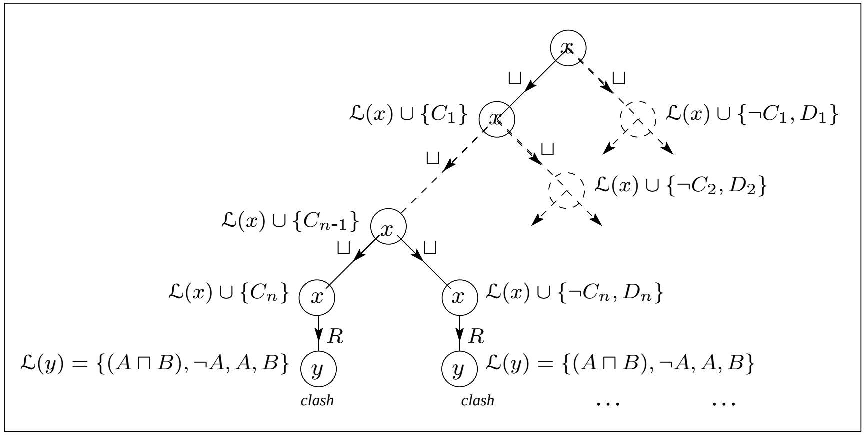
Backjumping

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



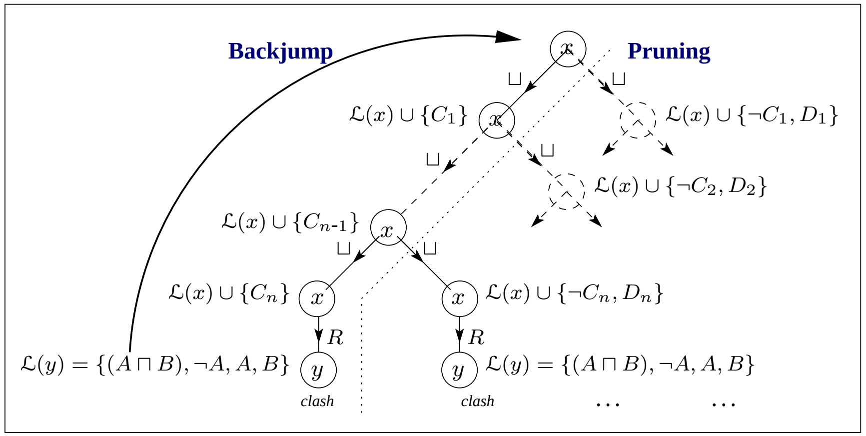
Backjumping

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

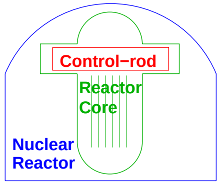


Backjumping

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



Inverse Roles



Consider the following TBox

$$\begin{array}{rl} \texttt{Control-rod} \ \dot{\sqsubseteq} & \texttt{Device} \sqcap \exists\texttt{part-of}.\texttt{Reactor-core} \\ \texttt{Reactor-core} \ \dot{\sqsubseteq} & \texttt{Device} \sqcap \exists\texttt{has-part}.\texttt{Control-rod} \sqcap \\ & \exists\texttt{part-of}.\texttt{N-reactor}, \\ \texttt{Reactor-core} \sqcap \exists\texttt{has\_part}.\texttt{Faulty} \ \dot{\sqsubseteq} & \texttt{Dangerous}, \end{array}$$

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

$\texttt{Control\_rod} \sqcap \texttt{Faulty}$ should be subsumed by $\exists\texttt{part-of}.\texttt{Dangerous}$

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

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

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

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

$$(R^-)^{\mathcal{I}} := \{\langle y, x\rangle \mid \langle x, y\rangle \in R^{\mathcal{I}}\}$$

Example: does $\forall$parent.$\forall$child.Blond $\sqsubseteq$ Blond w.r.t. $\{\top \ \dot{\sqsubseteq} \ \exists$parent.$\top\}$?

does $\forall$parent.$\forall$parent$^-$.Blond $\sqsubseteq$ Blond w.r.t. $\{\top \ \dot{\sqsubseteq} \ \exists$parent.$\top\}$?

Example: is $C_0 = \exists R.\exists S.\exists T.A$ satisf. w.r.t. $\{ C \ \dot{\sqsubseteq} \ \exists R.C \sqcap \forall R.B$, $\top \ \dot{\sqsubseteq} \ \forall T^-.\forall S^-.\forall R^-.C\}$?

Clear: inverse roles $\rightsquigarrow$ tableau algorithm must reason *up and down* edges

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

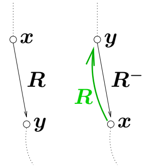
Modifications necessary to handle inverse roles:

① extend edge labels in c-trees to inverse roles,

② call y an R-**neighbour** of x if either

y is an R-successor of x or

x is an R^- successor of y,



③ substitute "R-successor" in the $\forall$- and $\exists$-rule with "R-neighbour"

still create an R-successor of x if no R-neighbour exists for $\exists R.C \in \mathcal{L}(x)$

R^--successor of x if no R^--neighbour exists for an $\exists R^-.C \in \mathcal{L}(x)$

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

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

$\sqcup$-rule: if $C_1 \sqcup C_2 \in \mathcal{L}(x)$, $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$, and x is not blocked
then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C\}$ for some $C \in \{C_1, C_2\}$

$\exists$-rule: if $\exists S.C \in \mathcal{L}(x)$, x has no S-neighbour y with $C \in \mathcal{L}(y)$,
and x is not blocked
then create a new node y with $\mathcal{L}(\langle x, y\rangle) = \{S\}$ and $\mathcal{L}(y) = \{C\}$

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

$\mathcal{T}$-rule: if $C_1 \ \dot{\sqsubseteq} \ C_2 \in \mathcal{T}$, NNF$(\neg C_1 \sqcup C_2) \notin \mathcal{L}(x)$
and x is not blocked
then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{\text{NNF}(\neg C_1 \sqcup C_2)\}$

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

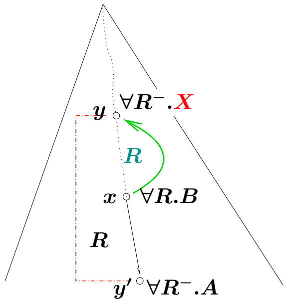
Example: is A satisfiable w.r.t. $\{A \dot{\sqsubseteq} \exists R^-.A \sqcap \forall R.(\neg A \sqcup \exists S.B)\}$?

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

Problem: algorithm returns *"satisfiable"* for unsatisfiable input $\rightsquigarrow$ **incorrect!**

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

Solution: tighten blocking condition to $\mathcal{L}(y') = \mathcal{L}(y)$



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

④ A node x is **directly blocked** if it has an ancestor y with $\mathcal{L}(x) = \mathcal{L}(y)$.

Lemma: Let $\mathcal{T}$ be a general $\mathcal{ALCI}$-Tbox and C_0 an $\mathcal{ALCI}$-concept. Then

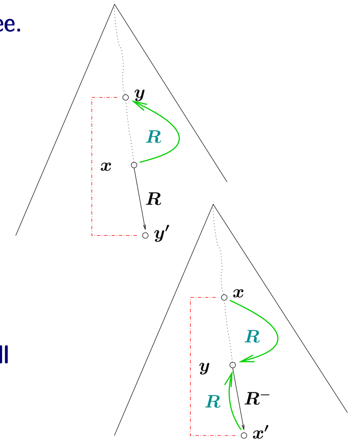
1. the algorithm terminates when applied to $\mathcal{T}$ and C_0,
2. the rules can be applied such that they generate a clash-free and complete completion tree iff C_0 is satisfiable w.r.t. $\mathcal{T}$.

Proof: (1) termination is identical to the $\mathcal{ALC}$ case.

Proof of the Lemma: Soundness

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

$$\begin{array}{lcl}
\Delta^{\mathcal{I}} & := & \{x \mid x \text{ is a node in } \mathcal{T},\ x \text{ is not blocked}\} \\
A^{\mathcal{I}} & := & \{x \in \Delta^{\mathcal{I}} \mid A \in \mathcal{L}(x)\} \text{ for concept names } A \\
R^{\mathcal{I}} & := & \{\langle x, y\rangle \in \Delta^{\mathcal{I}^2} \mid y \text{ is an } R\text{-succ of } x \text{ or} \\
 & & \quad y \text{ blocks an } R\text{-succ of } x \text{ or} \\
 & & \quad x \text{ is an } R^-\text{-succ of } y \text{ or} \\
 & & \quad x \text{ blocks an } R^-\text{-succ of } y\ \}
\end{array}$$



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

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

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

Proof of the Lemma: Completeness

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

That's it!

I hope you got an idea of how we can

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

Research Challenges

Challenges

- **Increased expressive power**
  - Existing DL systems implement (at most) $\mathcal{SHIQ}$
  - 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

- **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

- 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
- 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

- 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)

- 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

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

Summary

- **Description Logics** are family of logical KR formalisms
- **Applications** of DLs include DataBases and **Semantic Web**
  - Ontologies will provide vocabulary for semantic markup
  - OWL web ontology language based on $\mathcal{SHIQ}$ 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

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.