UNIKASSEL VERSITÄT

ENDOWED CHAIR OF THE HERTIE FOUNDATION
Knowledge and Data Engineering
ELECTRICAL ENGINEERING & COMPUTER SCIENCE, UNIVERSITY OF KASSEL

Vorlesung Künstliche Intelligenz Wintersemester 2007/08

Teil III: Wissensrepräsentation und Inferenz

Kap.10: Beschreibungslogiken

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

Beschreibungslogiken (Description Logics)

A family of logic based Knowledge Representation formalisms

- Descendants of semantic networks and KL-ONE
- Describe domain in terms of concepts (classes), roles (relationships) and individuals

Distinguished by:

- Formal semantics (typically model theoretic)
  - Decidable fragments of FOL
  - Closely related to Propositional Modal & Dynamic Logics
- Provision of inference services
  - Sound and complete decision procedures for key problems
  - Implemented systems (highly optimised)

- Einfache Sprache zum Start: $\mathcal{ALC}$ (Attributive Language with Complement)
- Im Semantic Web wird $\mathcal{SHOIN}(\mathbf{D}_n)$ eingesetzt. Hierauf basiert die Semantik von OWL DL.

Geschichte

- Ihre Entwicklung wurde inspiriert durch semantische Netze und Frames.
- Frühere Namen:
  - KL-ONE like languages
  - terminological logics

- Ziel war eine Wissensrepräsentation mit formaler Semantik.

- Das erste Beschreibungslogik-basierte System war KL-ONE (1985).
- Weitere Systeme u.a. LOOM (1987), BACK (1988), KRIS (1991), CLASSIC (1991), FaCT (1998), RACER (2001), KAON 2 (2005).

Literatur

- D. Nardi, R. J. Brachman. An Introduction to Description Logics. In: F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, P.F. Patel-Schneider (eds.): Description Logic Handbook, Cambridge University Press, 2002, 5-44.
- F. Baader, W. Nutt: Basic Description Logics. In: Description Logic Handbook, 47-100.
- Ian Horrocks, Peter F. Patel-Schneider and Frank van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. http://www.cs.man.ac.uk/%7Ehorrocks/Publications/download/2003/HoPH03a.pdf

Ontology/KR languages aim to model (part of) world

Terms in language correspond to entities in world

Meaning given by, e.g.:

- Mapping to another formalism, such as FOL, with own well defined semantics
- or a Model Theory (MT)

MT defines relationship between syntax and *interpretations*

- There can be many interpretations (models) of one piece of syntax
- Models supposed to be analogue of (part of) world
  - E.g., elements of model correspond to objects in world
- Formal relationship between syntax and models
  - Structure of models reflect relationships specified in syntax
- Inference (e.g., subsumption) defined in terms of MT
  - E.g., $\mathcal{T} \vDash A \sqsubseteq B$ iff in every model of $\mathcal{T}$, ext(A) $\subseteq$ ext(B)

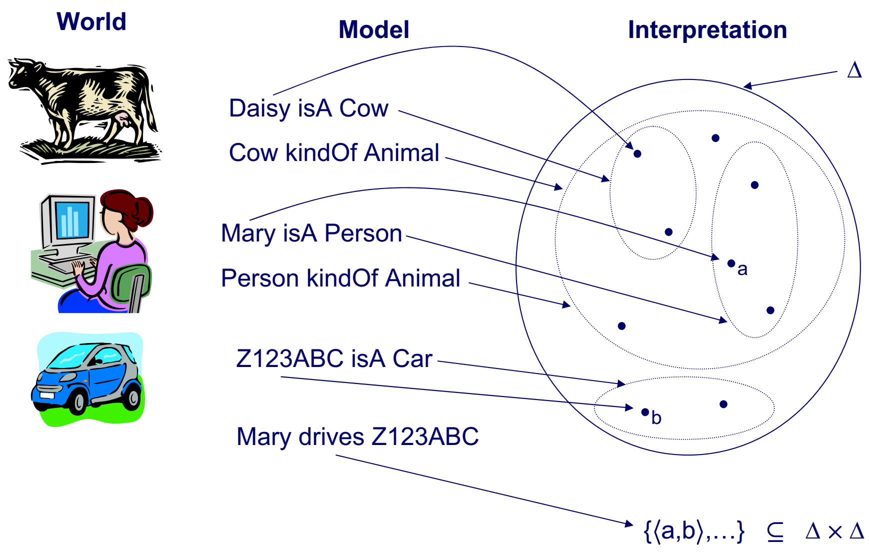
Many logics (including standard First Order Logic) use a model theory based on (Zermelo-Frankel) set theory

The domain of discourse (i.e., the part of the world being modelled) is represented as a set (often refered as Δ)

Objects in the world are interpreted as elements of Δ

- Classes/concepts (unary predicates) are subsets of Δ
- Properties/roles (binary predicates) are subsets of $\Delta \times \Delta$ (i.e., Δ^2)
- Ternary predicates are subsets of Δ^3 etc.

The sub-class relationship between classes can be interpreted as set inclusion.



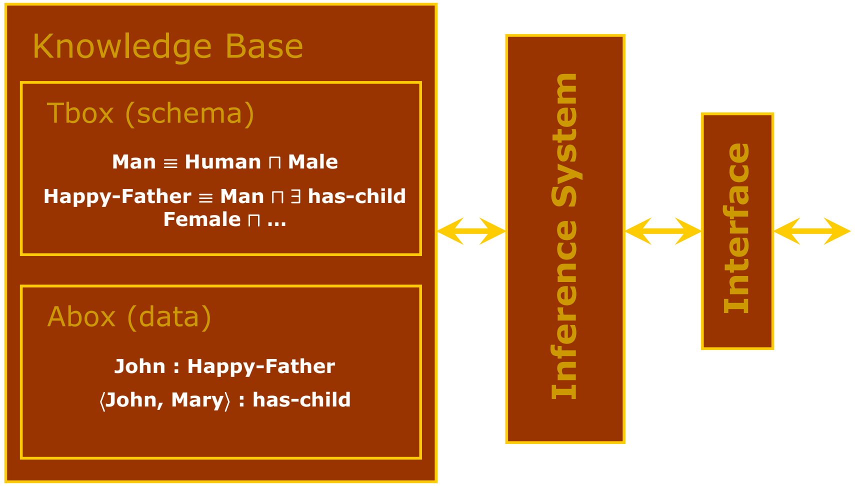
Formally, the vocabulary is the set of names we use in our model of (part of) the world

- {Daisy, Cow, Animal, Mary, Person, Z123ABC, Car, drives, …}

An interpretation $\mathcal{I}$ is a tuple $\langle \Delta, \cdot^{\mathcal{I}} \rangle$

- Δ is the domain (a set)
- $\cdot^{\mathcal{I}}$ is a mapping that maps
  - **Names of objects to elements of Δ**
  - **Names of unary predicates (classes/concepts) to subsets of Δ**
  - **Names of binary predicates (properties/roles) to subsets of $\Delta \times \Delta$**
  - **And so on for higher arity predicates (if any)**

DL Architecture



DL Knowledge Base

DL Knowledge Base (KB) normally separated into 2 parts:

- TBox is a set of axioms describing structure of domain (i.e., a conceptual schema), e.g.:
  - HappyFather ≡ Man ∧ ∃hasChild.Female ∧ …
  - Elephant ≡ Animal ∧ Large ∧ Grey
  - transitive(ancestor)

- ABox is a set of axioms describing a concrete situation (data), e.g.:
  - John:HappyFather
  - <John,Mary>:hasChild

Separation has no logical significance

- But may be conceptually and implementationally convenient

DL Semantics

Interpretation function $\cdot^{\mathcal{I}}$ extends to concept expressions in the obvious way, i.e.:

$$
\begin{aligned}
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
\{x\}^{\mathcal{I}} &= \{x^{\mathcal{I}}\} \\
(\exists R.C)^{\mathcal{I}} &= \{x \mid \exists y. \langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \\
(\forall R.C)^{\mathcal{I}} &= \{x \mid \forall y. (x, y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\} \\
(\leqslant n R)^{\mathcal{I}} &= \{x \mid \#\{y \mid \langle x, y\rangle \in R^{\mathcal{I}}\} \leqslant n\} \\
(\geqslant n R)^{\mathcal{I}} &= \{x \mid \#\{y \mid \langle x, y\rangle \in R^{\mathcal{I}}\} \geqslant n\}
\end{aligned}
$$

DL Knowledge Bases (Ontologies)

A DL Knowledge Base is of the form $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$

- $\mathcal{T}$ (Tbox) is a set of axioms of the form:
  - $C \sqsubseteq D$ (concept inclusion)
  - $C \equiv D$ (concept equivalence)
  - $R \sqsubseteq S$ (role inclusion)
  - $R \equiv S$ (role equivalence)
  - $R^{+} \sqsubseteq R$ (role transitivity)

- $\mathcal{A}$ (Abox) is a set of axioms of the form
  - $x \in D$ (concept instantiation)
  - $\langle x,y \rangle \in R$ (role instantiation)

Two sorts of Tbox axioms often distinguished

- “Definitions”
  - $C \sqsubseteq D$ or $C \equiv D$ where C is a concept name
- General Concept Inclusion axioms (GCIs)
  - $C \sqsubseteq D$ where C is an arbitrary concept

Knowledge Base Semantics

An interpretation $\mathcal{I}$ satisfies (models) an axiom A ($\mathcal{I} \models A$):

- $\mathcal{I} \models C \sqsubseteq D$ iff $C^\mathcal{I} \subseteq D^\mathcal{I}$
- $\mathcal{I} \models C \equiv D$ iff $C^\mathcal{I} = D^\mathcal{I}$
- $\mathcal{I} \models R \sqsubseteq S$ iff $R^\mathcal{I} \subseteq S^\mathcal{I}$
- $\mathcal{I} \models R \equiv S$ iff $R^\mathcal{I} = S^\mathcal{I}$
- $\mathcal{I} \models R^+ \sqsubseteq R$ iff $(R^\mathcal{I})^+ \subseteq R^\mathcal{I}$
- $\mathcal{I} \models x \in D$ iff $x^\mathcal{I} \in D^\mathcal{I}$
- $\mathcal{I} \models \langle x,y \rangle \in R$ iff $(x^\mathcal{I},y^\mathcal{I}) \in R^\mathcal{I}$

$\mathcal{I}$ satisfies a Tbox $\mathcal{T}$ ($\mathcal{I} \models \mathcal{T}$) iff $\mathcal{I}$ satisfies every axiom A in $\mathcal{T}$

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

$\mathcal{I}$ satisfies an KB $\mathcal{K}$ ($\mathcal{I} \models \mathcal{K}$) iff $\mathcal{I}$ satisfies both $\mathcal{T}$ and $\mathcal{A}$

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)

Inference Tasks

Knowledge is correct (captures intuitions)

- C subsumes D w.r.t. $\mathcal{K}$ iff for *every* model $\mathcal{I}$ of $\mathcal{K}$, $C^\mathcal{I} \subseteq D^\mathcal{I}$

Knowledge is minimally redundant (no unintended synonyms)

- C is equivalent to D w.r.t. $\mathcal{K}$ iff for *every* model $\mathcal{I}$ of $\mathcal{K}$, $C^\mathcal{I} = D^\mathcal{I}$

Knowledge is meaningful (classes can have instances)

- C is satisfiable w.r.t. $\mathcal{K}$ iff there exists *some* model $\mathcal{I}$ of $\mathcal{K}$ s.t. $C^\mathcal{I} \neq \emptyset$

Querying knowledge

- x is an instance of C w.r.t. $\mathcal{K}$ iff for *every* model $\mathcal{I}$ of $\mathcal{K}$, $x^\mathcal{I} \in C^\mathcal{I}$
- $\langle x,y \rangle$ is an instance of R w.r.t. $\mathcal{K}$ iff for, *every* model $\mathcal{I}$ of $\mathcal{K}$, $(x^\mathcal{I},y^\mathcal{I}) \in R^\mathcal{I}$

Knowledge base consistency

- A KB $\mathcal{K}$ is consistent iff there exists *some* model $\mathcal{I}$ of $\mathcal{K}$

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

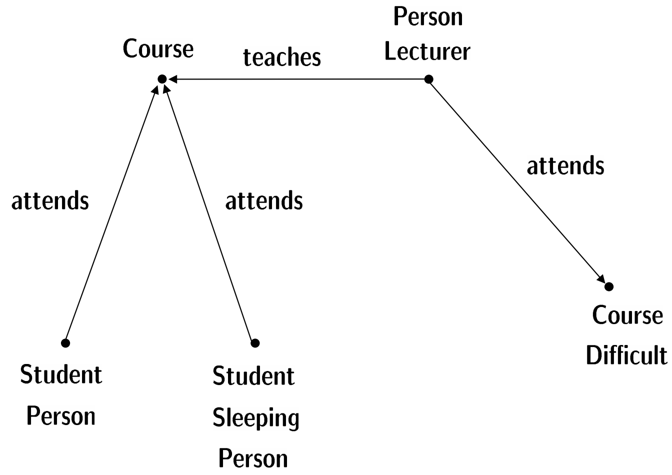
  each concept name A to a subset $A^\mathcal{I}$ of $\Delta^\mathcal{I}$ and

  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

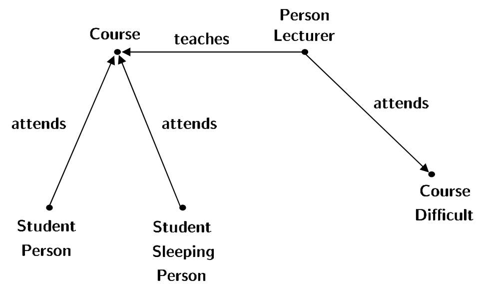


Semantics of Complex Concepts

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

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

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



Person ⊓ ∃attends.Course

Person ⊓ ∀attends.(¬Course ⊔ 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:

Woman $\doteq$ Person $\sqcap$ Female

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

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

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

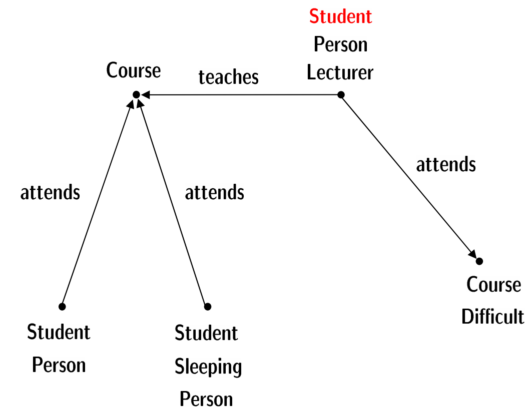
BadLecturer $\doteq$ Person $\sqcap$ $\forall$teaches.(Course $\rightarrow$ Boring)

TBox: Example II

A TBox restricts the set of admissible interpretations.

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

Student $\doteq$ Person $\sqcap$ $\exists$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:

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

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

Then

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

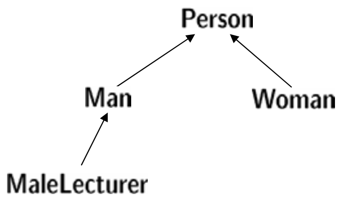
Reasoning Tasks — Classification

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

Woman $\doteq$ Person $\sqcap$ Female

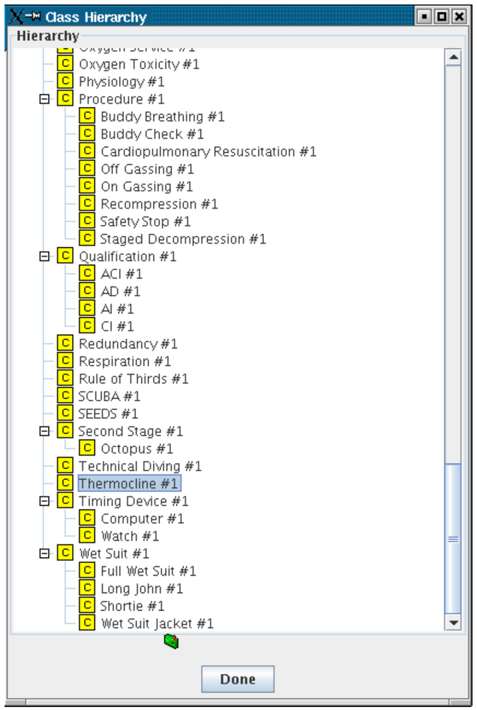
Man $\doteq$ Person $\sqcap$ $\neg$Woman

MaleLecturer $\doteq$ Man $\sqcap$ $\exists$teaches.Course



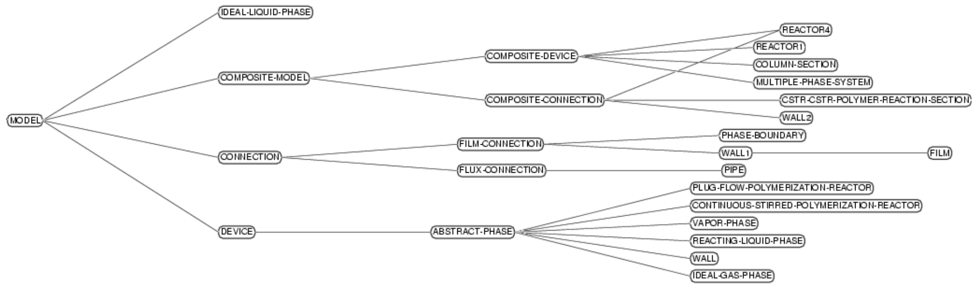
Can be computed using multiple subsumption tests

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

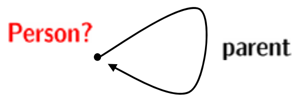
- the primitive concept names in $\mathcal{T}$
- all role names

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 $\doteq$ ∃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 ⊓ ∃teaches.Course~~

~~Course $\doteq$ ∃has-title.Title ⊓ ∃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 ⊓ ∀attended-by.Sleeping ⊑ Boring

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

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

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

e.g. Student ⊓ ∃has-favourite.SoccerTeam $\doteq$ Student ⊓ ∃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

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

Example for ABox Reasoning

ABox

dumbo : Mammal

~~g23 : Darkgrey~~

(dumbo, g23) : color

dumbo : ∀color.Lightgrey

t14 : Trunk

(dumbo, t14) : bodypart

TBox

Elephant $\doteq$ Mammal ⊓ ∃bodypart.Trunk ⊓ ∀color.Grey

Grey $\doteq$ Lightgrey ⊔ Darkgrey

$\bot \doteq$ Lightgrey ⊓ Darkgrey

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

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):

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

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 $\Delta^{\mathcal{I}}$
  $\leadsto$ 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
  $\leadsto$ 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.

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

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. $\mathcal{T}$.

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 $\mathcal{L}(x) = \mathcal{L}(x) \cup \{\mathsf{NNF}(\neg C_1 \sqcup C_2)\}$

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

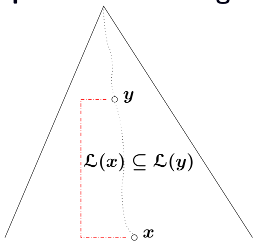
Tableau algorithm no longer terminates!

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



- 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**

$\sqcap$-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 $\mathcal{L}(y) = \{C\}$

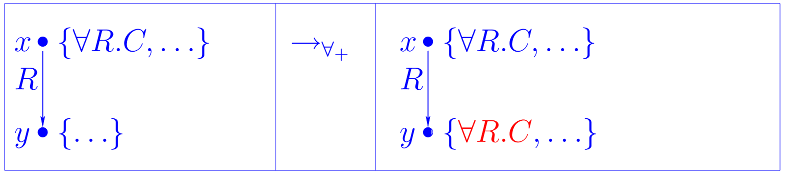
$\forall$-rule: if $\forall S.C \in \mathcal{L}(x)$, there is an S-successor y of x with $C \notin \mathcal{L}(y)$
and x is not blocked
then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$

$\mathcal{T}$-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 $\mathcal{ALC}$

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

Tableaux Rule for Transitive Roles



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

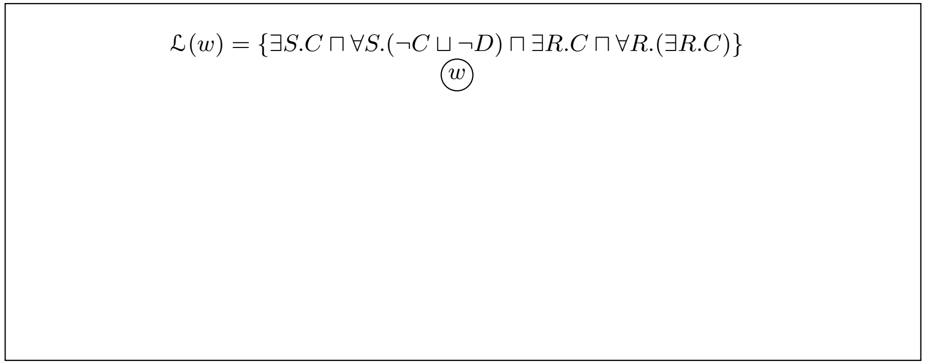
- No longer naturally terminating (e.g., if $C = \exists R.\top$)
- Need blocking
  - Simple blocking suffices for $\mathcal{ALC}$ plus transitive roles
  - I.e., do not expand node label if ancestor has superset label
  - More expressive logics (e.g., with inverse roles) need more sophisticated blocking strategies

Tableaux Algorithm — Example

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

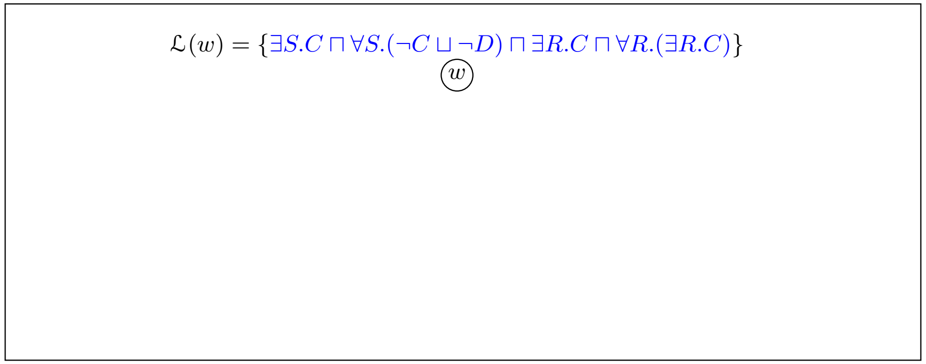
Tableaux Algorithm — Example

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



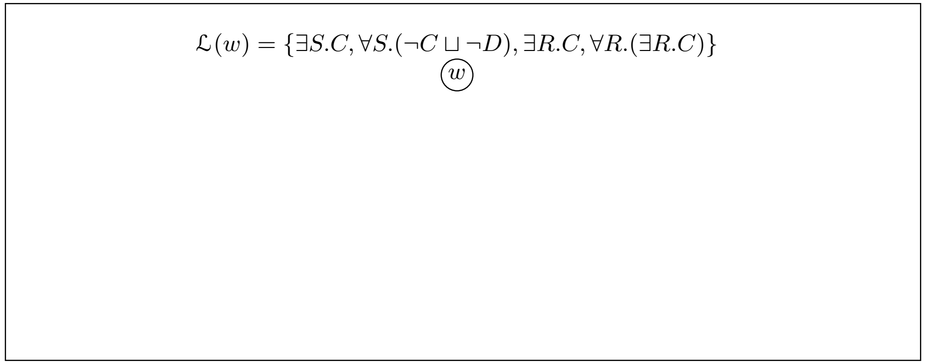
Tableaux Algorithm — Example

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



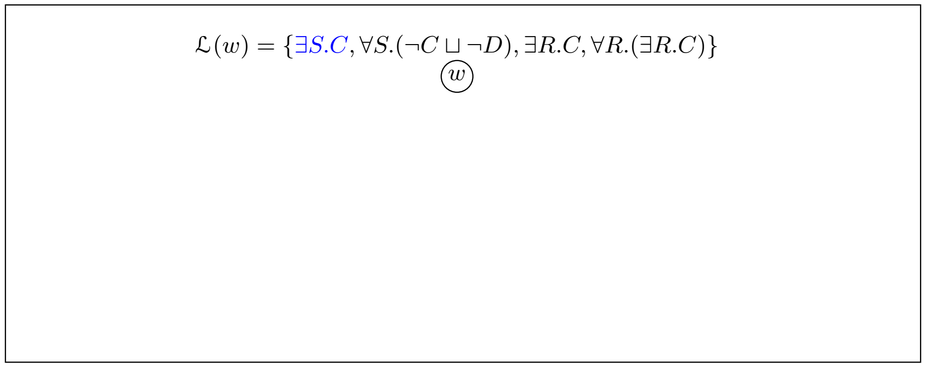
Tableaux Algorithm — Example

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



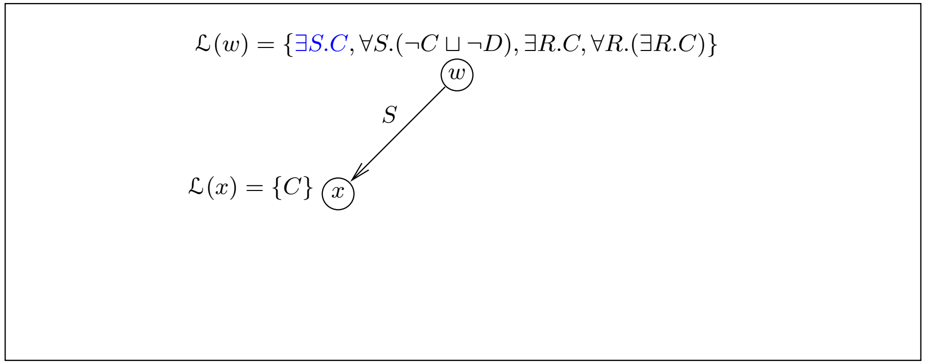
Tableaux Algorithm — Example

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



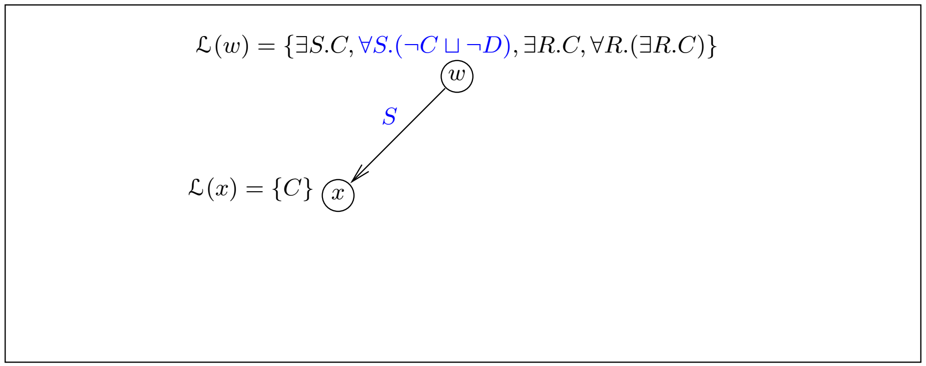
Tableaux Algorithm — Example

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



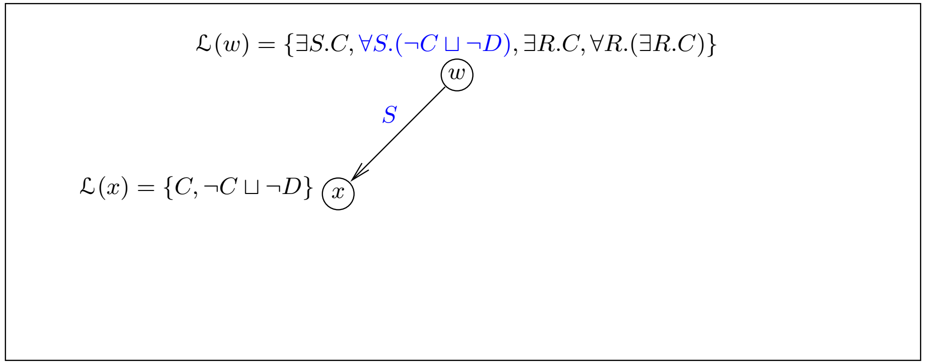
Tableaux Algorithm — Example

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



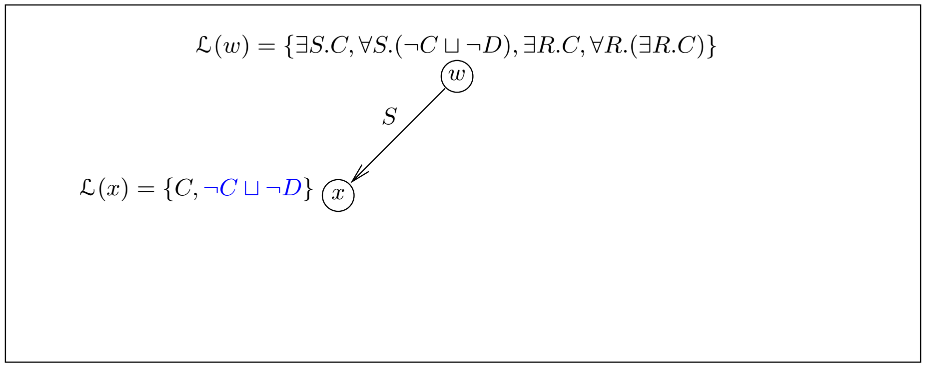
Tableaux Algorithm — Example

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



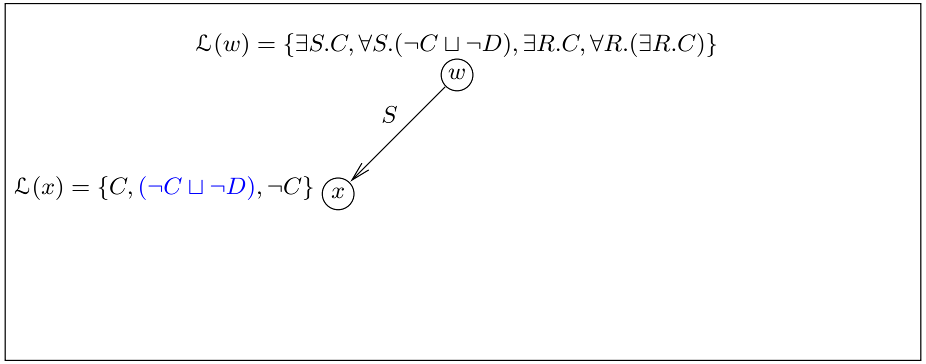
Tableaux Algorithm — Example

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



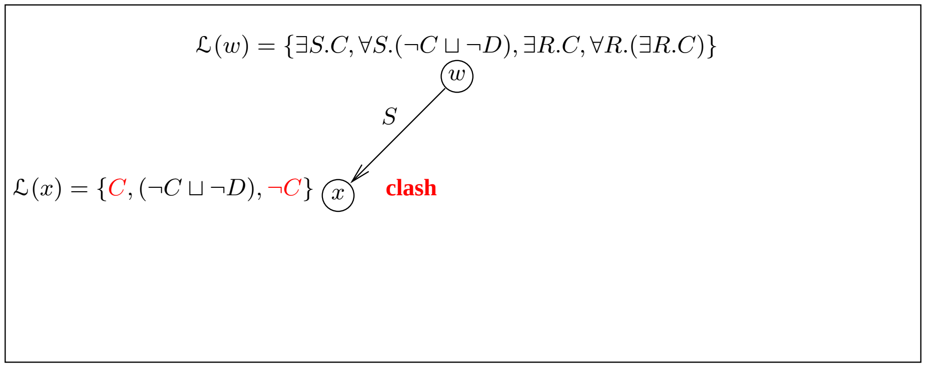
Tableaux Algorithm — Example

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



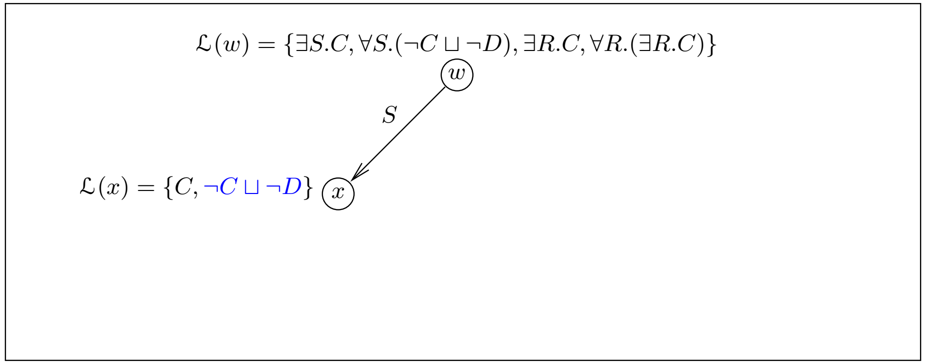
Tableaux Algorithm — Example

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



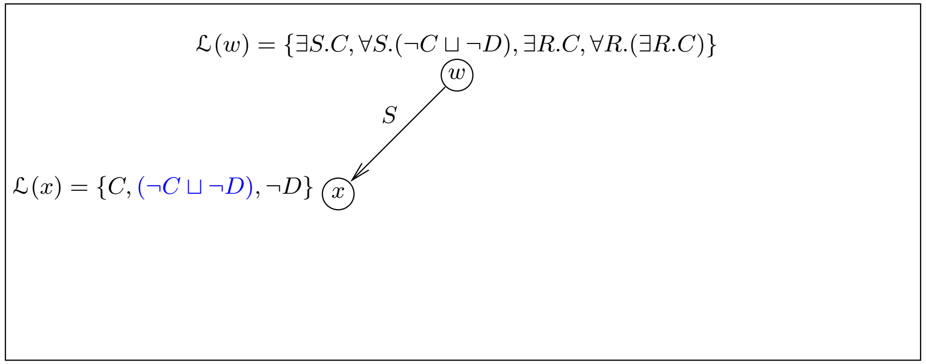
Tableaux Algorithm — Example

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



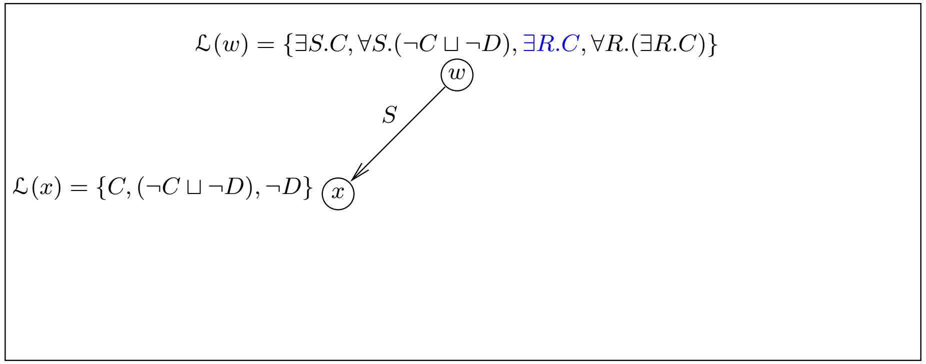
Tableaux Algorithm — Example

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



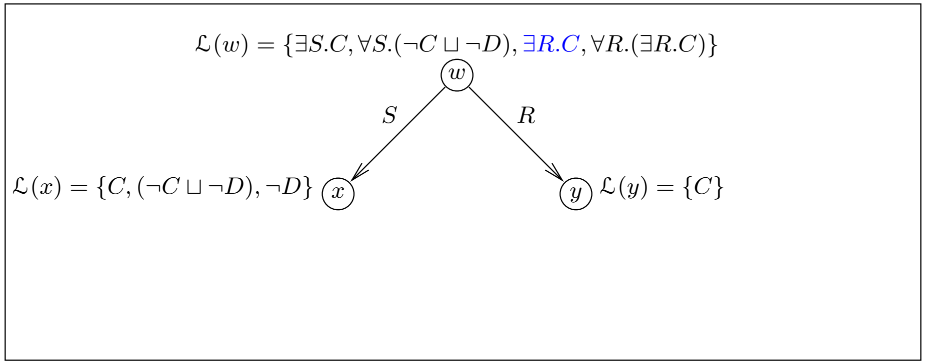
Tableaux Algorithm — Example

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



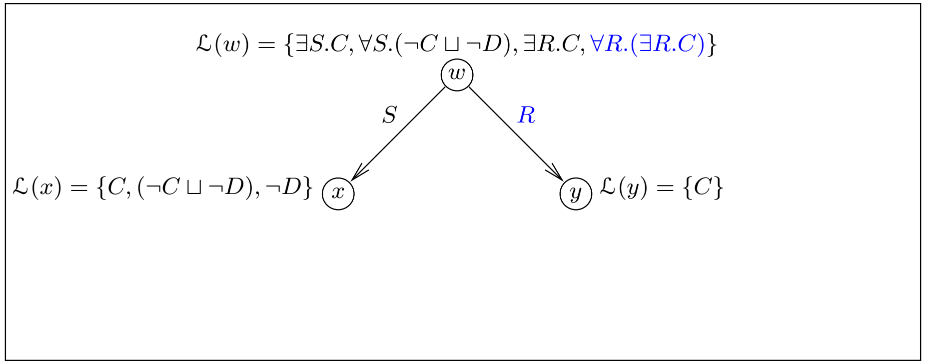
Tableaux Algorithm — Example

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



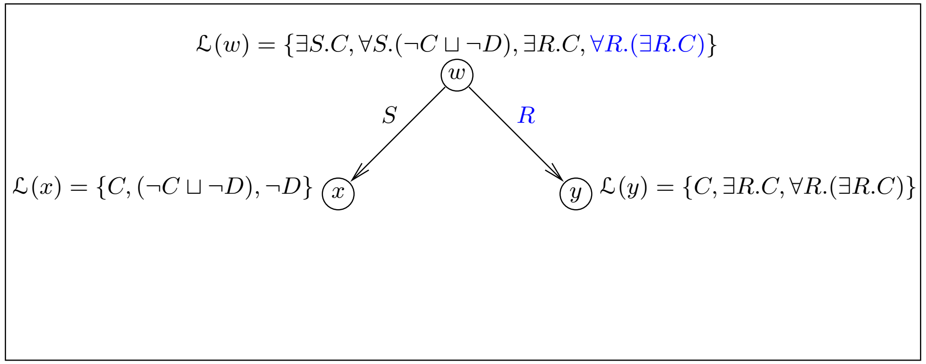
Tableaux Algorithm — Example

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



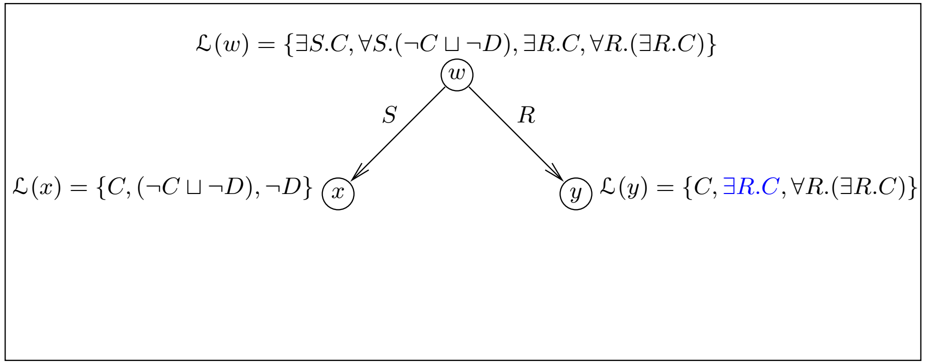
Tableaux Algorithm — Example

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



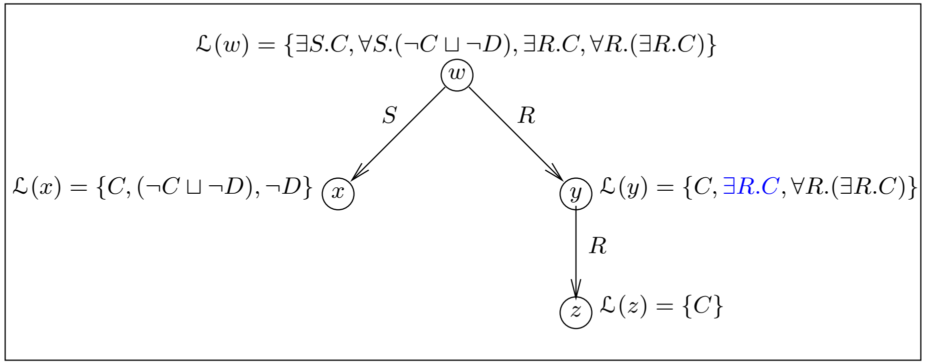
Tableaux Algorithm — Example

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



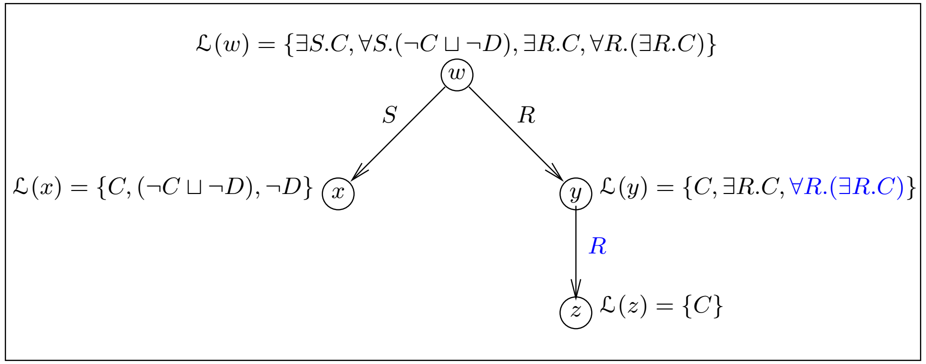
Tableaux Algorithm — Example

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



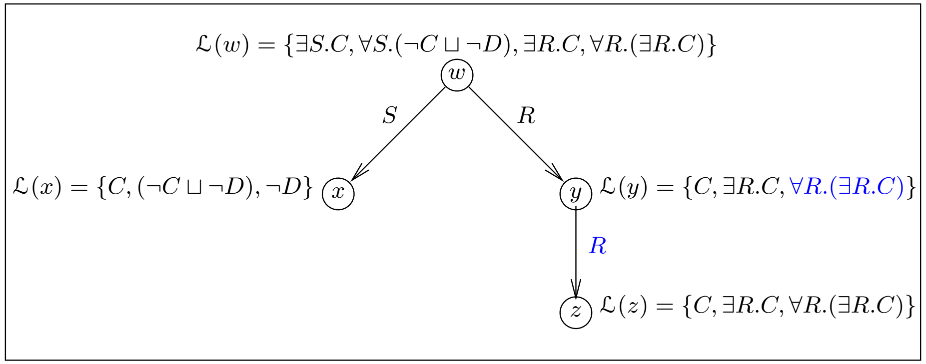
Tableaux Algorithm — Example

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



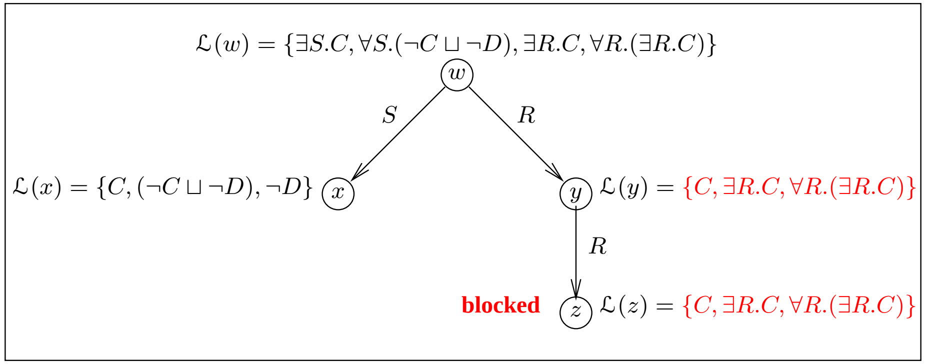
Tableaux Algorithm — Example

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



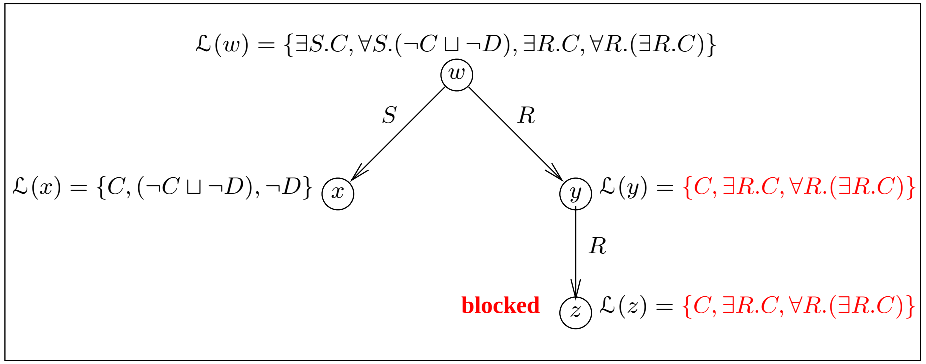
Tableaux Algorithm — Example

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



Tableaux Algorithm — Example

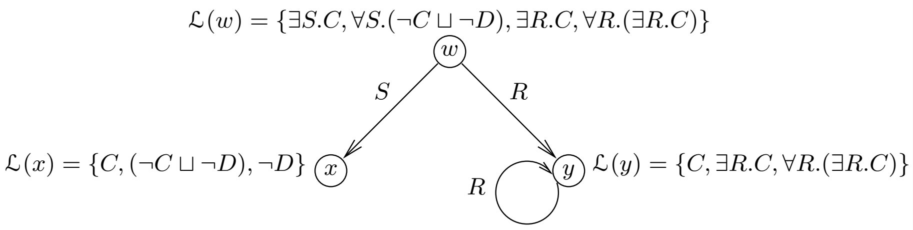
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



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

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 $\rightarrow_{\sqcup}$-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

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

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