

Description Logics

Carsten Lutz

TU Dresden, Germany

Ulrike Sattler

Univ. of Manchester, UK



Knowledge Representation

General goal of **knowledge representation**:

"develop formalisms for providing high-level descriptions of the world
that can be effectively used to build intelligent applications."

- "formalisms": formal syntax and formal and unambiguous semantics
- "high-level descriptions": which aspects should be represented, which left out?
- "intelligent applications": are able to infer new knowledge from given knowledge
- "effectively used": reasoning techniques should allow "usable" implementation



Terminological Knowledge

DLs focus on the representation of terminological knowledge:

formalize the basic terminology adopted in the modeled application

- terminologies are formalized as a collection of **concepts** and **relations**,
e.g. "Course" and "Lecturer" / "gives-course" and "attends-course"
- definition of concepts and relations stored in an **ontology**

Many applications need to reason about terminological knowledge, e.g.:

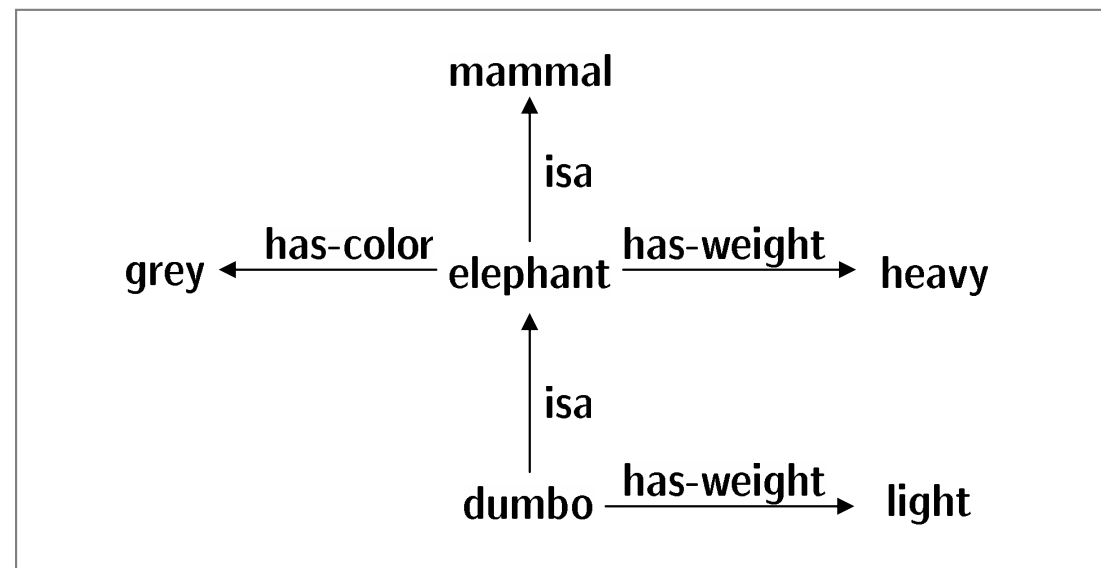
- **Bioinformatics**
e.g. the GeneOntology (GO): some 17.000 concepts about gene products
- **Semantic Web**
goal: provide a semantic description of the **content** of web pages
this is done by referring to concepts defined in an ontology



Early Formalisms

How to represent terminological knowledge?

Early days of AI: KR through obscure pictures (**semantic networks**)



Problems: missing semantics (reasoning!), complex pictures

Remedy: Use a logical formalism for KR rather than pictures



DL's answers to the challenges of knowledge representation:

- "formalisms": formal syntax and formal and unambiguous semantics
DLs have this, they are logics
- "high-level descriptions": which aspects should be represented, which left out?
DLs concentrate on terminological knowledge / ontologies
- "intelligent applications": are able to infer new knowledge from given knowledge
DLs come with reasoning algorithms and are decidable
- "effectively used": reasoning techniques should allow "usable" implementation
DLs have been implemented in very efficient systems



Representing Terminological Knowledge with DLs

The core part of any DL is the **concept language**

Mammal $\sqcap$ $\exists$ bodypart.Trunk $\sqcap$ $\forall$ color.Grey

- **concept names** assign a name to groups of objects
- **role names** assign a name to relations between objects
- **constructors** allow to related concept names and role names

Complex concepts can be used in **concept definitions**:

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



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

$\mathcal{ALC}$ is the smallest propositionally closed Description Logic.

Atomic types: concept names $A, B, \dots$ (unary predicates)
role names $R, S, \dots$ (binary predicates)

Constructors:

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

For example: $\neg(A \sqcup \exists R.(\forall S.B \sqcap \neg A))$

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



Semantics of $\mathcal{ALC}$

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

- each concept name A to a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$.
- each role name R to a binary relation $R^{\mathcal{I}}$ over $\Delta^{\mathcal{I}}$.

Semantics of complex concepts:

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

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

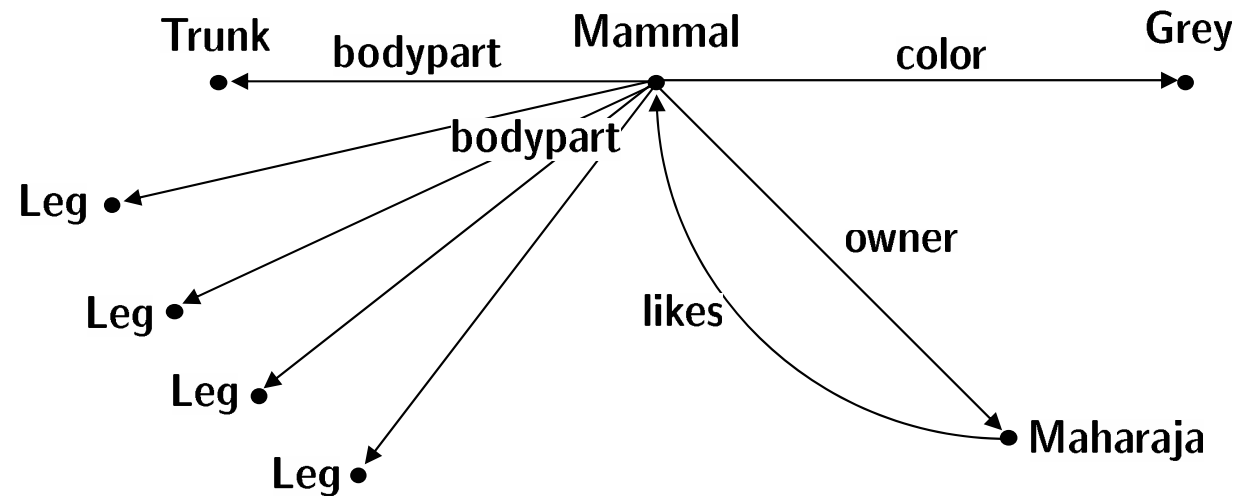
$$(\forall R.C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} \mid (d, e) \in R^{\mathcal{I}} \text{ implies } e \in C^{\mathcal{I}}\}$$

An interpretation $\mathcal{I}$ is a **model** for a concept C if $C^{\mathcal{I}} \neq \emptyset$.



Example

Intuition: interpretation is **complete** description of the world



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

$$\text{Mammal} \sqcap \forall \text{bodypart} . (\text{Trunk} \sqcup \text{Leg})$$


Reasoning Tasks

Most important reasoning tasks for constructing and using ontologies:

1. **Concept satisfiability** — does there exist a model of C ?

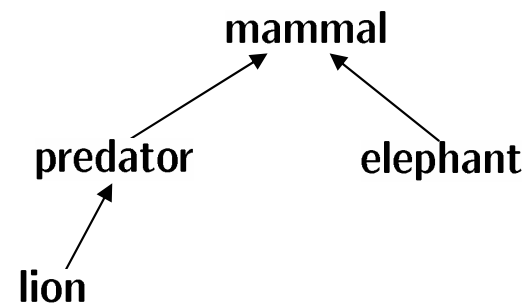
If unsatisfiable, the concept contains a contradiction.

2. **Concept subsumption** — does $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ hold for all $\mathcal{I}$?

written $C \sqsubseteq D$

If $C \sqsubseteq D$, then D is **more general** than C

$\Rightarrow$ Subsumption can be used to compute a concept hierarchy:



Examples

● $\forall \text{owner.Rich} \sqcap \forall \text{owner.Famous} \sqsubseteq \forall \text{owner.}(\text{Rich} \sqcap \text{Famous})$

● Set $\top := A \sqcup \neg A$. Then $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$ for all $\mathcal{I}$.
Thus $C \sqsubseteq \top$ for all C .

● Set $\perp := A \sqcap \neg A$. Then $\perp^{\mathcal{I}} = \emptyset$ for all $\mathcal{I}$.
Thus $\perp \sqsubseteq C$ for all C .

● $C \sqsubseteq D$ if and only if $C \sqcap \neg D$ is not satisfiable

● C is satisfiable if not $C \sqsubseteq \perp$.

⇒ subsumption can be reduced to (un)satisfiability and vice versa.



Expressive Power vs. Computational Complexity

In many cases, the expressive power of $\mathcal{ALC}$ does not suffice:

- an elephant has precisely four bodyparts that are legs
- every elephant **has a bodypart** which is a trunk
and every trunk **is a bodypart** of an elephant

Many extensions of $\mathcal{ALC}$ have been developed, for example:

- qualified number restrictions ($\leq n R C$) and ($\geq n R C$)
- inverse roles R^- to be used in existential and universal restriction

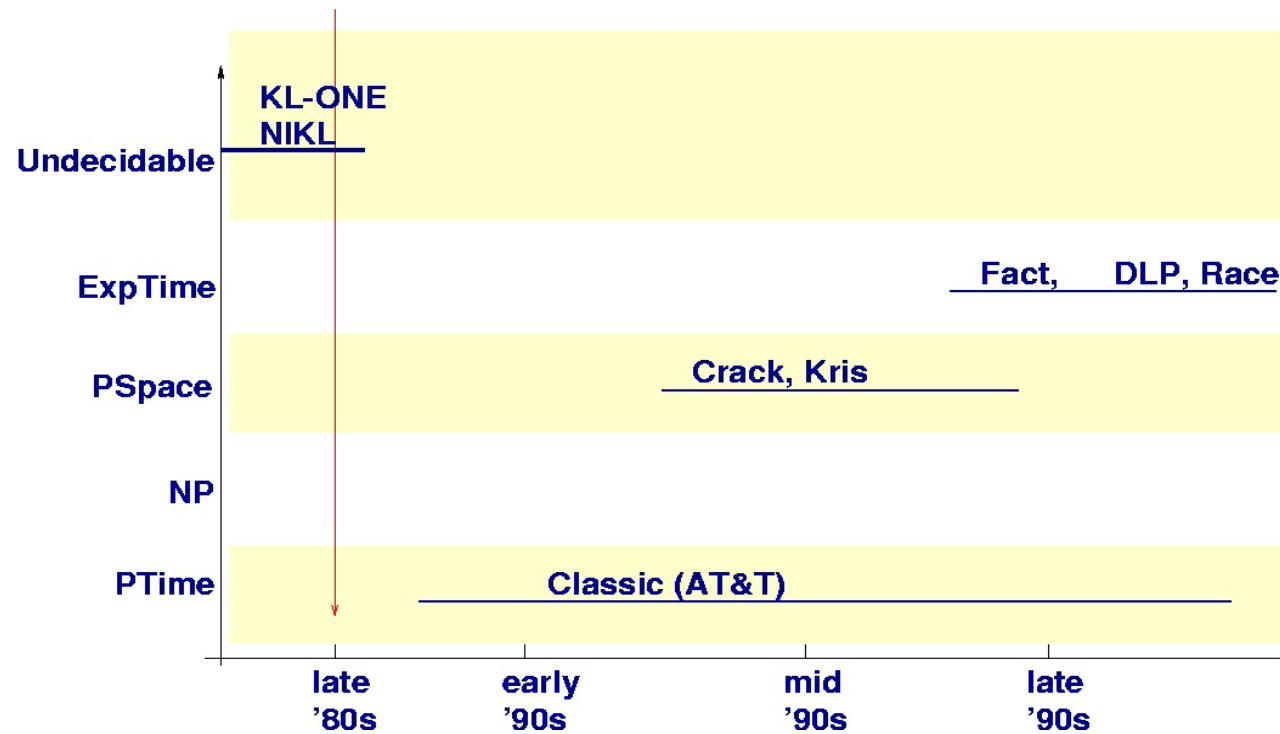
But: Increasing expressivity also increases computational complexity

$\Rightarrow$ **!! tradeoff between expressivity and computational complexity !!**

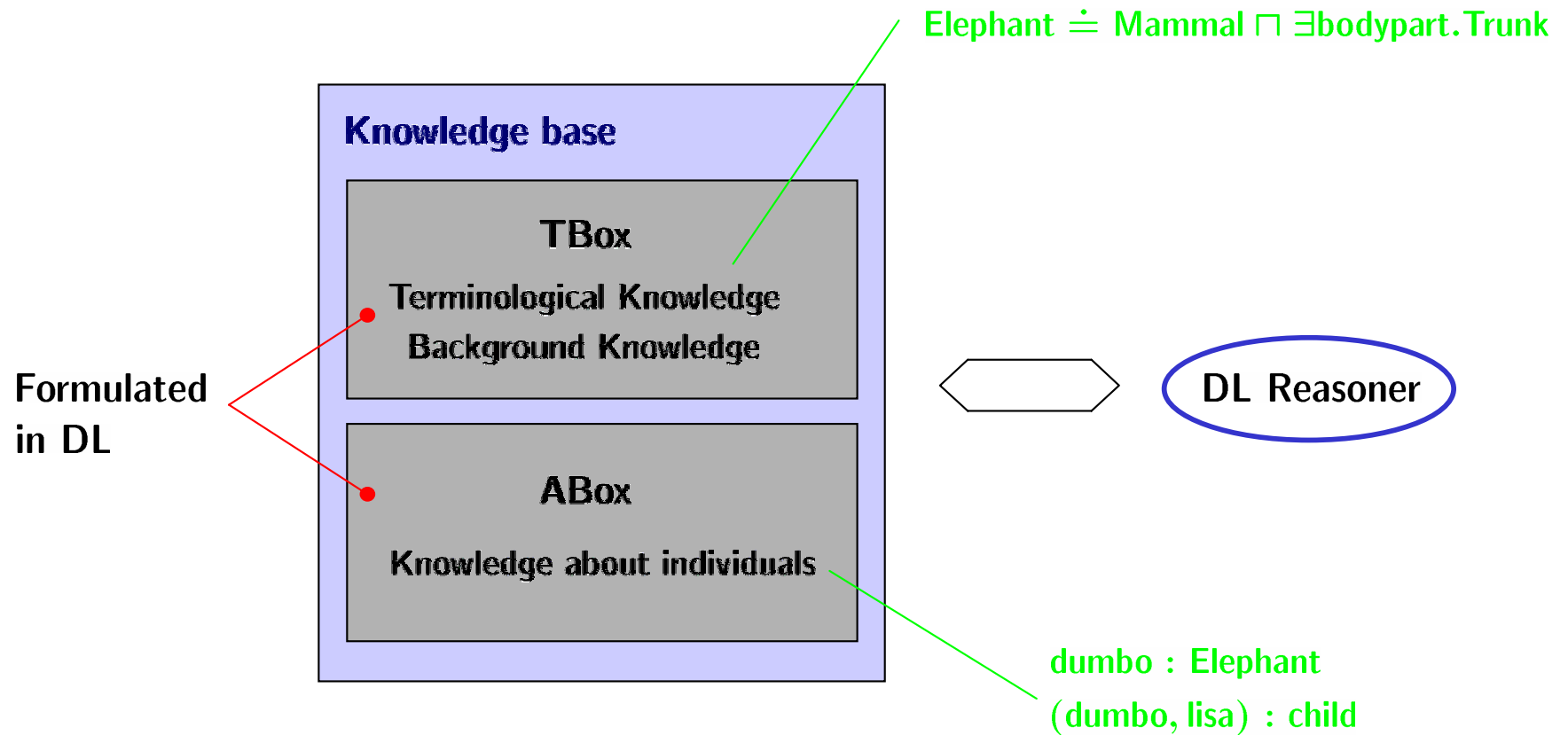


Development of DL Systems

Description Logics should be decidable. But what complexity is “ok”?



DLs are more than a Concept Language



General TBoxes

There exist several kinds of TBoxes.

General TBox: finite set of concept equations $C \doteq D$

e.g. $\{\text{WildAnimal} \doteq \text{Animal} \sqcap \neg \exists \text{owner}.\top$

$\text{Mammal} \sqcap \exists \text{bodypart.Hunch} \doteq \text{Camel} \sqcup \text{Dromedary}\}$

An interpretation $\mathcal{I}$ is a **model** of a TBox $\mathcal{T}$ if

$$C^{\mathcal{I}} = D^{\mathcal{I}} \text{ for all } C \doteq D \in \mathcal{T}.$$

Reasoning tasks with TBoxes:

1. **Concept satisfiability w.r.t. TBoxes**

Given C and $\mathcal{T}$, does there exist a common model of C and $\mathcal{T}$?

2. **Concept subsumption w.r.t. TBoxes** ($C \sqsubseteq_{\mathcal{T}} D$)

Given C, D , and $\mathcal{T}$, does $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ hold in all models of $\mathcal{T}$?



Acyclic TBoxes

Concept definition: expression $A \doteq C$ with A concept **name** and C concept

e.g. $\text{Elephant} \doteq \text{Mammal} \sqcap \exists \text{bodypart}.\text{Trunk}$

A finite set $\mathcal{T}$ of concept definitions is an **acyclic TBox** if

- a) the left-hand sides of concept definitions in $\mathcal{T}$ are unique
- b) it contains no "cycles"

not an acyclic TBox: ~~$\{\text{Elephant} \doteq \text{Mammal} \sqcap \exists \text{bodypart}.\text{Trunk}$
 $\text{Mammal} \doteq \text{Elephant} \sqcup \text{Lion} \sqcup \text{Zebra}\}$~~

Acyclic TBoxes can be conceived as macro definitions.



ABoxes

Fix a set of **individual names**.

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. {dumbo : Elephant, (dumbo, lisa) : child}

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 $\mathcal{A}$.



Reasoning tasks with ABoxes:

1. **ABox consistency**

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

2. **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 $\mathcal{A}, \mathcal{T} \models a : C$)

Instance checking can be reduced to ABox consistency.

Concept satisfiability can be reduced to ABox consistency.



Example

ABox

dumbo : Elephant

t14 : Trunk

~~g23 : Darkgrey~~

(dumbo, t14) : bodypart

~~(dumbo, g23) : color~~

dumbo : $\forall \text{color}.$ Lightgrey

TBox

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

Grey $\doteq$ Lightgrey $\sqcup$ Darkgrey

$\perp \doteq$ Lightgrey $\sqcap$ Darkgrey

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



Description Logics and First-order Logic

concept names A	$\iff$	unary predicates P_A
role names R	$\iff$	binary predicates P_R
concepts	$\iff$	formulas with one free variable

$$\begin{aligned}
 \varphi^x(A) &= P_A(x) \\
 \varphi^x(\neg C) &= \neg \varphi^x(C) \\
 \varphi^x(C \sqcap D) &= \varphi^x(C) \wedge \varphi^x(D) \\
 \varphi^x(C \sqcup D) &= \varphi^x(C) \vee \varphi^x(D) \\
 \varphi^x(\exists R.C) &= \exists y. P_R(x, y) \wedge \varphi^y(C) \\
 \varphi^x(\forall R.C) &= \forall y. P_R(x, y) \rightarrow \varphi^y(C)
 \end{aligned}$$

φ^y symmetric
with x and y exchanged

- Note:**
- two variables suffices (no "=", no constants, no function symbols)
 - formulas obtained by translation have "guarded" structure
 - not all DLs are purely first-order (transitive closure, etc.)



TBoxes:

Let C be a concept and $\mathcal{T}$ a (general or acyclic) TBox.

$$\varphi(\mathcal{T}) = \forall x. \bigwedge_{D \dot{=} E \in \mathcal{T}} \varphi^x(D) \leftrightarrow \varphi^x(E)$$

ABoxes:

individual names a $\longleftrightarrow$ constants c_a

$$\varphi(a : C) = \varphi^x(C)[c_a]$$

$$\varphi((a, b) : R) = P_R(c_a, c_b)$$

$$\varphi(\mathcal{A}) = \bigwedge_{\beta \in \mathcal{A}} \varphi(\beta)$$



Description Logics and Modal Logics

Obvious translation:

concept names	$\longleftrightarrow$	propositional variables
role names	$\longleftrightarrow$	modal parameters
concepts $\exists R.C$	$\longleftrightarrow$	formulas $\Diamond\psi$
concepts $\forall R.C$	$\longleftrightarrow$	formulas $\Box\psi$

- Notes:
- Interpretations can be viewed as Kripke structures
 - $\mathcal{ALC}$ is a notational variant of modal K_ω
 - TBoxes related to universal modality: $\Box_u \bigwedge_{D \dot{=} E \in \mathcal{T}} D \leftrightarrow E$
 - ABoxes related to nominals / hybrid modal logic
 - Extensions of $\mathcal{ALC}$ are related to graded modal logic, PDL, etc.



Overview of the Course

- Tableau Algorithm for $\mathcal{ALC}$
- Tableau algorithms for expressive Description Logics
- Automata-based decision procedures for expressive Description Logics
- Computational complexity
- Applications, System demonstration, other topics of DL research



Goal

Find algorithm for satisfiability and subsumption
(for now: only concept, no TBoxes, no ABoxes)

Subsumption can be reduced to (un)satisfiability

$\Rightarrow$ we can restrict ourselves to satisfiability

Tableau Algorithms

- Frequently used to prove decidability/complexity bounds of DLs
- All state-of-the-art DL reasoners are based on tableau algorithms



Tableau Algorithms: Idea

Try to construct a model for the input concept C_0 as follows:

- Represent models by **completion trees**
- To decide satisfiability of C_0 , start with initial completion tree T_{C_0}
- Repeatedly apply completion rules and check for obvious contradictions
- Return “satisfiable” iff a complete and contradiction-free completion tree was found



Negation Normal Form

A concept C is in **negation normal form (NNF)** if

negation occurs only in front of concept names.

Transformation rules:

$$\neg\neg C \rightsquigarrow C$$

$$\neg(C \sqcap D) \rightsquigarrow \neg C \sqcup \neg D$$

$$\neg(C \sqcup D) \rightsquigarrow \neg C \sqcap \neg D$$

$$\neg(\exists R.C) \rightsquigarrow \forall R.\neg C$$

$$\neg(\forall R.C) \rightsquigarrow \exists R.\neg C$$



Completion Tree

Completion tree:

Finite tree $T = (V, E, \mathcal{L})$ where $\mathcal{L}$ labels

- each node $x \in V$ with a set $\mathcal{L}(x) \subseteq \text{sub}(C_0)$
- each edge $(x, y) \in E$ with a role $\mathcal{L}(x, y)$ occurring in C_0 .

Initial completion tree for concept C_0 : $(\{x_0\}, \emptyset, \mathcal{L})$ where $\mathcal{L}(x_0) = \{C_0\}$

Apply completion rules until

- the completion tree is **complete**.

or - there exists a node $x \in V$ such that

$\{A, \neg A\} \subseteq \mathcal{L}(x)$ for some concept name A (**Clash**)



Completion Rules I

$x \bullet \{C_1 \sqcap C_2, \dots\}$	$\rightarrow_{\sqcap}$	$x \bullet \{C_1 \sqcap C_2, \mathbf{C}_1, \mathbf{C}_2, \dots\}$
$x \bullet \{C_1 \sqcup C_2, \dots\}$	$\rightarrow_{\sqcup}$	$x \bullet \{C_1 \sqcup C_2, \mathbf{C}, \dots\}$ for $C \in \{C_1, C_2\}$
$x \bullet \{\exists R.C, \dots\}$	$\rightarrow_{\exists}$	$x \bullet \{\exists R.C, \dots\}$ $\mathbf{R}$ $\downarrow$ $y \bullet \{\mathbf{C}\}$
$x \bullet \{\forall R.C, \dots\}$ $\mathbf{R}$ $\downarrow$ $y \bullet \{\dots\}$	$\rightarrow_{\forall}$	$x \bullet \{\forall R.C, \dots\}$ $\mathbf{R}$ $\downarrow$ $y \bullet \{\dots, \mathbf{C}\}$



Lemma

1. The algorithm terminates on any input
2. If the algorithm returns “satisfiable”, then the input concept has a model.
3. If the input concept has a model, then the algorithm returns “satisfiable”.

Corollary

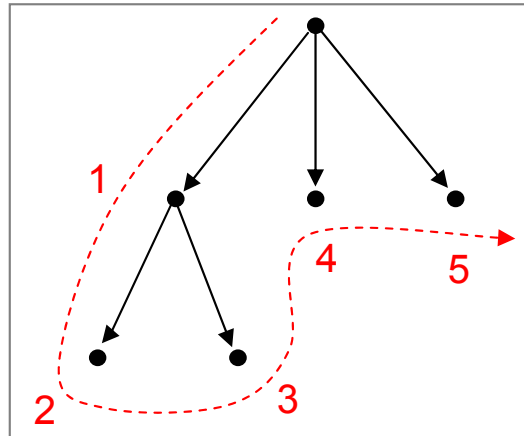
1. $\mathcal{ALC}$ -concept satisfiability and subsumption are decidable
2. $\mathcal{ALC}$ has the **tree model property**
3. $\mathcal{ALC}$ has the **finite model property**



A PSPACE upper bound for $\mathcal{ALC}$

Modify Tableau algorithm:

1. Construct completion tree in a depth-first manner:



2. Keep only paths of the tree in memory!

Yields a PSPACE algorithm:

- paths are of length polynomial in $|C_0|$
- the outdegree is polynomial in $|C_0|$.

PSPACE lower bound will be proved later!



Reasoning with Acyclic TBoxes

Naive approach: **unfolding**

$\implies$ reduce satisfiability w.r.t. TBoxes to satisfiability without TBoxes

Let C_0 be concept, $\mathcal{T}$ acyclic TBox

1. replace concept names on right hand sides of definitions $A \doteq C$ with their defining concept
2. replace each concept name in C_0 defined in $\mathcal{T}$ with its definition.

Terminates due to acyclicity!

But: exponential blowup in the worst case

$$\begin{aligned} A_0 &\doteq \forall R.A_1 \sqcap \forall S.A_1 \\ A_1 &\doteq \forall R.A_2 \sqcap \forall S.A_2 \\ &\vdots \\ A_{k-1} &\doteq \forall R.A_k \sqcap \forall S.A_k \end{aligned}$$



Unfolding on the Fly

Idea:

Modify existing tableau algorithm to directly deal with acyclic TBoxes

Roadmap:

- convert concept definitions into one of the forms

$$A \doteq \neg X, A \doteq B_1 \sqcap B_2, A \doteq B_1 \sqcup B_2, A \doteq \forall R.B, A \doteq \exists R.B$$

with A, B, B_1, B_2 concept **names** and X **primitive** concept name

- restrict node labels to concept **names**
- make **on the fly** TBox lookups for rule application

Result: Satisfiability of $\mathcal{ALC}$ -concepts w.r.t. acyclic TBoxes is PSPACE-complete.



That's it

More on tableau algorithms tomorrow!



Termination

Role depth of concepts:

$$d(A) = d(\leq n R) = 0 \qquad d(\geq n R) = 1$$

$$d(\neg C) = d(C)$$

$$d(C \sqcap D) = d(C \sqcup D) = \max\{d(C), d(D)\}$$

$$d(\exists R.C) = d(\forall R.C) = d(C) + 1$$

The algorithm terminates since:

1. depth of the completion tree bounded by $d(C_0)$.
2. for each node, at most $\#sub(C_0)$ successors are generated
3. each node label contains at most $\#sub(C_0)$ concepts
4. concepts are never deleted from node labels
5. nodes may be deleted (via identification), but 1 and 2 is independent from this



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

Extend $\mathcal{ALC}$ -tableau algorithm from first session with

- ① general TBoxes
- ② inverse roles
- ③ number restrictions

Discuss some optimisation issues

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

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

Remember:

- A concept equation is of the form $C \doteq D$ for C, D (complex) concepts
- A (general) TBox is a finite set of concept equations
- $\mathcal{I}$ satisfies $C \doteq D$ iff $C^{\mathcal{I}} = D^{\mathcal{I}}$
- $\mathcal{I}$ is a model of TBox $\mathcal{T} = \{C_i \doteq D_i \mid 1 \leq i \leq n\}$ iff $\mathcal{I}$ satisfies each concept equation in $\mathcal{T}$

Extend $\mathcal{ALC}$ tableau algorithm to TBox $\rightsquigarrow$ transform a TBox

$$\mathcal{T} = \{C_i \doteq D_i \mid 1 \leq i \leq n\},$$

into a single, equivalent concept equation

$$\top \doteq \bigwedge_{1 \leq i \leq n} (\neg C_i \sqcup D_i) \sqcap (C_i \sqcup \neg D_i) =: C_{\mathcal{T}},$$

$$\bigwedge_{1 \leq i \leq n} (C_i \Leftrightarrow D_i)$$

Lemma: $\mathcal{I}$ is a model of $\mathcal{T}$ iff $C_{\mathcal{T}}^{\mathcal{I}} = \Delta^{\mathcal{I}}$.

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

- Thus, 1. since nodes of completion tree represent elements of $\Delta^{\mathcal{I}}$ and
2. because of previous Lemma ($\mathcal{I}$ is a model of $\mathcal{T}$ iff $C_{\mathcal{T}}^{\mathcal{I}} = \Delta^{\mathcal{I}}$),

to decide satisfiability of $\mathcal{ALC}$ -concepts w.r.t. TBoxes,
we extend $\mathcal{ALC}$ -tableau algorithm with the following rule:

$\rightarrow_{CE}$: If $C_{\mathcal{T}} \notin \mathcal{L}(x)$
 Then $\mathcal{L}(x) \rightarrow \mathcal{L}(x) \cup \{C_{\mathcal{T}}\}$

Example: Consider satisfiability of C w.r.t. $\{C \doteq \exists R.C\}$
 $C_{\mathcal{T}} = (\neg C \sqcup \exists R.C) \sqcap (C \sqcup \forall R.\neg C)$

Tableau algorithm no longer terminates!

Reason: The role depth of concepts no longer decreases along paths
in a completion tree

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

Regain termination with cycle-detection

Blocking: if x and one of its ancestors y are very similar, only extend y :

- 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

$\rightsquigarrow$ Tableau algorithm for $\mathcal{ALC}$ w.r.t. TBoxes

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

Satisfiability of $\mathcal{ALC}$ -concept w.r.t. general TBoxes is decidable and $\mathcal{ALC}$ has the finite model property.

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

Proof: 1. **(Termination)** (the algorithm terminates when applied to $\mathcal{T}$ and C_0):
is due to the following properties: let $n = |C_0| + |C_{\mathcal{T}}|$

1. **outdegree** of the completion tree is bounded by n
2. **depth** of the completion tree is bounded by 2^n :
on a path of length 2^n , blocking occurs, and thus it does not get longer
3. **node labels** are sets of sub-concepts of C_0 and $C_{\mathcal{T}}$

(1+2+3): the **size** of the completion tree built is bounded

4. the completion tree is built in a **monotonic way**: each rule only
 - extends node labels or
 - adds a node (with a label)

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

Proof: 2. “ $\Rightarrow$ ” (**Soundness**): for $\mathcal{T}$ a complete, clash-free completion tree for $C_0, \mathcal{T}$, construct a finite (non-tree) model as follows:

$$\Delta^{\mathcal{I}} := \{x \mid x \text{ is a node in } \mathcal{T}, x \text{ is not blocked}\}$$

$$A^{\mathcal{I}} := \{x \in \Delta^{\mathcal{I}} \mid A \in \mathcal{L}(x)\} \quad \text{for concept names } A$$

$$R^{\mathcal{I}} := \{\langle x, y \rangle \in \Delta^{\mathcal{I}^2} \mid \langle x, y \rangle \in \mathcal{E}(R) \text{ or } \langle x, y' \rangle \in \mathcal{E}(R) \text{ and } y \text{ blocks } y'\} \quad \text{for role names } R$$

Prove (by induction on the structure of concepts) that, for all $x \in \Delta^{\mathcal{I}}, C \in \text{sub}(C_0, \mathcal{T})$:

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

Then, since C_0 is in the label of the root node and $C_{\mathcal{T}}$ in the label of all nodes, $\mathcal{I}$ is indeed a model of C_0 and $\mathcal{T}$.

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

Proof: 3. $\Leftarrow$ (Completeness):

use model $\mathcal{I}$ of C_0 and $\mathcal{T}$ to steer the application of the $\rightarrow_{\sqcup}$ -rule:

Inductively define a total mapping

$$\pi : \text{nodes of completion tree} \rightarrow \Delta^{\mathcal{I}},$$

start with $\pi(x_0) = a$ for some $a \in C_0^{\mathcal{I}}$ and show that

each rule can be applied in such a way that $(*)$ is preserved
if π is extended appropriately for the $\rightarrow_{\exists}$ -rule:

$$\begin{aligned} &\text{if } C \in \mathcal{L}(x), \text{ then } \pi(x) \in C^{\mathcal{I}} \\ &\text{if } \langle x, y \rangle \in \mathcal{E}(R), \text{ then } \langle \pi(x), \pi(y) \rangle \in R^{\mathcal{I}} \end{aligned} \quad (*)$$

E.g.: $\rightarrow_{\forall}$ -rule, applied to x and y with $\forall R.C \in \mathcal{L}(x)$: $(*)$ implies $\pi(x) \in (\forall R.C)^{\mathcal{I}}$ and $\langle \pi(x), \pi(y) \rangle \in R^{\mathcal{I}}$. Hence semantics implies $\pi(y) \in C^{\mathcal{I}}$, and thus adding C to $\mathcal{L}(y)$ preserves $(*)$.

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

Proof: 3. $\Leftarrow$ (Completeness):

use model $\mathcal{I}$ of C_0 and $\mathcal{T}$ to steer the application of the $\rightarrow_{\sqcup}$ -rule:

Inductively define a total mapping

$$\pi : \text{nodes of completion tree} \rightarrow \Delta^{\mathcal{I}},$$

start with $\pi(x_0) = a$ for some $a \in C_0^{\mathcal{I}}$ and show that

each rule can be applied in such a way that $(*)$ is preserved
if π is extended appropriately for the $\rightarrow_{\exists}$ -rule:

$$\begin{aligned} &\text{if } C \in \mathcal{L}(x), \text{ then } \pi(x) \in C^{\mathcal{I}} \\ &\text{if } \langle x, y \rangle \in \mathcal{E}(R), \text{ then } \langle \pi(x), \pi(y) \rangle \in R^{\mathcal{I}} \end{aligned} \quad (*)$$

E.g.: $\rightarrow_{\exists}$ -rule, applied to x with $\exists R.C \in \mathcal{L}(x)$: $(*)$ implies $\pi(x) \in (\exists R.C)^{\mathcal{I}}$.
Hence semantics implies the existence of $b \in \Delta^{\mathcal{I}}$ with $\langle \pi(x), b \rangle \in R^{\mathcal{I}}$ and $b \in C^{\mathcal{I}}$.
Extending π with $\pi(y) = b$ for y the new R -successor of x thus preserves $(*)$.

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) $2^{NExpTime}$,
- can be implemented in various ways,
 - order/priorities of rules
 - data structure
 - etc.
- is amenable to optimisations (see end of session).

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:

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

Motivation: with inverse roles, one can use both
 has-child and is-child-of
 has-part and is-part-of
and their interaction

Example: Does $\forall \text{parent}.\forall \text{child}.\text{Blond} \sqsubseteq \text{Blond}$ w.r.t. $\mathcal{T} = \{ \top \dot{=} \exists \text{parent}.\top \}$?

Example: Is $C_0 = \exists R.\exists S.\exists T.A$ satisfiable w.r.t. $\mathcal{T} = \{ \begin{array}{l} C \dot{=} \exists R.C \\ \top \dot{=} \forall T^-. \forall S^-. \forall R^-. C \end{array} \}$

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

Modifications necessary to handle inverse roles:

- ① extend edges $\mathcal{E}(\cdot)$ 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 $\rightarrow_{\forall}$ - and $\rightarrow_{\exists}$ -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 $\exists R^-.C \in \mathcal{L}(x)$)

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

Does this suffice? **No!**

Example: Is $\exists R.B$ satisfiable w.r.t. $\mathcal{T} = \{B \doteq \exists R.B \sqcap \forall R^-. \forall R^-. (A \sqcap \neg A)\}$?

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

“subset-blocking” (with $\mathcal{L}(x) \subseteq \mathcal{L}(x')$) no longer suffices:

④ A node x is called **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. $\mathcal{T}$.

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

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

2. $\Rightarrow$ (Soundness): again, construct a finite (non-tree) model from a complete, clash-free c-tree $\mathbb{T}$ for C_0 and $\mathcal{T}$ (take into account edges labelled with inverse roles):

$$\Delta^{\mathcal{I}} := \{x \mid x \text{ is a node in } \mathcal{T}, x \text{ is not blocked}\}$$

$$A^{\mathcal{I}} := \{x \in \Delta^{\mathcal{I}} \mid A \in \mathcal{L}(x)\} \quad \text{for concept names } A$$

$$R^{\mathcal{I}} := \{\langle x, y \rangle \in \Delta^{\mathcal{I}^2} \mid \langle x, y \rangle \in \mathcal{E}(R), \langle y, x \rangle \in \mathcal{E}(R^-) \text{ or} \quad \text{for role names } R$$

$$\langle x, y' \rangle \in \mathcal{E}(R) \text{ and } y \text{ blocks } y' \text{ or}$$

$$\langle y, x' \rangle \in \mathcal{E}(R^-) \text{ and } x \text{ blocks } x'\}$$

Again, prove that, for all $x \in \Delta^{\mathcal{I}}$:

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

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

3. $\Leftarrow$ (Completeness): again, use model $\mathcal{I}$ of C_0 and $\mathcal{T}$ and mapping π to steer the application of the $\rightarrow_{\sqcup}$ -rule.

Corollary:

Satisfiability of $\mathcal{ALCI}$ -concept w.r.t. general TBoxes is decidable and $\mathcal{ALCI}$ has the finite model property.

It can be shown that pure $\mathcal{ALCI}$ -concept satisfiability (without TBoxes) is PSpace-complete, just like $\mathcal{ALC}$.

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

Further add **qualifying number restrictions** $(\geq nr.C)$ and $(\leq nr.C)$:

$$\begin{aligned} (\geq nr.C)^{\mathcal{I}} &:= \{x \in \Delta^{\mathcal{I}} \mid \#\{y \mid \langle x, y \rangle \in r^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \geq n\} \\ (\leq nr.C)^{\mathcal{I}} &:= \{x \in \Delta^{\mathcal{I}} \mid \#\{y \mid \langle x, y \rangle \in r^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \leq n\} \end{aligned}$$

$\mathcal{ALCQI}$ is $\mathcal{ALCI}$ extended with qualifying number restrictions.

Observation: $\mathcal{ALCQI}$ with general TBoxes loses **finite model property**:

$$C_0 := \neg A, \quad \mathcal{T} := \{\top \doteq \exists R.A \sqcap (\leq 1 R^-. \top)\}$$

C_0 is satisfiable w.r.t. $\mathcal{T}$, but **only in infinite models**

Obvious: 2 new rules for tableau algorithm

- $\rightarrow_{\geq}$: **If** $(\geq nr.C) \in \mathcal{L}(x)$, x is not blocked, and x has less than n R -neighbours y_i with $C \in \mathcal{L}(y_i)$
Then create n new R -successor $y_1, \dots, y_n$ of x with $\mathcal{L}(y_i) := \{C\}$ for each $1 \leq i \leq n$
- $\rightarrow_{\leq}$: **If** $(\leq nr.C) \in \mathcal{L}(x)$, x is not indirectly blocked, x has $n + 1$ R -neighbours $y_0, \dots, y_n$ with $C \in \mathcal{L}(y_i)$, and there are i, j with y_j is **not** an ancestor of y_i
Then $\mathcal{L}(y_i) \rightarrow \mathcal{L}(y_i) \cup \mathcal{L}(y_j)$,
make y_j 's successors to successors of y_i ,
remove y_j from the tree

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

Problem: Consider what the tableau algorithm does for

satisfiability of $C_0 = (\geq 3R.(A \sqcap B)) \sqcap (\leq 2R.A)$ w.r.t. the empty TBox

1. it starts with $\mathcal{L}(x_0) = \{C_0\}$,
2. applies $\rightarrow_{\sqcap}$ -rule to $\mathcal{L}(x_0) = \{C_0, (\geq 3R.(A \sqcap B)), (\leq 2R.A)\}$,
3. applies $\rightarrow_{\geq}$ -rule and generates 3 R -successors y_i of x_0 with $\mathcal{L}(y_i) = \{A \sqcap B\}$,
4. applies $\rightarrow_{\sqcap}$ -rule to yield $\mathcal{L}(y_i) = \{A \sqcap B, A, B\}$,
5. applies $\rightarrow_{\leq}$ -rule and removes one of the y_i
6. goes back to (3), and thus loops
 $\Rightarrow$ algorithm does not terminate due to yoyo effect

Solution: Use explicit inequality $\neq$ to prevent yoyo effect

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

Solution: Use explicit inequality $\neq$ to prevent yoyo effect:

- $\rightarrow_{\geq}$: **If** $(\geq nr.C) \in \mathcal{L}(x)$, x is not blocked, and x has less than n R -neighbours y_i with $C \in \mathcal{L}(y_i)$
 Then create n new R -successor $y_1, \dots, y_n$ of x with
 $\mathcal{L}(y_i) := \{C\}$ for each $1 \leq i \leq n$ and $y_i \neq y_j$ for all $i \neq j$
- $\rightarrow_{\leq}$: **If** $(\leq nr.C) \in \mathcal{L}(x)$, x is not indirectly blocked, x has $n + 1$ R -neighbours $y_0, \dots, y_n$ with $C \in \mathcal{L}(y_i)$, and there are i, j with **not** $y_i \neq y_j$ and y_j is **not** an ancestor of y_i
 Then $\mathcal{L}(y_i) \rightarrow \mathcal{L}(y_i) \cup \mathcal{L}(y_j)$,
 make y_j 's successors to successors of y_i ,
 add $y_i \neq z$ for each z with $y_j \neq z$,
 remove y_j from the tree

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

Extend definition of a clash to NRs:

Additionally, x contains a clash if

- $(\leq n r.C) \in \mathcal{L}(x)$ and
- x has more than n R -neighbours $y_0, \dots, y_n$ with $y_i \neq y_j$ for all $i \neq j$.

Does this suffice? No:

$$(\leq 1 R.A) \sqcap (\leq 1 R.\neg A) \sqcap (\geq 3 R.B)$$

is **unsatisfiable**, but the algorithm would answer “satisfiable”

Reason: if $(\leq n R.C) \in \mathcal{L}(x)$ and x has an R -neighbour y ,
we need to know whether y is a C or not

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

Solution:

① use $\dot{\neg}C$ as an abbreviation for the NNF of $\neg C$,

② extend the node labels to

$$\text{cl}(C_0, \mathcal{T}) := \text{sub}(C_0, \mathcal{T}) \cup \{\dot{\neg}C \mid C \in \text{sub}(C_0, \mathcal{T})\}$$

② add a third new rule:

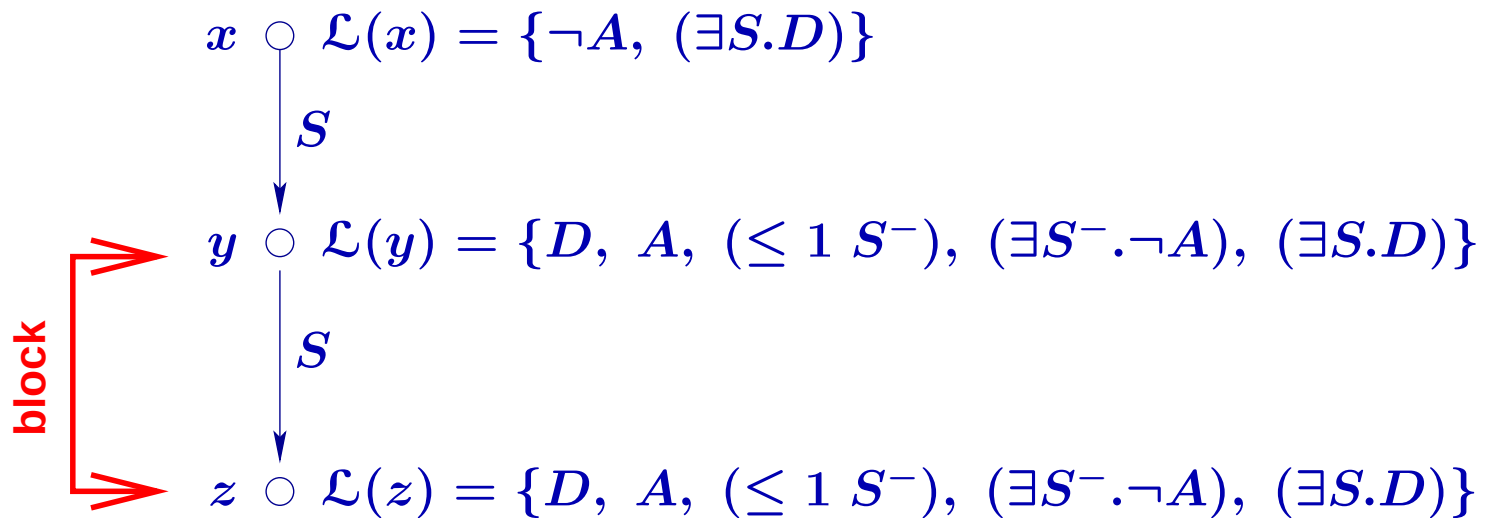
$\rightarrow_{\text{choice}}$: **If** $(\leq n R.C) \in \mathcal{L}(x)$, x is not indirectly blocked,
 x has an R -neighbour y with $\{C, \dot{\neg}C\} \cap \mathcal{L}(y) = \emptyset$
Then $\mathcal{L}(y) \rightarrow \mathcal{L}(y) \cup \{D\}$ for some $D \in \{C, \dot{\neg}C\}$

Does this suffice? No ...

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

Example: test satisfiability of $\neg A$ w.r.t. $\mathcal{T}$:

$$\mathcal{T} = \{\top \doteq \exists S. \underbrace{(A \sqcap (\leq 1 S^-. \top) \sqcap (\exists S^-. \neg A))}_D\}, \text{ i.e., } C_{\mathcal{T}} = \exists S.D$$



z would block y but we cannot construct a model from this

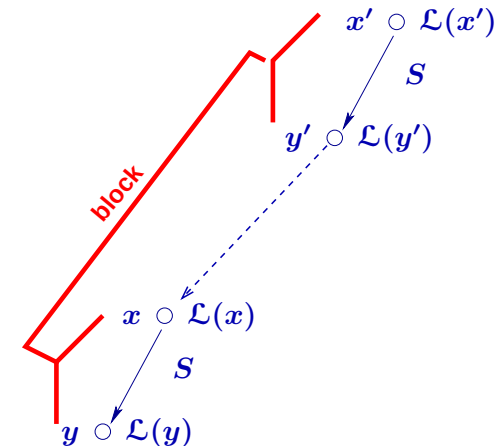
...and $\neg A$ is unsatisfiable w.r.t. $\mathcal{T}$, i.e., our algorithm is still incorrect!

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

Solution: Double-blocking:

y is **directly blocked** if there are ancestors x , x' , and y' of y with

- x is predecessor of y , x' is predecessor of y' ,
- $\mathcal{E}(\langle x, y \rangle) = \mathcal{E}(\langle x', y' \rangle)$,
- $\mathcal{L}(x) = \mathcal{L}(x')$, and $\mathcal{L}(y) = \mathcal{L}(y')$.



Lemma:

Let $\mathcal{T}$ be a general $\mathcal{ALCQI}$ -Tbox and C_0 an $\mathcal{ALCQI}$ -concept. Then the

1. 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: beyond the scope of this course

Implementing the $\mathcal{ALCQI}$ Tableau Algorithm

Naive implementation of $\mathcal{ALCQI}$ tableau algorithm is **doomed to failure**:

It constructs a tree of **exponential depth** in a **non-deterministic way**

~> requires double exponential non-deterministic time and

~> requires backtracking in a deterministic implementation

Optimisations are crucial

concern every aspect of the

help in “many” cases (which?)

In the following: a selection of some vital optimisations

Optimising the $\mathcal{ALCQI}$ Tableau Algorithm: Classification

FaCT provides service “classify all concepts defined $\mathcal{T}$ ”, i.e.,

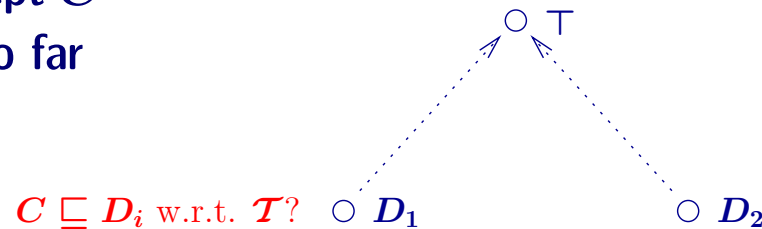
for all concepts C, D defined in $\mathcal{T}$, FaCT decides is $C \sqsubseteq D$ w.r.t. $\mathcal{T}$?

$\rightsquigarrow$ test satisfiability of $C \sqcap \neg D$ w.r.t. $\mathcal{T}$

$\rightsquigarrow n^2$ satisfiability tests!

Goal: reduce number of satisfiability tests when classifying TBox

Idea: “trickle” new concept C
into hierarchy computed so far



Optimising the $\mathcal{ALCQI}$ Tableau Algorithm: Classification

FaCT provides service “classify all concepts defined $\mathcal{T}$ ”, i.e.,

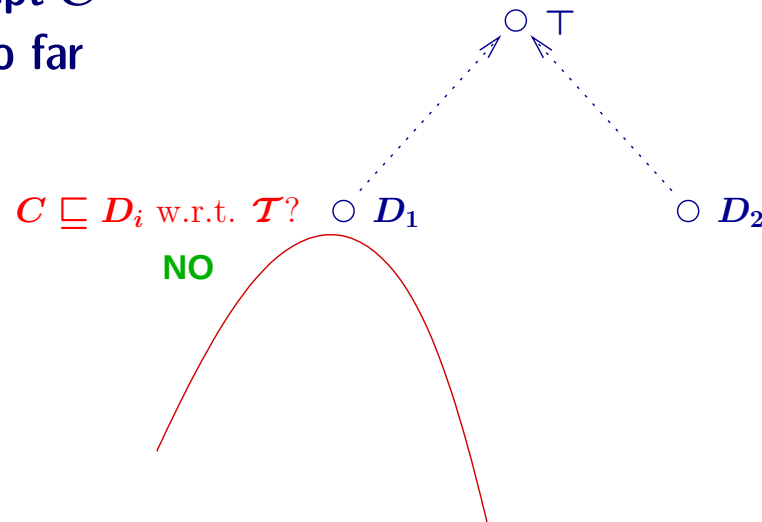
for all concepts C, D defined in $\mathcal{T}$, FaCT decides is $C \sqsubseteq D$ w.r.t. $\mathcal{T}$?

$\rightsquigarrow$ test satisfiability of $C \sqcap \neg D$ w.r.t. $\mathcal{T}$

$\rightsquigarrow n^2$ satisfiability tests!

Goal: reduce number of satisfiability tests when classifying TBox

Idea: “trickle” new concept C
into hierarchy computed so far



Optimising the $\mathcal{ALCQI}$ Tableau Algorithm: Classification

FaCT provides service “classify all concepts defined $\mathcal{T}$ ”, i.e.,

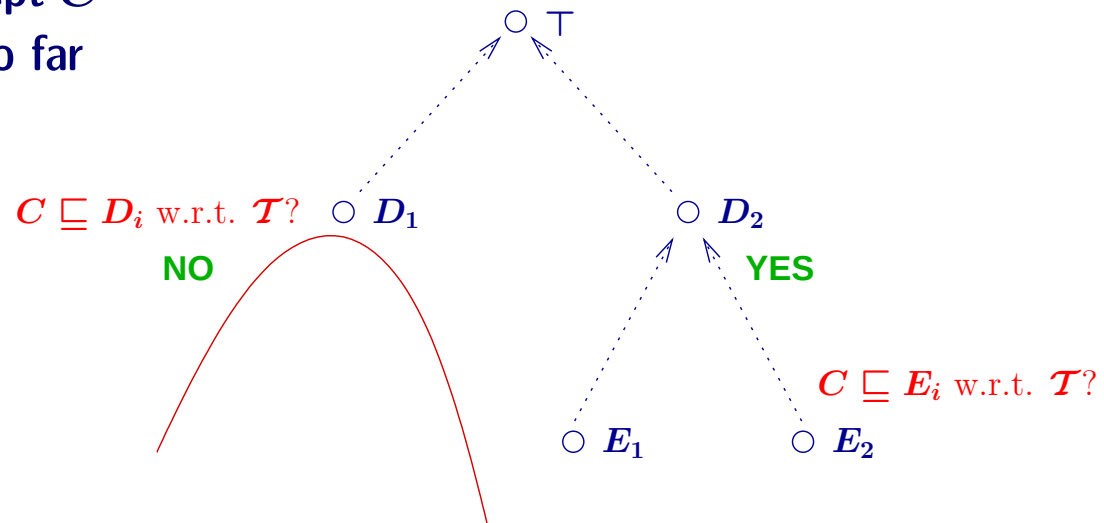
for all concepts C, D defined in $\mathcal{T}$, FaCT decides is $C \sqsubseteq D$ w.r.t. $\mathcal{T}$?

$\rightsquigarrow$ test satisfiability of $C \sqcap \neg D$ w.r.t. $\mathcal{T}$

$\rightsquigarrow n^2$ satisfiability tests!

Goal: reduce number of satisfiability tests when classifying TBox

Idea: “trickle” new concept C into hierarchy computed so far



Optimising the $\mathcal{ALCQI}$ Tableau Algorithm: Absorption

Remember: for $\mathcal{T} = \{C_i \doteq D_i \mid 1 \leq i \leq n\}$, and

$$C_{\mathcal{T}} = \bigcap_{1 \leq i \leq n} (\neg C_i \sqcup D_i) \sqcap (C_i \sqcup \neg D_i)$$

$\rightarrow_{\text{CE}}:$ If $C_{\mathcal{T}} \notin \mathcal{L}(x)$
 Then $\mathcal{L}(x) \rightarrow \mathcal{L}(x) \cup \{C_{\mathcal{T}}\}$

Problem: 2 disjunctions/GCI $\rightsquigarrow$ high degree of non-determinism and huge search space

Observation: split “equivalence” $C_i \doteq D_i$ into “implications” $C_i \sqsubseteq D_i$ and $D_i \sqsubseteq C_i$, then many GCIs are of the form $A \sqcap \dots \sqsubseteq C$ for concept name A
 e.g., Human $\sqcap \dots \sqsubseteq C$ versus Device $\sqcap \dots \sqsubseteq C$

Optimising the $\mathcal{ALCQI}$ Tableau Algorithm: Absorption

Idea: localise GCIs to concept names by transforming

$$A \sqcap X \sqsubseteq C \text{ into equivalent } A \sqsubseteq \neg X \sqcup C$$

$$\text{e.g., } \text{Human} \sqcap \exists \text{owns.Pet} \sqsubseteq C \text{ becomes } \text{Human} \sqsubseteq \neg \exists \text{owns.Pet} \sqcup C$$

For “absorbed” $\mathcal{T} = \{A_i \sqsubseteq D_i \mid 1 \leq i \leq n_1\} \cup \{C_i \sqsubseteq D_i \mid 1 \leq i \leq n_2\}$
replace $\rightarrow_{\text{CE}}$ -rule with $\rightarrow_{\text{GCI}}$ -rules:

$$\begin{array}{l} \rightarrow_{\text{GCI1}}: \text{ If } A_i \in \mathcal{L}(x) \\ \quad \text{for some } 1 \leq i \leq n_1 \\ \text{Then } \mathcal{L}(x) \rightarrow \mathcal{L}(x) \cup \{D_i\} \end{array}$$

$$\begin{array}{l} \rightarrow_{\text{GCI2}}: \text{ If } (\neg C_i \sqcup D_i) \notin \mathcal{L}(x) \\ \quad \text{for some } 1 \leq i \leq n_2 \\ \text{Then } \mathcal{L}(x) \rightarrow \mathcal{L}(x) \cup \{(\neg C_i \sqcup D_i)\} \end{array}$$

Observations: If no GCI is absorbable, this yields the same as $\rightarrow_{\text{CE}}$ -rule
Each absorption safes 1 disjunction **per node**,
in the best case, this supersedes **all** disjunctions from TBox axioms!

Optimising the $\mathcal{ALCQI}$ Tableau Algorithm: Optimised Blocking

For $\mathcal{ALCQI}$, the blocking condition is:

y is blocked by y' if

for x the predecessor of y , x' the predecessor of y'

1. $\mathcal{L}(x) = \mathcal{L}(x')$
2. $\mathcal{L}(y) = \mathcal{L}(y')$
3. (x, R, y) iff (x', R, y')

$\rightsquigarrow$ blocking occurs late

$\rightsquigarrow$ search space is huge

Optimising the $\mathcal{ALCQI}$ Tableau Algorithm: Optimised Blocking

For $\mathcal{ALCQI}$, the blocking condition is:

y is blocked by y' if

for x the predecessor of y , x' the predecessor of y'

1. $\mathcal{L}(x) = \mathcal{L}(x')$

2. $\mathcal{L}(y) = \mathcal{L}(y')$

3. (x, R, y) iff (x', R, y')

1. $\mathcal{L}(x) \cap RC = \mathcal{L}(x') \cap RC$

2. $\mathcal{L}(y) \cap RC = \mathcal{L}(y') \cap RC$

3. (x, R, y) iff (x', R, y')

for “relevant concepts RC ”

$\rightsquigarrow$ blocking occurs late

$\rightsquigarrow$ search space is huge

$\rightsquigarrow$ blocking occurs earlier

$\rightsquigarrow$ search space is smaller

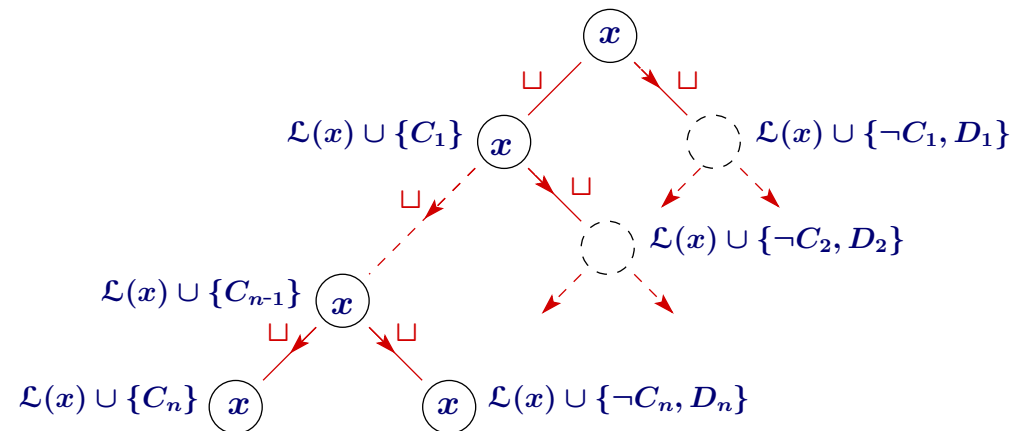
... details are beyond the scope of this course

Optimising the $\mathcal{ALCQI}$ Tableau Algorithm: Backjumping

Remember If a **clash** ($A, \neg A \in \mathcal{L}(x)$) is encountered, algorithm **backtracks**

i.e., returns to last non-deterministic choice and
tries other possibility

Example $\exists R.(A \sqcap B) \sqcap ((C_1 \sqcup D_1) \sqcap \dots \sqcap (C_n \sqcup D_n)) \sqcap \forall R.\neg A \in \mathcal{L}(x)$

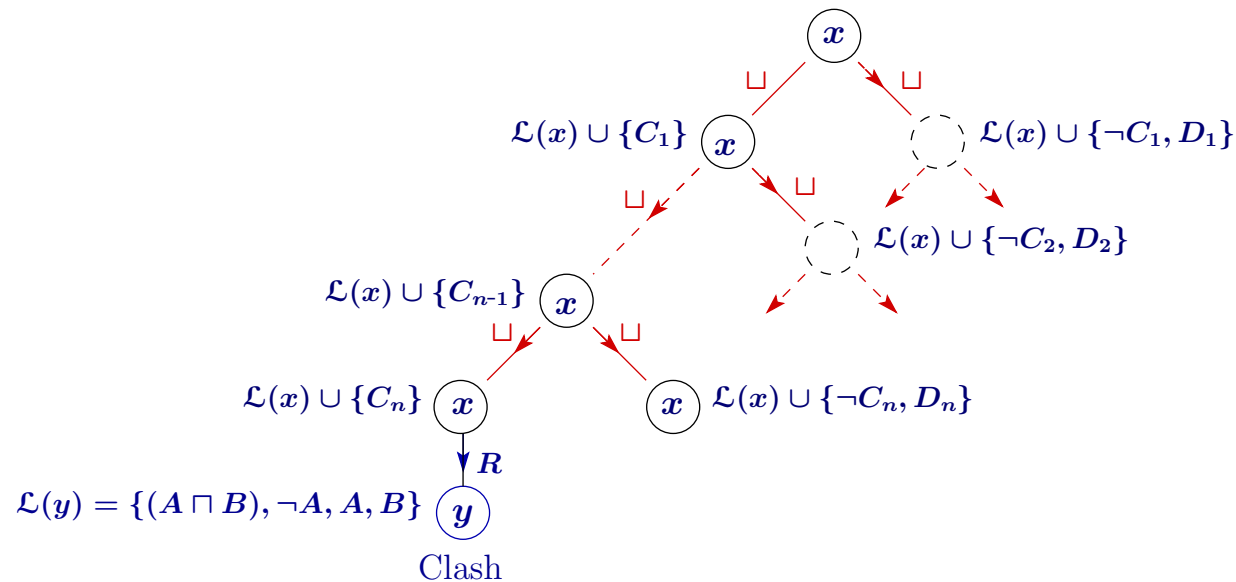


Optimising the $\mathcal{ALCQI}$ Tableau Algorithm: Backjumping

Remember If a **clash** ($A, \neg A \in \mathcal{L}(x)$) is encountered, algorithm **backtracks**

i.e., returns to last non-deterministic choice and
tries other possibility

Example $\exists R.(A \sqcap B) \sqcap ((C_1 \sqcup D_1) \sqcap \dots \sqcap (C_n \sqcup D_n)) \sqcap \forall R.\neg A \in \mathcal{L}(x)$

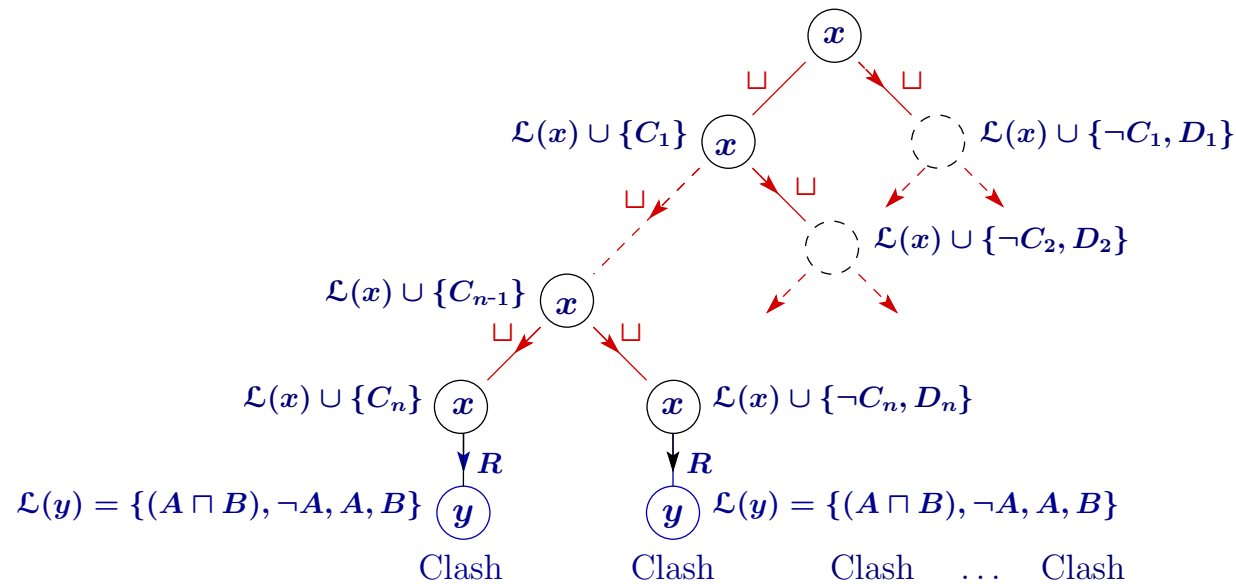


Optimising the $\mathcal{ALCQI}$ Tableau Algorithm: Backjumping

Remember If a **clash** ($A, \neg A \in \mathcal{L}(x)$) is encountered, algorithm **backtracks**

i.e., returns to last non-deterministic choice and
tries other possibility

Example $\exists R.(A \sqcap B) \sqcap ((C_1 \sqcup D_1) \sqcap \dots \sqcap (C_n \sqcup D_n)) \sqcap \forall R.\neg A \in \mathcal{L}(x)$

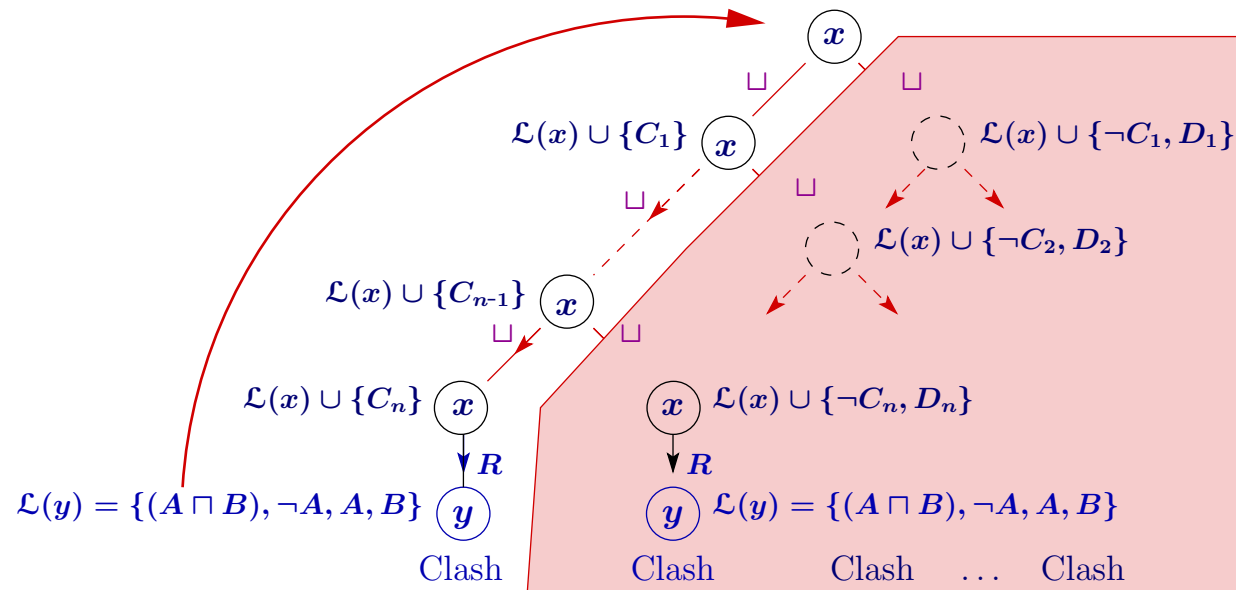


Optimising the $\mathcal{ALCQI}$ Tableau Algorithm: Backjumping

Remember If a **clash** ($A, \neg A \in \mathcal{L}(x)$) is encountered, algorithm **backtracks**

i.e., returns to last non-deterministic choice and
tries other possibility

Example $\exists R.(A \sqcap B) \sqcap ((C_1 \sqcup D_1) \sqcap \dots \sqcap (C_n \sqcup D_n)) \sqcap \forall R.\neg A \in \mathcal{L}(x)$



Optimising the $\mathcal{ALCQI}$ Tableau Algorithm: SAT Optimisations

Finally: $\mathcal{ALCQI}$ extends **propositional logic**

⇒ heuristics developed for **SAT** are relevant

Summing up: optimisations are possible at each aspect of tableau algorithm

can dramatically enhance performance

⇒ do they interact?

⇒ how?

⇒ which combination works best for which “cases”?

⇒ is the optimised algorithm still correct?

Tableau Algorithms: What was Left out

- here: $\mathcal{ALCQI}$,
in SOTA DL reasoners FaCT and Racer: $\mathcal{SHIQ}$, the extension of $\mathcal{ALCQI}$ with
 - **transitive roles**: if $\text{Trans}(R)$, then $R^{\mathcal{I}}$ must be transitive
 - **role hierarchies**: if $R \sqsubseteq S$, then $\mathcal{I}$ must satisfy $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$...extension of $\mathcal{ALCQI}$ tableau algorithm and proofs tedious, but not too difficult
- **ABox**: assertions of the form $a : C$ and $(a, b) : R$ to describe a specific situation
 - ...extension of $\mathcal{ALCQI}$ tableau algorithm and proofs relatively easy
 - ... implementation challenging
- **concrete domains** to describe “concrete” properties such as age, height, weight, etc.
 - ...extension of $\mathcal{ALCQI}$ tableau algorithm only possible for restricted cases
- **tools** such as OilEd that use a DL reasoner
 - ...recent development: **Protégé** has an OWL plugin that connects to DL reasoner

Thank you!

Description Logics

Carsten Lutz

TU Dresden, Germany

Ulrike Sattler

Univ. of Manchester, UK

Part III: Automata-based Decision Procedures



Automata vs. Tableaux

Tableau algorithms

- + based on a simple idea (model construction)
- + amenable to optimization techniques
- + basis for all state-of-the-art DL reasoners
- bad for proving deterministic upper time bounds
- termination proofs can become very hard

Automata-based algorithms

- + elegant and simple
- + well-suited for proving EXPTIME upper bounds
- + no termination proofs
- no optimized implementations exist (?)

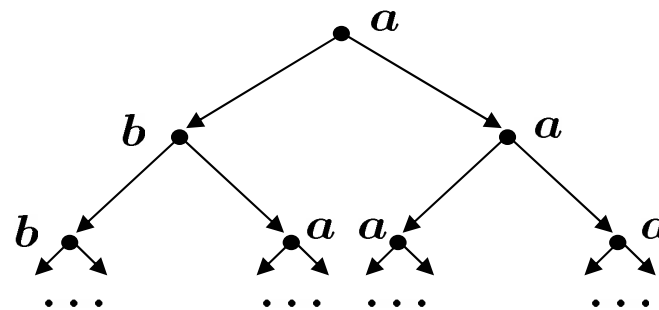


Automata

There are many types of automata:

- DFA/NFA: working on finite words
- ω -automata: working on infinite words
- automata on finite trees
- automata on infinite trees

k -ary M -tree T : infinite k -ary tree whose nodes are labelled with elements of M



2-ary $\{a, b\}$ -tree



Idea for deciding satisfiability w.r.t. TBoxes:

1. choose a DL that has the tree model property (infinite trees ok)
2. for concept C_0 and TBox $\mathcal{T}$, define automaton $\mathcal{A}_{C_0, \mathcal{T}}$ such that $\mathcal{A}_{C_0, \mathcal{T}}$ accepts precisely the tree models of C_0 and $\mathcal{T}$
3. check whether $\mathcal{A}_{C_0, \mathcal{T}}$ satisfies the empty language

If you do not have the tree model property: try some tricks

Can be used to establish EXPTIME upper bound:

- size of automata is usually exponential in size of C_0 and $\mathcal{T}$.
- emptiness can be decided in deterministic polynomial time



Looping Tree Automata

Looping tree automaton $\mathcal{A}$: tuple (Q, M, I, Δ) with

Q a finite set of **states**

M an alphabet

$I \subseteq Q$ a set of **initial states**

$\Delta \subseteq Q \times M \times Q^k$ a **transition relation**

Language recognized by automaton: set of k -ary M -trees



Runs

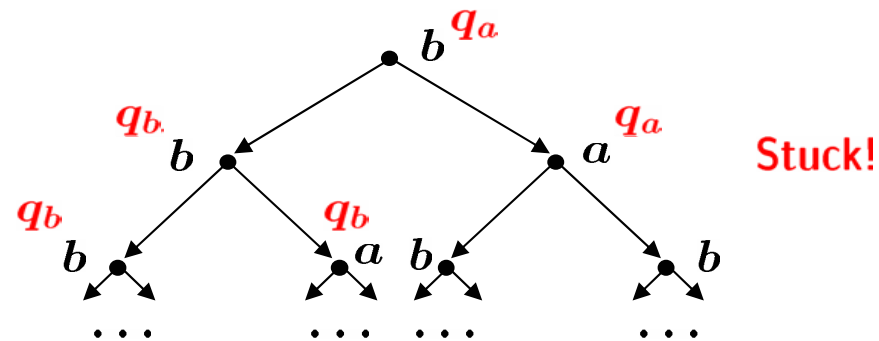
Looping tree automaton on alphabet $\{a, b\}$

$$Q = \{q_a, q_b\}$$

$$M = \{a, b\}$$

$$I = \{q_a\}$$

$$\Delta = \{(q_a, a, q_a, q_b), (q_b, a, q_b, q_b), \\ (q_a, a, q_b, q_a), (q_b, b, q_b, q_b)\}$$



Language: all trees with infinite a -path starting at root

Looping tree automaton on alphabet $\{a, b\}$

$$Q = \{q_a, q_b\}$$

$$M = \{a, b\}$$

$$I = \{q_a\}$$

$$\Delta = \{(q_a, a, q_a, q_b), (q_b, a, q_b, q_b), \\ (q_a, a, q_b, q_a), (q_b, b, q_b, q_b)\}$$

Run of $\mathcal{A}$ on T : assigns to each node in T a state from Q such that

1. T 's root is labelled with a state from I
2. $(r(\alpha), T(\alpha), r(\alpha 1), \dots, r(\alpha k)) \in \Delta$ for all $\alpha \in \{1, \dots, k\}^*$

$$L(\mathcal{A}) = \{T \mid \text{there is a run of } \mathcal{A} \text{ on } T\}$$



Roadmap

Recall: $\mathcal{ALCI} = \mathcal{ALC} + \text{inverse roles } \exists R^-.C \text{ and } \forall R^-.C$

Goal: prove that $\mathcal{ALCI}$ -satisfiability w.r.t. TBoxes is in EXPTIME

We proceed in two steps:

1. abstract tree interpretations into **Hintikka trees**
 - tree models have **labelled** edges (roles), automata trees don't
 - it is convenient to label nodes also with **complex** concepts
2. define automata that accepts exactly the Hintikka-trees
representing models for the input concept and TBox



Hintikka Sets

Hintikka sets are used as **node labels** in Hintikka trees (set M).

Intuitively, Hintikka set is the set of all (relevant) concepts
satisfied by some domain element.

Let $C_0, \mathcal{T}$ be in NNF and $\dot{\neg}$ and $\text{cl}(C_0, \mathcal{T})$ be defined as before.

Hintikka set for C_0 and $\mathcal{T}$: subset $\Psi \subseteq \text{cl}(C_0, \mathcal{T})$ such that

- (S1) if $C \sqcap D \in \Psi$, then $\{C, D\} \subseteq \Psi$
- (S2) if $C \sqcup D \in \Psi$, then $\{C, D\} \cap \Psi \neq \emptyset$
- (S3) $\{C, \dot{\neg}C\} \not\subseteq \Psi$ for all $C \in \text{cl}(C_0, \mathcal{T})$
- (S4) $C \doteq D \in \mathcal{T}$ implies $C \leftrightarrow D \in \Psi$



$\mathcal{H}_{C_0, \mathcal{T}}$: set of all Hintikka-sets for C_0 and $\mathcal{T}$

Hintikka Trees

Let k be the number of successors a domain element can be **forced** to have:

$$k := \#\{D \in \text{cl}(C_0, \mathcal{T}) \mid D \text{ is of the form } \exists R.C\}$$

Hintikka-trees will be k -ary $\mathcal{H}_{C_0, \mathcal{T}}$ -trees

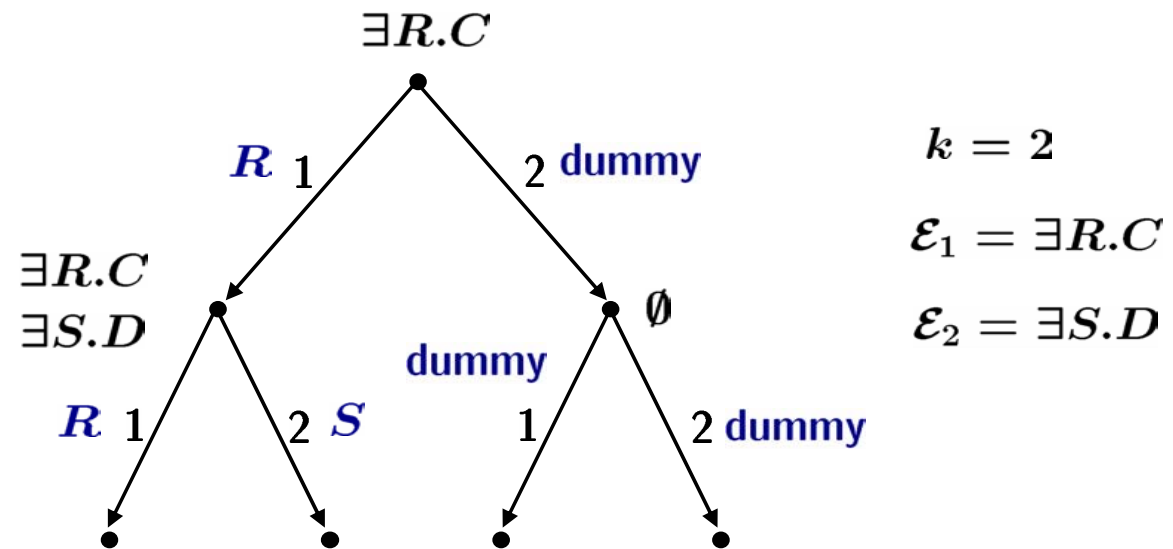
may be inverse

How can we deal with the non-labelled edges?

- intuitively, there is one **potential** successor for each $\exists R.C$
- hence, the **connecting role** for each successor is already fixed!
- enumerate all concepts $\exists R.C$ with $\mathcal{E}_i$ denoting the i 'th one.
- if $\mathcal{E}_i = \exists R.C \in \mathcal{L}(\alpha)$, then the role connecting α 's i -th successor is R
- if $\mathcal{E}_i = \exists R.C \notin \mathcal{L}(\alpha)$, then connection to α 's i -th successor is a "dummy"



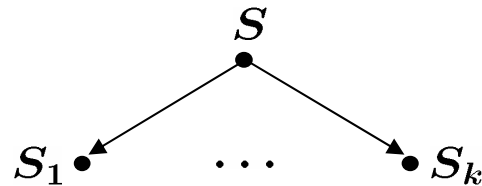
Example



Hintikka Trees II

Next step:

describe relationship between Hintikka-set of nodes and
Hintikka-set of it's successors



A $k + 1$ -tuple of Hintikka-sets $S, S_1, \dots, S_k$ is **matching** iff

(M1) if $\mathcal{E}_i = \exists R.C \in S$, then $C \in S_i$

(M2) if $\forall R.C \in S$ and $\mathcal{E}_i = \exists R.D \in S$, then $C \in S_i$

(M3) if $\forall R.C \in S_i$ and $\mathcal{E}_i = \exists \text{Inv}(R).D \in S$, then $C \in S$



Note: for satisfying $\exists R.C$, we only consider the i -th successor

Hintikka tree for C_0 and $\mathcal{T}$:

k -ary $\mathcal{H}_{C_0, \mathcal{T}}$ -tree T such that, for all nodes α :

(T1) $C_0 \in T(\epsilon)$

(T2) the tuple $(T(\alpha), T(\alpha 1), \dots, T(\alpha k))$ is matching

Lemma. C_0 and $\mathcal{T}$ have a model iff there exists a Hintikka-tree for C_0 and $\mathcal{T}$.



Constructing Automata I

Basic idea:

- use Hintikka sets as states and define Δ such that

$$q_0 = \ell \text{ for all } (q_0, \ell, q_1, \dots, q_k) \in \Delta$$

(recall that $\Delta \subseteq Q \times M \times Q^k$)

$\Rightarrow$ if there is an accepting run, it will be identical to the tree!!!

- use initial states to ensure that $C_0 \in T(\epsilon)$
- Check matching properties via transition relation:

e.g. (M1) if $(q_0, \ell, q_1, \dots, q_k) \in \Delta$ and $\mathcal{E}_i = \exists R.C \in q_0$
then $C \in q_i$



Automaton for C_0 and $\mathcal{T}$:

$\mathcal{A}_{C_0, \mathcal{T}} = (Q, M, I, \Delta)$ where

$$Q = \mathcal{H}_{C_0, \mathcal{T}}$$

$$M = \mathcal{H}_{C_0, \mathcal{T}}$$

$$I = \{q \in Q \mid C_0 \in q\}$$

$$q_0, \ell, q_1, \dots, q_k \in \Delta \text{ iff}$$

$$\text{- } q_0 = \ell$$

- the tuple $(S_0, S_1, \dots, S_k)$ is matching

Lemma. $T \in L(\mathcal{A}_{C_0, \mathcal{T}})$ iff T is a Hintikka-tree for C_0 and $\mathcal{T}$



Results

Size of $\mathcal{A}_{C_0, \mathcal{T}}$:

1. number of hintikka-sets exponential in $|C_0, \mathcal{T}|$

1 $\implies$ 2. $|Q|$, $|I|$, and $|M|$ exponential in $|C_0, \mathcal{T}|$.

2 $\implies$ 3. $|\Delta|$ exponential in $|C_0, \mathcal{T}|$.

$$(|\Delta| = |M| * |Q|^{k+1})$$

2 + 3 $\implies$ size of $|\mathcal{A}_{C_0, \mathcal{T}}|$ exponential in $|C_0, \mathcal{T}|$.

Since the emptiness problem for looping automata is in PTIME:

Theorem. $\mathcal{ALCI}$ -concept satisfiability and subsumption w.r.t. TBoxes
is in EXPTIME.

This complexity bound is optimal as $\mathcal{ALC}$ with TBox is EXPTIME-hard



The Emptiness Problem of Looping Automata

Determine in $|Q|$ rounds the set of **blocking states** $B \subseteq Q$:

Initialization:

Set $B_0 := \{q \in Q \mid \text{there is no } (q, a, q_1, \dots, q_k) \in \Delta\}$

Round i :

Set $B_i := B_{i-1} \cup \{q \in Q \mid \text{for all } (q, a, q_1, \dots, q_k) \in \Delta$
there is an $1 \leq i \leq k$ such that $q_i \in B_{i-1}\}$

Lemma. $L(\mathcal{A}) = \emptyset$ iff $I \subseteq B$.

Computation of the set $B = B_{|Q|}$ is clearly in PTime.



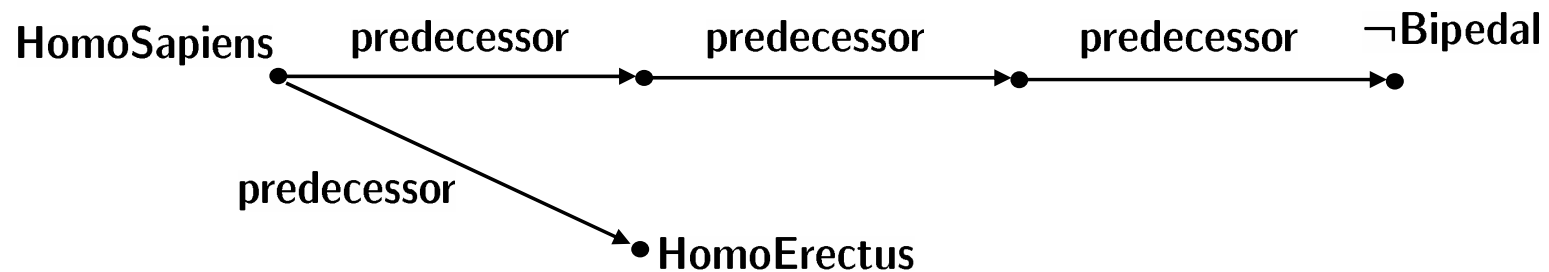
$\mathcal{ALC}$ with Transitive Closure

$$\mathcal{ALC}^+ = \mathcal{ALC}$$

+ transitive closure operator on roles R^+

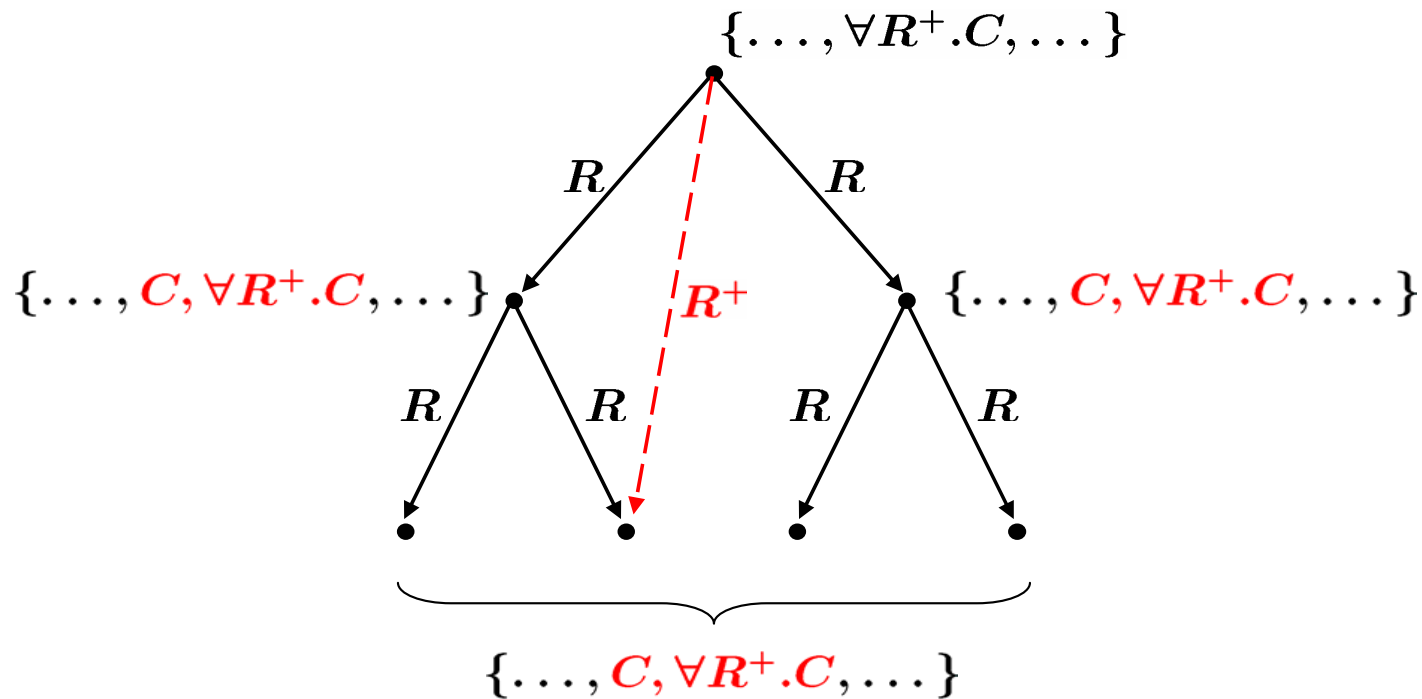
Semantics: $(R^+)^{\mathcal{I}} = (R^{\mathcal{I}})^+$

Example: $\text{HomoSapiens} \doteq \exists \text{predecessor.HomoErectus} \sqcap \exists \text{predecessor}^+.\neg \text{Bipedal}$

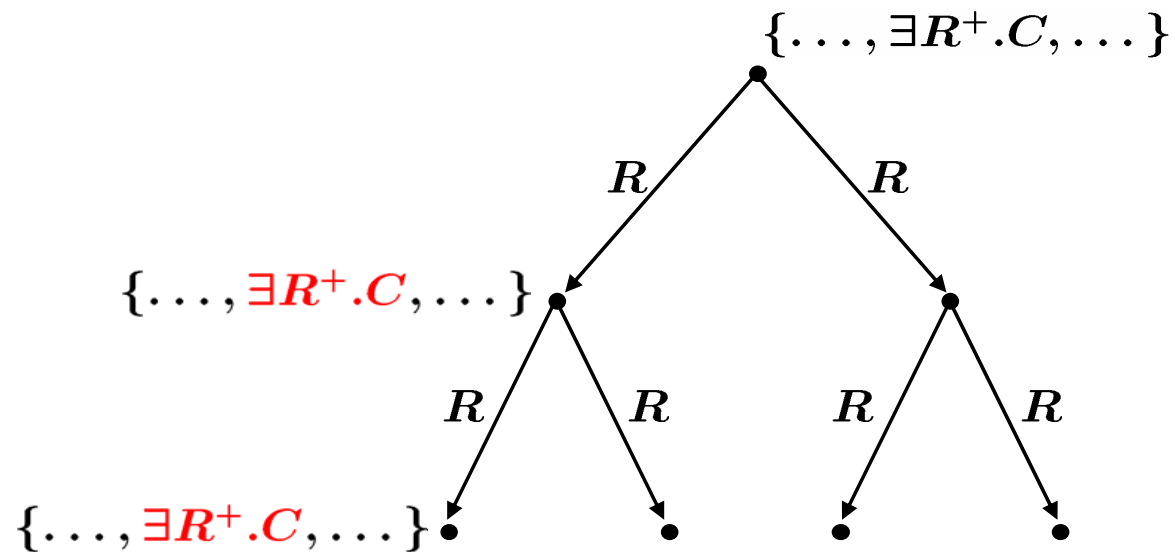


Idea for dealing with concepts $\forall R^+.C$:

not only put C to all R -successors, **but also $\forall R^+.C$!**



Can we do something similar for $\exists R.C$ (eventualities)?



Problem: Satisfaction of existential restriction may be delayed **forever**

$\Rightarrow$ Looping tree automata do not suffice!

Büchi Tree Automata

Büchi tree automata: $\mathcal{A} = (Q, M, I, \Delta, F)$

like looping tree automata with additional set of **accepting states** $F \subseteq Q$

branch of a tree: infinite path starting at the root

A run is **accepting** iff for **every** branch B of the tree

there **exists** a final state $q \in F$

such that q occurs infinitely often on B

$L(\mathcal{A}) = \{T \mid \text{there is an accepting run of } \mathcal{A} \text{ on } T\}$

Lemma. The emptiness problem of Büchi tree automata is in PTIME.



Generalized Büchi Tree Automata

Generalized Büchi tree automata: possibly **many** sets of accepting states

$$\mathcal{A} = (Q, M, I, \Delta, \mathbf{F_1}, \dots, \mathbf{F_\ell})$$

A run is **accepting** iff for **every** branch B of the tree

for **every** $i \in \{1, \dots, \ell\}$

there **exists** a state $q \in F_i$

such that q occurs infinitely often on B

Lemma. For every generalized Büchi tree automaton, an equivalent Büchi tree automaton can be constructed.



Hintikka Trees I

Let k be the number of concepts $\exists R.C$ or $\exists R^+.C$ in $\text{cl}(C_0, \mathcal{T})$

Again, $\mathcal{E}_i$ denotes the i 'th such concept.

A $k + 1$ -tuple of Hintikka-sets $S, S_1, \dots, S_k$ is **matching** iff

(M1) if $\mathcal{E}_i = \exists R.C \in S$, then $C \in S_i$

(M2) if $\mathcal{E}_i = \exists R^+.C \in S$, then $\{\exists R^+.C, C\} \cap S_i \neq \emptyset$

(M3) if $\forall R.C \in S$ and $\mathcal{E}_i = \exists R.D \in S$ or $\mathcal{E}_i = \exists R^+.D \in S$,
then $C \in S_i$

(M4) if $\forall R^+.C \in S$ and $\mathcal{E}_i = \exists R.D \in S$ or $\mathcal{E}_i = \exists R^+.D \in S$,
then $\{\forall R^+.C, C\} \subseteq S_i$

Note: for satisfying $\exists R^+.C$, we only consider the i -th successor



Hintikka tree for C_0 and $\mathcal{T}$:

k -ary $\mathcal{H}_{C_0, \mathcal{T}}$ -tree T such that, for all nodes α :

(T1) $C_0 \in T(\epsilon)$

(T2) The tuple $(T(\alpha), T(\alpha 1), \dots, T(\alpha k))$ is matching

(T3) if $\mathcal{E}_i = \exists R^+.C \in T(\alpha)$, then $C \in T(\alpha i^n)$ for some $n > 0$

start at α and go to the i -th successor n times

Lemma. C_0 and $\mathcal{T}$ have a model iff there exists a Hintikka-tree for C_0 and $\mathcal{T}$.



Eventualities and Flags

Büchi acceptance condition needed for:

(T3) if $\mathcal{E}_i = \exists R^+.C \in T(\alpha)$, then $C \in T(\alpha i^n)$ for some $n > 0$

Idea: - States are pairs (S, f) with

- * S Hintikka set

- * $f \in \{0, \dots, k\}$ a **flag**.

- Flags are used to keep track of unsatisfied eventualities

- * eventuality $\mathcal{E}_i$ is waiting for satisfaction: $f = i$

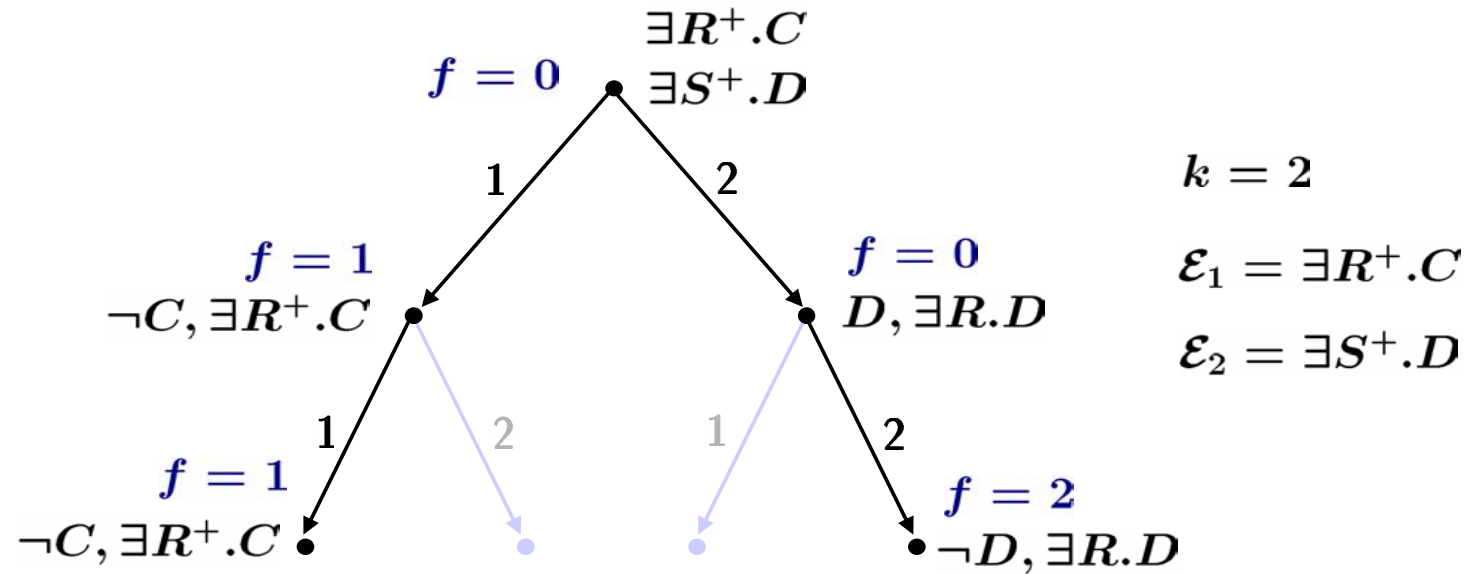
- * no eventuality waiting for satisfaction: $f = 0$ (recall: there is no $\mathcal{E}_0$!)

- Starving eventuality means that the flag sticks to the same non-null value on a branch

$\Rightarrow$ Use Büchi-condition to detect that.



Eventualities and Flags II



If $r(\alpha) = (S, f)$, then

- $f = i$ iff * $\mathcal{E}_f = \exists R^+.C$ occurred in α 's predecessor
- * α is f 'th successor of the predecessor
- * $C \notin S$



Eventualities and Flags III

Assume eventuality $\mathcal{E}_i$ is starving on path $\alpha i i i i i i i \dots$

- $\Rightarrow$ Flag sticks to the same (non-null) value i
- $\Rightarrow$ All values **different** from i occur only finitely often on the branch $\epsilon \dots \alpha i i i i i i i \dots$
- $\Rightarrow$ for each i , introduce a set F_i containing those states with $f \neq i$

Use k sets of accepting states $F_1, \dots, F_k$:

$$F_i = \{(S, j) \in Q \mid j \neq i\}$$



Automaton for C_0 and $\mathcal{T}$:

$\mathcal{A}_{C_0, \mathcal{T}} = (Q, M, I, \Delta, F_1, \dots, F_k)$ where

$$Q = \mathcal{H}_{C_0, \mathcal{T}} \times \{0, \dots, k\}$$

$$M = \mathcal{H}_{C_0, \mathcal{T}}$$

$$I = \{(S, f) \in Q \mid C_0 \in S \text{ and } f = 0\}$$

$$F_i = \{(S, j) \in Q \mid j \neq i\}$$

$$((S_0, f_0), S, (S_1, f_1), \dots, (S_k, f_k)) \in \Delta \text{ iff}$$

- $S_0 = S$
- the tuple $(S_0, S_1, \dots, S_k)$ is matching
- $f_i = \begin{cases} i & \text{if } \mathcal{E}_i = \exists R^+.C \in S_0 \text{ and } C \notin S_i \\ 0 & \text{otherwise} \end{cases}$



Results

Emptiness test for Büchi-automata is more complicated, but still P_{TIME}

Lemma. $T \in L(\mathcal{A}_{C_0, \mathcal{T}})$ iff T is a Hintikka-tree for C_0 and $\mathcal{T}$

Theorem. $\mathcal{ALC}^+$ -concept satisfiability and subsumption w.r.t. TBoxes
is EXP_{TIME} -complete.

Can be easily extended to $\mathcal{ALC}^{reg}$:

- $\mathcal{ALC}$ + role constructors “ $\cup$ ” (union), “ $;$ ” (composition),
and “ $\cdot^*$ ” (refl-trans closure)
- Notational variant of PDL (Propositional Dynamic Logic)



Final Remarks

- Automata-based approach also works for logics from Part II

$\implies$ they are in EXPTIME .

- There exist other acceptance conditions than Büchi
- There exist other automata models, for example **alternating automata**:

Idea:

- states are formulas, not sets of formulas
- size of automaton is polynomial in the size of the input concept + TBox
- emptiness check is EXPTIME -complete



4. Complexity of selected DLs

Devise (and prove) upper and lower complexity bounds for a selection of DLs

So far: upper complexity bounds for the satisfiability of:

- **CSAT($\mathcal{ALC}$) is in NEXP TIME**
(tableau algorithm is non-deterministic, builds c-tree of linear depth)
- **SAT($\mathcal{ALCQI}$) w.r.t. TBoxes is in 2NEXP TIME**
(tableau algorithm is non-deterministic, builds c-tree of exponential depth)
- **SAT($\mathcal{ALCIO}$) w.r.t. TBoxes is in EXP TIME**
(automata built for $C_0, \mathcal{T}$ is exponential in $|C_0| + |\mathcal{T}|$,
testing its emptiness is polynomial in automata's size)

The NExpTime tableau algorithm deciding $\text{CSAT}(\mathcal{ALC})$ can be modified easily to run in PSpace:

For an $\mathcal{ALC}$ -concept C_0 ,

1. the c-tree can be built **depth-first**
2. **branches are independent** $\rightsquigarrow$ keep only one branch in memory at any time
3. **length of branch** $\leq |C_0|$
4. for each node x , $\mathcal{L}(x) \subseteq \text{sub}(C_0)$ and $\# \text{sub}(C_0)$ is linear in $|C_0|$

$\rightsquigarrow$ non-deterministic PSpace decision procedure for $\text{CSAT}(\mathcal{ALC})$

inverse roles: use “ R -neighbour” instead of “ R -successor”
to decide $\text{CSAT}(\mathcal{ALCI}) \rightsquigarrow$ tableau algorithm satisfying

1. the c-tree can be built depth-first
2. **branches are independent** $\rightsquigarrow$ keep only one branch in memory at any time
3. **length** of branch $\leq |C_0|$
4. for each node x , $\mathcal{L}(x) \subseteq \text{sub}(C_0)$ and $\# \text{sub}(C_0)$ is linear in $|C_0|$

branches are not **independent**: effects go up one branch, down another branch

Solution: reset-restart technique:

5. when node label $\mathcal{L}(x)$ changes, remove the subtree below x (reset) and re-construct it (restart).

$\rightsquigarrow$ **PSpace** realisation of the tableau algorithm for $\text{CSAT}(\mathcal{ALCI})$

PSpace DLs: $\mathcal{ALC}$ (and thus $\mathcal{ALCI}$) is PSpace-hard

PSpace-hardness of $\mathcal{ALC}$ proved by a reduction of

the validity problem for
quantified Boolean formulae

to

satisfiability of $\mathcal{ALC}$

QBFs are of the form $Q_1p_1.Q_2p_2.\dots.Q_np_n.\varphi$
for $Q_i \in \{\forall, \exists\}$ and φ a Boolean formula over $p_1, \dots, p_n$

Validity of QBFs defined inductively:

$\exists p.\Phi$ is valid if $\Phi[p/t]$ or $\Phi[p/f]$ is valid

$\forall p.\Phi$ is valid if $\Phi[p/t]$ and $\Phi[p/f]$ are valid

Known: validity of QBFs is PSPACE-hard

Obligation: for QBF Φ , define C_Φ such that

Lemma: Φ is valid iff C_Φ is satisfiable

$\rightsquigarrow$ **Corollary:** $\text{CSAT}(\mathcal{ALC})$ is PSPACE-complete

Each model of $C_\Phi = Q_1 p_1 . Q_2 p_2 . \dots . Q_n p_n . \varphi$ contains a tree of depth n where, for $x_0 \in C_\Phi$,

- if $Q_i = \exists$, then each $R^{(i-1)}$ -successor of x_0 has 1 R -successor, and
- if $Q_i = \forall$, then each $R^{(i-1)}$ -successor of x_0 has 2 R -successors, one in p_i and one in $\neg p_i$.
- and all leave nodes satisfy φ .

$C_\Phi := L_1 \sqcap \forall R . (L_2 \sqcap \forall R . (L_3 \sqcap \dots \forall R . (L_n \sqcap \varphi)) \dots))$ where

$$L_i := D_i \sqcap \begin{cases} \exists R . \top & \text{if } Q_i = \exists \\ \exists R . p_i \sqcap \exists R . \neg p_i & \text{if } Q_i = \forall \text{ and} \end{cases}$$

$$D_i := \bigsqcap_{j \leq i} (p_j \Rightarrow \forall R . p_j) \sqcap (\neg p_j \Rightarrow \forall R . \neg p_j)$$



ExpTime DLs: $\mathcal{ALC}$ with TBoxes is ExpTime-hard

We know: $\text{SAT}(\mathcal{ALCIO})$ w.r.t. TBoxes is in ExpTime

Lemma: $\text{SAT}(\mathcal{ALC})$ w.r.t. TBoxes is ExpTime-hard

Proof idea: reduce the halting problem of a polynomial-space-bounded alternating TM to $\text{SAT}(\mathcal{ALC})$ w.r.t. TBoxes.

Basic Ideas: for PSB-A-TM $\mathcal{M}$, build TBox $\mathcal{T}$ with

- tape-cell i contains letter σ is coded as concept name σ_i
(possible due to polynomial space bound)
- use CEs to ensure that each tape-cell contains exactly one letter
- use CEs to encode $\mathcal{M}$'s $(\forall/\exists)$ transitions relation
(e.g., cell contents changes only near head)

Then $\mathcal{M}$ holds on w iff $\hat{w}$ is satisfiable w.r.t. $\mathcal{T}$

An alternating TM $\mathcal{M} = (Q, \Sigma, \Delta, q_0)$ consists of

$Q = U \dot{\cup} E$, a set of states with
 U a set of universal states and
 E a set of existential states,
 $q_0 \in Q$ the initial state,
 Σ , the alphabet,

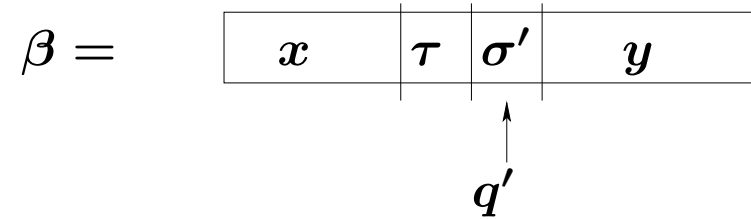
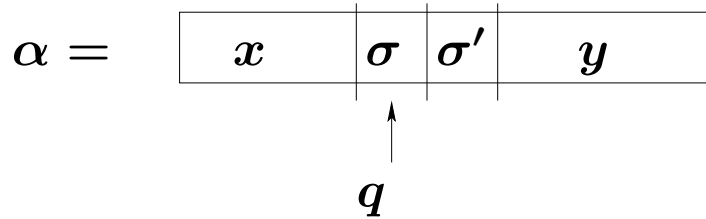
$\Delta \subseteq (Q \times \Sigma) \times (Q \times \Sigma \times \{\ell, r\})$, the transition relation.

- $\alpha \in \Sigma^* Q \Sigma^+$ is a configuration,
- $\alpha \in \Sigma^* U \Sigma^+$ is a universal configuration,
- $\alpha \in \Sigma^* E \Sigma^+$ is an existential configuration.

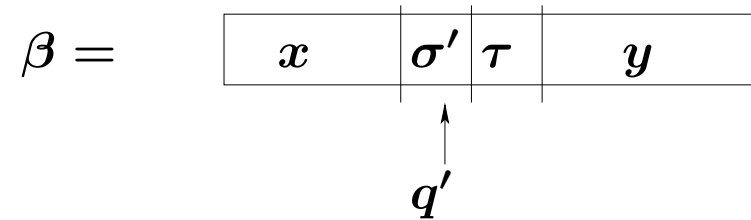
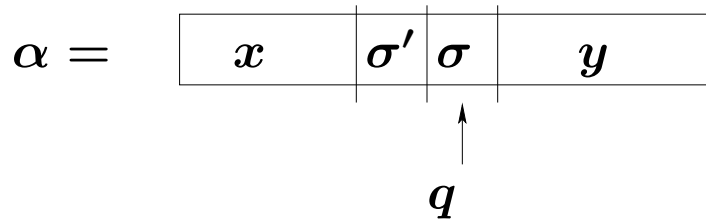
ExpTime DLs: $\mathcal{ALC}$ with TBoxes is ExpTime-hard

The transitions are defined as usual:

β is a next-config of α
if $(q, \sigma, q', \tau, r) \in \Delta$ and



or
if $(q, \sigma, q', \tau, l) \in \Delta$ and



A **trace** of $\mathcal{M}$ is a finite set $T = \{\alpha_0, \alpha_1, \dots\}$ where each α_i is a configuration and if

- $\alpha \in T$ is a universal configuration, then $\forall$ next configuration β of α : $\beta \in T$
- $\alpha \in T$ is an existential configuration, then $\exists$ a next configuration β of α : $\beta \in T$.

$$L(\mathcal{M}) := \{w \in \Sigma^* \mid \exists T \text{ with } q_0 w \in T\}$$

$s(\cdot)$ — bounded A-TM:

$$L(\mathcal{M}, s(\cdot)) := \{w \in \Sigma^* \mid \exists T \text{ with } q_0 w \in T \text{ and } \forall \alpha \in T : |\alpha| \leq s(|w|)\}$$

Example: an alternating TM $\mathcal{M} = (Q, \Sigma, \Delta, q_0)$ with

$$L(\mathcal{M}) = \{w \in \{a, b\}^* \mid w \in (aa)^+ \text{ or } w \in (ab)^+\} = (aa)^+ \cup (ab)^+$$

$Q = U \dot{\cup} E$, set of states with

$\{A_1, A_2, B_1, B_2, F\}$ set of universal states and

$\{q_0\}$ set of existential states,

$q_0 \in Q$ the initial state,

$\{a, b\}$, the alphabet,

, the transition relation.

$$\begin{aligned} \Delta &\subseteq (Q \times \Sigma) \times (Q \times \Sigma \times \{\ell, r\}) \\ &= \{ (q_0, a, A_1, a, r), (q_0, a, B_1, a, r) \\ &\quad (A_1, a, A_0, a, r), (A_0, a, A_1, a, r) \\ &\quad (B_1, b, B_0, b, r), (B_0, a, B_1, a, r) \\ &\quad (B_0, \text{blank}, F, ,), (A_0, \text{blank}, F, ,) \} \end{aligned}$$

ExpTime DLs: $\mathcal{ALC}$ with TBoxes is ExpTime-hard

For an ATM $\mathcal{M} = (Q, \Sigma, b, \Delta, q_0)$, a word $w \in \Sigma^*$ and $m := s(|w|) + 1$, the TBox $\mathcal{T}_{\mathcal{M}}$ contains the following CEs to ensure that each element in a model corresponds to a configuration of $\mathcal{M}$:

exactly one state and one head position

$$\top \doteq \bigsqcup_{q \in Q} (A_q \sqcap \bigsqcap_{q' \neq q} (\neg A_{q'})) \sqcap \bigsqcup_{i=0}^m (H_i \sqcap \bigsqcap_{j \neq i} \neg H_j)$$

exactly one letter per cell

$$\top \doteq \bigsqcap_{i=0}^m \bigsqcup_{\sigma \in \Sigma} (C_{i,\sigma} \sqcap \bigsqcap_{\tau \neq \sigma} \neg C_{i,\tau})$$

unread cells are maintained

$$\top \doteq \bigsqcap_{i=0}^m \bigsqcup_{\sigma \in \Sigma} (C_{i,\sigma} \sqcap \neg H_i \Rightarrow \forall N. C_{i,\sigma})$$

head moves only 1 cell

$$\top \doteq \bigsqcap_{i=0}^m (H_i \Rightarrow \forall N. (H_{i-1} \sqcup H_{i+1}))$$

universal transitions are ok

$$\top \doteq \bigwedge_{i=1}^{m-1} \bigwedge_{\sigma, q \in U} H_i \sqcap C_{i,\sigma} \sqcap A_q \Rightarrow \begin{array}{l} \bigwedge_{(q,\sigma,q',\sigma',r) \in \Delta} \exists N.(H_{i+1} \sqcap C_{i,\sigma'} \sqcap A_{q'}) \sqcap \\ \bigwedge_{(q,\sigma,q',\sigma',l) \in \Delta} \exists N.(H_{i-1} \sqcap C_{i,\sigma'} \sqcap A_{q'}) \end{array}$$

existential transitions are ok

$$\top \doteq \bigwedge_{i=1}^{m-1} \bigwedge_{\sigma, q \in E} H_i \sqcap C_{i,\sigma} \sqcap A_q \Rightarrow \begin{array}{l} \bigwedge_{(q,\sigma,q',\sigma',r) \in \Delta} \exists N.(H_{i+1} \sqcap C_{i,\sigma'} \sqcap A_{q'}) \sqcup \\ \bigwedge_{(q,\sigma,q',\sigma',l) \in \Delta} \exists N.(H_{i-1} \sqcap C_{i,\sigma'} \sqcap A_{q'}) \end{array}$$

Then, for $w = \sigma_1 \cdots \sigma_n$, we have that

$$A_{q_0} \sqcap H_1 \sqcap \bigwedge_{i=1}^n C_{i,\sigma_i}$$

is satisfiable w.r.t. $\mathcal{T}_{\mathcal{M}}$ iff $w \in L(\mathcal{M})$.

ExpTime-hard DLs

Each Description Logic extending $\mathcal{ALC}$ where we can

polynomially reduce satisfiability w.r.t. TBoxes to
pure concept satisfiability is
ExpTime-hard

This reduction is called **internalisation** of TBoxes

ExpTime DLs: $\mathcal{ALC}\mathcal{IO}$ is ExpTime-hard, even without TBoxes

Lemma: $\text{CSAT}(\mathcal{ALC}\mathcal{IO})$ is ExpTime-hard

Proof idea: use a spy to internalise TBox

Spy: an object N related to **all** other objects via (new) role U :
for input concept C , define

$$C_1 = N \sqcap \exists U.C \sqcap \forall U. \bigwedge_{\text{role } R} \forall R. \exists U^-.N$$

where the conjunction is over all roles R, R^- in C or $\mathcal{T}$

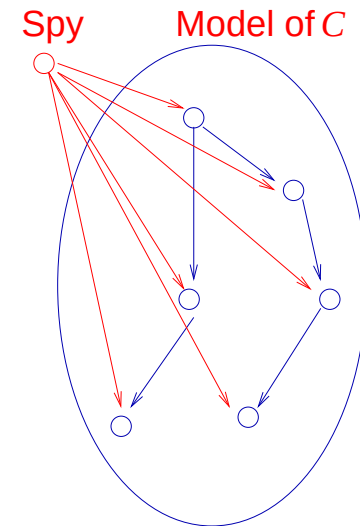
Internalise: ensure that each object satisfies $\mathcal{T} = \{\top = C_{\mathcal{T}}\}$:

$$C_2 = C_1 \sqcap \forall U.C_{\mathcal{T}}$$

$\rightsquigarrow C$ is satisfiable w.r.t. $\mathcal{T}$ iff C_2 is satisfiable

$\rightsquigarrow |C_2|$ is polynomial in $|C| + |C_{\mathcal{T}}|$

$\rightsquigarrow \text{SAT}(\mathcal{ALC}\mathcal{IO})$ w.r.t. TBoxes is polyn. reducible to $\text{CSAT}(\mathcal{ALC}\mathcal{IO})$



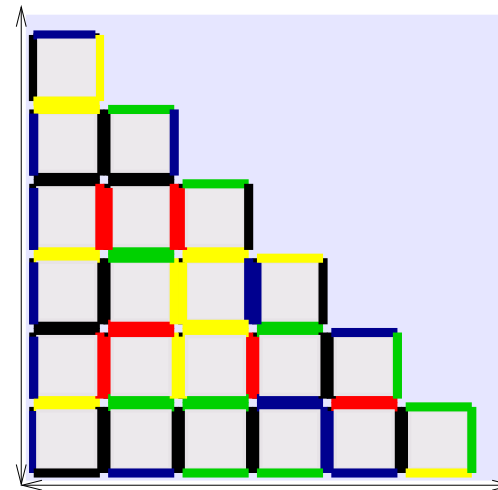
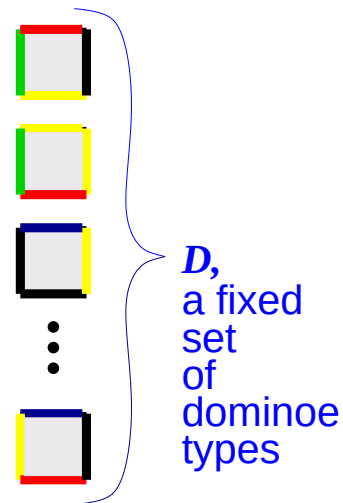
NExpTime DLs: $\mathcal{ALCQIO}$ is NExpTime-hard

We know: $\text{SAT}(\mathcal{ALCIO})$ w.r.t. TBoxes is ExpTime-complete

Also: $\text{SAT}(\mathcal{ALCQI})$ w.r.t. TBoxes is ExpTime-complete
where Q stands for qualifying number restrictions ($\geq nr.C$) and ($\leq nr.C$)

Lemma: their combination is NExpTime-hard

Proof: by reduction of a NExpTime version of the domino problem:



can we tile a
 $2^n \times 2^n$ square
using D ?

Definition: A domino system $\mathcal{D} = (D, H, V)$

- set of domino types $D = \{D_1, \dots, D_d\}$, and
- horizontal and vertical matching conditions $H \subseteq D \times D$ and $V \subseteq D \times D$

A tiling of the $\mathbb{N} \times \mathbb{N}$ grid using $\mathcal{D}$:

$$t : \mathbb{N} \times \mathbb{N} \rightarrow D \text{ such that}$$

$$\langle t(m, n), t(m + 1, n) \rangle \in H \text{ and}$$

$$\langle t(m, n), t(m, n + 1) \rangle \in V$$

Domino problem standard: has $\mathcal{D}$ a tiling?
 NExpTime: has $\mathcal{D}$ a tiling for a $2^n \times 2^n$ square?

Reducing the NExpTime domino problem to $\text{CSAT}(\mathcal{ALCQIO}) \rightsquigarrow$ four tasks:

- ① each object carries exactly one domino type D_i
 $\rightsquigarrow$ use concept name D_i for each domino type and

$$\top \doteq \bigsqcup_{1 \leq i \leq d} (D_i \sqcap \bigsqcap_{j \neq i} \neg D_j)$$

- ② each element x has exactly one H -successor
 exactly one V -successor

whose domino types satisfy the horizontal/vertical matching conditions:

$$\begin{aligned} \top \doteq \bigsqcap_{1 \leq i \leq n} D_i \Rightarrow & ((\leq 1V.\top) \sqcap (\exists V. \bigsqcup_{(D_i, D_j) \in V} D_j)) \sqcap \\ & ((\leq 1H.\top) \sqcap (\exists H. \bigsqcup_{(D_i, D_j) \in H} D_j)) \end{aligned}$$

- ③ the model must be large enough, i.e., have $2^n \times 2^n$ elements
 $\rightsquigarrow$ encode the position (x, y) of each point using binary coding in
the concept names $X_1, \dots, X_n, Y_1, \dots, Y_n$:

E.g., an instance of $\neg X_3, X_2, X_1, Y_3, \neg Y_2$, and Y_1 represents the tuple (011, 101),
and thus the point (3, 5).

$$\begin{aligned} \top &\doteq (\tilde{X}_1 \sqcap \tilde{X}_2 \sqcap \dots \sqcap \tilde{X}_n) \\ \tilde{X}_1 &\doteq (X_1 \sqcap (\forall H. \neg X_1)) \sqcup (\neg X_1 \sqcap (\forall H. X_1)) \quad \% \text{ switch lowest bit} \\ \tilde{X}_i &\doteq \left(\bigcap_{1 \leq j < i} X_j \sqcap ((X_i \sqcap \forall H. \neg X_i) \sqcup (\neg X_i \sqcap \forall H. X_i)) \right) \sqcup \% \text{ switch } i\text{-th bit, if} \\ &\hspace{25em} \% \text{ all lower ones are 1} \\ &\quad (\neg \bigcap_{1 \leq j < i} X_j \sqcap ((X_i \sqcap \forall H. X_i) \sqcup (\neg X_i \sqcap \forall H. \neg X_i))). \% \text{ else keep it} \end{aligned}$$

- ④ ensure that the $V \circ H$ -successor of each node coincides with its $H \circ V$ -successor
 $\rightsquigarrow$ enforce that each object is the H -successor of at most one element (and the same for V):

$$\top \doteq (\leq 1 V^-. \top) \sqcap (\leq 1 H^-. \top)$$

- $\rightsquigarrow$ enforce that there is ≤ 1 object in the upper right corner:

$$N \doteq X_1 \sqcap \dots \sqcap X_n \sqcap Y_1 \sqcap \dots \sqcap Y_n$$

Harvest:

$$\neg X_1 \sqcap \dots \sqcap \neg X_n \sqcap \neg Y_1 \sqcap \dots \sqcap \neg Y_n$$

is satisfiable w.r.t. to $\mathcal{T}_D$ defined above iff D has a $2^n \times 2^n$ -tiling.

NExpTime DLs: $\mathcal{ALCQIO}$ is in NExpTime

Known:

- ① 2-variable fragment of FOL is decidable
- ② 2-variable fragment of FOL with counting quantifiers is decidable
 $\exists^{\geq n} x \dots$ and $\exists^{\leq n} x \dots$ NexpTime-complete
- ③ in $\mathcal{ALCQIO}$, TBoxes can be internalized

$\rightsquigarrow$ to prove $\text{SAT}(\mathcal{ALCQIO})$ w.r.t. TBoxes being in NExpTime, it suffices to reduce $\text{CSAT}(\mathcal{ALCQIO})$ to satisfiability of C-FOL2

In $\mathcal{ALCQIO}$, $\exists R.C$ and $\forall R.C$ are syntactic sugar:

$$\exists R.C \equiv (\geq 1 R.C) \quad \text{and} \quad \forall R.C \equiv (\leq 0 R.\neg C)$$

NExpTime DLs: $\mathcal{ALCQIO}$ is in NExpTime

$$t_x(A) = A(x),$$

$$t_x(N) = (x = N),$$

$$t_x(\neg D) = \neg t_x(D),$$

$$t_x(C \sqcap D) = t_x(C) \wedge t_x(D),$$

$$t_x(C \sqcup D) = t_x(C) \vee t_x(D),$$

$$t_x(\geq n R.C) = \exists^{\geq n} y. R(x, y) \wedge t_y(C),$$

$$t_x(\geq n R^-.C) = \exists^{\geq n} y. R(y, x) \wedge t_y(C),$$

$$t_x(\leq n R.C) = \exists^{\leq n} y. R(x, y) \wedge t_y(C),$$

$$t_x(\leq n R^-.C) = \exists^{\leq n} y. R(y, x) \wedge t_y(C).$$

$\rightsquigarrow$ **SAT**($\mathcal{ALCQIO}$) w.r.t. TBoxes is NExpTime-complete

NExpTime DLs: $\mathcal{ALCQIO}$ is in NExpTime

$\text{SAT}(\mathcal{ALCQIO})$ w.r.t. TBoxes is NExpTime-complete

provided that $|(\geq n R.C)| = |(\leq n R.C)| = n + 1 + |C|$.

For $|(\geq n R.C)| = |(\leq n R.C)| = \log(n) + 1 + |C|$,
no exact upper bound is known.

The DL $\mathcal{SHIQ}$

The DL $\mathcal{SHIQ}$ extends $\mathcal{ALCQI}$ with **transitive roles** and **role hierarchies**:

transitive roles: certain role names must be interpreted as **transitive relations**
e.g., ancestor, has-part, part-of, etc.

role hierarchy: set of implications $R \sqsubseteq S$, which require $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
e.g., daughter $\sqsubseteq$ child, has-component $\sqsubseteq$ has-part

FaCT and **Racer** are highly optimised $\mathcal{SHIQ}$ -implementations

Note: If $\text{Trans}(S)$ and $R \sqsubseteq S$, then $S^{\mathcal{I}}$ is a transitive relation containing $R^{\mathcal{I}}$

$\neq$

$(R^+)^{\mathcal{I}}$ is the **smallest** transitive relation containing $R^{\mathcal{I}}$

Known: $\text{SAT}(\mathcal{SHIQ})$ w.r.t. TBoxes is ExpTime-complete if
roles in NRs are **simple** (don't have transitive subroles)

SHIQ is undecidable with non-simple roles in NRs

Restricting NRs to simple roles is indeed necessary

Proof: *SHIQ* without inverse roles and with un-qualifying NRs only, but with non-simple roles in NRs is **undecidable**

by reduction of the domino problem:

For a domino system $\mathcal{D} = (D, H, V)$, define a TBox $\mathcal{T}_{\mathcal{D}}$ and a *SHIQ*-concept $C_{\mathcal{D}}$ such that

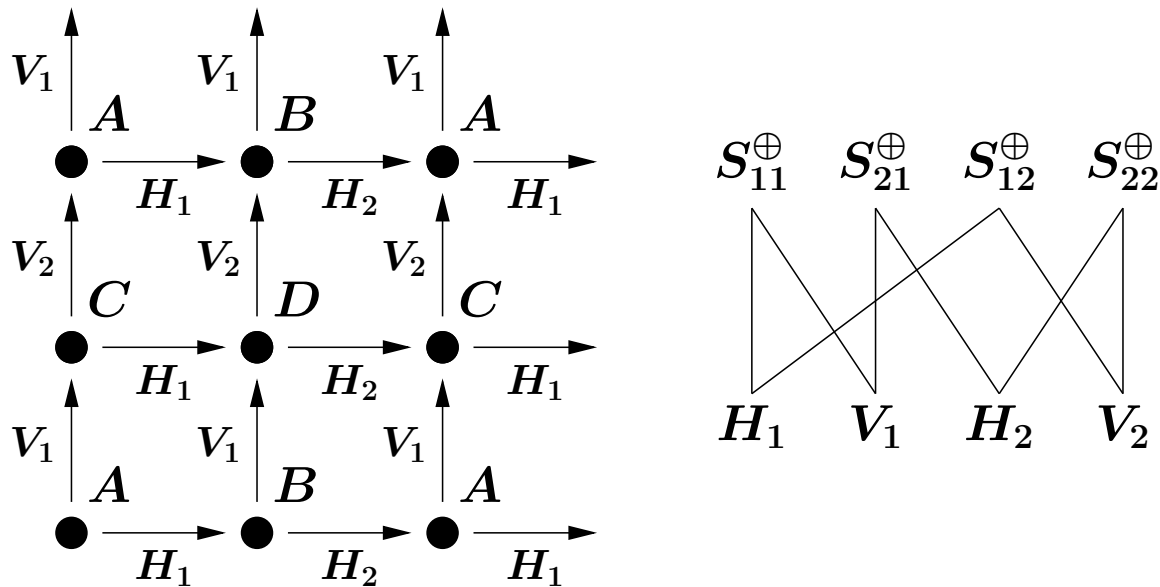
$\mathcal{D}$ has a tiling (of the whole grid) iff $C_{\mathcal{D}}$ is satisfiable w.r.t. $\mathcal{T}_{\mathcal{D}}$

We can (almost) re-use the concept equations

$$\begin{aligned} \top &\doteq \left(\bigsqcup_{1 \leq i \leq n} D_i \right) \sqcap \left(\bigsqcap_{1 \leq i < j \leq n} \neg(D_i \sqcap D_j) \right) \\ \top &\doteq \bigsqcap_{1 \leq i \leq n} D_i \Rightarrow ((\leq 1 V. \top) \sqcap (\exists V. \bigsqcup_{(D_i, D_j) \in V} D_j) \sqcap \\ &\quad \bigsqcap_{1 \leq i \leq n} D_i \Rightarrow ((\leq 1 H. \top) \sqcap (\exists H. \bigsqcup_{(D_i, D_j) \in H} D_j))) \end{aligned}$$

$\mathcal{SHIQ}$ is undecidable with non-simple roles in NRs

... and impose grid-structure as follows:



$$\top \dot{=} A \Rightarrow (\neg B \sqcap \neg C \sqcap \neg D \sqcap \exists H_1.B \sqcap \exists V_1.C \sqcap \leq 3S_{11}),$$

$$\top \dot{=} B \Rightarrow (\neg A \sqcap \neg C \sqcap \neg D \sqcap \exists H_2.A \sqcap \exists V_1.D \sqcap \leq 3S_{21}),$$

$$\top \dot{=} C \Rightarrow (\neg A \sqcap \neg B \sqcap \neg D \sqcap \exists H_1.D \sqcap \exists V_2.A \sqcap \leq 3S_{12}),$$

...

Description Logics

Carsten Lutz

TU Dresden, Germany

Ulrike Sattler

Univ. of Manchester, UK

Part Va: SubBoolean DLs



I. Applications

II. Many Extensions

- concrete domains: numbers and arithmetics in concepts
- temporal / epistemic / non-monotonic DLs
- some constructors: Booleans on roles, role value maps, etc.
- DLs with n -ary relations

III. Sub-Boolean DLs

- DLs below $\mathcal{ALC}$ (tractable?!)
- Non-standard reasoning services



SubBoolean DLs

Even good old $\mathcal{ALC}$ is PSPACE-complete / EXPTIME-complete

$\Rightarrow$ What if tractability is needed? (e.g. in large-scale applications)

Name	Operators	Satisfiability	Subsumption	Subsumption TBoxes
$\mathcal{EL}$	$\sqcap, \exists, \top$	—	P _{TIME}	P _{TIME}
$\mathcal{FL}_0$	$\sqcap, \forall, \top$	—	P _{TIME}	PSPACE
$\mathcal{FL}$	$\sqcap, \forall, \exists^\top, \top$	—	PSPACE	$\geq$ PSPACE
—	$\sqcap, \sqcup$	—	co-NP	—



SubBoolean DLs

$\mathcal{EL}$ ($\sqcap, \exists, \top$) is used in medical and biological ontologies

$$\top \doteq \text{Inflammation} \rightarrow (\text{Disease} \sqcap \exists \text{acts-on.Tissue})$$
$$\text{HeartDisease} \doteq \text{Disease} \sqcap \exists \text{has-loc.} \exists \text{comp-of.Heart}$$
$$\top \doteq \text{Retina} \rightarrow (\text{Tissue} \sqcap \exists \text{has-loc.Eye})$$

Every $\mathcal{EL}$ -concept is satisfiable w.r.t. every $\mathcal{EL}$ -TBox

Goal: show that subsumption w.r.t. TBoxes is in P_{TIME}

1. Normalize TBox
2. Compute “Implication Sets”
3. Read off subsumption relationships



TBox Normalization I

Assumptions:

- consider only the subsumption of concept **names** w.r.t. TBoxes

$$C \sqsubseteq_{\mathcal{T}} D \text{ iff } A \sqsubseteq_{\mathcal{T}'} B, \text{ where } \mathcal{T}' = \mathcal{T} \cup \{A \dot{=} C, B \dot{=} D\}$$

- TBox assertions have the form $C \sqsubseteq D$

$$\mathcal{I} \models C \sqsubseteq D \text{ iff } C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$$

$$C \dot{=} D \text{ equivalent to } C \sqsubseteq D, D \sqsubseteq C$$

Normalized TBox: only assertions of the following form:

$$A \sqsubseteq B$$

$$A \sqcap A' \sqsubseteq B$$

$$A \sqsubseteq \exists R.B$$

$$\exists R.A \sqsubseteq B$$

A, A', B : concept **names** or $\top$



Normalization can be achieved by introducing new concept names:

Let $C \sqsubseteq D \in \mathcal{T}$ with C, D complex

Top-most operator in D (C treated analogously):

- $C = \exists R.E$ $C \sqsubseteq D \rightsquigarrow A \sqsubseteq E, E \sqsubseteq A, C \sqsubseteq \exists R.A$
- $C = E \sqcap F$ introduce "new names" for E and F

Additional case:

$C \sqsubseteq D \sqcap E$ replaced by $C \sqsubseteq D$ and $C \sqsubseteq E$



Implication Sets

Compute an **implication set** $S(A)$ for every concept name A in $\mathcal{T}$

Intuition: $S(A)$ contains all concept names B with $A \sqsubseteq_{\mathcal{T}} B$

● Initially, $S(A) = \{A\}$

● Then saturate $S(A)$ using these rules:

R1 $A' \in S(A) \quad A' \sqsubseteq B \in \mathcal{T} \quad \rightsquigarrow \quad \text{add } B \text{ to } S(A)$

R2 $A_1, A_2 \in S(A) \quad A_1 \sqcap A_2 \sqsubseteq B \in \mathcal{T} \quad \rightsquigarrow \quad \text{add } B \text{ to } S(A)$

R3 $A' \in S(A) \quad A' \sqsubseteq \exists R.B \in \mathcal{T} \quad \rightsquigarrow \quad \text{add } A \text{ to } S(A)$
 $B' \in S(B) \quad \exists R.B' \sqsubseteq A'' \in \mathcal{T}$



Implication Sets II

Lemma. 1. $B \in S(A)$ iff $A \sqsubseteq_{\mathcal{T}} B$.

2. the implication sets can be computed in polynomial time.

ad 2.

- there are at most $|\mathcal{T}|$ implication sets to be computed;
- each implication set contains at most $|\mathcal{T}|$ elements

$\Rightarrow$ at most $|\mathcal{T}|^2$ rule applications

- searching for an applicable rule needs time at most $|\mathcal{T}|^2$

$\Rightarrow$ algorithm runs in time $|\mathcal{T}|^4$



Extensions of $\mathcal{EL}$

Negative results (all with TBoxes):

- $\mathcal{EL} + “\sqcup”$: CO-NP-hard
- $\mathcal{EL} + “\forall R.C”$: EXPTIME-complete
- $\mathcal{EL} +$ inverse roles: PSPACE-hard
- $\mathcal{EL} +$ atomic negation: EXPTIME-complete
- $\mathcal{EL} +$ functional roles: NP-hard

Positive result: we can add the following (simultaneously):

- nominals
- role inclusions $R_1 \circ \dots \circ R_k \sqsubseteq R$
 - $\Rightarrow$ role hierarchies and transitive roles
- a weak form of concrete domains
- $\perp$
- transitive closure



Non-Standard Inferences

Reasoning tasks “satisfiability” and “subsumption” have longest tradition in DL

But there are also “non-standard-inferences”:

- main purpose: provide support for engineering large ontologies
- often only applicable to small languages
or not well-understood for expressive languages
- often do not correspond to standard reasoning in other areas of logic
- usually computation problems, not decision problems

Here we consider: “Least Common Subsumer” and “Most Specific Concept”



Least Common Subsumer (LCS)

Intuition:

The LCS of two concepts C and D is a concept describing
the **commonalities** of C and D .

Helps to build engineer large knowledge bases:

if the concept hierarchy is flat, the **system** can propose intermediary concepts

Definition:

E is the **LCS** of C and D if

1. $C \sqsubseteq E$ and $D \sqsubseteq E$
2. for every F with $C \sqsubseteq F$ and $D \sqsubseteq F$, we have $E \sqsubseteq F$.



Least Common Subsumer (LCS) II

Example: $C = \exists R.A \sqcap \exists R.B \sqcap \exists R.C$

$D = \exists R.(A \sqcap B)$

● LCS of C and D in $\mathcal{EL}$: $\exists R.A \sqcap \exists R.B$

● LCS of C and D in $\mathcal{ALC}$: $C \sqcup D$

$\Rightarrow$ LCS only meaningful in DLs without disjunction

LCS often exists and is computable, but can be hard huge

$\mathcal{FL}_0$: LCS polynomial

$\mathcal{EL}$: binary LCS polynomial; n -ary LCS can be exponential

$\mathcal{ALE}$ [$(\neg)$, $\sqcap$, $\forall$, and $\exists$]: binary LCS can be exponential



Most Specific Concept (MSC)

Idea:

Allow the knowledge engineer to define concepts by giving examples:

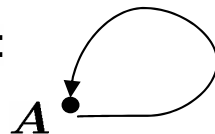
1. examples given by ABox individuals
2. for each example, compute the most specific concept describing the individual
3. extract their commonalities by computing the LCS

Definition:

The **MSC** of an individual a in an ABox $\mathcal{A}$ is the concept C such that

1. $\mathcal{A} \models a : C$
2. for each D with $\mathcal{A} \models a : D$, we have $C \sqsubseteq D$

Exists in $\mathcal{FL}_0$, but not in $\mathcal{EL}$:



$\Rightarrow$ consider only acyclic ABoxes or the “ k -approximation”



LCS does not always exist in the presence of TBoxes:

$$\begin{array}{l} A \doteq \exists R.A \\ B \doteq \exists R.B \end{array} \quad \text{LCS}(A, B) \text{ w.r.t. } \mathcal{T}: ???$$

Thus we work without TBoxes here.

Strategy:

1. Represent $\mathcal{EL}$ -concepts by $\mathcal{EL}$ -trees
2. Characterize subsumption in terms of **homomorphisms** between $\mathcal{EL}$ -trees
3. Compute the LCS as the **product** of $\mathcal{EL}$ -trees.



$\mathcal{EL}$ -trees

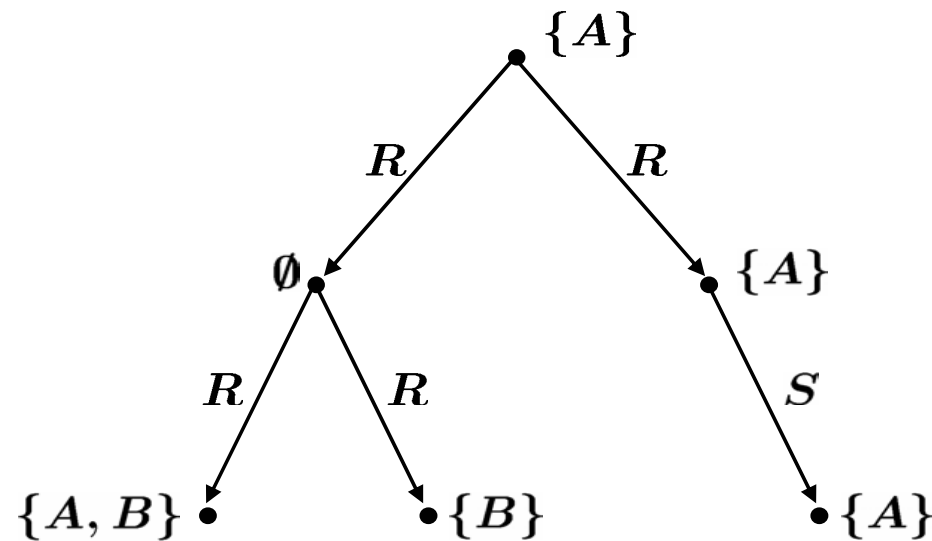
$\mathcal{EL}$ -trees: labelled tree $T = (V, E, \ell)$, where ℓ labels

- $E \subseteq V \times \text{Roles} \times V$
- ℓ labels each node with a set of concept names

For example:

$$A \sqcap \exists R.(\exists R.(A \sqcap B) \sqcap \exists S.B) \sqcap \exists R.(A \sqcap \exists S.A)$$

becomes:



Subsumption Through Homomorphisms

Let $T = (V, E, \ell)$ and $T' = (V', E', \ell')$

$\mathcal{EL}$ -homomorphism from T to T' :

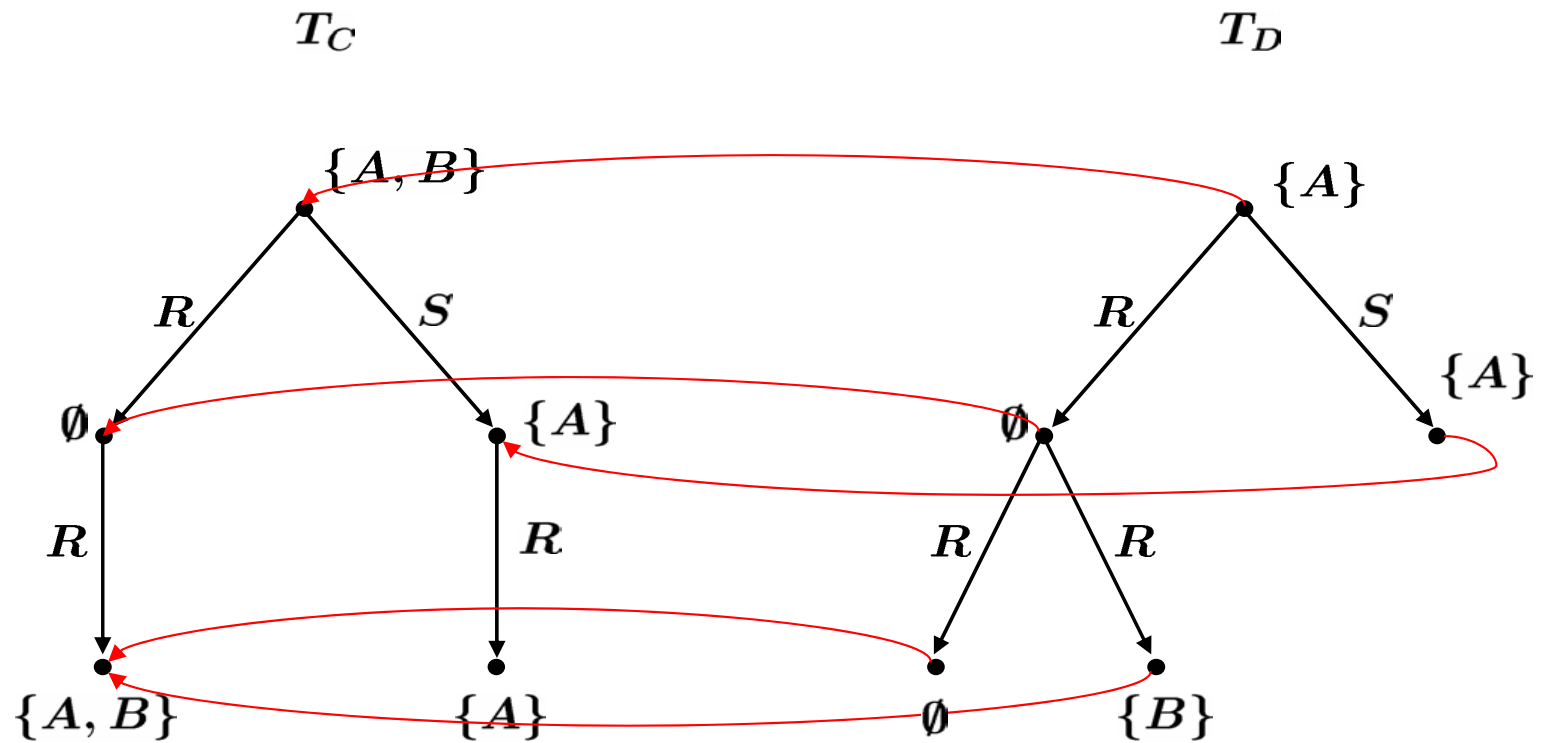
mapping $h : V \rightarrow V'$ such that

- the root of T is mapped to the root of T'
- $(x, R, y) \in E \implies (h(x), R, h(y)) \in E'$
- $\ell(x) \subseteq \ell'(h(x))$

Lemma. $C \sqsubseteq D$ iff there exists an $\mathcal{EL}$ -homomorphism from T_D to T_C .



Subsumption Through Homomorphisms II



$C \sqsubseteq D!!!$



LCS as Product

Let $T = (V, E, \ell)$ and $T' = (V', E', \ell')$ with roots r and r'

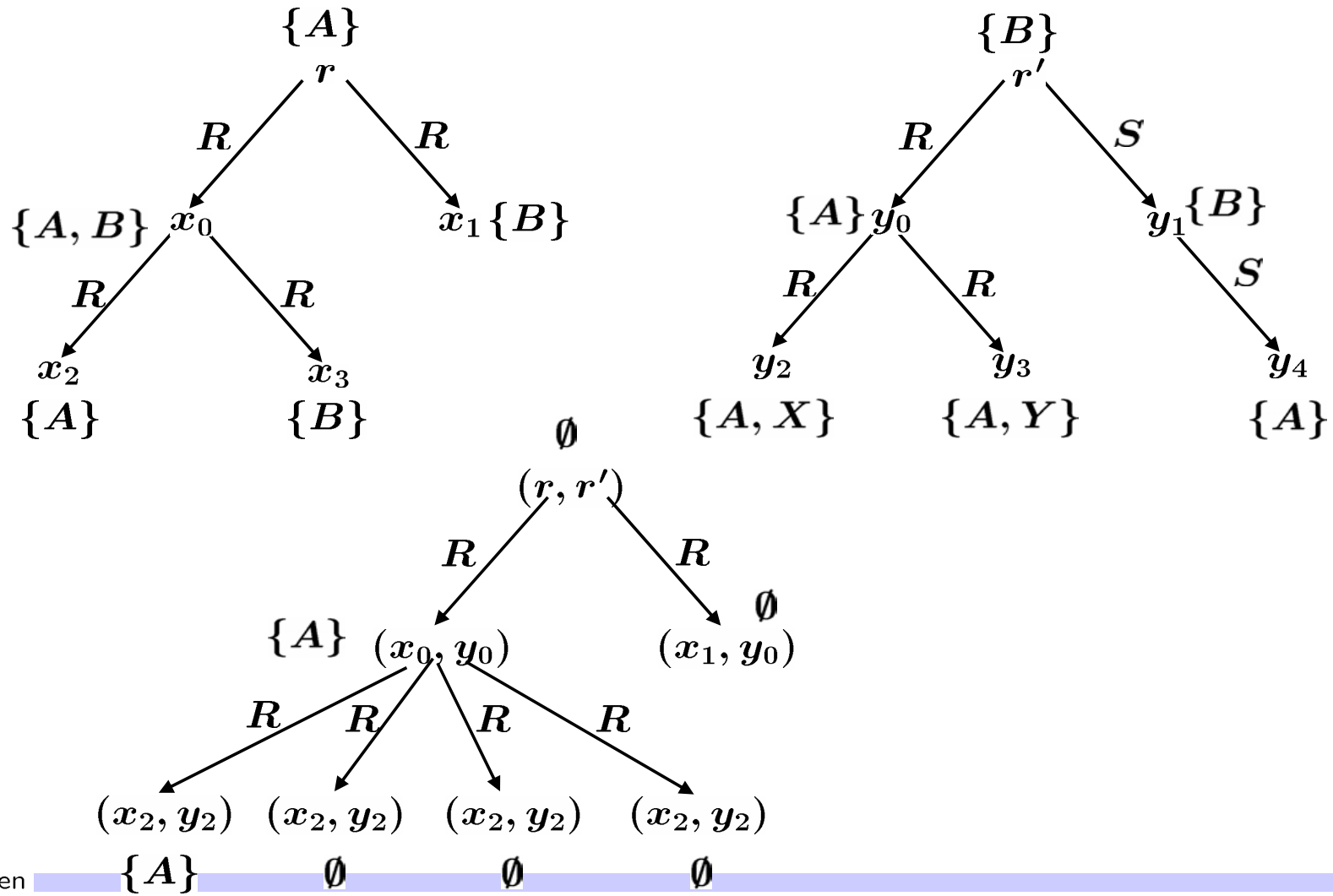
Their **product** $T \times T'$ is defined inductively:

- $T_1 \times T_2$ has root (r, r')
- $\ell(r, r') = \ell(r) \cap \ell'(r')$
- for each pair of edges $(r, R, x) \in E$ and $(r', R, y) \in E'$
in $T \times T'$ there is an edge $((r, r'), R, (x, y))$
with (x, y) root of $T|_x \times T'|_y$

Lemma. The LCS of C and D corresponds to the product $T_C \times T_D$



LCS as Product II



LCS as Product III

Lemma. The LCS of C and D corresponds to the product $T_C \times T_D$

Proof idea:

denote with E the concept corresponding to $T_C \times T_D$

1. $C \sqsubseteq E$ and $D \sqsubseteq E$

$\mathcal{EL}$ -homomorphisms from T_E to T_C and T_D : projection
(map (x, y) to x and y , respectively)

2. $C \sqsubseteq F$ and $D \sqsubseteq F$ implies $E \sqsubseteq F$

there exist $\mathcal{EL}$ -homomorphisms h_C from T_F to T_C
and h_D from T_F to T_D

$\mathcal{EL}$ -homomorphisms from T_F to T_E :

map each node x of T_F to $(h_C(x), h_D(x))$



Want more?

More information:

- The Description Logic Handbook
- Our slides (available on the web at <http://lat.inf.tu-dresden.de/~clu/>)
- Send us a mail: sattler@cs.man.ac.uk
lutz@tcs.inf.tu-dresden.de



n -ary Products of $\mathcal{EL}$ -trees

150



TU
Dresden

5.a: Two Applications of DLs

1. DLs for reasoning about (database) conceptual models
2. DLs for ontology engineering

State of the art systems are
highly optimised versions of the tableau algorithm for $\mathcal{SHIQ}$: FaCT and Racer

Optimisations implemented in FaCT and Racer

- are known from other “logic implementations”,
- include minimisation of subsumption tests for the classification of all concepts defined in the TBox
- include SAT-heuristics, e.g., semantic branching or Davis-Putnam,
- include dependency-directed backtracking
(non-determinism is implemented through search with backtracking)
- include strong pre-processing of TBox (localisation/absorption of CEs)
- include optimised double-blocking conditions
(otherwise, blocking occurs dramatically too late)

DLs for conceptual modelling

Idea: Translate EER/UML database conceptual model E into $\mathcal{SHIQ}$ TBox $\mathcal{T}_E$:

each model of E is a model of $\mathcal{T}_E$ and vice versa.

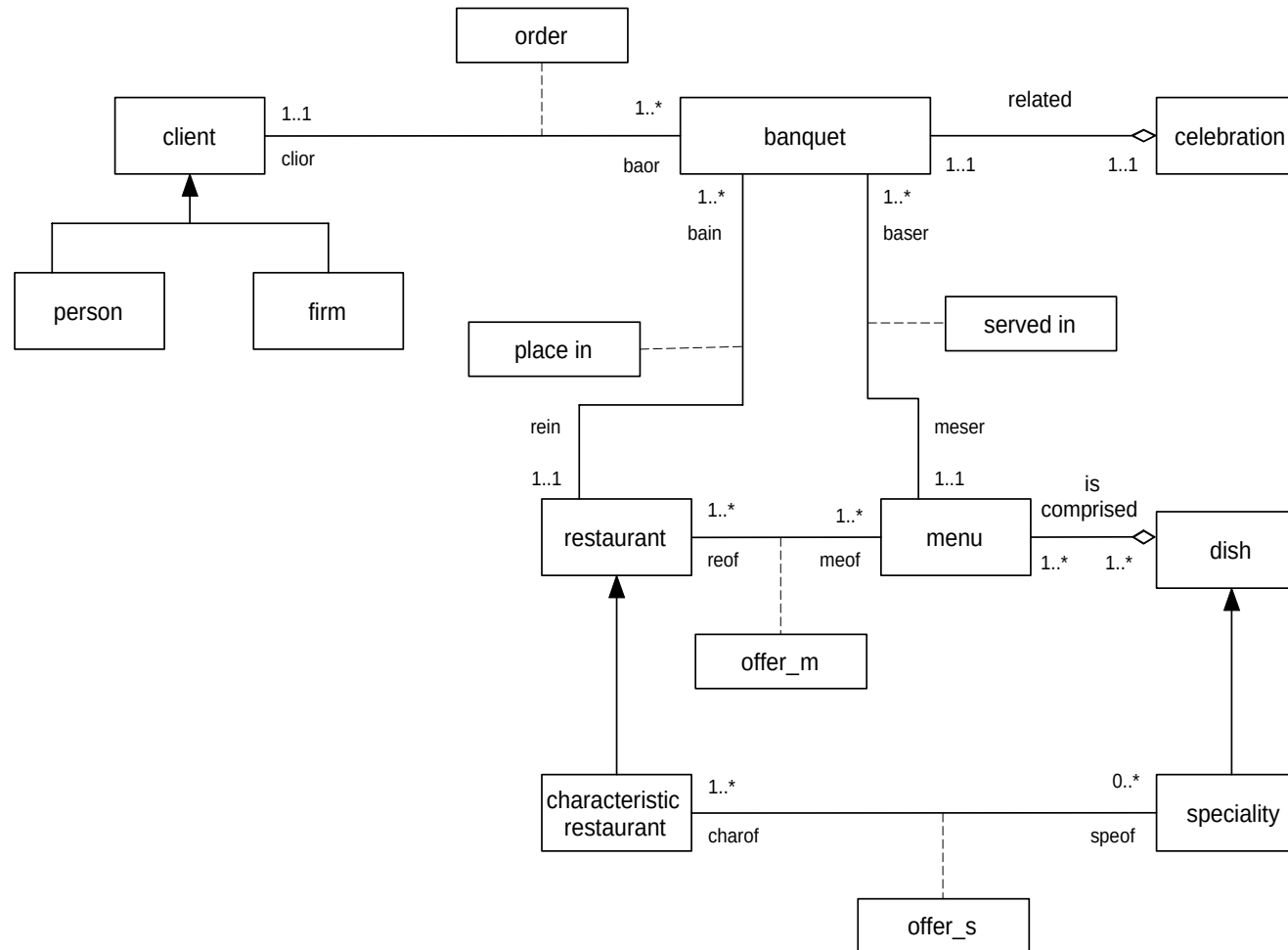
use DL system to infer inconsistencies and implicit subsumption relations
useful for integrating databases/in datawarehousing

Technicalities: If only **binary** relations are present in E ,
translation into $\mathcal{SHIQ}$ is rather straightforward

If n -ary relations are present in E ,
translation into $\mathcal{SHIQ}$ requires **reification** of relations

DLs for conceptual modelling

Example:



SHIQ TBox for the restaurant UML

$$\text{client} \doteq (\geq 1 \text{order}.\top) \sqcap (\forall \text{order}.\text{banquet})$$

$$\begin{aligned} \text{banquet} \doteq & (= 1 \text{order}^{\neg}.\top) \sqcap (\forall \text{order}^{\neg}.\text{client}) \sqcap \\ & (= 1 \text{placed-in}.\top) \sqcap (\forall \text{placed-in}.\text{restaurant}) \sqcap \\ & (= 1 \text{served-in}^{\neg}.\top) \sqcap (\forall \text{served-in}^{\neg}.\text{menu}) \\ & (= 1 \text{related-to}.\top) \sqcap (\forall \text{related-to}.\text{celebration}) \end{aligned}$$

$$\text{celebration} \doteq (= 1 \text{related-to}^{\neg}.\top) \sqcap (\forall \text{related-to}^{\neg}.\text{banquet})$$

$$\text{person} \sqsubseteq^{\dot{}} \text{client} \sqcap \neg \text{firm}$$

$$\text{firm} \sqsubseteq^{\dot{}} \text{client} \sqcap \neg \text{person}$$

$$\vdots \quad \vdots$$

System demo of i.com,
a tool for intelligent conceptual modeling,
implemented by Enrico Franconi and Gary Ng
using FaCT by Ian Horrocks

Ontologies are formalization of a conceptualization
fix the relevant concepts of an application domain (like a TBox)
are widely used in enterprise knowledge management systems
will widely be used in the semantic web

Useful ontologies

- should be consistent and not redundant
- are built possibly by several knowledge engineers
- evolve over time
- are possibly integrated with other ontologies

An ontology language

- should have a clear semantics $\rightsquigarrow$ **logic**
- should allow for tractable inference services to support the design, evolution, maintenance, and integration of ontologies

DAML+OIL, OWL, and OIL are DL-based ontology language

System demo of **OilEd**,
a tool for ontology engineering,
implemented by Sean Bechhofer and Gary Ng
using **FaCT** by Ian Horrocks