Description Logics: background

What are Description Logics?

There is no precise definition of what a description logic is. They form a **huge family** of logic-based **knowledge representation formalisms** with a number of common properties:

- They are descendants of **semantic networks** and **KL-ONE** from the 1960-70s.
- They describe a domain of interest in terms of
  - **concepts** (also called classes),
  - **roles** (also called relations or properties),
  - **individuals**
- Modulo a simple translation, they are subsets of predicate logic.
- Distinction between terminology and data (see next slide).

DL architecture

Knowledge Base (KB)

TBox (terminological box, schema)

$$\mathsf{Man} \equiv \mathsf{Human} \sqcap \mathsf{Male}$$
$$\mathsf{Father} \equiv \mathsf{Man} \sqcap \exists \mathsf{hasChild}.\top$$
...

ABox (assertion box, data)

$$\mathsf{john} : \mathsf{Man}$$
$$(\mathsf{john}, \mathsf{mary}) : \mathsf{hasChild}$$
...

Reasoning System

Description Logics to be discussed

We first discuss the **terminological part** of the description logics

- $\mathcal{EL}$ (the DL underpinning OWL2 EL);
- DL-Lite (the DL underpinning OWL2 QL);
- The DL underpinning Schema.org;
- $\mathcal{ALC}$ and some extensions (the DL underpinning OWL2).

We will later discuss how description logics are used to access **instance data**.

$\mathcal{ALC}$ TBox

Definition (Syntax)

Let C and D be possibly complex $\mathcal{ALC}$ concepts.

- $C \sqsubseteq D$ (general concept inclusion, or simply GCI)
- $C \equiv D$ (concept equivalence, shortcut for $C \sqsubseteq D$ and $D \sqsubseteq C$)

Both GCI and concept equivalences are called Axioms.
An $\mathcal{ALC}$ TBox $\mathcal{T}$ is a finite set of GCIs.

Definition (Semantics)

Let $\mathcal{I}$ be an interpretation. $\mathcal{I}$ satisfies a GCI $C \sqsubseteq D$ ($C \sqsubseteq D$ is true in $\mathcal{I}$), written $\mathcal{I} \models C \sqsubseteq D$, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.
In this case, $\mathcal{I}$ is called a model of $C \sqsubseteq D$.
$\mathcal{I}$ is a model of $\mathcal{T}$ iff it satisfies each GCI in $\mathcal{T}$.

Examples of $\mathcal{ALC}$ TBox

Interpretation $\mathcal{I}_1$:

$$\begin{aligned}
\Delta^{\mathcal{I}_1} &= \{m, c_6, c_7, et\}, \\
\mathsf{Teacher}^{\mathcal{I}_1} &= \{m\}, \\
\mathsf{Course}^{\mathcal{I}_1} &= \{c_6, c_7, et\}, \\
\mathsf{Person}^{\mathcal{I}_1} &= \{m, et\}, \\
\mathsf{UGC}^{\mathcal{I}_1} &= \{c_7\}, \\
\mathsf{teaches}^{\mathcal{I}_1} &= \{(m, c_6), (m, c_7), (et, et)\}.
\end{aligned}$$

Interpretation $\mathcal{I}_2$:

$$\begin{aligned}
\mathsf{Teacher}^{\mathcal{I}_2} &= \{m\}, \\
\mathsf{Course}^{\mathcal{I}_2} &= \{c_6, c_7\}, \\
\mathsf{Person}^{\mathcal{I}_2} &= \{m, et\}, \\
\mathsf{UGC}^{\mathcal{I}_2} &= \{c_7\}, \\
\mathsf{teaches}^{\mathcal{I}_2} &= \{(m, c_6), (m, c_7), (et, et)\}.
\end{aligned}$$

Examples of $\mathcal{ALC}$ TBox

A TBox $\mathcal{T}_1$:

$$\begin{aligned} \mathsf{Teacher} &\sqsubseteq \mathsf{Person} \\ \mathsf{UGC} &\sqsubseteq \neg\mathsf{Person} \\ \mathsf{Teacher} &\sqsubseteq \exists\mathsf{teaches}.\mathsf{Course} \\ \exists\mathsf{teaches}.\mathsf{Course} &\sqsubseteq \mathsf{Person} \end{aligned}$$

Both $\mathcal{I}_1$ and $\mathcal{I}_2$ are models of $\mathcal{T}_1$.

A TBox $\mathcal{T}_2$ extending $\mathcal{T}_1$ by:

$$\begin{aligned} \mathsf{Course} &\sqsubseteq \neg\mathsf{Person} \\ \exists\mathsf{teaches}.\mathsf{Course} &\sqsubseteq \mathsf{Teacher} \end{aligned}$$

$\mathcal{I}_1$ is no longer a model of $\mathcal{T}_1$, but $\mathcal{I}_2$ is.

$\mathcal{ALC}$ ABox

Definition (Syntax)

Let a and b be two individuals, C a possibly complex $\mathcal{ALC}$ concept, and r a role name:

- $a : C$ (concept assertion, sometimes written $C(a)$)
- $(a, b) : r$ (role assertion, sometime written $r(a, b)$)

Both are called Assertions.
An $\mathcal{ALC}$ ABox $\mathcal{A}$ is a finite set of concept and role assertions.

Definition (Semantics)

Let $\mathcal{I}$ be an interpretation. $\mathcal{I}$ satisfies a concept assertion $a : C$ ($a : C$ is true in $\mathcal{I}$), written $\mathcal{I} \models a : C$, if $a^{\mathcal{I}} \in C^{\mathcal{I}}$. $\mathcal{I}$ satisfies a role assertion $(a, b) : r$, written $\mathcal{I} \models (a, b) : r$, if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$.
In this case, $\mathcal{I}$ is called a model of the assertion.
$\mathcal{I}$ is a model of $\mathcal{A}$ iff it satisfies each assertion in $\mathcal{A}$.

Basic Reasoning Problems

Definition

Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be an $\mathcal{ALC}$ knowledge base, C and D be possibly $\mathcal{ALC}$ concepts, and a an individual name. We say that:

- C is <u>satisfiable</u> with respect to $\mathcal{T}$ if there exists a model $\mathcal{I}$ of $\mathcal{T}$ and some $d \in \Delta^{\mathcal{I}}$ with $d \in C^{\mathcal{I}}$;
- C is <u>subsumed</u> by D with respect to $\mathcal{T}$, written $\mathcal{T} \models C \sqsubseteq D$ or $C \sqsubseteq_{\mathcal{T}} D$, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model of $\mathcal{T}$;
- $\mathcal{K}$ is <u>consistent</u> (<u>satisfiable</u>) if there exists a model of $\mathcal{K}$;
- a is an <u>instance</u> of C with respect to $\mathcal{K}$, written $\mathcal{K} \models a : C$, if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ for every model of $\mathcal{K}$.

Reasoning for $\mathcal{ALC}$ (without TBox)

We first consider reasoning without TBoxes:

- **Subsumption**. We say that a concept inclusion $C \sqsubseteq D$ follows from the empty TBox (or that C is subsumed by D) if, and only if, for all interpretations $\mathcal{I}$ we have that $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. In this case, we often write $\emptyset \models C \sqsubseteq D$.

- **Concept satisfiability**. A concept C is satisfiable if, and only if, there exists an interpretation $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \emptyset$.

We have: $\emptyset \models C \sqsubseteq D$ if, and only if, $C \sqcap \neg D$ is not satisfiable. Thus, in $\mathcal{ALC}$, subsumption is reducible to concept satisfiability.

We give an algorithm deciding whether a $\mathcal{ALC}$-concept C is satisfiable.

Remark This problem is *not* tractable. Its complexity is between NP-complete and ExpTime-complete (precisely: PSpace-complete). The algorithm we present requires exponential time.

Satisfiability of Concepts: example 1

Q: Is $\boxed{(\forall \text{hasChild.Male}) \sqcap (\exists \text{hasChild.}\neg\text{Male})}$ satisfiable?

Let us try to construct an **interpretation** satisfying this concept

(1)		$x : (\forall \text{hasChild.Male}) \sqcap (\exists \text{hasChild.}\neg\text{Male})$
(2)	from (1)	$x : \forall \text{hasChild.Male}$
(3)	from (1)	$x : \exists \text{hasChild.}\neg\text{Male}$
(4)	from (3)	$(x, y) :$ hasChild and $y : \neg\text{Male}$, for fresh y
(5)	from (2) & (4)	$y :$ Male
(6)	from (4) & (5)	**contradiction:** $y :$ Male and $y : \neg\text{Male}$

A: the concept is **not satisfiable!**

Satisfiability of Concepts: example 2

Q: Is $\boxed{(\forall \text{hasChild.Male}) \sqcap (\exists \text{hasChild.Male})}$ satisfiable?

Let us try to construct a **interpretation** satisfying this concept

(1)		x : $(\forall \text{hasChild.Male}) \sqcap (\exists \text{hasChild.Male})$
(2)	from (1)	x : $\forall \text{hasChild.Male}$
(3)	from (1)	x : $\exists \text{hasChild.Male}$
(4)	from (3)	(x, y) : hasChild and y : Male, for fresh y

A: the concept is **satisfiable** and a satisfying model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is

$$\Delta^{\mathcal{I}} = \{x, y\}, \qquad \text{Male}^{\mathcal{I}} = \{y\}, \qquad \text{hasChild}^{\mathcal{I}} = \{(x, y)\}$$

Then $x \in \big((\forall \text{hasChild.Male}) \sqcap (\exists \text{hasChild.Male})\big)^{\mathcal{I}}$

Satisfiability of Concepts: example 3

Q: Is $\boxed{\forall r.(\neg C \sqcup D) \sqcap \exists r.(C \sqcap D)}$ satisfiable?

(1)		$x : \forall r.(\neg C \sqcup D) \sqcap \exists r.(C \sqcap D)$
(2)	from (1)	$x : \forall r.(\neg C \sqcup D)$
(3)	from (1)	$x : \exists r.(C \sqcap D)$
(4)	from (3)	$(x, y) : r$ and $y : C \sqcap D$, for fresh y
(5)	from (4)	$y : C$
(6)	from (4)	$y : D$
(7)	from (2)	$y : \neg C \sqcup D$

Two ways of continue **(branching!)**:

(8.1)	from (7)	$y : \neg C$
(8.2)	from (7)	$y : D$

A: (8.1) is a **contradiction**, while (8.2) is not and yields a **satisfying model**

Tableau Methods

How can we prove satisfiability of a concept?

Achieved by applying **tableau methods**
(set of **completion rules** operating on **constraint systems** or **tableaux**)

Proof procedure:

- transform a given concept into **Negation Normal Form** (NNF)
  (all occurrences of negations are in front of concept names)
- apply **completion rules** in arbitrary order as long as possible.
- the concept is **satisfiable** if, and only if, a **clash-free** tableau can be derived to which no completion rule is applicable.

Negation Normal Form (NNF)

A concept is in **Negation Normal Form** (NNF)
if all occurrences of negations in it are in front of *concept names*

Every $\mathcal{ALC}$-concept can be transformed into an equivalent one in NNF
using the following rules:

$$
\begin{array}{rcll}
\neg\top & \equiv & \bot & \\
\neg\bot & \equiv & \top & \\
\neg\neg C & \equiv & C & \\
\neg(C \sqcap D) & \equiv & \neg C \sqcup \neg D & \text{(De Morgan's law)} \\
\neg(C \sqcup D) & \equiv & \neg C \sqcap \neg D & \text{(De Morgan's law)} \\
\neg\forall r.C & \equiv & \exists r.\neg C & \\
\neg\exists r.C & \equiv & \forall r.\neg C &
\end{array}
$$

Negation Normal Form: example

Transform the concept

$$\neg\exists r.(A \sqcap \neg B) \ \sqcup \ \neg\forall r.(\neg A \sqcup \neg B)$$

to an equivalent concept in negation normal form.

$\neg\exists r.(A \sqcap \neg B) \ \sqcup \ \neg\forall r.(\neg A \sqcup \neg B) \ \equiv$ (use $\neg\exists r.D \ \equiv \ \forall r.\neg D$)

$\forall r.\neg(A \sqcap \neg B) \ \sqcup \ \neg\forall r.(\neg A \sqcup \neg B) \ \equiv$ (use $\neg(A \sqcap D) \ \equiv \ \neg A \sqcup \neg D$)

$\forall r.(\neg A \sqcup \neg\neg B) \ \sqcup \ \neg\forall r.(\neg A \sqcup \neg B) \ \equiv$ (use $\neg\neg B \ \equiv \ B$)

$\forall r.(\neg A \sqcup B) \ \sqcup \ \neg\forall r.(\neg A \sqcup \neg B) \ \equiv$ (use $\neg\forall r.D \ \equiv \ \exists r.\neg D$)

$\forall r.(\neg A \sqcup B) \ \sqcup \ \exists r.\neg(\neg A \sqcup \neg B) \ \equiv$ (use $\neg(C \sqcup D) \ \equiv \ \neg C \sqcap \neg D$)

$\forall r.(\neg A \sqcup B) \ \sqcup \ \exists r.(\neg\neg A \sqcap \neg\neg B) \ \equiv$ (use $\neg\neg C \ \equiv \ C$)

$\forall r.(\neg A \sqcup B) \ \sqcup \ \exists r.(A \sqcap B)$

Tableau Calculus for $\mathcal{ALC}$ concept satisfiability

Constraint: expression of the form $x\colon C$ or $(x,y)\colon r$,
where C is a concept in NNF and r a role name

Constraint system: a finite non-empty set S of constraints

Completion rules: $S \to S'$, where S' is a constraint system containing S

Clash: S contains **clash** if

$\{\, x\colon A, \quad x\colon \neg A \,\} \subseteq S$, for some x and concept name A

Aim: starting from $S_0 = \{x\colon C\}$ apply completion rules to construct a clash-free system S_n to which no completion rule is applicable

- If this is possible, then we can extract a **model satisfying** C
- Otherwise, C is **not satisfiable**.

Completion Rules for $\mathcal{ALC}$ concept satisfiability (1)

$$S \;\rightarrow_{\sqcap}\; S \cup \{\, x\colon C,\; x\colon D \,\}$$

if (a) $x\colon C \sqcap D$ is in S

(b) $x\colon C$ and $x\colon D$ are not both in S

$$S \;\rightarrow_{\sqcup}\; S \cup \{\, x\colon E \,\}$$

if (a) $x\colon C \sqcup D$ is in S

(b) neither $x\colon C$ nor $x\colon D$ is in S

(c) $E = C$ or $E = D$ **(branching!)**

NB: Non-deterministically add any of the disjuncts to the constraint system

NB: Clashes eliminate branches in the OR tree

Completion Rules for $\mathcal{ALC}$ concept satisfiability (2)

$$S \;\rightarrow_{\forall}\; S \cup \{\, y : C \,\}$$

if (a) $x : \forall r.C$ is in S

(b) $(x, y) : r$ is in S

(c) $y : C$ is not in S

NB: Only applicable if role successors can be found

$$S \;\rightarrow_{\exists}\; S \cup \{\, (x, y) : r,\; y : C \,\}$$

if (a) $x : \exists r.C$ is in S

(b) y is a fresh individual

(c) there is no z such that both $(x, z) : r$ and $z : C$ are in S

NB: The only rule that creates new individuals in a constraint system

Tableau Example 1

We check whether $(A \sqcap \neg A) \sqcup B$ is satisfiable.

It is in NNF, so we can directly apply the tableau algorithm to

$$S_0 = \{x : (A \sqcap \neg A) \sqcup B\}$$

The only rule applicable is $\rightarrow_\sqcup$. We have two possibilities.

Firstly we can try

$$S_1 = S_0 \cup \{x : A \sqcap \neg A\}.$$

Then we can apply $\rightarrow_\sqcap$ and obtain

$$S_2 = S_1 \cup \{x : A, x : \neg A\}$$

We have obtained a clash, thus this choice was unsuccessful.

Secondly, we can try

$$S_1^* = S_0 \cup \{x : B\}.$$

No rule is applicable to S_1^* and it does not contain a clash. Thus, $(A \sqcap \neg A) \sqcup B$ is satisfiable.

A model $\mathcal{I}$ satisfying it is given by

$$\boldsymbol{\Delta^{\mathcal{I}} = \{x\}, \quad B^{\mathcal{I}} = \{x\}, \quad A^{\mathcal{I}} = \emptyset.}$$

Tableau Example 2

We check whether $C = A \sqcap \exists r.\exists s.B \sqcap \forall r.\neg B$ is satisfiable.

It is in NNF, so we can directly apply the tableau algorithm to

$$S_0 = \{x : A \sqcap \exists r.\exists s.B \sqcap \forall r.\neg B\}$$

An application of $\rightarrow_\sqcap$ gives

$$S_1 = S_0 \cup \{x : A, x : \exists r.\exists s.B \sqcap \forall r.\neg B\}$$

An application of $\rightarrow_\sqcap$ gives

$$S_2 = S_1 \cup \{x : \exists r.\exists s.B, x : \forall r.\neg B\}$$

An application of $\rightarrow_\exists$ gives

$$S_3 = S_2 \cup \{(x, y) : r, y : \exists s.B\}$$

An application of $\rightarrow_{\exists}$ gives

$$\boldsymbol{S_4 = S_3 \cup \{(y, z) : s, z : B\}}$$

Tableau Example 2

Recall that

$$S_4 = S_3 \cup \{(y, z) : s, z : B\}$$

An application of $\rightarrow_\forall$ gives

$$S_5 = S_4 \cup \{y : \neg B\}$$

No rule is applicable to S_5 and S_5 contains no clash. Thus, the concept C is satisfiable.

A model $\mathcal{I}$ of C is given by

$$\Delta^{\mathcal{I}} = \{x, y, z\}, \quad A^{\mathcal{I}} = \{x\}, \quad B^{\mathcal{I}} = \{z\}, \quad r^{\mathcal{I}} = \{(x, y)\}, \quad s^{\mathcal{I}} = \{(y, z)\}$$

Tableau Example 3

We check whether $C = \exists r.A \sqcap \exists r.\neg A$ is satisfiable.

C is in NNF, so we can directly apply the tableau algorithm to

$$S_0 = \{x : \exists r.A \sqcap \exists r.\neg A\}$$

An application of $\rightarrow_\sqcap$ gives

$$S_1 = S_0 \cup \{x : \exists r.A, x : \exists r.\neg A\}$$

An application of $\rightarrow_\exists$ gives

$$S_2 = S_1 \cup \{(x, y) : r, y : A\}$$

Tableau Example 3

Recall that

$$\boldsymbol{S_2 = S_1 \cup \{(x,y) : r, y : A\}}$$

Another application of $\rightarrow_\exists$ gives

$$\boldsymbol{S_3 = S_2 \cup \{(x,z) : r, z : \neg A\}}$$

No rule is applicable to $\boldsymbol{S_3}$ and $\boldsymbol{S_3}$ contains no clash. Thus, $\boldsymbol{C}$ is satisfiable.

A model $\boldsymbol{\mathcal{I}}$ of $\boldsymbol{C}$ is given by

$$\boldsymbol{\Delta^{\mathcal{I}} = \{x, y, z\}, \quad A^{\mathcal{I}} = \{y\}, \quad r^{\mathcal{I}} = \{(x,y),(x,z)\}}$$

Analysing the Tableau Calculus

To show that the tableau does what it is supposed to do one has to show

- Soundness: If the concept is satisfiable, then there is a branch without clash such that no rule is applicable;
- Termination: The tableau terminates after finitely many steps for any input concept in NNF;
- Completeness: If there is a branch without clash such that no rule is applicable, then the concept is satisfiable.

Tableau Calculus: Soundness

- Suppose that a constraint system $\boldsymbol{S}$ is satisfiable and

$$\boldsymbol{S} \rightarrow_{\sqcap} \boldsymbol{S}', \quad \boldsymbol{S} \rightarrow_{\forall} \boldsymbol{S}' \quad \text{or} \quad \boldsymbol{S} \rightarrow_{\exists} \boldsymbol{S}'.$$

  Then $\boldsymbol{S}'$ is also satisfiable.

- If

$$\boldsymbol{S} \rightarrow_{\sqcup} \boldsymbol{S}' \quad \text{and} \quad \boldsymbol{S} \rightarrow_{\sqcup} \boldsymbol{S}''$$

  then one of $\boldsymbol{S}'$ and $\boldsymbol{S}''$ is satisfiable (or perhaps both).

Thus, having started with a satisfiable constraint system
we cannot derive clashes in all branches

Tableau Calculus: Termination

For every constraint system S_0,
there is no infinite sequence of the form

$$S_0, S_1, S_2, \ldots$$

such that S_{i+1} is obtained form S_i
by an application of one of the completion rules

Proof: All rules but $\rightarrow_\forall$ are never applied twice to the same constraint

$\rightarrow_\forall$ is never applied to an individual x more times than
the number of direct successors of x (i.e., y such that $(x, y) : r$),
which is bounded by the length of the concept

Each rule application to a constraint $y : C$
adds constraints $z : D$ such that D is a subconcept of C

Tableau Calculus: Completeness

If starting from $S_0 = \{x : C\}$ and applying the completion rules we construct a **clash-free** constraint system S_n to which no rule is applicable then C is satisfiable

S_n determines an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$:

- $\Delta^{\mathcal{I}}$ contains all individuals in S_n
- for $x \in \Delta^{\mathcal{I}}$ and a concept name A,
$$x \in A^{\mathcal{I}} \quad \text{iff} \quad x : A \text{ is in } S_n$$
- for $x, y \in \Delta^{\mathcal{I}}$ and a role name r,
$$(x, y) \in r^{\mathcal{I}} \quad \text{iff} \quad (x, y) : r \text{ is in } S_n$$

It is easy to check that C is satisfied in $\mathcal{I}$, i.e., $C^{\mathcal{I}} \neq \emptyset$

Reasoning Services for $\mathcal{ALC}$ (with TBox)

- **Subsumption w.r.t. TBoxes**. We say that a concept inclusion $C \sqsubseteq D$ follows from a TBox $\mathcal{T}$ if, and only if, every interpretation $\mathcal{I}$ that is a model of $\mathcal{T}$ is a model of $C \sqsubseteq D$. In this case, we often write $\mathcal{T} \models C \sqsubseteq D$.

- **Concept satisfiability w.r.t. TBoxes**. A concept C is satisfiable w.r.t. a TBox $\mathcal{T}$ if, and only if, there exists an interpretation $\mathcal{I}$ that is a model of $\mathcal{T}$ such that $C^{\mathcal{I}} \neq \emptyset$.

- **TBox satisfiability**. A TBox $\mathcal{T}$ is satisfiable if, and only if, there exists a model of $\mathcal{T}$.

We have the following reductions to concept satisfiability w.r.t. TBoxes:

- $\mathcal{T} \models C \sqsubseteq D$ if, and only if, $C \sqcap \neg D$ is not satisfiable w.r.t. $\mathcal{T}$.

- $\mathcal{T}$ is satisfiable if, and only if, A is satisfiable w.r.t. $\mathcal{T}$ (A a fresh concept name).

Thus, it is sufficient to design an algorithm checking concept satisfiability w.r.t. TBoxes.

Discussion

The concept satisfiability problem w.r.t. $\mathcal{ALC}$-TBoxes is ExpTime-complete. Thus, it is not tractable:

- There is no guarantee that existing implementations (or future implementations) of algorithms for this problem will terminate in a reasonable amount of time for every $\mathcal{ALC}$-TBox.
- Nevertheless, there are a number of systems (FACT, PELLET, RACER) which work for most currently existing TBoxes.

Reasoning with TBoxes

Given a TBox $\mathcal{T}$ and a concept C,

how to determine whether $\mathcal{T} \cup \{ x\colon C \}$ has a model

(**concept satisfiability w.r.t. a TBox**)

Note that, for any interpretation $\mathcal{I}$ and any two concepts C and D,

$$\mathcal{I} \models C \sqsubseteq D \qquad \text{iff} \qquad \mathcal{I} \models \top \sqsubseteq \neg C \sqcup D$$

So, $C \sqsubseteq D$ is equivalent to $\top \sqsubseteq \neg C \sqcup D$.

The **initial constraint system** S_0 for $\mathcal{T} \cup \{ x\colon C \}$ is defined by

$$S_0 = \{ x\colon C \} \cup \{\top \sqsubseteq \neg C \sqcup D \mid C \sqsubseteq D \in \mathcal{T}\}$$

So, now we have three different types of constraints:

$$y\colon D \qquad (x, y)\colon r \qquad \top \sqsubseteq D$$

Reasoning with TBoxes (cont.)

$$\boldsymbol{S} \;\to_U\; \boldsymbol{S} \cup \{\, x : D \,\}$$

if (a) $\top \sqsubseteq D$ is in S

(b) x occurs in S

(c) $x : D$ is not in S

The tableau algorithm based on rules

$$\to_\sqcap, \quad \to_\sqcup, \quad \to_\forall, \quad \to_\exists \quad \text{and} \quad \to_U$$

does not terminate:

in general, even if $\mathcal{T} \cup \{\, x : C \,\}$ has model,

the algorithm can produce an **infinite** model for it

(although finite models exist)

see the next slide for an example. . .

Reasoning with TBoxes: example

$$
\begin{array}{ll}
S_0 & = \{\, x_0 \colon \top, \quad \top \sqsubseteq \exists r.A \,\} \\
S_0 \rightarrow_U S_1 & = S_0 \cup \{\, x_0 \colon \exists r.A \,\} \\
S_1 \rightarrow_\exists S_2 & = S_1 \cup \{\, (x_0, x_1) \colon r, \quad x_1 \colon A \,\} \\
S_2 \rightarrow_U S_3 & = S_2 \cup \{\, x_1 \colon \exists r.A \,\} \\
S_3 \rightarrow_\exists S_4 & = S_3 \cup \{\, (x_1, x_2) \colon r, \quad x_2 \colon A \,\} \\
S_4 \rightarrow_U S_5 & = S_4 \cup \{\, x_2 \colon \exists r.A \,\} \\
\quad \ldots & \ldots
\end{array}
$$

This gives an **infinite model** which can easily be reconstructed into a finite one

Rule $\rightarrow_\exists$ can be modified in such a way that

the resulting algorithm always **terminates**

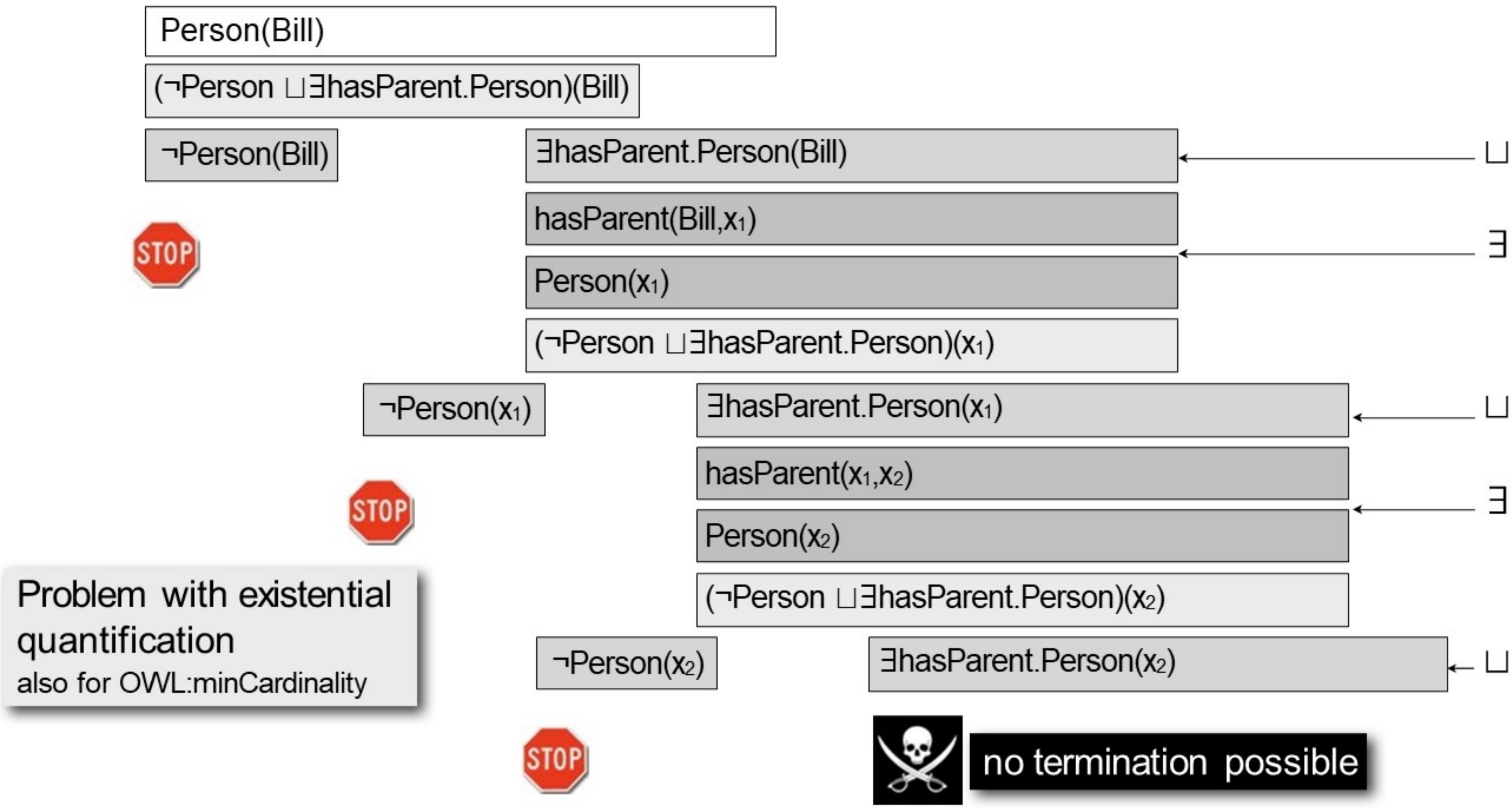
(using so-called *blocking technique*)

Reasoning with TBoxes (Blocking Example)

- TBox: $\{ \text{Person} \sqsubseteq \exists \text{hasParent.Person} \}$
- infer: $\neg\text{Person(Bill)}$ $\{ \neg\text{Person} \sqcup \exists \text{hasParent.Person},\ \text{Person(Bill)} \}$

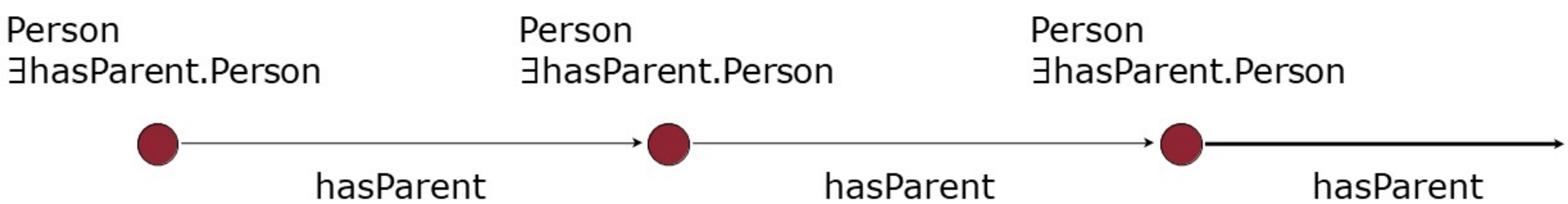
Reasoning with TBoxes (Blocking Example)

- TBox: $\{ \text{Person} \sqsubseteq \exists \text{hasParent.Person} \}$
- infer: $\neg\text{Person(Bill)}$ $\{ \neg\text{Person} \sqcup \exists\text{hasParent.Person},\ \text{Person(Bill)} \}$

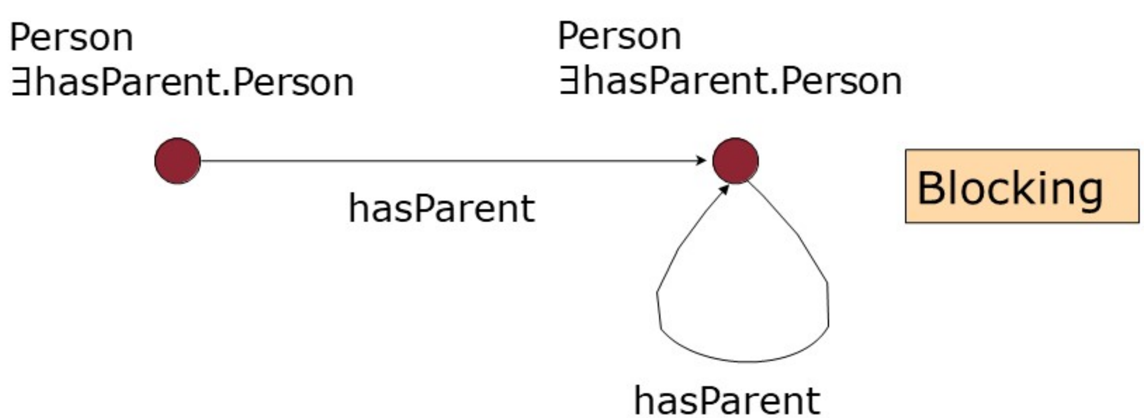


Reasoning with TBoxes (Blocking Example)

- the following had been constructed in the Tableaux:



- Idea: reuse old nodes



Correctness of this short cut must be proven!

Reasoning with TBoxes (Blocking Example)

- TBox: { Person ⊑ ∃hasParent.Person }
- infer: ¬Person(Bill)

Person(Bill)

(¬Person ⊔ ∃hasParent.Person)(Bill)

¬Person(Bill) | ∃hasParent.Person(Bill) ← ⊔

STOP

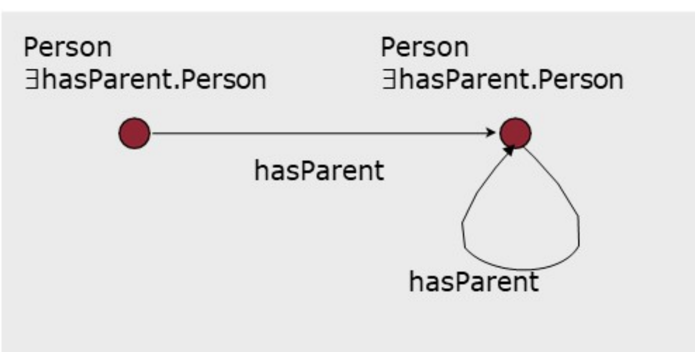
hasParent(Bill,x_1)

Person(x_1) ← ∃

(¬Person ⊔ ∃hasParent.Person)(x_1)

¬Person(x_1) | ∃hasParent.Person(x_1) ← ⊔

STOP



σ(Bill) = {Person,
¬Person ⊔ ∃hasParent.Person,
∃hasParent.Person}

σ(x_1) = { Person,
¬Person ⊔ ∃hasParent.Person,
∃hasParent.Person}

σ(x_1) ⊆ σ(Bill), so Bill blocks x_1 STOP **termination**