Description Logics and Reasoning on Data
2: Reasoning in $\mathcal{ALC}$

C. Bourgaux, M. Thomazo

Outline

Reminder: $\mathcal{ALC}$

The $\mathcal{ALC}$ DL is defined as follows:

- if A is an atomic concept, then A is an $\mathcal{ALC}$ concept
- if C, D are $\mathcal{ALC}$ concepts and R is an atomic role, then the following are $\mathcal{ALC}$ concepts:
    - $C \sqcap D$ (conjunction)
    - $C \sqcup D$ (disjunction)
    - $\neg C$ (negation)
    - $\exists R.C$ (existential restriction)
    - $\forall R.C$ (universal restriction)
- an $\mathcal{ALC}$ TBox contains only concept inclusions

Note that $A \sqcap \neg A$ can be abbreviated by $\bot$ and $A \sqcup \neg A$ by $\top$.

Reminder: Concept and KB Satisfiability

- Concept satisfiability w.r.t. an empty TBox: Given a concept C, is there an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ such that $C^{\mathcal{I}} \neq \emptyset$?
  - $A \sqcap B$ is satisfiable, $A \sqcap \neg A$ is not satisfiable
- Concept satisfiability w.r.t. a TBox: Given a concept C and a TBox $\mathcal{T}$, is there a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}} \neq \emptyset$?
  - $A \sqcap B$ is not satisfiable w.r.t. $\mathcal{T} = \{A \sqsubseteq \neg B\}$
- KB satisfiability: Given a KB $\langle \mathcal{T}, \mathcal{A} \rangle$, does $\langle \mathcal{T}, \mathcal{A} \rangle$ have a model?
  - $\langle \{A \sqsubseteq \neg B\},\ \{A(a), B(a)\} \rangle$ is not satisfiable,
    $\langle \{A \sqsubseteq \neg B\},\ \{A(a), B(b)\} \rangle$ is satisfiable
- Important in practice to build and debug ontologies
  - we usually don't want to use an unsatisfiable concept when defining an ontology
  - we may want to check that the model is sufficiently constrained to prevent some situation captured by a concept that should be unsatisfiable w.r.t. the TBox
  - an unsatisfiable KB indicates a modelisation problem

Reminder: Reduction Between Reasoning Tasks in $\mathcal{ALC}$

- From subsumption to concept satisfiability:
  $\mathcal{T} \models C \sqsubseteq D$ iff $C \sqcap \neg D$ is not satisfiable w.r.t. $\mathcal{T}$
  - note that if C and D are $\mathcal{ALC}$ concepts, so is $C \sqcap \neg D$
- From concept satisfiability to KB satisfiability:
  C is satisfiable w.r.t. $\mathcal{T}$ iff $\langle \mathcal{T} \cup \{A \sqsubseteq C\}, \mathcal{A} \cup \{C(a)\} \rangle$ is satisfiable
- From instance checking to KB satisfiability:
  $\langle \mathcal{T}, \mathcal{A} \rangle \models C(a)$ iff $\langle \mathcal{T} \cup \{C \sqsubseteq \neg A\}, \mathcal{A} \cup \{A(a)\} \rangle$ is not satisfiable

In this course: Algorithms to decide concept satisfiability w.r.t. an empty TBox and KB satisfiability
$\rightarrow$ concept satisfiability w.r.t. a non-empty TBox, subsumption and instance checking can be solved via reduction to KB satisfiability

Tableau Algorithms

- Tableau-based methods are used to decide satisfiability of a formula or theory by using rules to construct a model
  - if it succeeds, the theory is satisfiable
  - if it fails, despite having considered all possibilities, the theory is unsatisfiable
- Classical approach used for different kinds of logics (propositional, FOL, modal...)
- Popular approach for reasoning in expressive DLs ($\mathcal{ALC}$ and its extensions), implemented in state-of-the-art DL reasoners (with variants and optimizations)

Negation Normal Form

- The algorithms we consider need $\mathcal{ALC}$ concepts to be in negation normal form (NNF):
  An $\mathcal{ALC}$ concept C is in NNF if the symbol $\neg$ appears only in front of atomic concepts:
  - in NNF: $A \sqcap \neg B$, $\exists R.\neg A$, $A \sqcup B$
  - not in NNF: $\neg(A \sqcap B)$, $\exists R.\neg(\forall S.B)$, $A \sqcap \neg(B \sqcup C)$
- Every $\mathcal{ALC}$ concept C is equivalent to an $\mathcal{ALC}$ concept $\mathsf{nnf}(C)$ in NNF
  - $C^{\mathcal{I}} = \mathsf{nnf}(C)^{\mathcal{I}}$ for every interpretation $\mathcal{I}$
- $\mathsf{nnf}(C)$ can be computed in linear time by "pushing the negation inside" using the following equivalences

$$\neg(C \sqcap D) \equiv \neg C \sqcup \neg D \qquad \neg(\exists R.C) \equiv \forall R.\neg C \qquad \neg(\neg C) \equiv C$$

$$\neg(C \sqcup D) \equiv \neg C \sqcap \neg D \qquad \neg(\forall R.C) \equiv \exists R.\neg C$$

Negation Normal Form

Given an $\mathcal{ALC}$ concept C, $\mathrm{nnf}(C)$ is computed by the recursive algorithm:

- $\mathrm{nnf}(A) = A$ for A atomic concept
- $\mathrm{nnf}(\neg A) = \neg A$ for A atomic concept
- $\mathrm{nnf}(C \sqcap D) = \mathrm{nnf}(C) \sqcap \mathrm{nnf}(D)$
- $\mathrm{nnf}(C \sqcup D) = \mathrm{nnf}(C) \sqcup \mathrm{nnf}(D)$
- $\mathrm{nnf}(\exists R.C) = \exists R.\mathrm{nnf}(C)$
- $\mathrm{nnf}(\forall R.C) = \forall R.\mathrm{nnf}(C)$
- $\mathrm{nnf}(\neg(\neg C)) = \mathrm{nnf}(C)$
- $\mathrm{nnf}(\neg(C \sqcap D)) = \mathrm{nnf}(\neg C) \sqcup \mathrm{nnf}(\neg D)$
- $\mathrm{nnf}(\neg(C \sqcup D)) = \mathrm{nnf}(\neg C) \sqcap \mathrm{nnf}(\neg D)$
- $\mathrm{nnf}(\neg(\exists R.C)) = \forall R.\mathrm{nnf}(\neg C)$
- $\mathrm{nnf}(\neg(\forall R.C)) = \exists R.\mathrm{nnf}(\neg C)$

Tableau Algorithm for Concept Satisfiability

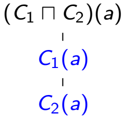
Overview

- Take as input an $\mathcal{ALC}$ concept C in NNF
- Decide the satisfiability of C by trying to construct an interpretation $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \emptyset$
- Represent an interpretation $\mathcal{I}$ by an ABox $\mathcal{A}_{\mathcal{I}}$ such that $a \in A^{\mathcal{I}}$ (resp. $(a, b) \in R^{\mathcal{I}}$) iff $A(a) \in \mathcal{A}_{\mathcal{I}}$ (resp. $R(a, b) \in \mathcal{A}_{\mathcal{I}}$)
- Initialize a set S of ABoxes, containing a single ABox $\{C(a_0)\}$
- At each stage, apply a tableau rule to some $\mathcal{A} \in S$ (see rules next slide)
- A rule application replaces $\mathcal{A}$ by one or two ABoxes that extend $\mathcal{A}$ with new assertions
- Stop applying rules when either:
  1. every $\mathcal{A} \in S$ contains a clash, that is, a pair $\{A(a_i), \neg A(a_i)\}$
  2. some $\mathcal{A} \in S$ is clash-free and complete, meaning that no rule can be applied to $\mathcal{A}$
- Return "yes" if some $\mathcal{A} \in S$ is clash-free, "no" otherwise

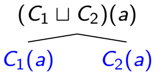
Tableau Algorithm for Concept Satisfiability

Tableau rules

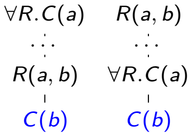
$\sqcap$-rule: if $(C_1 \sqcap C_2)(a) \in \mathcal{A}$
and $\{C_1(a), C_2(a)\} \not\subseteq \mathcal{A}$
replace $\mathcal{A}$ with $\mathcal{A} \cup \{C_1(a), C_2(a)\}$.



$\sqcup$-rule: if $(C_1 \sqcup C_2)(a) \in \mathcal{A}$
and $\{C_1(a), C_2(a)\} \cap \mathcal{A} = \emptyset$
replace $\mathcal{A}$ with $\mathcal{A} \cup \{C_1(a)\}$ and $\mathcal{A} \cup \{C_2(a)\}$.



$\forall$-rule: if $\{\forall R.C(a), R(a,b)\} \subseteq \mathcal{A}$
and $C(b) \notin \mathcal{A}$
replace $\mathcal{A}$ with $\mathcal{A} \cup \{C(b)\}$.



$\exists$-rule: if $\exists R.C(a) \in \mathcal{A}$
and there is no b with $\{R(a,b), C(b)\} \subseteq \mathcal{A}$
create a new individual name c and
replace $\mathcal{A}$ with $\mathcal{A} \cup \{R(a,c), C(c)\}$.

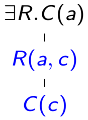


Tableau Algorithm for Concept Satisfiability

Example

$$(A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)(a_0)$$

Tableau Algorithm for Concept Satisfiability

Example

$$\begin{tikzcd}
(A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)(a_0) \arrow[d, dash] \\
A \sqcup B(a_0) \\
((\neg B \sqcup D) \sqcap \neg A)(a_0)
\end{tikzcd}$$

Tableau Algorithm for Concept Satisfiability

Example

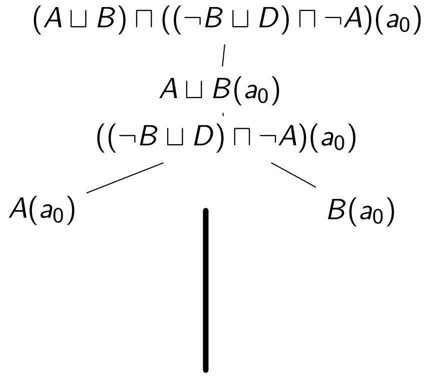
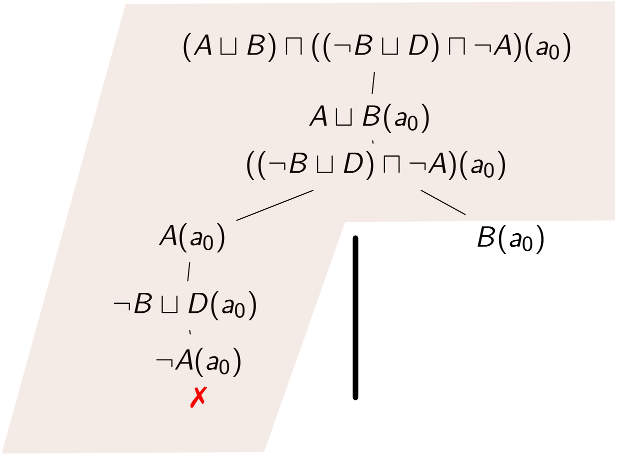

$(A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)(a_0)$
$A \sqcup B(a_0)$
$((\neg B \sqcup D) \sqcap \neg A)(a_0)$
$A(a_0)$
$B(a_0)$

Tableau Algorithm for Concept Satisfiability

Example



$(A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)(a_0)$
$A \sqcup B(a_0)$
$((\neg B \sqcup D) \sqcap \neg A)(a_0)$
$A(a_0)$
$B(a_0)$
$\neg B \sqcup D(a_0)$
$\neg A(a_0)$
✗

Tableau Algorithm for Concept Satisfiability

Example

$$\begin{tikzcd}
& (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)(a_0) \arrow[d, dash] & \\
& A \sqcup B(a_0) \arrow[d, dash] & \\
& ((\neg B \sqcup D) \sqcap \neg A)(a_0) \arrow[dl, dash] \arrow[dr, dash] & \\
A(a_0) \arrow[d, dash] & & B(a_0) \arrow[d, dash] \\
\neg B \sqcup D(a_0) \arrow[d, dash] & & \neg B \sqcup D(a_0) \arrow[d, dash] \\
\neg A(a_0) & & \neg A(a_0) \\
\text{✗} & &
\end{tikzcd}$$

Tableau Algorithm for Concept Satisfiability

Example

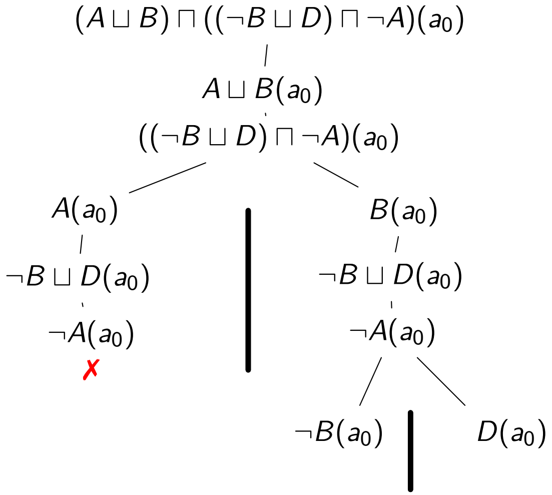


Tableau Algorithm for Concept Satisfiability

Example

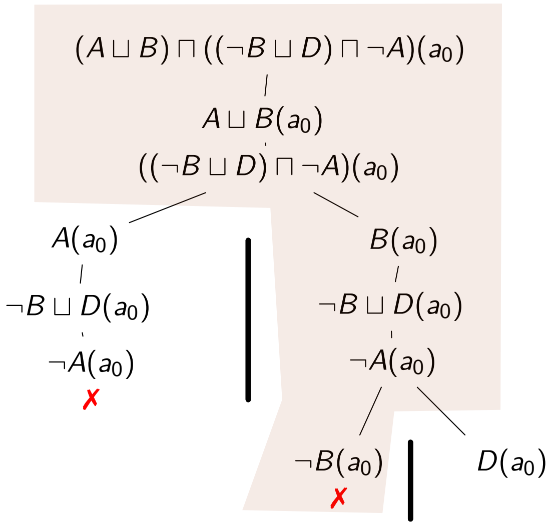


Tableau Algorithm for Concept Satisfiability

Example

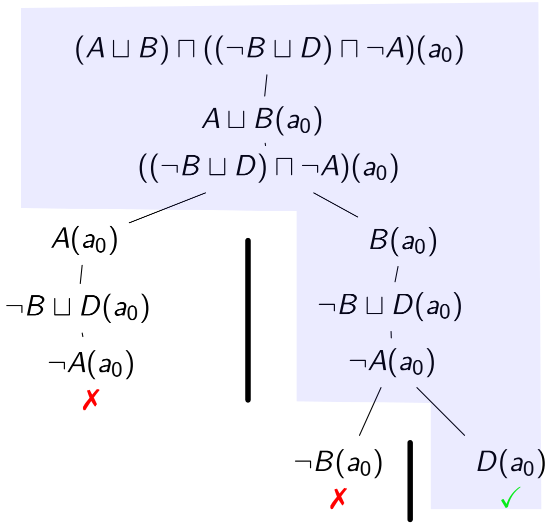


Tableau Algorithm for Concept Satisfiability

Another example

$$(\exists R.A \sqcap \forall R.\neg A)(a_0)$$

Tableau Algorithm for Concept Satisfiability

Another example

$$\begin{array}{cc}
(\exists R.A \sqcap \forall R.\neg A)(a_0) & \\
| & \\
\exists R.A(a_0) & \sqcap\text{-rule} \\
\forall R.\neg A(a_0) &
\end{array}$$

Tableau Algorithm for Concept Satisfiability

Another example

$$(\exists R.A \sqcap \forall R.\neg A)(a_0)$$

$$\exists R.A(a_0)$$

$$\forall R.\neg A(a_0)$$

$$R(a_0, a_1) \qquad \exists\text{-rule}$$

$$A(a_1)$$

Tableau Algorithm for Concept Satisfiability

Another example

$$(\exists R.A \sqcap \forall R.\neg A)(a_0)$$

$$\exists R.A(a_0)$$

$$\forall R.\neg A(a_0)$$

$$R(a_0, a_1)$$

$$A(a_1)$$

$$\neg A(a_1) \qquad \forall\text{-rule}$$

✗

Exercise

Use the tableau algorithm to decide which of the following concepts is satisfiable:

- $\exists R.(A \sqcap B) \sqcap \forall R.(\neg A \sqcup C) \sqcap \forall R.(\neg B \sqcup \neg C)$
- $\exists R.A \sqcap \forall R.(\exists R.A \sqcup \neg A)$

Tableau Algorithm for Concept Satisfiability

Let us call our tableau algorithm `CSat` (for concept satisfiability)

Theorem

`CSat` terminates and it answers yes if and only if the input concept is satisfiable.

To prove this theorem, we must show:

- termination: `CSat` always terminates
- soundness: if `Csat` outputs “yes” on input C_0, then the concept C_0 is satisfiable
- completeness: if C_0 is satisfiable, then `CSat` outputs “yes” on input C_0

Preliminary Definitions

Subconcepts of a concept

$$\begin{aligned}
\mathrm{sub}(A) &= \{A\} \\
\mathrm{sub}(\neg C) &= \{\neg C\} \cup \mathrm{sub}(C) \\
\mathrm{sub}(\exists R.C) &= \{\exists R.C\} \cup \mathrm{sub}(C) \\
\mathrm{sub}(\forall R.C) &= \{\forall R.C\} \cup \mathrm{sub}(C) \\
\mathrm{sub}(C_1 \sqcup C_2) &= \{C_1 \sqcup C_2\} \cup \mathrm{sub}(C_1) \cup \mathrm{sub}(C_2) \\
\mathrm{sub}(C_1 \sqcap C_2) &= \{C_1 \sqcap C_2\} \cup \mathrm{sub}(C_1) \cup \mathrm{sub}(C_2)
\end{aligned}$$

Preliminary Definitions

Subconcepts of a concept

$$
\begin{aligned}
\mathsf{sub}(A) &= \{A\} \\
\mathsf{sub}(\neg C) &= \{\neg C\} \cup \mathsf{sub}(C) \\
\mathsf{sub}(\exists R.C) &= \{\exists R.C\} \cup \mathsf{sub}(C) \\
\mathsf{sub}(\forall R.C) &= \{\forall R.C\} \cup \mathsf{sub}(C) \\
\mathsf{sub}(C_1 \sqcup C_2) &= \{C_1 \sqcup C_2\} \cup \mathsf{sub}(C_1) \cup \mathsf{sub}(C_2) \\
\mathsf{sub}(C_1 \sqcap C_2) &= \{C_1 \sqcap C_2\} \cup \mathsf{sub}(C_1) \cup \mathsf{sub}(C_2)
\end{aligned}
$$

Example

$$
\begin{aligned}
\mathsf{sub}(\exists R.(A \sqcap \forall S.(B \sqcup \neg C))) = \{ & \\
& \exists R.(A \sqcap \forall S.(B \sqcup \neg C)), \quad A \sqcap \forall S.(B \sqcup \neg C), \quad A, \\
& \forall S.(B \sqcup \neg C), \quad B \sqcup \neg C, \quad B, \quad \neg C, \quad C \\
& \}
\end{aligned}
$$

Preliminary Definitions

Role depth of a concept

$$\begin{aligned}
\mathrm{depth}(A) &= 0\\
\mathrm{depth}(\neg C) &= \mathrm{depth}(C)\\
\mathrm{depth}(\exists R.C) &= \mathrm{depth}(\forall R.C) = \mathrm{depth}(C) + 1\\
\mathrm{depth}(C_1 \sqcup C_2) &= \mathrm{depth}(C_1 \sqcap C_2) = \max(\mathrm{depth}(C_1), \mathrm{depth}(C_2))
\end{aligned}$$

Preliminary Definitions

Role depth of a concept

$$\begin{aligned}
\mathrm{depth}(A) &= 0\\
\mathrm{depth}(\neg C) &= \mathrm{depth}(C)\\
\mathrm{depth}(\exists R.C) &= \mathrm{depth}(\forall R.C) = \mathrm{depth}(C) + 1\\
\mathrm{depth}(C_1 \sqcup C_2) &= \mathrm{depth}(C_1 \sqcap C_2) = \max(\mathrm{depth}(C_1), \mathrm{depth}(C_2))
\end{aligned}$$

Example

$$\mathrm{depth}(\exists R.(A \sqcap \forall S.(B \sqcup C))) = 2$$

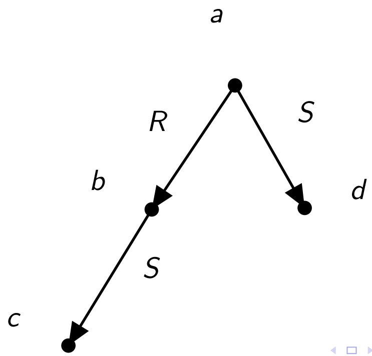
Preliminary Definitions

Tree-shaped ABox

Graph representation of an ABox $\mathcal{A}$: graph whose vertices are individual names of $\mathcal{A}$ and such that there is a (directed) edge from a to b labelled by R iff $R(a,b) \in \mathcal{A}$.
If this graph is a tree, $\mathcal{A}$ is tree-shaped.

Example

$\{R(a,b), S(b,c), S(a,d)\}$ is tree-shaped



Termination of `CSat` (Informal Proof)

Suppose we run `CSat` starting from $S = \{\{C(a_0)\}\}$. Let us make the following observations for every ABox $\mathcal{A}$ generated by `CSat`:

1. $\mathcal{A}$ is tree-shaped
2. The depth of the tree is bounded by the role depth of C: each individual in $\mathcal{A}$ is at distance $k \leq \text{depth}(C)$ from a_0
   - if $D(b) \in \mathcal{A}$ and the unique path from a_0 to b has length k, then $\text{depth}(D) \leq \text{depth}(C) - k$
3. The degree of the tree is bounded by the number of existentials in C
4. The number of concept assertions per individual is bounded by the number of subconcepts $|\text{sub}(C)|$
   - if $D(b) \in \mathcal{A}$, then $D \in \text{sub}(C)$

Hence there is a bound on the size of generated ABoxes. Since `CSat` only adds assertions to ABoxes, every generated ABox will eventually be complete or contain a clash. Hence `CSat` terminates.

Soundness of CSat

Assume that CSat returns "yes" on input C.

- Then S must contain a complete and clash-free ABox $\mathcal{A}$.
- Define an interpretation $\mathcal{I}$ as follows:
  - $\Delta^{\mathcal{I}} = \{a \mid a \text{ is an individual in } \mathcal{A}\}$
  - $A^{\mathcal{I}} = \{a \mid A(a) \in \mathcal{A}\}$
  - $R^{\mathcal{I}} = \{(a, b) \mid R(a, b) \in \mathcal{A}\}$
- Claim: $\mathcal{I}$ is such that $C^{\mathcal{I}} \neq \emptyset$

  To show the claim, we prove by induction on the size of concepts that:

  $$D(b) \in \mathcal{A} \Rightarrow b \in D^{\mathcal{I}}$$

  Since the completion algorithm never deletes assertions, $C(a_0) \in \mathcal{A}$ for every $\mathcal{A} \in S$ and the claim follows.

It follows from the claim that C is satisfiable.

Soundness of CSat

Proof of the claim: $D(b) \in \mathcal{A} \Rightarrow b \in D^{\mathcal{I}}$

Base Case: $D = A$ or $D = \neg A$

- If $D = A$, then $b \in A^{\mathcal{I}}$ by definition of $\mathcal{I}$
- If $D = \neg A$, then $A(b) \notin \mathcal{A}$ because $\mathcal{A}$ is clash-free, hence $b \notin A^{\mathcal{I}}$, i.e., $b \in \neg A^{\mathcal{I}}$

Soundness of CSat

Proof of the claim: $D(b) \in \mathcal{A} \Rightarrow b \in D^{\mathcal{I}}$

Base Case: $D = A$ or $D = \neg A$

- If $D = A$, then $b \in A^{\mathcal{I}}$ by definition of $\mathcal{I}$
- If $D = \neg A$, then $A(b) \notin \mathcal{A}$ because $\mathcal{A}$ is clash-free, hence $b \notin A^{\mathcal{I}}$, i.e., $b \in \neg A^{\mathcal{I}}$

Induction Hypothesis: statement holds whenever $|D| \leq k$

Induction Step: show statement holds for D with $|D| = k + 1$

Soundness of CSat

Proof of the claim: $D(b) \in \mathcal{A} \Rightarrow b \in D^{\mathcal{I}}$

Base Case: $D = A$ or $D = \neg A$

- If $D = A$, then $b \in A^{\mathcal{I}}$ by definition of $\mathcal{I}$
- If $D = \neg A$, then $A(b) \notin \mathcal{A}$ because $\mathcal{A}$ is clash-free, hence $b \notin A^{\mathcal{I}}$, i.e., $b \in \neg A^{\mathcal{I}}$

Induction Hypothesis: statement holds whenever $|D| \leq k$

Induction Step: show statement holds for D with $|D| = k + 1$

- $D = E \sqcap F$: since $\mathcal{A}$ is complete, $\mathcal{A}$ contains $E(b)$ and $F(b)$. By the induction hypothesis, $b \in E^{\mathcal{I}}$ and $b \in F^{\mathcal{I}}$, so $b \in (E \sqcap F)^{\mathcal{I}}$

Soundness of CSat

Proof of the claim: $D(b) \in \mathcal{A} \Rightarrow b \in D^{\mathcal{I}}$

Base Case: $D = A$ or $D = \neg A$

- If $D = A$, then $b \in A^{\mathcal{I}}$ by definition of $\mathcal{I}$
- If $D = \neg A$, then $A(b) \notin \mathcal{A}$ because $\mathcal{A}$ is clash-free, hence $b \notin A^{\mathcal{I}}$, i.e., $b \in \neg A^{\mathcal{I}}$

Induction Hypothesis: statement holds whenever $|D| \leq k$

Induction Step: show statement holds for D with $|D| = k + 1$

- $D = E \sqcap F$: since $\mathcal{A}$ is complete, $\mathcal{A}$ contains $E(b)$ and $F(b)$. By the induction hypothesis, $b \in E^{\mathcal{I}}$ and $b \in F^{\mathcal{I}}$, so $b \in (E \sqcap F)^{\mathcal{I}}$
- $D = \exists R.E$: since $\mathcal{A}$ is complete, there is some c such that $R(b, c) \in \mathcal{A}$ and $E(c) \in \mathcal{A}$. Then $(b, c) \in R^{\mathcal{I}}$, and by the induction hypothesis, we get that $c \in E^{\mathcal{I}}$, so $b \in (\exists R.E)^{\mathcal{I}}$

Soundness of CSat

Proof of the claim: $D(b) \in \mathcal{A} \Rightarrow b \in D^{\mathcal{I}}$

Base Case: $D = A$ or $D = \neg A$

- If $D = A$, then $b \in A^{\mathcal{I}}$ by definition of $\mathcal{I}$
- If $D = \neg A$, then $A(b) \notin \mathcal{A}$ because $\mathcal{A}$ is clash-free, hence $b \notin A^{\mathcal{I}}$, i.e., $b \in \neg A^{\mathcal{I}}$

Induction Hypothesis: statement holds whenever $|D| \leq k$

Induction Step: show statement holds for D with $|D| = k + 1$

- $D = E \sqcap F$: since $\mathcal{A}$ is complete, $\mathcal{A}$ contains $E(b)$ and $F(b)$. By the induction hypothesis, $b \in E^{\mathcal{I}}$ and $b \in F^{\mathcal{I}}$, so $b \in (E \sqcap F)^{\mathcal{I}}$
- $D = \exists R.E$: since $\mathcal{A}$ is complete, there is some c such that $R(b, c) \in \mathcal{A}$ and $E(c) \in \mathcal{A}$. Then $(b, c) \in R^{\mathcal{I}}$, and by the induction hypothesis, we get that $c \in E^{\mathcal{I}}$, so $b \in (\exists R.E)^{\mathcal{I}}$
- $D = E \sqcup F$: left as practice
- $D = \forall R.E$: left as practice

Completeness of `CSat`

Suppose that C is satisfiable.

- This implies that the ABox $\{C(a_0)\}$ is satisfiable.
- Claim: Tableau rules are satisfiability-preserving:
  - if an ABox $\mathcal{A}$ is satisfiable and $\mathcal{A}'$ is the result of applying a rule to $\mathcal{A}$, then $\mathcal{A}'$ is also satisfiable
  - if an ABox $\mathcal{A}$ is satisfiable and $\mathcal{A}_1$ and $\mathcal{A}_2$ are obtained when applying a rule to $\mathcal{A}$, then either $\mathcal{A}_1$ or $\mathcal{A}_2$ is satisfiable
- Since ABoxes containing a clash are not satisfiable and we start with the satisfiable ABox $\{C(a_0)\}$, `CSat` will eventually generate a complete satisfiable (thus clash-free) ABox.

Hence `CSat` returns "yes" on input C.

Completeness of CSat

Proof of the claim: Tableau rules are satisfiability-preserving

Let $\mathcal{A}$ be a satisfiable ABox and $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be a model of $\mathcal{A}$

- If $\mathcal{A}'$ is the result of applying the $\sqcap$-rule to $\mathcal{A}$, there is $(C_1 \sqcap C_2)(b) \in \mathcal{A}$ and $\mathcal{A}' = \mathcal{A} \cup \{C_1(b), C_2(b)\}$
  - since $b^{\mathcal{I}} \in (C_1 \sqcap C_2)^{\mathcal{I}}$, then $b^{\mathcal{I}} \in C_1^{\mathcal{I}}$ and $b^{\mathcal{I}} \in C_2^{\mathcal{I}}$
  - it follows that $\mathcal{I}$ is a model of $\mathcal{A}'$, thus $\mathcal{A}'$ is satisfiable
- If $\mathcal{A}_1$ and $\mathcal{A}_2$ are the result of applying the $\sqcup$-rule to $\mathcal{A}$, there is $(C_1 \sqcup C_2)(b) \in \mathcal{A}$, $\mathcal{A}_1 = \mathcal{A} \cup \{C_1(b)\}$, and $\mathcal{A}_2 = \mathcal{A} \cup \{C_2(b)\}$
  - since $b^{\mathcal{I}} \in (C_1 \sqcup C_2)^{\mathcal{I}}$, then $b^{\mathcal{I}} \in C_1^{\mathcal{I}}$ or $b^{\mathcal{I}} \in C_2^{\mathcal{I}}$
  - it follows that $\mathcal{I}$ is a model of $\mathcal{A}_1$ or of $\mathcal{A}_2$, thus $\mathcal{A}_1$ or $\mathcal{A}_2$ is satisfiable
- $\forall$-rule: left as practice
- $\exists$-rule: left as practice

Tree Model Property

`CSat` produces tree-shaped ABoxes, so we get that for every $\mathcal{ALC}$ concept C, if C has a model, then it has a tree-shaped model

This is an important property

- We only need to look at tree-shaped structures when reasoning about $\mathcal{ALC}$ concepts
- Trees are computationally "friendly"
- This property exposes a limitation in the expressive power of $\mathcal{ALC}$ (for example they cannot describe structures with cycles)

Extension to KB Satisfiability

We want to modify `CSat` to check the satisfiability of a knowledge base $\langle \mathcal{T}, \mathcal{A} \rangle$

Adding the ABox is easy:

- start from $S = \{\mathcal{A}\}$ instead of $S = \{\{C(a)\}\}$

Extension to KB Satisfiability

We want to modify `CSat` to check the satisfiability of a knowledge base $\langle \mathcal{T}, \mathcal{A} \rangle$

Adding the ABox is easy:

- start from $S = \{\mathcal{A}\}$ instead of $S = \{\{C(a)\}\}$

For the TBox, note that $C \sqsubseteq D \equiv \top \sqsubseteq \neg C \sqcup D$ and add the following rule to the tableau rules:

TBox-rule:
if $C \sqsubseteq D \in \mathcal{T}$,
a is an individual of $\mathcal{A}$
and $(\text{nnf}(\neg C \sqcup D))(a) \notin \mathcal{A}$
replace $\mathcal{A}$ with $\mathcal{A} \cup \{(\text{nnf}(\neg C \sqcup D))(a)\}$.

$$\begin{tikzcd} X(a) \arrow[d, dash] \\ (\text{nnf}(\neg C \sqcup D))(a) \end{tikzcd}$$

Exercise

Use the tableau algorithm to check whether the following KBs are satisfiable:

- $\langle \mathcal{T}, \{A(a)\} \rangle$
- $\langle \mathcal{T}, \{R(c, a), B(a)\} \rangle$

where

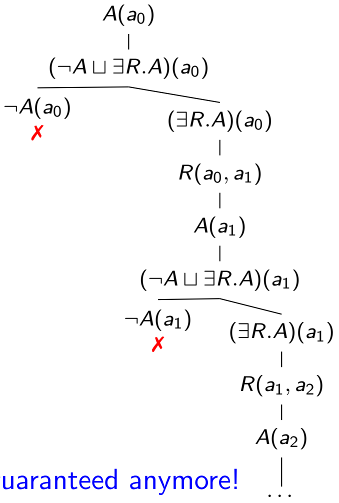
$$\mathcal{T} = \{A \sqsubseteq \exists R.B,\ B \sqsubseteq D,\ \exists R.D \sqsubseteq \neg A\}$$

Exercise

Now try on the following KB: $\langle\{A \sqsubseteq \exists R.A\}, \{A(a_0)\}\rangle$

Exercise

Now try on the following KB: $\langle \{A \sqsubseteq \exists R.A\}, \{A(a_0)\} \rangle$



Termination is not guaranteed anymore!

Making the Algorithm Terminate

Basic idea: if two individuals “look the same”, explore only one

Making the Algorithm Terminate

Basic idea: if two individuals “look the same”, explore only one

Blocking

An individual a blocks an individual b in an ABox $\mathcal{A}$ if:

- $\{C \mid C(b) \in \mathcal{A}\} \subseteq \{C \mid C(a) \in \mathcal{A}\}$
- a was in $\mathcal{A}$ when b has been introduced

An individual b is blocked if some a blocks b

Making the Algorithm Terminate

Basic idea: if two individuals "look the same", explore only one

Blocking

An individual a blocks an individual b in an ABox $\mathcal{A}$ if:

- $\{C \mid C(b) \in \mathcal{A}\} \subseteq \{C \mid C(a) \in \mathcal{A}\}$
- a was in $\mathcal{A}$ when b has been introduced

An individual b is blocked if some a blocks b

The blocked individual b can use the role successors of a instead of generating new ones

Modify the tableau rules to apply them only to individuals that are not blocked

Tableau Algorithm for KB Satisfiability

Tableau rules

$\sqcap$-rule: if $(C_1 \sqcap C_2)(a) \in \mathcal{A}$, a is not blocked, and $\{C_1(a), C_2(a)\} \not\subseteq \mathcal{A}$, replace $\mathcal{A}$ with $\mathcal{A} \cup \{C_1(a), C_2(a)\}$.

$\sqcup$-rule: if $(C_1 \sqcup C_2)(a) \in \mathcal{A}$, a is not blocked, and $\{C_1(a), C_2(a)\} \cap \mathcal{A} = \emptyset$ replace $\mathcal{A}$ with $\mathcal{A} \cup \{C_1(a)\}$ and $\mathcal{A} \cup \{C_2(a)\}$.

$\forall$-rule: if $\{\forall R.C(a), R(a,b)\} \subseteq \mathcal{A}$, a is not blocked, and $C(b) \notin \mathcal{A}$, replace $\mathcal{A}$ with $\mathcal{A} \cup \{C(b)\}$.

$\exists$-rule: if $\exists R.C(a) \in \mathcal{A}$, a is not blocked, and there is no b with $\{R(a,b), C(b)\} \subseteq \mathcal{A}$, create a new individual name c and replace $\mathcal{A}$ with $\mathcal{A} \cup \{R(a,c), C(c)\}$.

TBox-rule: if $C \sqsubseteq D \in \mathcal{T}$, a is not blocked, and $(\text{nnf}(\neg C \sqcup D))(a) \notin \mathcal{A}$, replace $\mathcal{A}$ by $\mathcal{A} \cup \{(\text{nnf}(\neg C \sqcup D)(a))\}$.

Tableau Algorithm for KB Satisfiability

Example

Apply blocking to the previous KB: $\langle \{A \sqsubseteq \exists R.A\}, \{A(a_0)\} \rangle$

Tableau Algorithm for KB Satisfiability

Example

Apply blocking to the previous KB: $\langle\{A \sqsubseteq \exists R.A\}, \{A(a_0)\}\rangle$

$$A(a_0)$$

Tableau Algorithm for KB Satisfiability

Example

Apply blocking to the previous KB: $\langle\{A \sqsubseteq \exists R.A\}, \{A(a_0)\}\rangle$

$$\begin{tikzcd}
A(a_0) \arrow[d, dash] \\
(\neg A \sqcup \exists R.A)(a_0)
\end{tikzcd}$$

Tableau Algorithm for KB Satisfiability

Example

Apply blocking to the previous KB: $\langle\{A \sqsubseteq \exists R.A\}, \{A(a_0)\}\rangle$

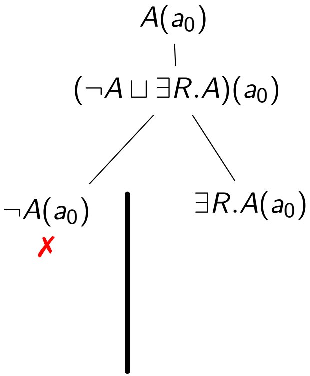


Tableau Algorithm for KB Satisfiability

Example

Apply blocking to the previous KB: $\langle \{A \sqsubseteq \exists R.A\}, \{A(a_0)\} \rangle$

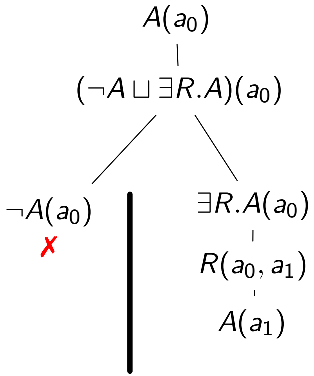
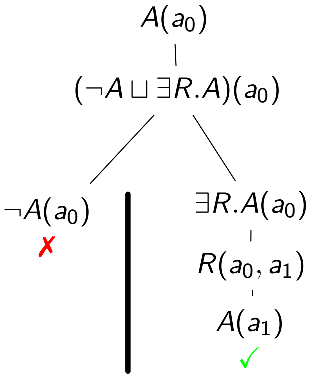


Tableau Algorithm for KB Satisfiability

Example

Apply blocking to the previous KB: $\langle \{A \sqsubseteq \exists R.A\}, \{A(a_0)\} \rangle$

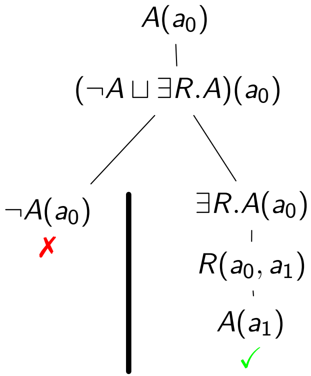


a_1 is blocked by a_0!

Tableau Algorithm for KB Satisfiability

Example

Apply blocking to the previous KB: $\langle \{A \sqsubseteq \exists R.A\}, \{A(a_0)\} \rangle$



a_1 is blocked by a_0!

We obtain a complete, clash-free ABox

$\rightarrow \langle \{A \sqsubseteq \exists R.A\}, \{A(a_0)\} \rangle$ is satisfiable

Tableau Algorithm for KB Satisfiability

Another example

Consider

$$\mathcal{T} = \{A \sqsubseteq \exists R.A,\ A \sqsubseteq B,\ \exists R.B \sqsubseteq D\}$$

We want to test whether $\mathcal{T} \models A \sqsubseteq D$ using the tableau algorithm

$\rightarrow$ check whether the following KB is satisfiable

$$\langle \mathcal{T}, \{(A \sqcap \neg D)(a_0)\}\rangle$$

Tableau Algorithm for KB Satisfiability

Another example

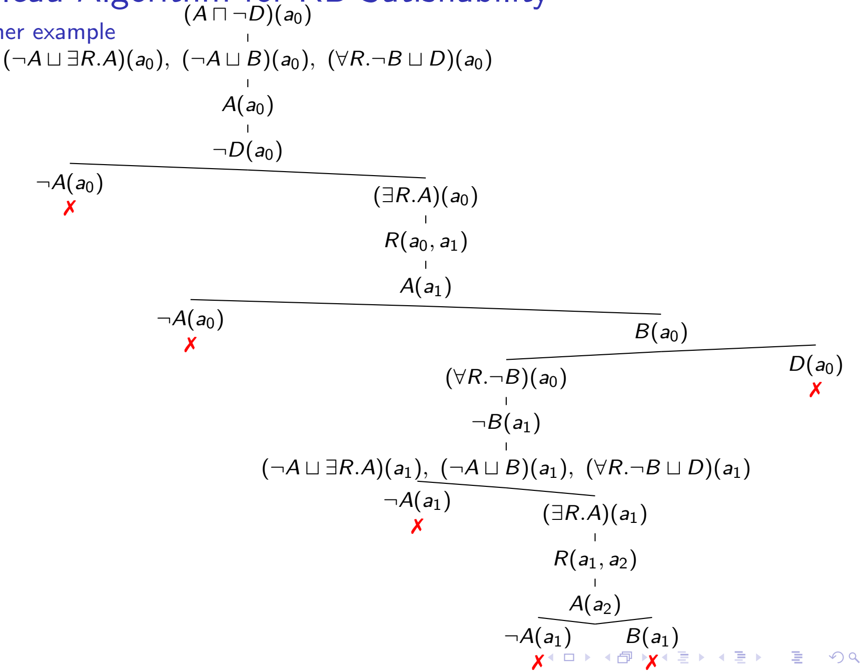


Tableau Algorithm for KB Satisfiability

Another example

Consider

$$\mathcal{T} = \{A \sqsubseteq \exists R.A,\ A \sqsubseteq B,\ \exists R.B \sqsubseteq D\}$$

We want to test whether $\mathcal{T} \models A \sqsubseteq D$ using the tableau algorithm

$\rightarrow$ check whether the following KB is satisfiable

$$\langle \mathcal{T}, \{(A \sqcap \neg D)(a_0)\} \rangle$$

$\langle \mathcal{T}, \{(A \sqcap \neg D)(a_0)\} \rangle$ is unsatisfiable so $\mathcal{T} \models A \sqsubseteq D$

Remark: an individual can be blocked then later become unblocked

Tableau Algorithm for KB Satisfiability

Let us call our tableau algorithm `KBSat` (for KB satisfiability)

Theorem

`KBSat` terminates and it answers yes if and only if the input KB is satisfiable.

Termination of KBSat (Informal Proof)

KBSat terminates on every input $\langle \mathcal{T}, \mathcal{A} \rangle$.

Similar to the proof of termination for CSat: Show that there is a bound on the size of the generated ABoxes

For every ABox $\mathcal{A}'$ generated by KBSat:

1. The number of concept assertions per individual is bounded by the total number of subconcepts of concepts that occur in $\mathcal{A}$ or in $\{\text{nnf}(\neg C \sqcup D) \mid C \sqsubseteq D \in \mathcal{T}\}$
2. The individuals generated by the $\exists$-rule form trees whose roots are individuals from $\mathcal{A}$
3. Blocking ensures that the depth of each tree is finite (bounded by the number of sets of subconcepts of concepts that occur in $\mathcal{A}$ or in $\{\text{nnf}(\neg C \sqcup D) \mid C \sqsubseteq D \in \mathcal{T}\}$)
4. The degree of each tree is bounded by the number of existentials in $\mathcal{T}$

Soundness of KBSat

If KBSat returns "yes" on input $\langle\mathcal{T},\mathcal{A}\rangle$, then $\langle\mathcal{T},\mathcal{A}\rangle$ is satisfiable.

- Build a model $\mathcal{I}$ from a complete and clash-free ABox $\mathcal{A}'$
- Difference with CSat: deal with the blocked individuals
  - $\Delta^{\mathcal{I}} = \{a \mid a$ is an individual in $\mathcal{A}'$ which is not blocked$\}$
  - $A^{\mathcal{I}} = \{a \mid A(a) \in \mathcal{A}', a$ not blocked$\}$
  - $R^{\mathcal{I}} = \{(a,b) \mid R(a,b) \in \mathcal{A}', a, b$ not blocked$\} \cup \{(a,b) \mid R(a,c) \in \mathcal{A}', a$ not blocked$, c$ blocked by b, b not blocked$\}$
- Claim: $\mathcal{I}$ is a model of $\langle\mathcal{T},\mathcal{A}\rangle$

Soundness of KBSat

If KBSat returns "yes" on input $\langle \mathcal{T}, \mathcal{A} \rangle$, then $\langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable.

- Build a model $\mathcal{I}$ from a complete and clash-free ABox $\mathcal{A}'$
- Difference with CSat: deal with the blocked individuals
  - $\Delta^{\mathcal{I}} = \{a \mid a \text{ is an individual in } \mathcal{A}' \text{ which is not blocked}\}$
  - $A^{\mathcal{I}} = \{a \mid A(a) \in \mathcal{A}', a \text{ not blocked}\}$
  - $R^{\mathcal{I}} = \{(a, b) \mid R(a, b) \in \mathcal{A}', a, b \text{ not blocked}\} \cup \{(a, b) \mid R(a, c) \in \mathcal{A}', a \text{ not blocked}, c \text{ blocked by } b, b \text{ not blocked}\}$
- Claim: $\mathcal{I}$ is a model of $\langle \mathcal{T}, \mathcal{A} \rangle$
  - Since individuals from $\mathcal{A}$ are never blocked, $\mathcal{I} \models \mathcal{A}$
  - Let $C \sqsubseteq D \in \mathcal{T}$ and $b \in C^{\mathcal{I}}$
    - since b is not blocked in $\mathcal{A}'$ and $\mathcal{A}'$ is complete, $\text{nnf}(\neg C \sqcup D)(b) \in \mathcal{A}'$ (TBox-rule) so $\text{nnf}(\neg C)(b)$ or $\text{nnf}(D)(b)$ is in $\mathcal{A}'$ ($\sqcup$-rule)
    - we prove that $E(b) \in \mathcal{A}'$ and b not blocked $\Rightarrow b \in E^{\mathcal{I}}$ for every concept E by induction on the size of E
    - since $b \in C^{\mathcal{I}}$ (so that $b \notin \text{nnf}(\neg C)^{\mathcal{I}}$), it follows that $\text{nnf}(\neg C)(b) \notin \mathcal{A}'$
    - thus $\text{nnf}(D)(b)$ is in $\mathcal{A}'$ and $b \in \text{nnf}(D)^{\mathcal{I}} = D^{\mathcal{I}}$

    It follows that $\mathcal{I} \models C \sqsubseteq D$
  - Hence $\mathcal{I} \models \mathcal{T}$

Soundness of `KBSat`

Proof of the claim: $E(b) \in \mathcal{A}'$ and b not blocked $\Rightarrow b \in E^{\mathcal{I}}$

Base Case: $E = A$ or $E = \neg A$

- If $E = A$, then $b \in A^{\mathcal{I}}$, by definition of $\mathcal{I}$
- If $E = \neg A$, then $A(b) \notin \mathcal{A}'$ because $\mathcal{A}'$ is clash-free, hence $b \in \neg A^{\mathcal{I}}$

Soundness of `KBSat`

Proof of the claim: $E(b) \in \mathcal{A}'$ and b not blocked $\Rightarrow b \in E^{\mathcal{I}}$

Base Case: $E = A$ or $E = \neg A$

- If $E = A$, then $b \in A^{\mathcal{I}}$, by definition of $\mathcal{I}$
- If $E = \neg A$, then $A(b) \notin \mathcal{A}'$ because $\mathcal{A}'$ is clash-free, hence $b \in \neg A^{\mathcal{I}}$

Induction Hypothesis: statement holds whenever $|E| \leq k$

Induction Step: show statement holds for $|E| = k + 1$

Soundness of KBSat

Proof of the claim: $E(b) \in \mathcal{A}'$ and b not blocked $\Rightarrow b \in E^{\mathcal{I}}$

Base Case: $E = A$ or $E = \neg A$

- If $E = A$, then $b \in A^{\mathcal{I}}$, by definition of $\mathcal{I}$
- If $E = \neg A$, then $A(b) \notin \mathcal{A}'$ because $\mathcal{A}'$ is clash-free, hence $b \in \neg A^{\mathcal{I}}$

Induction Hypothesis: statement holds whenever $|E| \leq k$

Induction Step: show statement holds for $|E| = k + 1$

- $E = \exists R.F$: since $\mathcal{A}'$ is complete and b is not blocked, there is some c such that $R(b, c) \in \mathcal{A}'$ and $F(c) \in \mathcal{A}'$
  - if c is not blocked, $(b, c) \in R^{\mathcal{I}}$, and by the induction hypothesis, $c \in F^{\mathcal{I}}$, so $b \in (\exists R.F)^{\mathcal{I}}$
  - if c is blocked, it must be blocked by some d which is not blocked, so $(b, d) \in R^{\mathcal{I}}$, and $F(d) \in \mathcal{A}'$ so by the induction hypothesis, $d \in F^{\mathcal{I}}$, so $b \in (\exists R.F)^{\mathcal{I}}$

Soundness of KBSat

Proof of the claim: $E(b) \in \mathcal{A}'$ and b not blocked $\Rightarrow b \in E^{\mathcal{I}}$

Base Case: $E = A$ or $E = \neg A$

- If $E = A$, then $b \in A^{\mathcal{I}}$, by definition of $\mathcal{I}$
- If $E = \neg A$, then $A(b) \notin \mathcal{A}'$ because $\mathcal{A}'$ is clash-free, hence $b \in \neg A^{\mathcal{I}}$

Induction Hypothesis: statement holds whenever $|E| \leq k$

Induction Step: show statement holds for $|E| = k + 1$

- $E = \exists R.F$: since $\mathcal{A}'$ is complete and b is not blocked, there is some c such that $R(b, c) \in \mathcal{A}'$ and $F(c) \in \mathcal{A}'$
  - if c is not blocked, $(b, c) \in R^{\mathcal{I}}$, and by the induction hypothesis, $c \in F^{\mathcal{I}}$, so $b \in (\exists R.F)^{\mathcal{I}}$
  - if c is blocked, it must be blocked by some d which is not blocked, so $(b, d) \in R^{\mathcal{I}}$, and $F(d) \in \mathcal{A}'$ so by the induction hypothesis, $d \in F^{\mathcal{I}}$, so $b \in (\exists R.F)^{\mathcal{I}}$
- $E = \forall R.F$: left as practice
- $E = F \sqcap G$: left as practice
- $E = F \sqcup G$: left as practice

Completeness of `KBSat`

If $\langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable, then `KBSat` returns "yes" on input $\langle \mathcal{T}, \mathcal{A} \rangle$.

Similar to the proof of completeness of `CSat`: Show that tableau rules are satisfiability-preserving

Let $\langle \mathcal{T}, \mathcal{A} \rangle$ be a satisfiable KB and $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be a model of $\langle \mathcal{T}, \mathcal{A} \rangle$

- For the new TBox-rule: If $\mathcal{A}'$ is the result of applying the TBox-rule to $\mathcal{A}$, there is $C \sqsubseteq D \in \mathcal{T}$ and $\mathcal{A}' = \mathcal{A} \cup \{(\mathsf{nnf}(\neg C \sqcup D)(a))\}$
  - if $a^{\mathcal{I}} \notin (\neg C)^{\mathcal{I}}$, i.e., $a^{\mathcal{I}} \in C^{\mathcal{I}}$, since $\mathcal{I} \models \mathcal{T}$, then $a^{\mathcal{I}} \in D^{\mathcal{I}}$
  - hence $a^{\mathcal{I}} \in (\neg C)^{\mathcal{I}} \cup D^{\mathcal{I}}$, i.e., $a^{\mathcal{I}} \in (\neg C \sqcup D)^{\mathcal{I}} = \mathsf{nnf}(\neg C \sqcup D)^{\mathcal{I}}$
  - it follows that $\mathcal{I} \models \langle \mathcal{T}, \mathcal{A}' \rangle$, thus $\langle \mathcal{T}, \mathcal{A}' \rangle$ is satisfiable
- Adding the condition that a is not blocked only restricts the rules applicability

Forest Model Property

- An interpretation $\mathcal{I}$ is forest-shaped if the graph whose vertices are the domain elements and edges are

$$\{(d, d') \mid (d, d') \in R^{\mathcal{I}} \text{ for some } R \text{ and } d, d' \notin \{a^{\mathcal{I}} \mid a \text{ individual name}\}\}$$

  is a set of (disconnected) trees
- The model built in the proof of tableau algorithm soundness need not be forest-shaped because of the way it handles blocked individuals
- It can be shown that every satisfiable $\mathcal{ALC}$ KB has a forest-shaped model
- Unlike the case of $\mathcal{ALC}$ concepts, trees may be infinite

Tableau Algorithm for Expressive DLs

Tableau algorithm can be modified to handle extensions of $\mathcal{ALC}$ (with number restrictions, role inclusions, transitive roles...)

- additional tableau rule for each constructor
- new types of clashes
- different blocking conditions

Complexity Issues

- `CSat` decides whether an $\mathcal{ALC}$ concept is satisfiable
- `KBSat` decides whether an $\mathcal{ALC}$ KB is satisfiable
  - also concept satisfiability w.r.t. a TBox, subsumption and instance checking via polynomial reduction

Two questions for each case:

- What is the complexity of the algorithm?
  - what amount of ressources (time, memory) is required to run the algorithm, expressed as a function of the input size, in the worst possible case?
- What is the complexity of the decision problem solved?
  - what is the complexity of the best algorithms that solve the problem?

Complexity of `CSat`

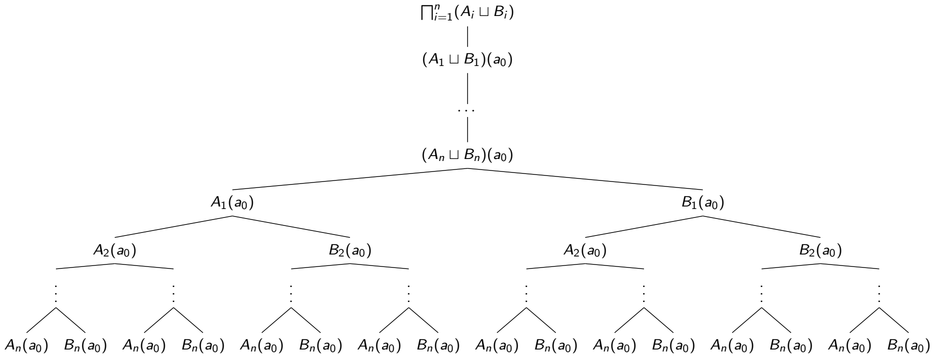
`CSat` needs exponential time and space:

- Due to the $\sqcup$-rule, exponentially many complete ABoxes may be generated
  - consider $C = \bigsqcap_{i=1}^{n}(A_i \sqcup B_i)$

Complexity of CSat

CSat needs exponential time and space:

- Due to the $\sqcup$-rule, exponentially many complete ABoxes may be generated
  - consider $C = \bigsqcap_{i=1}^{n}(A_i \sqcup B_i)$
  - $|C|$ is linear w.r.t. n and $\texttt{CSat}(C)$ generates 2^n complete ABoxes



Complexity of CSat

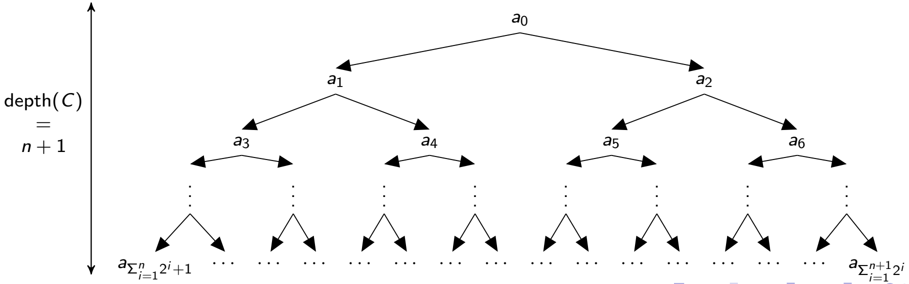
CSat needs exponential time and space:

- Due to the $\sqcup$-rule, exponentially many complete ABoxes may be generated
  - consider $C = \bigsqcap_{i=1}^{n} (A_i \sqcup B_i)$
- Due to the interaction of $\forall$- and $\exists$-rules, complete ABoxes may be exponentially large
  - consider $C = \bigsqcap_{i=0}^{n} \underbrace{\forall R. \ldots \forall R}_{i \text{ times}} (\exists R.B \sqcap \exists R.\neg B)$

Complexity of CSat

CSat needs exponential time and space:

- Due to the $\sqcup$-rule, exponentially many complete ABoxes may be generated
  - consider $C = \bigsqcap_{i=1}^{n}(A_i \sqcup B_i)$
- Due to the interaction of $\forall$- and $\exists$-rules, complete ABoxes may be exponentially large
  - consider $C = \bigsqcap_{i=0}^{n} \underbrace{\forall R. \ldots \forall R}_{i \text{ times}}(\exists R.B \sqcap \exists R.\neg B)$
  - $|C|$ is polynomial w.r.t. n and CSat(C) generates a complete ABox with $2^{n+2}-1$ individuals



Complexity of `CSat`

`CSat` can be modified so that it runs in polynomial space

- Keep only one ABox in memory at a time:
  - when applying the $\sqcup$-rule, first examine $\mathcal{A}_1$, then afterwards examine $\mathcal{A}_2$
  - keep in memory that the second disjunct needs to be checked

Complexity of CSat

CSat can be modified so that it runs in polynomial space

- Keep only one ABox in memory at a time:
  - when applying the $\sqcup$-rule, first examine $\mathcal{A}_1$, then afterwards examine $\mathcal{A}_2$
  - keep in memory that the second disjunct needs to be checked
- Keep at most $\mathsf{depth}(C) + 1$ individuals in memory:
  - explore the children of an individual one at a time, in a depth-first manner
  - possible because no interaction between individuals in different branches
  - store which $\exists R.C$ have been explored and which are left to do

Complexity of $\mathcal{ALC}$ Concept Satisfiability (No TBox)

- `CSat` runs in polynomial space so the problem of deciding whether an $\mathcal{ALC}$ concept is satisfiable is in PSpace
- Any hope for better algorithms?

  $\text{PTime} \subseteq \text{NP} \subseteq \text{PSpace} \subseteq \text{ExpTime} \subseteq \text{NExpTime} \subseteq \text{ExpSpace}$
  - inclusions are believed to be strict
- It can be shown that deciding whether an $\mathcal{ALC}$ concept is satisfiable is PSpace-hard
  - reduction from a PSpace-complete problem (for instance deciding whether a quantified Boolean formula is valid)

Theorem
Checking the satisfiability of an $\mathcal{ALC}$ concept in the absence of a TBox is PSpace-complete.

Complexity of KBSat

We cannot make KBSat run in polynomial space as we did for CSat because we may need to generate exponentially many individuals on a single "branch"

- consider $\mathcal{A} = \{F_1(a_0), \ldots, F_n(a_0)\}$ and

$$
\begin{aligned}
\mathcal{T} = &\{\bigsqcup_{i=1}^{n} F_i \sqsubseteq \exists R.\top\} \cup \{F_i \sqsubseteq \neg T_i \mid 1 \leq i \leq n\} \\
&\cup \{T_1 \sqcap \cdots \sqcap T_{k-1} \sqcap F_k \sqsubseteq \\
&\qquad \forall R.(F_1 \sqcap \cdots \sqcap F_{k-1} \sqcap T_k) \sqcap \\
&\quad \bigsqcap_{k+1 \leq \ell \leq n} ((T_\ell \sqcap \forall R.T_\ell) \sqcup (F_\ell \sqcap \forall R.F_\ell)) \mid 1 \leq k \leq n\}
\end{aligned}
$$

Complexity of $\mathcal{ALC}$ KB Satisfiability

- What is the complexity of KB satisfiability?

Theorem

Checking the satisfiability of an $\mathcal{ALC}$ KB is ExpTime-complete.

- we will next show membership
- hardness can be shown by reduction from an ExpTime-complete problem (for instance the problem of deciding the existence of a winning strategy for infinite Boolean games)

Complexity of $\mathcal{ALC}$ KB Satisfiability

EXPTIME membership

- Show that concept satisfiability w.r.t. a TBox is in EXPTIME
    - to decide whether a KB $\langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable, let $\mathcal{T}' = \mathcal{T} \cup \{C_a \sqsubseteq A \mid A(a) \in \mathcal{A}\} \cup \{C_a \sqsubseteq \exists R.C_b \mid R(a,b) \in \mathcal{A}\}$ and decide whether $\sqcap_{a \text{ individual of } \mathcal{A}} \exists S.C_a$ is satisfiable w.r.t. $\mathcal{T}'$ where S and all C_a are fresh role and concept names

Complexity of $\mathcal{ALC}$ KB Satisfiability

ExpTime membership

- Show that concept satisfiability w.r.t. a TBox is in ExpTime
  - to decide whether a KB $\langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable, let $\mathcal{T}' = \mathcal{T} \cup \{C_a \sqsubseteq A \mid A(a) \in \mathcal{A}\} \cup \{C_a \sqsubseteq \exists R.C_b \mid R(a,b) \in \mathcal{A}\}$ and decide whether $\sqcap_{a \text{ individual of } \mathcal{A}} \exists S.C_a$ is satisfiable w.r.t. $\mathcal{T}'$ where S and all C_a are fresh role and concept names
- We consider an atomic concept A_0
  - C is satisfiable w.r.t. $\mathcal{T}$ iff A_0 is satisfiable w.r.t. $\mathcal{T} \cup \{A_0 \sqsubseteq C\}$
- We assume that $\mathcal{T}$ contains a single axiom of the form $\top \sqsubseteq C_{\mathcal{T}}$ with $C_{\mathcal{T}}$ an $\mathcal{ALC}$ concept in NNF
  - A_0 is satisfiable w.r.t. $\mathcal{T}$ iff A_0 is satisfiable w.r.t. $\{\top \sqsubseteq \bigsqcap_{C \sqsubseteq D \in \mathcal{T}} \mathrm{nnf}(\neg C \sqcup D)\}$
- We assume that $A_0 \in \mathrm{sub}(C_{\mathcal{T}})$
  - otherwise A_0 is satisfiable w.r.t. $\{\top \sqsubseteq C_{\mathcal{T}}\}$ iff $C_{\mathcal{T}}$ is satisfiable

Complexity of $\mathcal{ALC}$ KB Satisfiability

Type elimination algorithm

We use a type elimination algorithm to decide whether A_0 is satisfiable w.r.t. $\{\top \sqsubseteq C_\mathcal{T}\}$

- A $\mathcal{T}$-type is a set of concepts $\tau \subseteq \mathrm{sub}(C_\mathcal{T})$ such that
  - $C \in \tau$ implies $\mathrm{nnf}(\neg C) \notin \tau$ for all $C \in \mathrm{sub}(C_\mathcal{T})$
  - $C \sqcap D \in \tau$ implies $C \in \tau$ and $D \in \tau$
  - $C \sqcup D \in \tau$ implies $C \in \tau$ or $D \in \tau$
  - $C_\mathcal{T} \in \tau$
- There are at most $2^{|\mathrm{sub}(C_\mathcal{T})|}$ types

Complexity of $\mathcal{ALC}$ KB Satisfiability

Type elimination algorithm

We use a type elimination algorithm to decide whether A_0 is satisfiable w.r.t. $\{\top \sqsubseteq C_\mathcal{T}\}$

- A $\mathcal{T}$-type is a set of concepts $\tau \subseteq \text{sub}(C_\mathcal{T})$ such that
  - $C \in \tau$ implies $\text{nnf}(\neg C) \notin \tau$ for all $C \in \text{sub}(C_\mathcal{T})$
  - $C \sqcap D \in \tau$ implies $C \in \tau$ and $D \in \tau$
  - $C \sqcup D \in \tau$ implies $C \in \tau$ or $D \in \tau$
  - $C_\mathcal{T} \in \tau$
- There are at most $2^{|\text{sub}(C_\mathcal{T})|}$ types
- The algorithm starts with the set of all types and iteratively removes the bad types that contain some existential restriction that cannot be satisfied in models of $\mathcal{T}$
  - Given a set of types T, τ is bad in T if there exists $\exists R.C \in \tau$ such that the set $\{C\} \cup \{D \mid \forall R.D \in \tau\}$ is not a subset of any type in T
- If at the end of the algorithm there remains some type that contains A_0, return "satisfiable", otherwise return "not satisfiable"

Complexity of $\mathcal{ALC}$ KB Satisfiability

Type elimination algorithm: Complexity

The type elimination algorithm runs in exponential time w.r.t. the size of $C_\mathcal{T}$

- At most $2^{|\mathsf{sub}(C_\mathcal{T})|}$ iterations and $|\mathsf{sub}(C_\mathcal{T})|$ is linear in the size of $C_\mathcal{T}$
- Each step takes polynomial time in the number of remaining types, thus is in $O(2^{|\mathsf{sub}(C_\mathcal{T})|})$
- Hence the algorithm runs in $O(2^{2*|\mathsf{sub}(C_\mathcal{T})|})$

Complexity of $\mathcal{ALC}$ KB Satisfiability

Type elimination algorithm: Soundness

The type elimination algorithm is sound

- Assume that the algorithm returns “satisfiable”
- At the end of the algorithm, T is a set of types such that every $\tau \in T$ is good in T and there exists $\tau_0 \in T$ such that $A_0 \in \tau_0$
- Let $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ with
  - $\Delta^\mathcal{I} = T$
  - $A^\mathcal{I} = \{\tau \mid A \in \tau\}$
  - $R^\mathcal{I} = \{(\tau_1, \tau_2) \mid \exists R.C \in \tau_1, \{C\} \cup \{D \mid \forall R.D \in \tau_1\} \subseteq \tau_2\}$
- Since $A_0 \in \tau_0$, $\tau_0 \in A_0^\mathcal{I}$ and $A_0^\mathcal{I} \neq \emptyset$
- Claim: $\mathcal{I} \models \top \sqsubseteq C_\mathcal{T}$
- Hence A_0 is satisfiable w.r.t. $\{\top \sqsubseteq C_\mathcal{T}\}$

Complexity of $\mathcal{ALC}$ KB Satisfiability

Type elimination algorithm: Soundness – Proof of the claim

$\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ with $\Delta^{\mathcal{I}} = T$, $A^{\mathcal{I}} = \{\tau \mid A \in \tau\}$ and
$R^{\mathcal{I}} = \{(\tau_1, \tau_2) \mid \exists R.C \in \tau_1, \{C\} \cup \{D \mid \forall R.D \in \tau_1\} \subseteq \tau_2\}$

- Show by induction that for every concept E, for every $\tau \in T$ such that $E \in \tau$, $\tau \in E^{\mathcal{I}}$
  - Base case: $E = A$ or $E = \neg A$.
    - if $E = A$, $A \in \tau$ implies $\tau \in A^{\mathcal{I}}$ by definition of $\mathcal{I}$
    - if $E = \neg A$, $\neg A \in \tau$ implies that $A \notin \tau$ because τ is a type, so $\tau \notin A^{\mathcal{I}}$

Complexity of $\mathcal{ALC}$ KB Satisfiability

Type elimination algorithm: Soundness – Proof of the claim

$\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ with $\Delta^{\mathcal{I}} = T$, $A^{\mathcal{I}} = \{\tau \mid A \in \tau\}$ and
$R^{\mathcal{I}} = \{(\tau_1, \tau_2) \mid \exists R.C \in \tau_1, \{C\} \cup \{D \mid \forall R.D \in \tau_1\} \subseteq \tau_2\}$

- Show by induction that for every concept E, for every $\tau \in T$ such that $E \in \tau$, $\tau \in E^{\mathcal{I}}$
  - Base case: $E = A$ or $E = \neg A$.
    - if $E = A$, $A \in \tau$ implies $\tau \in A^{\mathcal{I}}$ by definition of $\mathcal{I}$
    - if $E = \neg A$, $\neg A \in \tau$ implies that $A \notin \tau$ because τ is a type, so $\tau \notin A^{\mathcal{I}}$
  - Induction step:
    - if $E = C \sqcap D$, since τ is a type, then C and D are in τ, and by induction hypothesis, $\tau \in C^{\mathcal{I}}$ and $\tau \in D^{\mathcal{I}}$ so $\tau \in (C \sqcap D)^{\mathcal{I}}$
    - if $E = \exists R.C$, since τ is good in T, there exists τ' such that $(\tau, \tau') \in R^{\mathcal{I}}$ and $C \in \tau'$, so by induction hypothesis $\tau' \in C^{\mathcal{I}}$ so $\tau \in \exists R.C^{\mathcal{I}}$
    - $E = C \sqcup D$: left as practice
    - $E = \forall R.C$: left as practice

Complexity of $\mathcal{ALC}$ KB Satisfiability

Type elimination algorithm: Soundness – Proof of the claim

$\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ with $\Delta^{\mathcal{I}} = T$, $A^{\mathcal{I}} = \{\tau \mid A \in \tau\}$ and
$R^{\mathcal{I}} = \{(\tau_1, \tau_2) \mid \exists R.C \in \tau_1, \{C\} \cup \{D \mid \forall R.D \in \tau_1\} \subseteq \tau_2\}$

- Show by induction that for every concept E, for every $\tau \in T$ such that $E \in \tau$, $\tau \in E^{\mathcal{I}}$
  - Base case: $E = A$ or $E = \neg A$.
    - if $E = A$, $A \in \tau$ implies $\tau \in A^{\mathcal{I}}$ by definition of $\mathcal{I}$
    - if $E = \neg A$, $\neg A \in \tau$ implies that $A \notin \tau$ because τ is a type, so $\tau \notin A^{\mathcal{I}}$
  - Induction step:
    - if $E = C \sqcap D$, since τ is a type, then C and D are in τ, and by induction hypothesis, $\tau \in C^{\mathcal{I}}$ and $\tau \in D^{\mathcal{I}}$ so $\tau \in (C \sqcap D)^{\mathcal{I}}$
    - if $E = \exists R.C$, since τ is good in T, there exists τ' such that $(\tau, \tau') \in R^{\mathcal{I}}$ and $C \in \tau'$, so by induction hypothesis $\tau' \in C^{\mathcal{I}}$ so $\tau \in \exists R.C^{\mathcal{I}}$
    - $E = C \sqcup D$: left as practice
    - $E = \forall R.C$: left as practice
- For every $\tau \in T$, since τ is a $\mathcal{T}$-type, then $C_{\mathcal{T}} \in \tau$ so $\tau \in C_{\mathcal{T}}^{\mathcal{I}}$

Hence $\mathcal{I} \models \top \sqsubseteq C_{\mathcal{T}}$

Complexity of $\mathcal{ALC}$ KB Satisfiability

Type elimination algorithm: Completeness

The type elimination algorithm is complete

- Assume that A_0 is satisfiable w.r.t. $\{\top \sqsubseteq C_{\mathcal{T}}\}$
- There is a model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of $\top \sqsubseteq C_{\mathcal{T}}$ such that $A_0^{\mathcal{I}} \neq \emptyset$
- Claim: $T = \{\tau \mid e \in \Delta^{\mathcal{I}}, \tau = \{C \mid C \in \mathsf{sub}(C_{\mathcal{T}}), e \in C^{\mathcal{I}}\}\}$ is a set of $\mathcal{T}$-types such that there is $\tau \in T$ with $A_0 \in \tau$ and the type elimination algorithm does not remove any of the types in T

Complexity of $\mathcal{ALC}$ KB Satisfiability

Type elimination algorithm: Completeness – Proof of the claim

$$T = \{\tau \mid e \in \Delta^{\mathcal{I}}, \tau = \{C \mid C \in \mathsf{sub}(C_{\mathcal{T}}), e \in C^{\mathcal{I}}\}\}$$

- Since $A_0^{\mathcal{I}} \neq \emptyset$, there is $\tau \in T$ such that $A_0 \in \tau$
- T is a set of $\mathcal{T}$-types: for every $\tau \in T$
  - $e \in C^{\mathcal{I}}$ implies $e \notin \mathsf{nnf}(\neg C)^{\mathcal{I}}$, so $C \in \tau$ implies $\mathsf{nnf}(\neg C) \notin \tau$
  - $e \in (C \sqcap D)^{\mathcal{I}}$ implies $e \in C^{\mathcal{I}}$ and $e \in D^{\mathcal{I}}$, so $C \sqcap D \in \tau$ implies $C \in \tau$ and $D \in \tau$
  - similarly for $C \sqcup D$
  - $\mathcal{I} \models \top \sqsubseteq C_{\mathcal{T}}$, so $C_{\mathcal{T}} \in \tau$

Complexity of $\mathcal{ALC}$ KB Satisfiability

Type elimination algorithm: Completeness – Proof of the claim

$$T = \{\tau \mid e \in \Delta^{\mathcal{I}}, \tau = \{C \mid C \in \mathsf{sub}(C_{\mathcal{T}}), e \in C^{\mathcal{I}}\}\}$$

- Since $A_0^{\mathcal{I}} \neq \emptyset$, there is $\tau \in T$ such that $A_0 \in \tau$
- T is a set of $\mathcal{T}$-types: for every $\tau \in T$
  - $e \in C^{\mathcal{I}}$ implies $e \notin \mathsf{nnf}(\neg C)^{\mathcal{I}}$, so $C \in \tau$ implies $\mathsf{nnf}(\neg C) \notin \tau$
  - $e \in (C \sqcap D)^{\mathcal{I}}$ implies $e \in C^{\mathcal{I}}$ and $e \in D^{\mathcal{I}}$, so $C \sqcap D \in \tau$ implies $C \in \tau$ and $D \in \tau$
  - similarly for $C \sqcup D$
  - $\mathcal{I} \models \top \sqsubseteq C_{\mathcal{T}}$, so $C_{\mathcal{T}} \in \tau$
- Every $\tau \in T$ is good in T
  - let $\tau \in T$ and $\exists R.C \in \tau$
  - there is $e \in \Delta^{\mathcal{I}}$ such that $\tau = \{C \mid C \in \mathsf{sub}(C_{\mathcal{T}}), e \in C^{\mathcal{I}}\}$
  - $e \in \exists R.C^{\mathcal{I}}$ so there is $d \in \Delta^{\mathcal{I}}$ s.t. $(e, d) \in R^{\mathcal{I}}$ and $d \in C^{\mathcal{I}}$
  - for every D such that $\forall R.D \in \tau$, $e \in (\forall R.D)^{\mathcal{I}}$ so $d \in D^{\mathcal{I}}$
  - the type $\tau_d = \{E \mid E \in \mathsf{sub}(C_{\mathcal{T}}), d \in E^{\mathcal{I}}\}$ is such that $\{C\} \cup \{D \mid \forall R.D \in \tau\} \subseteq \tau_d$ and belongs to T
- The type elimination algorithm never removes any type $\tau \in T$: by induction on the number of iterations

In Practice: Optimizations

- Tableau algorithms are implemented and work well in practice
  - type elimination algorithm has optimal worst-case complexity but its best-case complexity is exponential!
- However, good performances crucially depends on optimizations
  - explore only one branch of one ABox at a time
  - strategies/heuristics for choosing next rule to apply
  - caching of results to reduce redundant computation
  - examine source of conflicts to prune search space
  - reduce numbers of $\sqcup$'s created by TBox inclusions
  - reduce number of satisfiability checks during classification

In Practice: Optimizations

Absorption: reduce number of disjunctions

If $\mathcal{T} = \{C_i \sqsubseteq D_i \mid 1 \leq i \leq n\}$, for each individual a, the TBox-rule builds n disjunctions $\mathrm{nnf}(\neg C_i \sqcup D_i)(a)$
$\rightarrow$ Try to reduce this number

- When C_i or D_i is an atomic concept, trigger the TBox-rule only when we have information about this concept
  - for inclusions $A \sqsubseteq D$ with atomic left-hand side, replace the TBox-rule by
    TBox-atomic-left-rule: if $A(a) \in \mathcal{A}$, a is not blocked, $A \sqsubseteq D \in \mathcal{T}$ (A atomic), and $D(a) \notin \mathcal{A}$, replace $\mathcal{A}$ with $\mathcal{A} \cup \{D(a)\}$.
  - for inclusions $D \sqsubseteq A$ with atomic right-hand side, replace the TBox-rule by
    TBox-atomic-right-rule: if $\neg A(a) \in \mathcal{A}$, a is not blocked, $D \sqsubseteq A \in \mathcal{T}$ (A atomic), and $\neg D(a) \notin \mathcal{A}$, replace $\mathcal{A}$ with $\mathcal{A} \cup \{\neg D(a)\}$.

In Practice: Optimizations

Absorption: reduce number of disjunctions

- Preprocess the TBox
  - to decrease the number of concept inclusions with non-atomic left- and right-hand sides
    - $(A \sqcap C \sqsubseteq D) \equiv (A \sqsubseteq \neg C \sqcup D)$
    - $(D \sqsubseteq A \sqcup C) \equiv (D \sqcap \neg C \sqsubseteq A)$
    - ...
  - to obtain a single concept inclusion per atomic concept with this concept as right- or left-hand side ("absorption")
    - $A \sqsubseteq C_1, A \sqsubseteq C_2 \Rightarrow A \sqsubseteq C_1 \sqcap C_2$
    - $C_1 \sqsubseteq A, C_2 \sqsubseteq A \Rightarrow C_1 \sqcup C_2 \sqsubseteq A$

In Practice: Optimizations

Classification: reduce number of satisfiability checks

Classification consists in finding all pairs of atomic concepts A, B such that $\mathcal{T} \models A \sqsubseteq B$

- Naïve approach: test satisfiability of $A \sqcap \neg B$ w.r.t. $\mathcal{T}$ for every pair A, B
- Reduce the number of satisfiability checks
    - some subsumptions are obvious
        - $A \sqsubseteq A$
        - $A \sqsubseteq B \in \mathcal{T}$
    - use simple reasoning to obtain new (non-)subsumptions
        - if we found that $\mathcal{T} \models A \sqsubseteq B$ and $\mathcal{T} \models B \sqsubseteq C$, then $\mathcal{T} \models A \sqsubseteq C$
        - if we found that $\mathcal{T} \models A \sqsubseteq B$ and $\mathcal{T} \not\models A \sqsubseteq C$, then $\mathcal{T} \not\models B \sqsubseteq C$

References

- Baader, Calvanese, McGuinness, Nardi, Patel-Schneider (2003): The Description Logic Handbook: Theory, Implementation, and Applications (book, can be found online)
- Bienvenu (2022): Ontologies & Description Logics (lecture: https://www.labri.fr/perso/meghyn/teaching/lola-2022/2-lola-tableau.pdf)
- Baader (2019): course on Description Logics (lecture: https://tu-dresden.de/ing/informatik/thi/lat/studium/lehrveranstaltungen/sommersemester-2019/description-logic)
- Ortiz (2012): course on Declarative Knowledge Processing (lecture: http://www.kr.tuwien.ac.at/education/dekl_slides/ws12/)