

Ontologies & Description Logics

KNOWLEDGE REPRESENTATION

Meghyn Bienvenu (LaBRI - CNRS & *Université de Bordeaux*)

REASONING IN EXPRESSIVE DLS

TABLEAU METHOD

Tableau method: popular approach for reasoning in **expressive DLs**

- implemented in **state-of-the-art DL reasoners**

Tableau algorithms are used to **decide satisfiability**

- solve other tasks (e.g. entailment) by reducing them to satisfiability

TABLEAU METHOD

Tableau method: popular approach for reasoning in **expressive DLs**

- implemented in **state-of-the-art DL reasoners**

Tableau algorithms are used to **decide satisfiability**

- solve other tasks (e.g. entailment) by reducing them to satisfiability

Idea: to determine whether a given (concept or KB) Ψ is satisfiable, **try to construct a (representation of a) model** of Ψ

- if we succeed, then we have shown that Ψ is satisfiable
- if we fail despite having **considered all possibilities**, then we have proven that Ψ is unsatisfiable

ALC CONCEPTS

Recall that \mathcal{ALC} concepts are built using the following constructors:

$\top \quad \perp \quad \neg \quad \sqcup \quad \sqcap \quad \forall r.C \quad \exists r.C$

ALC CONCEPTS

Recall that \mathcal{ALC} concepts are built using the following constructors:

$$\top \quad \perp \quad \neg \quad \sqcup \quad \sqcap \quad \forall r.C \quad \exists r.C$$

We say that an \mathcal{ALC} concept C is in **negation normal form (NNF)** if the symbol \neg only appears directly in front of atomic concepts.

- **in NNF**: $A \sqcap \neg B, \exists r. \neg A, \neg A \sqcup \neg B$
- **not in NNF**: $\neg(A \sqcap B), \exists r. \neg(\forall s. B), A \sqcup \neg \forall r. B, \neg \top$

ALC CONCEPTS

Recall that \mathcal{ALC} concepts are built using the following constructors:

$$\top \quad \perp \quad \neg \quad \sqcup \quad \sqcap \quad \forall r.C \quad \exists r.C$$

We say that an \mathcal{ALC} concept C is in **negation normal form (NNF)** if the symbol \neg only appears directly in front of atomic concepts.

- **in NNF**: $A \sqcap \neg B, \exists r. \neg A, \neg A \sqcup \neg B$
- **not in NNF**: $\neg(A \sqcap B), \exists r. \neg(\forall s.B), A \sqcup \neg \forall r.B, \neg \top$

Fact. Every \mathcal{ALC} concept C can be **transformed into an equivalent concept in NNF** in linear time by applying the following rewrite rules:

$$\begin{array}{lll} \neg \top \rightsquigarrow \perp & \neg(C \sqcap D) \rightsquigarrow \neg C \sqcup \neg D & \neg(\forall r.C) \rightsquigarrow \exists r. \neg C \\ \neg \perp \rightsquigarrow \top & \neg(C \sqcup D) \rightsquigarrow \neg C \sqcap \neg D & \neg(\exists r.C) \rightsquigarrow \forall r. \neg C \end{array}$$

Note: say C and D are equivalent if the empty TBox entails $C \equiv D$.

SATISFIABILITY OF ALC-CONCEPTS VIA TABLEAU

We begin by presenting a tableau algorithm for deciding
satisfiability of \mathcal{ALC} -concepts (in NNF) w.r.t. the empty TBox

Procedure for testing satisfiability of C_0 :

- We work with a set S of ABoxes
- Initially, S contains a single ABox $\{C_0(a_0)\}$

SATISFIABILITY OF ALC-CONCEPTS VIA TABLEAU

We begin by presenting a tableau algorithm for deciding
satisfiability of \mathcal{ALC} -concepts (in NNF) w.r.t. the empty TBox

Procedure for testing satisfiability of C_0 :

- We work with a set S of ABoxes
- Initially, S contains a single ABox $\{C_0(a_0)\}$
- At each stage, we **apply a tableau rule** to some $\mathcal{A} \in S$
(note: rules are detailed on next slide)

SATISFIABILITY OF ALC-CONCEPTS VIA TABLEAU

We begin by presenting a tableau algorithm for deciding
satisfiability of \mathcal{ALC} -concepts (in NNF) w.r.t. the empty TBox

Procedure for testing satisfiability of C_0 :

- We work with a set S of ABoxes
- Initially, S contains a single ABox $\{C_0(a_0)\}$
- At each stage, we **apply a tableau rule** to some $\mathcal{A} \in S$
(note: rules are detailed on next slide)
- A rule application involves replacing \mathcal{A} by one or two ABoxes that extend \mathcal{A} with new assertions

SATISFIABILITY OF ALC-CONCEPTS VIA TABLEAU

We begin by presenting a tableau algorithm for deciding **satisfiability of \mathcal{ALC} -concepts (in NNF) w.r.t. the empty TBox**

Procedure for testing satisfiability of C_0 :

- We work with a set S of ABoxes
- Initially, S contains a single ABox $\{C_0(a_0)\}$
- At each stage, we **apply a tableau rule** to some $\mathcal{A} \in S$
(note: rules are detailed on next slide)
- A rule application involves replacing \mathcal{A} by one or two ABoxes that extend \mathcal{A} with new assertions
- **Stop applying rules when either:**
 - every $\mathcal{A} \in S$ contains a **clash**, i.e. an assertion $\perp(b)$ or a pair of assertions $\{B(b), \neg B(b)\}$
 - some $\mathcal{A} \in S$ is **clash-free** and **complete**: no rule can be applied to \mathcal{A}

SATISFIABILITY OF ALC-CONCEPTS VIA TABLEAU

We begin by presenting a tableau algorithm for deciding **satisfiability of \mathcal{ALC} -concepts (in NNF) w.r.t. the empty TBox**

Procedure for testing satisfiability of C_0 :

- We work with a set S of ABoxes
- Initially, S contains a single ABox $\{C_0(a_0)\}$
- At each stage, we **apply a tableau rule** to some $\mathcal{A} \in S$
(note: rules are detailed on next slide)
- A rule application involves replacing \mathcal{A} by one or two ABoxes that extend \mathcal{A} with new assertions
- **Stop applying rules when either:**
 - every $\mathcal{A} \in S$ contains a **clash**, i.e. an assertion $\perp(b)$ or a pair of assertions $\{B(b), \neg B(b)\}$
 - some $\mathcal{A} \in S$ is **clash-free** and **complete**: no rule can be applied to \mathcal{A}
- Return **'yes, satisfiable'** if some $\mathcal{A} \in S$ is **clash-free**, else "no".

TABLEAU RULES FOR ALC

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

TABLEAU RULES FOR ALC

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

TABLEAU RULES FOR ALC

- \sqcap -rule:** if $(C_1 \sqcap C_2)(a) \in \mathcal{A}$ and $\{C_1(a), C_2(a)\} \not\subseteq \mathcal{A}$
then 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 $\{C_1(a), C_2(a)\} \cap \mathcal{A} = \emptyset$
then replace \mathcal{A} with $\mathcal{A} \cup \{C_1(a)\}$ **and** $\mathcal{A} \cup \{C_2(a)\}$
- \forall -rule:** if $\{\forall r.C(a), r(a, b)\} \in \mathcal{A}$ and $C(b) \notin \mathcal{A}$
then replace \mathcal{A} with $\mathcal{A} \cup \{C(b)\}$

TABLEAU RULES FOR ALC

- \sqcap -rule:** if $(C_1 \sqcap C_2)(a) \in \mathcal{A}$ and $\{C_1(a), C_2(a)\} \not\subseteq \mathcal{A}$
then 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 $\{C_1(a), C_2(a)\} \cap \mathcal{A} = \emptyset$
then replace \mathcal{A} with $\mathcal{A} \cup \{C_1(a)\}$ **and** $\mathcal{A} \cup \{C_2(a)\}$
- \forall -rule:** if $\{\forall r.C(a), r(a, b)\} \in \mathcal{A}$ and $C(b) \notin \mathcal{A}$
then replace \mathcal{A} with $\mathcal{A} \cup \{C(b)\}$
- \exists -rule:** if $\{\exists r.C(a)\} \in \mathcal{A}$ and no b with $\{r(a, b), C(b)\} \subseteq \mathcal{A}$,
then **pick a new individual name d** and
replace \mathcal{A} with $\mathcal{A} \cup \{r(a, d), C(d)\}$

FIRST EXAMPLE: \sqcap AND \sqcup

Test satisfiability of concept $C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$

FIRST EXAMPLE: \sqcap AND \sqcup

Test satisfiability of concept $C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$

Start with $S = \{ \mathcal{A}_0 \}$ where $\mathcal{A}_0 = \{ ((A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A))(a_0) \}$.

FIRST EXAMPLE: \sqcap AND \sqcup

Test satisfiability of concept $C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$

Start with $S = \{ \mathcal{A}_0 \}$ where $\mathcal{A}_0 = \{ ((A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A))(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_0 :

get $S = \{ \mathcal{A}'_0 \}$ where $\mathcal{A}'_0 = \mathcal{A}_0 \cup \{ (A \sqcup B)(a_0), ((\neg B \sqcup D) \sqcap \neg A)(a_0) \}$.

FIRST EXAMPLE: \sqcap AND \sqcup

Test satisfiability of concept $C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$

Start with $S = \{ \mathcal{A}_0 \}$ where $\mathcal{A}_0 = \{ ((A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A))(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_0 :

get $S = \{ \mathcal{A}'_0 \}$ where $\mathcal{A}'_0 = \mathcal{A}_0 \cup \{ (A \sqcup B)(a_0), ((\neg B \sqcup D) \sqcap \neg A)(a_0) \}$.

FIRST EXAMPLE: \sqcap AND \sqcup

Test satisfiability of concept $C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$

Start with $S = \{ \mathcal{A}_0 \}$ where $\mathcal{A}_0 = \{ ((A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A))(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_0 :

get $S = \{ \mathcal{A}'_0 \}$ where $\mathcal{A}'_0 = \mathcal{A}_0 \cup \{ (A \sqcup B)(a_0), ((\neg B \sqcup D) \sqcap \neg A)(a_0) \}$.

Apply \sqcup -rule to \mathcal{A}'_0 :

get $S = \{ \mathcal{A}_1, \mathcal{A}_2 \}$ where $\mathcal{A}_1 = \mathcal{A}'_0 \cup \{ A(a_0) \}$ and $\mathcal{A}_2 = \mathcal{A}'_0 \cup \{ B(a_0) \}$.

FIRST EXAMPLE: \sqcap AND \sqcup

Test satisfiability of concept $C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$

Start with $S = \{ \mathcal{A}_0 \}$ where $\mathcal{A}_0 = \{ ((A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A))(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_0 :

get $S = \{ \mathcal{A}'_0 \}$ where $\mathcal{A}'_0 = \mathcal{A}_0 \cup \{ (A \sqcup B)(a_0), ((\neg B \sqcup D) \sqcap \neg A)(a_0) \}$.

Apply \sqcup -rule to \mathcal{A}'_0 :

get $S = \{ \mathcal{A}_1, \mathcal{A}_2 \}$ where $\mathcal{A}_1 = \mathcal{A}'_0 \cup \{ A(a_0) \}$ and $\mathcal{A}_2 = \mathcal{A}'_0 \cup \{ B(a_0) \}$.

FIRST EXAMPLE: \sqcap AND \sqcup

Test satisfiability of concept $C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$

Start with $S = \{ \mathcal{A}_0 \}$ where $\mathcal{A}_0 = \{ ((A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A))(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_0 :

get $S = \{ \mathcal{A}'_0 \}$ where $\mathcal{A}'_0 = \mathcal{A}_0 \cup \{ (A \sqcup B)(a_0), ((\neg B \sqcup D) \sqcap \neg A)(a_0) \}$.

Apply \sqcup -rule to \mathcal{A}'_0 :

get $S = \{ \mathcal{A}_1, \mathcal{A}_2 \}$ where $\mathcal{A}_1 = \mathcal{A}'_0 \cup \{ A(a_0) \}$ and $\mathcal{A}_2 = \mathcal{A}'_0 \cup \{ B(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_1 :

get $S = \{ \mathcal{A}'_1, \mathcal{A}_2 \}$ where $\mathcal{A}'_1 = \mathcal{A}_1 \cup \{ (\neg B \sqcup D)(a_0), \neg A(a_0) \}$

FIRST EXAMPLE: \sqcap AND \sqcup

Test satisfiability of concept $C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$

Start with $S = \{ \mathcal{A}_0 \}$ where $\mathcal{A}_0 = \{ ((A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A))(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_0 :

get $S = \{ \mathcal{A}'_0 \}$ where $\mathcal{A}'_0 = \mathcal{A}_0 \cup \{ (A \sqcup B)(a_0), ((\neg B \sqcup D) \sqcap \neg A)(a_0) \}$.

Apply \sqcup -rule to \mathcal{A}'_0 :

get $S = \{ \mathcal{A}_1, \mathcal{A}_2 \}$ where $\mathcal{A}_1 = \mathcal{A}'_0 \cup \{ A(a_0) \}$ and $\mathcal{A}_2 = \mathcal{A}'_0 \cup \{ B(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_1 :

get $S = \{ \mathcal{A}'_1, \mathcal{A}_2 \}$ where $\mathcal{A}'_1 = \mathcal{A}_1 \cup \{ (\neg B \sqcup D)(a_0), \neg A(a_0) \}$

\mathcal{A}'_1 contains clash $\{ A(a_0), \neg A(a_0) \}$!

FIRST EXAMPLE: \sqcap AND \sqcup

Test satisfiability of concept $C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$

Start with $S = \{ \mathcal{A}_0 \}$ where $\mathcal{A}_0 = \{ ((A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A))(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_0 :

get $S = \{ \mathcal{A}'_0 \}$ where $\mathcal{A}'_0 = \mathcal{A}_0 \cup \{ (A \sqcup B)(a_0), ((\neg B \sqcup D) \sqcap \neg A)(a_0) \}$.

Apply \sqcup -rule to \mathcal{A}'_0 :

get $S = \{ \mathcal{A}_1, \mathcal{A}_2 \}$ where $\mathcal{A}_1 = \mathcal{A}'_0 \cup \{ A(a_0) \}$ and $\mathcal{A}_2 = \mathcal{A}'_0 \cup \{ B(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_1 :

get $S = \{ \mathcal{A}'_1, \mathcal{A}_2 \}$ where $\mathcal{A}'_1 = \mathcal{A}_1 \cup \{ (\neg B \sqcup D)(a_0), \neg A(a_0) \}$

FIRST EXAMPLE: \sqcap AND \sqcup

Test satisfiability of concept $C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$

Start with $S = \{ \mathcal{A}_0 \}$ where $\mathcal{A}_0 = \{ ((A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A))(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_0 :

get $S = \{ \mathcal{A}'_0 \}$ where $\mathcal{A}'_0 = \mathcal{A}_0 \cup \{ (A \sqcup B)(a_0), ((\neg B \sqcup D) \sqcap \neg A)(a_0) \}$.

Apply \sqcup -rule to \mathcal{A}'_0 :

get $S = \{ \mathcal{A}_1, \mathcal{A}_2 \}$ where $\mathcal{A}_1 = \mathcal{A}'_0 \cup \{ A(a_0) \}$ and $\mathcal{A}_2 = \mathcal{A}'_0 \cup \{ B(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_1 :

get $S = \{ \mathcal{A}'_1, \mathcal{A}_2 \}$ where $\mathcal{A}'_1 = \mathcal{A}_1 \cup \{ (\neg B \sqcup D)(a_0), \neg A(a_0) \}$

Apply \sqcap -rule to \mathcal{A}_2 :

get $S = \{ \mathcal{A}'_1, \mathcal{A}'_2 \}$ where $\mathcal{A}'_2 = \mathcal{A}_2 \cup \{ (\neg B \sqcup D)(a_0), \neg A(a_0) \}$

FIRST EXAMPLE: \sqcap AND \sqcup

Test satisfiability of concept $C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$

Start with $S = \{ \mathcal{A}_0 \}$ where $\mathcal{A}_0 = \{ ((A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A))(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_0 :

get $S = \{ \mathcal{A}'_0 \}$ where $\mathcal{A}'_0 = \mathcal{A}_0 \cup \{ (A \sqcup B)(a_0), ((\neg B \sqcup D) \sqcap \neg A)(a_0) \}$.

Apply \sqcup -rule to \mathcal{A}'_0 :

get $S = \{ \mathcal{A}_1, \mathcal{A}_2 \}$ where $\mathcal{A}_1 = \mathcal{A}'_0 \cup \{ A(a_0) \}$ and $\mathcal{A}_2 = \mathcal{A}'_0 \cup \{ B(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_1 :

get $S = \{ \mathcal{A}'_1, \mathcal{A}_2 \}$ where $\mathcal{A}'_1 = \mathcal{A}_1 \cup \{ (\neg B \sqcup D)(a_0), \neg A(a_0) \}$

Apply \sqcap -rule to \mathcal{A}_2 :

get $S = \{ \mathcal{A}'_1, \mathcal{A}'_2 \}$ where $\mathcal{A}'_2 = \mathcal{A}_2 \cup \{ (\neg B \sqcup D)(a_0), \neg A(a_0) \}$

FIRST EXAMPLE: \sqcap AND \sqcup

Test satisfiability of concept $C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$

Start with $S = \{ \mathcal{A}_0 \}$ where $\mathcal{A}_0 = \{ ((A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A))(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_0 :

get $S = \{ \mathcal{A}'_0 \}$ where $\mathcal{A}'_0 = \mathcal{A}_0 \cup \{ (A \sqcup B)(a_0), ((\neg B \sqcup D) \sqcap \neg A)(a_0) \}$.

Apply \sqcup -rule to \mathcal{A}'_0 :

get $S = \{ \mathcal{A}_1, \mathcal{A}_2 \}$ where $\mathcal{A}_1 = \mathcal{A}'_0 \cup \{ A(a_0) \}$ and $\mathcal{A}_2 = \mathcal{A}'_0 \cup \{ B(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_1 :

get $S = \{ \mathcal{A}'_1, \mathcal{A}_2 \}$ where $\mathcal{A}'_1 = \mathcal{A}_1 \cup \{ (\neg B \sqcup D)(a_0), \neg A(a_0) \}$

Apply \sqcap -rule to \mathcal{A}_2 :

get $S = \{ \mathcal{A}'_1, \mathcal{A}'_2 \}$ where $\mathcal{A}'_2 = \mathcal{A}_2 \cup \{ (\neg B \sqcup D)(a_0), \neg A(a_0) \}$

Apply \sqcup -rule to \mathcal{A}'_2 :

get $S = \{ \mathcal{A}'_1, \mathcal{A}_3, \mathcal{A}_4 \}$ where $\mathcal{A}_3 = \mathcal{A}'_2 \cup \{ \neg B(a_0) \}$, $\mathcal{A}_4 = \mathcal{A}'_2 \cup \{ D(a_0) \}$

FIRST EXAMPLE: \sqcap AND \sqcup

Test satisfiability of concept $C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$

Start with $S = \{ \mathcal{A}_0 \}$ where $\mathcal{A}_0 = \{ ((A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A))(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_0 :

get $S = \{ \mathcal{A}'_0 \}$ where $\mathcal{A}'_0 = \mathcal{A}_0 \cup \{ (A \sqcup B)(a_0), ((\neg B \sqcup D) \sqcap \neg A)(a_0) \}$.

Apply \sqcup -rule to \mathcal{A}'_0 :

get $S = \{ \mathcal{A}_1, \mathcal{A}_2 \}$ where $\mathcal{A}_1 = \mathcal{A}'_0 \cup \{ A(a_0) \}$ and $\mathcal{A}_2 = \mathcal{A}'_0 \cup \{ B(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_1 :

get $S = \{ \mathcal{A}'_1, \mathcal{A}_2 \}$ where $\mathcal{A}'_1 = \mathcal{A}_1 \cup \{ (\neg B \sqcup D)(a_0), \neg A(a_0) \}$

Apply \sqcap -rule to \mathcal{A}_2 :

get $S = \{ \mathcal{A}'_1, \mathcal{A}'_2 \}$ where $\mathcal{A}'_2 = \mathcal{A}_2 \cup \{ (\neg B \sqcup D)(a_0), \neg A(a_0) \}$

Apply \sqcup -rule to \mathcal{A}'_2 :

get $S = \{ \mathcal{A}'_1, \mathcal{A}_3, \mathcal{A}_4 \}$ where $\mathcal{A}_3 = \mathcal{A}'_2 \cup \{ \neg B(a_0) \}$, $\mathcal{A}_4 = \mathcal{A}'_2 \cup \{ D(a_0) \}$

\mathcal{A}_3 contains clash $\{ B(a_0), \neg B(a_0) \}$!

FIRST EXAMPLE: \sqcap AND \sqcup

Test satisfiability of concept $C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$

Start with $S = \{ \mathcal{A}_0 \}$ where $\mathcal{A}_0 = \{ ((A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A))(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_0 :

get $S = \{ \mathcal{A}'_0 \}$ where $\mathcal{A}'_0 = \mathcal{A}_0 \cup \{ (A \sqcup B)(a_0), ((\neg B \sqcup D) \sqcap \neg A)(a_0) \}$.

Apply \sqcup -rule to \mathcal{A}'_0 :

get $S = \{ \mathcal{A}_1, \mathcal{A}_2 \}$ where $\mathcal{A}_1 = \mathcal{A}'_0 \cup \{ A(a_0) \}$ and $\mathcal{A}_2 = \mathcal{A}'_0 \cup \{ B(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_1 :

get $S = \{ \mathcal{A}'_1, \mathcal{A}_2 \}$ where $\mathcal{A}'_1 = \mathcal{A}_1 \cup \{ (\neg B \sqcup D)(a_0), \neg A(a_0) \}$

Apply \sqcap -rule to \mathcal{A}_2 :

get $S = \{ \mathcal{A}'_1, \mathcal{A}'_2 \}$ where $\mathcal{A}'_2 = \mathcal{A}_2 \cup \{ (\neg B \sqcup D)(a_0), \neg A(a_0) \}$

Apply \sqcup -rule to \mathcal{A}'_2 :

get $S = \{ \mathcal{A}'_1, \mathcal{A}_3, \mathcal{A}_4 \}$ where $\mathcal{A}_3 = \mathcal{A}'_2 \cup \{ \neg B(a_0) \}$, $\mathcal{A}_4 = \mathcal{A}'_2 \cup \{ D(a_0) \}$

\mathcal{A}_4 is complete, so we can stop.

FIRST EXAMPLE: \sqcap AND \sqcup

Test satisfiability of concept $C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$

Start with $S = \{ \mathcal{A}_0 \}$ where $\mathcal{A}_0 = \{ ((A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A))(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_0 :

get $S = \{ \mathcal{A}'_0 \}$ where $\mathcal{A}'_0 = \mathcal{A}_0 \cup \{ (A \sqcup B)(a_0), ((\neg B \sqcup D) \sqcap \neg A)(a_0) \}$.

Apply \sqcup -rule to \mathcal{A}'_0 :

get $S = \{ \mathcal{A}_1, \mathcal{A}_2 \}$ where $\mathcal{A}_1 = \mathcal{A}'_0 \cup \{ A(a_0) \}$ and $\mathcal{A}_2 = \mathcal{A}'_0 \cup \{ B(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_1 :

get $S = \{ \mathcal{A}'_1, \mathcal{A}_2 \}$ where $\mathcal{A}'_1 = \mathcal{A}_1 \cup \{ (\neg B \sqcup D)(a_0), \neg A(a_0) \}$

Apply \sqcap -rule to \mathcal{A}_2 :

get $S = \{ \mathcal{A}'_1, \mathcal{A}'_2 \}$ where $\mathcal{A}'_2 = \mathcal{A}_2 \cup \{ (\neg B \sqcup D)(a_0), \neg A(a_0) \}$

Apply \sqcup -rule to \mathcal{A}'_2 :

get $S = \{ \mathcal{A}'_1, \mathcal{A}_3, \mathcal{A}_4 \}$ where $\mathcal{A}_3 = \mathcal{A}'_2 \cup \{ \neg B(a_0) \}$, $\mathcal{A}_4 = \mathcal{A}'_2 \cup \{ D(a_0) \}$

\mathcal{A}_4 is complete and contains no clash $\Rightarrow C_0$ is satisfiable

PREVIOUS EXAMPLE IN GRAPHICAL FORMAT

Test satisfiability of concept $C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$

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

PREVIOUS EXAMPLE IN GRAPHICAL FORMAT

Test satisfiability of concept $C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$

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

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

\sqcap -rule

$$(A \sqcup B) (a_0)$$

PREVIOUS EXAMPLE IN GRAPHICAL FORMAT

Test satisfiability of concept $C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$

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

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

$$(A \sqcup B) (a_0)$$

$$A(a_0)$$

$$B(a_0)$$

\sqcup -rule

PREVIOUS EXAMPLE IN GRAPHICAL FORMAT

Test satisfiability of concept $C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$

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

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

$$(A \sqcup B) (a_0)$$

$$A (a_0)$$

$$B (a_0)$$

\sqcap -rule $(\neg B \sqcup D) (a_0)$

$$\neg A (a_0)$$

PREVIOUS EXAMPLE IN GRAPHICAL FORMAT

Test satisfiability of concept $C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$

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

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

$$(A \sqcup B) (a_0)$$

$$A (a_0)$$

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

$$\neg A (a_0)$$

$$B (a_0)$$

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

$$\neg A (a_0)$$

\sqcap -rule

PREVIOUS EXAMPLE IN GRAPHICAL FORMAT

Test satisfiability of concept $C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$

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

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

$$(A \sqcup B) (a_0)$$

$$A(a_0)$$

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

$$\neg A(a_0)$$

$$B(a_0)$$

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

$$\neg A(a_0)$$

$$\neg B(a_0)$$

$$D(a_0) \quad \sqcup\text{-rule}$$

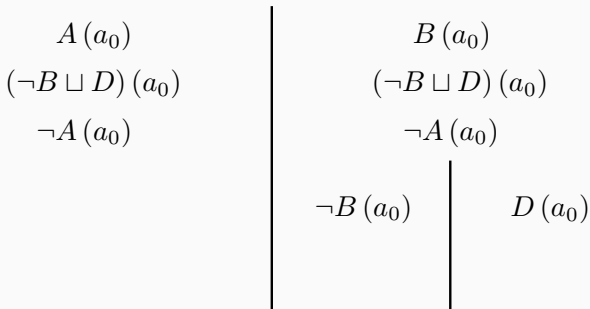
PREVIOUS EXAMPLE IN GRAPHICAL FORMAT

Test satisfiability of concept $C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$

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

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

$$(A \sqcup B) (a_0)$$






PREVIOUS EXAMPLE IN GRAPHICAL FORMAT

Test satisfiability of concept $C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$

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

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

$$(A \sqcup B) (a_0)$$

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

EXAMPLE: WITNESSING INTERPRETATION

In our example, we had the **complete and clash-free ABox** \mathcal{A}_4 :

$$\begin{aligned} & ((A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A))(a_0) \quad (A \sqcup B)(a_0) \\ & ((\neg B \sqcup D) \sqcap \neg A)(a_0) \quad B(a_0) \quad (\neg B \sqcup D)(a_0) \quad \neg A(a_0) \quad D(a_0) \end{aligned}$$

Can build from \mathcal{A}_4 the interpretation \mathcal{I} with:

- $\Delta^{\mathcal{I}} = \{a_0\}$ use individuals from \mathcal{A}_4
- $A^{\mathcal{I}} = \emptyset$ since \mathcal{A}_4 does not contain $A(a_0)$
- $B^{\mathcal{I}} = D^{\mathcal{I}} = \{a_0\}$ since \mathcal{A}_4 contains $B(a_0)$ and $D(a_0)$

We can verify that $(A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)^{\mathcal{I}} = \{a_0\}$.

- \mathcal{I} witnesses the satisfiability of $C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$

ANOTHER EXAMPLE: \forall AND \exists

Let's use the tableau procedure to test satisfiability of

$$C_0 = \exists r.A \sqcap \forall r.\neg A$$

ANOTHER EXAMPLE: \forall AND \exists

Let's use the tableau procedure to test satisfiability of

$$C_0 = \exists r.A \sqcap \forall r.\neg A$$

Start with $S = \{ \mathcal{A}_0 \}$ where $\mathcal{A}_0 = \{ (\exists r.A \sqcap \forall r.\neg A)(a_0) \}$.

ANOTHER EXAMPLE: \forall AND \exists

Let's use the tableau procedure to test satisfiability of

$$C_0 = \exists r.A \sqcap \forall r.\neg A$$

Start with $S = \{ \mathcal{A}_0 \}$ where $\mathcal{A}_0 = \{ (\exists r.A \sqcap \forall r.\neg A)(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_0 :

get $S = \{ \mathcal{A}'_0 \}$ where $\mathcal{A}'_0 = \mathcal{A}_0 \cup \{ (\exists r.A)(a_0), (\forall r.\neg A)(a_0) \}$.

ANOTHER EXAMPLE: \forall AND \exists

Let's use the tableau procedure to test satisfiability of

$$C_0 = \exists r.A \sqcap \forall r.\neg A$$

Start with $S = \{ \mathcal{A}_0 \}$ where $\mathcal{A}_0 = \{ (\exists r.A \sqcap \forall r.\neg A)(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_0 :

get $S = \{ \mathcal{A}'_0 \}$ where $\mathcal{A}'_0 = \mathcal{A}_0 \cup \{ (\exists r.A)(a_0), (\forall r.\neg A)(a_0) \}$.

ANOTHER EXAMPLE: \forall AND \exists

Let's use the tableau procedure to test satisfiability of

$$C_0 = \exists r.A \sqcap \forall r.\neg A$$

Start with $S = \{ \mathcal{A}_0 \}$ where $\mathcal{A}_0 = \{ (\exists r.A \sqcap \forall r.\neg A)(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_0 :

get $S = \{ \mathcal{A}'_0 \}$ where $\mathcal{A}'_0 = \mathcal{A}_0 \cup \{ (\exists r.A)(a_0), (\forall r.\neg A)(a_0) \}$.

Apply \exists -rule to \mathcal{A}'_0 :

get $S = \{ \mathcal{A}''_0 \}$ where $\mathcal{A}''_0 = \mathcal{A}'_0 \cup \{ r(a_0, a_1), A(a_1) \}$.

ANOTHER EXAMPLE: \forall AND \exists

Let's use the tableau procedure to test satisfiability of

$$C_0 = \exists r.A \sqcap \forall r.\neg A$$

Start with $S = \{ \mathcal{A}_0 \}$ where $\mathcal{A}_0 = \{ (\exists r.A \sqcap \forall r.\neg A)(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_0 :

get $S = \{ \mathcal{A}'_0 \}$ where $\mathcal{A}'_0 = \mathcal{A}_0 \cup \{ (\exists r.A)(a_0), (\forall r.\neg A)(a_0) \}$.

Apply \exists -rule to \mathcal{A}'_0 :

get $S = \{ \mathcal{A}''_0 \}$ where $\mathcal{A}''_0 = \mathcal{A}'_0 \cup \{ r(a_0, a_1), A(a_1) \}$.

ANOTHER EXAMPLE: \forall AND \exists

Let's use the tableau procedure to test satisfiability of

$$C_0 = \exists r.A \sqcap \forall r.\neg A$$

Start with $S = \{ \mathcal{A}_0 \}$ where $\mathcal{A}_0 = \{ (\exists r.A \sqcap \forall r.\neg A)(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_0 :

get $S = \{ \mathcal{A}'_0 \}$ where $\mathcal{A}'_0 = \mathcal{A}_0 \cup \{ (\exists r.A)(a_0), (\forall r.\neg A)(a_0) \}$.

Apply \exists -rule to \mathcal{A}'_0 :

get $S = \{ \mathcal{A}''_0 \}$ where $\mathcal{A}''_0 = \mathcal{A}'_0 \cup \{ r(a_0, a_1), A(a_1) \}$.

Apply \forall -rule to \mathcal{A}''_0 :

get $S = \{ \mathcal{A}'''_0 \}$ where $\mathcal{A}'''_0 = \mathcal{A}''_0 \cup \{ \neg A(a_1) \}$.

ANOTHER EXAMPLE: \forall AND \exists

Let's use the tableau procedure to test satisfiability of

$$C_0 = \exists r.A \sqcap \forall r.\neg A$$

Start with $S = \{ \mathcal{A}_0 \}$ where $\mathcal{A}_0 = \{ (\exists r.A \sqcap \forall r.\neg A)(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_0 :

get $S = \{ \mathcal{A}'_0 \}$ where $\mathcal{A}'_0 = \mathcal{A}_0 \cup \{ (\exists r.A)(a_0), (\forall r.\neg A)(a_0) \}$.

Apply \exists -rule to \mathcal{A}'_0 :

get $S = \{ \mathcal{A}''_0 \}$ where $\mathcal{A}''_0 = \mathcal{A}'_0 \cup \{ r(a_0, a_1), A(a_1) \}$.

Apply \forall -rule to \mathcal{A}''_0 :

get $S = \{ \mathcal{A}'''_0 \}$ where $\mathcal{A}'''_0 = \mathcal{A}''_0 \cup \{ \neg A(a_1) \}$.

\mathcal{A}'''_0 contains clash $\{A(a_1), \neg A(a_1)\}$!

ANOTHER EXAMPLE: \forall AND \exists

Let's use the tableau procedure to test satisfiability of

$$C_0 = \exists r.A \sqcap \forall r.\neg A$$

Start with $S = \{ \mathcal{A}_0 \}$ where $\mathcal{A}_0 = \{ (\exists r.A \sqcap \forall r.\neg A)(a_0) \}$.

Apply \sqcap -rule to \mathcal{A}_0 :

get $S = \{ \mathcal{A}'_0 \}$ where $\mathcal{A}'_0 = \mathcal{A}_0 \cup \{ (\exists r.A)(a_0), (\forall r.\neg A)(a_0) \}$.

Apply \exists -rule to \mathcal{A}'_0 :

get $S = \{ \mathcal{A}''_0 \}$ where $\mathcal{A}''_0 = \mathcal{A}'_0 \cup \{ r(a_0, a_1), A(a_1) \}$.

Apply \forall -rule to \mathcal{A}''_0 :

get $S = \{ \mathcal{A}'''_0 \}$ where $\mathcal{A}'''_0 = \mathcal{A}''_0 \cup \{ \neg A(a_1) \}$.

The only set in S contains a clash $\Rightarrow C_0$ is unsatisfiable

PREVIOUS EXAMPLE IN GRAPHICAL FORMAT

Test satisfiability of concept $C_0 = \exists r.A \sqcap \forall r.\neg A$

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

PREVIOUS EXAMPLE IN GRAPHICAL FORMAT

Test satisfiability of concept $C_0 = \exists r.A \sqcap \forall r.\neg A$

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

$$(\exists r.A)(a_0) \quad \sqcap\text{-rule}$$

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

PREVIOUS EXAMPLE IN GRAPHICAL FORMAT

Test satisfiability of concept $C_0 = \exists r.A \sqcap \forall r.\neg A$

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

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

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

$$r(a_0, a_1) \quad \exists\text{-rule}$$

$$A(a_1)$$

PREVIOUS EXAMPLE IN GRAPHICAL FORMAT

Test satisfiability of concept $C_0 = \exists r.A \sqcap \forall r.\neg A$

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

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

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

$$r(a_0, a_1)$$

$$A(a_1)$$

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

PREVIOUS EXAMPLE IN GRAPHICAL FORMAT

Test satisfiability of concept $C_0 = \exists r.A \sqcap \forall r.\neg A$

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

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

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

$$r(a_0, a_1)$$

$$\begin{array}{c} A(a_1) \\ \neg A(a_1) \end{array}$$



PREVIOUS EXAMPLE IN GRAPHICAL FORMAT

Test satisfiability of concept $C_0 = \exists r.A \sqcap \forall r.\neg A$

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

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

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

$$r(a_0, a_1)$$

$$\begin{array}{c} A(a_1) \\ \neg A(a_1) \end{array}$$



Conclude that C_0 is unsatisfiable

FURTHER EXAMPLE WITH \forall AND \exists

Suppose that we consider a slightly different concept

$$C_0 = \exists r.A \sqcap \forall r.\neg \mathbf{B}$$

Now the algorithm yields the following complete, clash-free ABox:

$$(\exists r.A \sqcap \forall r.\neg B)(a_0) \quad (\exists r.A)(a_0) \quad (\forall r.\neg B)(a_0) \quad r(a_0, a_1) \quad A(a_1) \quad \neg B(a_1)$$

FURTHER EXAMPLE WITH \forall AND \exists

Suppose that we consider a slightly different concept

$$C_0 = \exists r.A \sqcap \forall r.\neg B$$

Now the algorithm yields the following complete, clash-free ABox:

$$(\exists r.A \sqcap \forall r.\neg B)(a_0) \quad (\exists r.A)(a_0) \quad (\forall r.\neg B)(a_0) \quad r(a_0, a_1) \quad A(a_1) \quad \neg B(a_1)$$

Corresponding interpretation \mathcal{I} :

- $\Delta^{\mathcal{I}} = \{a_0, a_1\}$
- $A^{\mathcal{I}} = \{a_1\}$
- $B^{\mathcal{I}} = \emptyset$
- $r^{\mathcal{I}} = \{(a_0, a_1)\}$

Can check that \mathcal{I} is such that $C_0^{\mathcal{I}} = \{a_0\}$.

PROPERTIES OF THE TABLEAU ALGORITHM

Let's call our tableau algorithm **CSat** (for concept satisfiability).

To show that **CSat** is a decision procedure, we must show:

Termination: The algorithm **CSat** always terminates.

Soundness: **CSat** outputs “yes” on input $C_0 \Rightarrow C_0$ is satisfiable.

Completeness: C_0 satisfiable \Rightarrow **CSat** will output “yes”.

Subconcepts of a concept:

$$\text{sub}(A) = \{A\}$$

$$\text{sub}(\neg C) = \{\neg C\} \cup \text{sub}(C)$$

$$\text{sub}(\exists r.C) = \{\exists r.C\} \cup \text{sub}(C)$$

$$\text{sub}(\forall r.C) = \{\forall r.C\} \cup \text{sub}(C)$$

$$\text{sub}(C_1 \sqcup C_2) = \{C_1 \sqcup C_2\} \cup \text{sub}(C_1) \cup \text{sub}(C_2)$$

$$\text{sub}(C_1 \sqcap C_2) = \{C_1 \sqcap C_2\} \cup \text{sub}(C_1) \cup \text{sub}(C_2)$$

PRELIMINARY DEFINITIONS

Subconcepts of a concept:

$$\text{sub}(A) = \{A\}$$

$$\text{sub}(\neg C) = \{\neg C\} \cup \text{sub}(C)$$

$$\text{sub}(\exists r.C) = \{\exists r.C\} \cup \text{sub}(C)$$

$$\text{sub}(\forall r.C) = \{\forall r.C\} \cup \text{sub}(C)$$

$$\text{sub}(C_1 \sqcup C_2) = \{C_1 \sqcup C_2\} \cup \text{sub}(C_1) \cup \text{sub}(C_2)$$

$$\text{sub}(C_1 \sqcap C_2) = \{C_1 \sqcap C_2\} \cup \text{sub}(C_1) \cup \text{sub}(C_2)$$

Role depth of a concept:

$$\text{depth}(A) = \text{depth}(\top) = \text{depth}(\perp) = 0$$

$$\text{depth}(\neg C) = \text{depth}(C)$$

$$\text{depth}(\exists r.C) = \text{depth}(\forall r.C) = \text{depth}(C) + 1$$

$$\text{depth}(C_1 \sqcup C_2) = \text{depth}(C_1 \sqcap C_2) = \max(\text{depth}(C_1), \text{depth}(C_2))$$

PRELIMINARY DEFINITIONS

Subconcepts of a concept: $|\text{sub}(C)| \leq |C|$

$$\text{sub}(A) = \{A\}$$

$$\text{sub}(\neg C) = \{\neg C\} \cup \text{sub}(C)$$

$$\text{sub}(\exists r.C) = \{\exists r.C\} \cup \text{sub}(C)$$

$$\text{sub}(\forall r.C) = \{\forall r.C\} \cup \text{sub}(C)$$

$$\text{sub}(C_1 \sqcup C_2) = \{C_1 \sqcup C_2\} \cup \text{sub}(C_1) \cup \text{sub}(C_2)$$

$$\text{sub}(C_1 \sqcap C_2) = \{C_1 \sqcap C_2\} \cup \text{sub}(C_1) \cup \text{sub}(C_2)$$

Role depth of a concept: $\text{depth}(C) \leq |C|$

$$\text{depth}(A) = \text{depth}(\top) = \text{depth}(\perp) = 0$$

$$\text{depth}(\neg C) = \text{depth}(C)$$

$$\text{depth}(\exists r.C) = \text{depth}(\forall r.C) = \text{depth}(C) + 1$$

$$\text{depth}(C_1 \sqcup C_2) = \text{depth}(C_1 \sqcap C_2) = \max(\text{depth}(C_1), \text{depth}(C_2))$$

TERMINATION OF CSAT

Suppose we run `CSat` starting from $S = \{\{C_0(a_0)\}\}$.

TERMINATION OF CSAT

Suppose we run **CSat** starting from $S = \{\{C_0(a_0)\}\}$.

We observe that for every ABox \mathcal{A} generated by the procedure:

TERMINATION OF CSAT

Suppose we run **CSat** starting from $S = \{\{C_0(a_0)\}\}$.

We observe that for every ABox \mathcal{A} generated by the procedure:

1. if $D(b) \in \mathcal{A}$, then $D \in \text{sub}(C_0)$

TERMINATION OF CSAT

Suppose we run **CSat** starting from $S = \{\{C_0(a_0)\}\}$.

We observe that for every ABox \mathcal{A} generated by the procedure:

1. if $D(b) \in \mathcal{A}$, then $D \in \text{sub}(C_0)$
 - \mathcal{A} contains at most $|C_0|$ concept assertions per individual

TERMINATION OF CSAT

Suppose we run **CSat** starting from $S = \{\{C_0(a_0)\}\}$.

We observe that for every ABox \mathcal{A} generated by the procedure:

1. if $D(b) \in \mathcal{A}$, then $D \in \text{sub}(C_0)$
 - \mathcal{A} contains at most $|C_0|$ concept assertions per individual
2. the set of role assertions in \mathcal{A} forms a tree

TERMINATION OF CSAT

Suppose we run **CSat** starting from $S = \{\{C_0(a_0)\}\}$.

We observe that for every ABox \mathcal{A} generated by the procedure:

1. if $D(b) \in \mathcal{A}$, then $D \in \text{sub}(C_0)$
 - \mathcal{A} contains at most $|C_0|$ concept assertions per individual
2. the set of role assertions in \mathcal{A} forms a tree
3. if $D(b) \in \mathcal{A}$ and the unique path from a_0 to b has length k , then $\text{depth}(D) \leq \text{depth}(C_0) - k$

TERMINATION OF CSAT

Suppose we run **CSat** starting from $S = \{\{C_0(a_0)\}\}$.

We observe that for every ABox \mathcal{A} generated by the procedure:

1. if $D(b) \in \mathcal{A}$, then $D \in \text{sub}(C_0)$
 - \mathcal{A} contains at most $|C_0|$ concept assertions per individual
2. the set of role assertions in \mathcal{A} forms a tree
3. if $D(b) \in \mathcal{A}$ and the unique path from a_0 to b has length k , then $\text{depth}(D) \leq \text{depth}(C_0) - k$
 - each individual in \mathcal{A} is at distance $\leq \text{depth}(C_0)$ from a_0

TERMINATION OF CSAT

Suppose we run **CSat** starting from $S = \{\{C_0(a_0)\}\}$.

We observe that for every ABox \mathcal{A} generated by the procedure:

1. if $D(b) \in \mathcal{A}$, then $D \in \text{sub}(C_0)$
 - \mathcal{A} contains at most $|C_0|$ concept assertions per individual
2. the set of role assertions in \mathcal{A} forms a tree
3. if $D(b) \in \mathcal{A}$ and the unique path from a_0 to b has length k , then $\text{depth}(D) \leq \text{depth}(C_0) - k$
 - each individual in \mathcal{A} is at distance $\leq \text{depth}(C_0)$ from a_0
4. for every individual b in \mathcal{A} , there are at most $|C_0|$ individuals c such that $r(b, c) \in \mathcal{A}$ for some r (at most one per existential concept)

TERMINATION OF CSAT

Suppose we run **CSat** starting from $S = \{\{C_0(a_0)\}\}$.

We observe that for every ABox \mathcal{A} generated by the procedure:

1. if $D(b) \in \mathcal{A}$, then $D \in \text{sub}(C_0)$
 - \mathcal{A} contains at most $|C_0|$ concept assertions per individual
2. the set of role assertions in \mathcal{A} forms a tree
3. if $D(b) \in \mathcal{A}$ and the unique path from a_0 to b has length k , then $\text{depth}(D) \leq \text{depth}(C_0) - k$
 - each individual in \mathcal{A} is at distance $\leq \text{depth}(C_0)$ from a_0
4. for every individual b in \mathcal{A} , there are at most $|C_0|$ individuals c such that $r(b, c) \in \mathcal{A}$ for some r (at most one per existential concept)

Thus: **bound on the size of ABoxes** generated by the procedure

TERMINATION OF CSAT

Suppose we run **CSat** starting from $S = \{\{C_0(a_0)\}\}$.

We observe that for every ABox \mathcal{A} generated by the procedure:

1. if $D(b) \in \mathcal{A}$, then $D \in \text{sub}(C_0)$
 - \mathcal{A} contains at most $|C_0|$ concept assertions per individual
2. the set of role assertions in \mathcal{A} forms a tree
3. if $D(b) \in \mathcal{A}$ and the unique path from a_0 to b has length k , then $\text{depth}(D) \leq \text{depth}(C_0) - k$
 - each individual in \mathcal{A} is at distance $\leq \text{depth}(C_0)$ from a_0
4. for every individual b in \mathcal{A} , there are at most $|C_0|$ individuals c such that $r(b, c) \in \mathcal{A}$ for some r (at most one per existential concept)

Thus: **bound on the size of ABoxes** generated by the procedure

The tableau procedure **only adds assertions** to ABoxes

⇒ **eventually all ABoxes will contain a clash or will be complete**

SOUNDNESS OF CSAT (1)

Suppose that **CSat** returns “yes” on input C_0 .

Then S must contain a **complete and clash-free ABox** \mathcal{A} .

SOUNDNESS OF CSAT (1)

Suppose that **CSat** returns “yes” on input C_0 .

Then S must contain a **complete and clash-free ABox** \mathcal{A} .

Use \mathcal{A} to 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 $C_0^{\mathcal{I}} \neq \emptyset$

SOUNDNESS OF CSAT (1)

Suppose that **CSat** returns “yes” on input C_0 .

Then S must contain a **complete and clash-free ABox** \mathcal{A} .

Use \mathcal{A} to 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 $C_0^{\mathcal{I}} \neq \emptyset$

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

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

SOUNDNESS OF CSAT (2)

Base case: $D = A$ or $D = \neg A$ or $D = \top$ or $D = \perp$

SOUNDNESS OF CSAT (2)

Base case: $D = A$ or $D = \neg A$ or $D = \top$ or $D = \perp$

If $D = A$, then $b \in A^{\mathcal{I}}$.

If $D = \neg A$, then $A(b) \notin \mathcal{A}$, so $b \in \neg A^{\mathcal{I}}$.

If $D = \top$, trivially $b \in \top^{\mathcal{I}} = \Delta^{\mathcal{I}}$. Cannot have $D = \perp$ since clash-free.

Induction hypothesis (IH): suppose holds whenever $|D| \leq k$

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

Again, many cases to consider:

- $D = E \sqcap F$: since \mathcal{A} is complete, it must contain both $E(b)$ and $F(b)$. Applying the IH, we get $b \in E^{\mathcal{I}}$ and $b \in F^{\mathcal{I}}$, hence $b \in (E \sqcap F)^{\mathcal{I}}$
- $D = \exists r.E$: since \mathcal{A} is complete, there exists c such that $r(b, c) \in \mathcal{A}$ and $E(c) \in \mathcal{A}$. Then $(b, c) \in r^{\mathcal{I}}$. From IH, get $c \in E^{\mathcal{I}}$, 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 the **concept C_0 is satisfiable**.
Then the ABox $\{C_0(a_0)\}$ must be satisfiable too.

COMPLETENESS OF CSAT

Suppose that the **concept C_0 is satisfiable**.

Then the ABox $\{C_0(a_0)\}$ must be satisfiable too.

We observe that the **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 \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 is satisfiable or \mathcal{A}_2 is satisfiable**.

COMPLETENESS OF CSAT

Suppose that the **concept C_0 is satisfiable**.

Then the ABox $\{C_0(a_0)\}$ must be satisfiable too.

We observe that the **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 \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 is satisfiable or \mathcal{A}_2 is satisfiable**.

We start with a satisfiable ABox and the rules are satisfiability-preserving, so eventually we will **reach a complete, satisfiable (thus: clash-free) ABox** and output 'yes'.

COMPLEXITY OF CSAT

Bad news: our algorithm may require exponential time and space...
To see why, consider what happens if we run **CSat** on the concept

$$\bigwedge_{0 \leq i < n} \underbrace{\forall r. \dots \forall r.}_{i \text{ times}} (\exists r. B \sqcap \exists r. \neg B)$$

COMPLEXITY OF CSAT

Bad news: our algorithm may require exponential time and space...
To see why, consider what happens if we run **CSat** on the concept

$$\bigcap_{0 \leq i < n} \underbrace{\forall r. \dots \forall r.}_{i \text{ times}} (\exists r. B \sqcap \exists r. \neg B)$$

Good news: can **modify algorithm so it runs in polynomial space**

- instead of set of ABoxes, **keep only 1 ABox in memory at a time**
 - when apply the \sqcup -rule, first examine \mathcal{A}_1 , then afterwards examine \mathcal{A}_2
 - remember that second disjunct stills needs to be checked
- **explore the children of an individual one at a time**
 - possible because no interaction between the different “branches”
 - store which $\exists r.C$ concepts have been tested, which are left to do
- this allows us to keep at most $|C_0|$ individuals in memory at a time

Hierarchy of complexity classes

$\text{PTIME} \subseteq \text{NP} \subseteq \dots \subseteq \text{PSPACE} \subseteq \text{EXPTIME} \subseteq \text{NEXPTIME} \subseteq \dots \subseteq \text{EXPSPACE} \dots$

(it is believed that all inclusions are strict)

Hierarchy of complexity classes

$\text{PTIME} \subseteq \text{NP} \subseteq \dots \subseteq \text{PSPACE} \subseteq \text{EXPTIME} \subseteq \text{NEXPTIME} \subseteq \dots \subseteq \text{EXPSPACE} \dots$

(it is believed that all inclusions are strict)

PSPACE = class of decision **problems solvable in polynomial space**

PSPACE-complete problems = hardest problems in PSPACE

Hierarchy of complexity classes

$\text{PTIME} \subseteq \text{NP} \subseteq \dots \subseteq \text{PSPACE} \subseteq \text{EXPTIME} \subseteq \text{NEXPTIME} \subseteq \dots \subseteq \text{EXPSPACE} \dots$

(it is believed that all inclusions are strict)

PSPACE = class of decision **problems solvable in polynomial space**

PSPACE-complete problems = hardest problems in PSPACE

Theorem: *ALC* concept satisfiability (no TBox) is PSPACE-complete.

- Membership in PSPACE shown using modified tableau procedure
- Hardness for PSPACE shown by giving a reduction from some known PSPACE-hard problem (e.g. QBF validity)

EXTENSION TO KB SATISFIABILITY

Now we want to modify the algorithm to handle KB satisfiability.

Adding an ABox is easy: simply start with $\{\mathcal{A}\}$ instead of $\{C_0(a_0)\}$

EXTENSION TO KB SATISFIABILITY

Now we want to modify the algorithm to handle KB satisfiability.

Adding an ABox is easy: simply start with $\{\mathcal{A}\}$ instead of $\{C_0(a_0)\}$

Adding a TBox is a bit more tricky...

Idea: if $C \sqsubseteq D$, then **every element must satisfy either $\neg C$ or D**

EXTENSION TO KB SATISFIABILITY

Now we want to modify the algorithm to handle KB satisfiability.

Adding an ABox is easy: simply start with $\{\mathcal{A}\}$ instead of $\{C_0(a_0)\}$

Adding a TBox is a bit more tricky...

Idea: if $C \sqsubseteq D$, then **every element must satisfy either $\neg C$ or D**

Concretely, we might try adding the following rule:

TBox rule if a is in \mathcal{A} , $C \sqsubseteq D \in \mathcal{T}$, & $(\text{NNF}(\neg C) \sqcup \text{NNF}(D))(a) \notin \mathcal{A}$
then replace \mathcal{A} with $\mathcal{A} \cup \{(\text{NNF}(\neg C) \sqcup \text{NNF}(D))(a)\}$

EXAMPLE: NON-TERMINATION

Let's try the modified procedure on the KB $(\{F \sqsubseteq \exists s.F\}, \{F(a)\})$

EXAMPLE: NON-TERMINATION

Let's try the modified procedure on the KB $(\{F \sqsubseteq \exists s.F\}, \{F(a)\})$

Seems we have a problem... How can we ensure termination?

Basic idea: if two individuals “look the same”, then it is unnecessary to explore both of them

Basic idea: if two individuals “look the same”, then it is unnecessary to explore both of them

Formally: given individuals a, b from \mathcal{A} , we say that b **blocks** a if:

- $\{C \mid C(a) \in \mathcal{A}\} \subseteq \{C \mid C(b) \in \mathcal{A}\}$
- b was present in \mathcal{A} before a was introduced

Say that individual a **is blocked** (in \mathcal{A}) if some b blocks a .

Basic idea: if two individuals “look the same”, then it is unnecessary to explore both of them

Formally: given individuals a, b from \mathcal{A} , we say that b **blocks** a if:

- $\{C \mid C(a) \in \mathcal{A}\} \subseteq \{C \mid C(b) \in \mathcal{A}\}$
- b was present in \mathcal{A} before a was introduced

Say that individual a **is blocked** (in \mathcal{A}) if some b blocks a .

Modify rules so that they **only apply to unblocked individuals**.

TABLEAU RULES FOR KBS

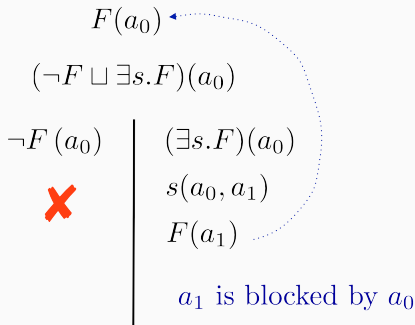
- \sqcap -rule:** if $(C_1 \sqcap C_2)(a) \in \mathcal{A}$, **a is not blocked**, and $\{C_1(a), C_2(a)\} \not\subseteq \mathcal{A}$, then replace \mathcal{A} with $\mathcal{A} \cup \{C_1(a), C_2(a)\}$
- \sqcup -rule:** if $(C_1 \sqcup C_2)(a) \in \mathcal{A}$, **a is not blocked**, and $\{C_1(a), C_2(a)\} \cap \mathcal{A} = \emptyset$, then replace \mathcal{A} with $\mathcal{A} \cup \{C_1(a)\}$ and $\mathcal{A} \cup \{C_2(a)\}$
- \forall -rule:** if $\{\forall r.C(a), r(a, b)\} \in \mathcal{A}$, **a is not blocked**, and $C(b) \notin \mathcal{A}$, then replace \mathcal{A} with $\mathcal{A} \cup \{C(b)\}$
- \exists -rule:** if $\{\exists r.C(a)\} \in \mathcal{A}$, **a is not blocked**, and no $\{r(a, b), C(b)\} \subseteq \mathcal{A}$, then **pick a new individual name** d and replace \mathcal{A} with $\mathcal{A} \cup \{r(a, d), C(d)\}$
- \sqsubseteq -rule:** if a appears in \mathcal{A} and **a is not blocked**, $C \sqsubseteq D \in \mathcal{T}$, and $(\text{NNF}(\neg C) \sqcup \text{NNF}(D))(a) \notin \mathcal{A}$, then replace \mathcal{A} with $\mathcal{A} \cup \{(\text{NNF}(\neg C) \sqcup \text{NNF}(D))(a)\}$

EXAMPLE: BLOCKING

Let's try blocking on the problematic KB $(\{F \sqsubseteq \exists s.F\}, \{F(a)\})$

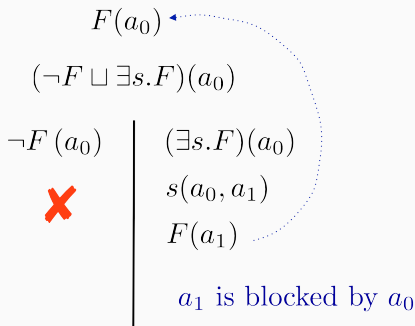
EXAMPLE: BLOCKING

Let's try blocking on the problematic KB $(\{F \sqsubseteq \exists s.F\}, \{F(a)\})$



EXAMPLE: BLOCKING

Let's try blocking on the problematic KB $(\{F \sqsubseteq \exists s.F\}, \{F(a)\})$



We obtain a **complete and clash-free ABox** \Rightarrow the KB is **satisfiable**

ANOTHER BLOCKING EXAMPLE

Consider the TBox $\mathcal{T} = \{A \sqsubseteq \exists r.A, A \sqsubseteq B, \exists r.B \sqsubseteq D\}$ and suppose we want to test whether $\mathcal{T} \models A \sqsubseteq D$.

We can do this by running the algorithm on $(\mathcal{T}, \{(A \sqcap \neg D)(a_0)\})$.

ANOTHER BLOCKING EXAMPLE

Consider the TBox $\mathcal{T} = \{A \sqsubseteq \exists r.A, A \sqsubseteq B, \exists r.B \sqsubseteq D\}$ and suppose we want to test whether $\mathcal{T} \models A \sqsubseteq D$.

We can do this by running the algorithm on $(\mathcal{T}, \{(A \sqcap \neg D)(a_0)\})$.

Result: the KB is unsatisfiable $\Rightarrow \mathcal{T} \models A \sqsubseteq D$

ANOTHER BLOCKING EXAMPLE

Consider the TBox $\mathcal{T} = \{A \sqsubseteq \exists r.A, A \sqsubseteq B, \exists r.B \sqsubseteq D\}$ and suppose we want to test whether $\mathcal{T} \models A \sqsubseteq D$.

We can do this by running the algorithm on $(\mathcal{T}, \{(A \sqcap \neg D)(a_0)\})$.

Result: the KB is unsatisfiable $\Rightarrow \mathcal{T} \models A \sqsubseteq D$

Observe: individual can be blocked, then later become unblocked

Let's call our new tableau algorithm **KBSat** (for KB satisfiability).

Termination: The algorithm **KBSat** always terminates.

- similar to before: bound the size of generated ABoxes

Let's call our new tableau algorithm **KBSat** (for KB satisfiability).

Termination: The algorithm **KBSat** always terminates.

- similar to before: bound the size of generated ABoxes

Soundness: **KBSat** outputs “yes” on $(\mathcal{T}, \mathcal{A}) \Rightarrow (\mathcal{T}, \mathcal{A})$ is satisfiable.

- again, we use complete, clash-free ABox to build a model
- tricky part: need to handle the blocked individuals

Let's call our new tableau algorithm **KBSat** (for KB satisfiability).

Termination: The algorithm **KBSat** always terminates.

- similar to before: bound the size of generated ABoxes

Soundness: **KBSat** outputs “yes” on $(\mathcal{T}, \mathcal{A}) \Rightarrow (\mathcal{T}, \mathcal{A})$ is satisfiable.

- again, we use complete, clash-free ABox to build a model
- tricky part: need to handle the blocked individuals

Completeness: $(\mathcal{T}, \mathcal{A})$ satisfiable \Rightarrow **KBSat** will output “yes”.

- again, show rules satisfiability-preserving

PROPERTIES OF KBSAT

Let's call our new tableau algorithm **KBSat** (for KB satisfiability).

Termination: The algorithm **KBSat** always terminates.

- similar to before: bound the size of generated ABoxes

Soundness: **KBSat** outputs “yes” on $(\mathcal{T}, \mathcal{A}) \Rightarrow (\mathcal{T}, \mathcal{A})$ is satisfiable.

- again, we use complete, clash-free ABox to build a model
- tricky part: need to handle the blocked individuals

Completeness: $(\mathcal{T}, \mathcal{A})$ satisfiable \Rightarrow **KBSat** will output “yes”.

- again, show rules satisfiability-preserving

So: **KBSat** is a **decision procedure for KB satisfiability**.

COMPLEXITY OF REASONING IN EXPRESSIVE DLS

Tableau procedure takes **exponential time and space**

- can have **exponentially long 'branches'** to explore

Complexity results tell us this is **unavoidable in worst case**:

Theorem: In \mathcal{ALC} , KB satisfiability is **EXPTIME-complete**

- for highly expressive DLs (\rightsquigarrow **OWL 2**): **complexity even higher**

Despite high worst-case complexity, **tableau algorithms** for \mathcal{ALC} and other expressive DLs **can work well in practice**.

Despite high worst-case complexity, **tableau algorithms** for \mathcal{ALC} and other expressive DLs **can work well in practice**.

However, **good performance crucially depends on optimizations!**

Despite high worst-case complexity, **tableau algorithms** for \mathcal{ALC} and other expressive DLs **can work well in practice**.

However, **good performance crucially depends on optimizations!**

Many types of optimizations:

- explore only one branch of one ABox at a time

Despite high worst-case complexity, **tableau algorithms** for \mathcal{ALC} and other expressive DLs **can work well in practice**.

However, **good performance crucially depends on optimizations!**

Many types of optimizations:

- explore only one branch of one ABox at a time
- strategies / heuristics for choosing next rule to apply

Despite high worst-case complexity, **tableau algorithms** for \mathcal{ALC} and other expressive DLs **can work well in practice**.

However, **good performance crucially depends on optimizations!**

Many types of 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

OPTIMIZATIONS

Despite high worst-case complexity, **tableau algorithms** for \mathcal{ALC} and other expressive DLs **can work well in practice**.

However, **good performance crucially depends on optimizations!**

Many types of 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 (backjumping)

Despite high worst-case complexity, **tableau algorithms** for \mathcal{ALC} and other expressive DLs **can work well in practice**.

However, **good performance crucially depends on optimizations!**

Many types of 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 (backjumping)
- **reduce number of \sqcup 's created by TBox inclusions (absorption)**

OPTIMIZATIONS

Despite high worst-case complexity, **tableau algorithms** for \mathcal{ALC} and other expressive DLs **can work well in practice**.

However, **good performance crucially depends on optimizations!**

Many types of 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 (backjumping)
- **reduce number of \sqcup 's created by TBox inclusions (absorption)**
- **reduce number of satisfiability checks during classification**

ABSORPTION (1)

When $\mathcal{T} = \{C_i \sqsubseteq D_i \mid 1 \leq i \leq n\}$, we get n disjunctions per individual:

$$(\text{NNF}(\neg \mathbf{C}_1) \sqcup \text{NNF}(\mathbf{D}_1))(\mathbf{a}), \dots, (\text{NNF}(\neg \mathbf{C}_n) \sqcup \text{NNF}(\mathbf{D}_n))(\mathbf{a})$$

ABSORPTION (1)

When $\mathcal{T} = \{C_i \sqsubseteq D_i \mid 1 \leq i \leq n\}$, we get n disjunctions per individual:

$$(\text{NNF}(\neg C_1) \sqcup \text{NNF}(D_1))(a), \dots, (\text{NNF}(\neg C_n) \sqcup \text{NNF}(D_n))(a)$$

Observation: if have $A \sqsubseteq D$ with A a concept name

- if don't have $A(a)$, can satisfy the inclusion by choosing $\neg A(a)$
- if have $A(a)$, then must have $D(a)$

ABSORPTION (1)

When $\mathcal{T} = \{C_i \sqsubseteq D_i \mid 1 \leq i \leq n\}$, we get n disjunctions per individual:

$$(\text{NNF}(\neg C_1) \sqcup \text{NNF}(D_1))(a), \dots, (\text{NNF}(\neg C_n) \sqcup \text{NNF}(D_n))(a)$$

Observation: if have $A \sqsubseteq D$ with A a concept name

- if don't have $A(a)$, can satisfy the inclusion by choosing $\neg A(a)$
- if have $A(a)$, then must have $D(a)$

So for **inclusions with atomic left-hand side**, can **replace \sqsubseteq -rule** by:

\sqsubseteq^{at} -rule: if $A(a) \in \mathcal{A}$, a is not blocked, $A \sqsubseteq D \in \mathcal{T}$ (with A atomic), and $D(a) \notin \mathcal{A}$, then replace \mathcal{A} with $\mathcal{A} \cup \{D(a)\}$

ABSORPTION (1)

When $\mathcal{T} = \{C_i \sqsubseteq D_i \mid 1 \leq i \leq n\}$, we get n disjunctions per individual:

$$(\text{NNF}(\neg C_1) \sqcup \text{NNF}(D_1))(\mathbf{a}), \dots, (\text{NNF}(\neg C_n) \sqcup \text{NNF}(D_n))(\mathbf{a})$$

Observation: if have $A \sqsubseteq D$ with A a concept name

- if don't have $A(a)$, can satisfy the inclusion by choosing $\neg A(a)$
- if have $A(a)$, then must have $D(a)$

So for **inclusions with atomic left-hand side**, can **replace \sqsubseteq -rule** by:

\sqsubseteq^{at} -rule: if $A(a) \in \mathcal{A}$, a is not blocked, $A \sqsubseteq D \in \mathcal{T}$ (with A atomic), and $D(a) \notin \mathcal{A}$, then replace \mathcal{A} with $\mathcal{A} \cup \{D(a)\}$

Good news: we've **lowered the number of disjunctions!**

ABSORPTION (2)

Second observation: can **transform** some inclusions with complex concept on left into **equivalent inclusions with atomic left-hand side**

ABSORPTION (2)

Second observation: can **transform** some inclusions with complex concept on left into **equivalent inclusions with atomic left-hand side**

$$(A \sqcap C) \sqsubseteq D \quad \rightsquigarrow \quad A \sqsubseteq (\neg C \sqcup D)$$

ABSORPTION (2)

Second observation: can **transform** some inclusions with complex concept on left into **equivalent inclusions with atomic left-hand side**

$$(A \sqcap C) \sqsubseteq D \quad \rightsquigarrow \quad A \sqsubseteq (\neg C \sqcup D)$$

Absorption technique:

1. **preprocess the TBox** by replacing inclusions with equivalent inclusions with atomic concept on left, whenever possible
2. when running tableau algorithm
 - use new \sqsubseteq^{at} -rule for inclusions $A \sqsubseteq D$ with A a concept name
 - use regular \sqsubseteq -rule for the other TBox inclusions

EXAMPLE: ABSORPTION

Let's use absorption on the KB $(\mathcal{T}, \{A(a)\})$ with:

$$\{ \quad A \sqsubseteq \exists r.B \quad B \sqsubseteq D \quad \exists r.D \sqsubseteq \neg A \quad \}$$

EXAMPLE: ABSORPTION

Let's use absorption on the KB $(\mathcal{T}, \{A(a)\})$ with:

$$\{ \quad A \sqsubseteq \exists r.B \quad B \sqsubseteq D \quad \exists r.D \sqsubseteq \neg A \quad \}$$

- first two inclusions in \mathcal{T} already have concept name on left
- third inclusion in \mathcal{T} can be equivalently written as $A \sqsubseteq \forall r.\neg D$
- so: only need to use \sqsubseteq^{at} -rule

EXAMPLE: ABSORPTION

Let's use absorption on the KB $(\mathcal{T}, \{A(a)\})$ with:

$$\{ \quad A \sqsubseteq \exists r.B \quad B \sqsubseteq D \quad \exists r.D \sqsubseteq \neg A \quad \}$$

- first two inclusions in \mathcal{T} already have concept name on left
- third inclusion in \mathcal{T} can be equivalently written as $A \sqsubseteq \forall r.\neg D$
- so: only need to use \sqsubseteq^{at} -rule

Result: avoid disjunction, algorithm terminates much faster!

OPTIMIZATIONS FOR CLASSIFICATION

Classification: find all pairs of concept names A, B with $\mathcal{T} \models A \sqsubseteq B$

Naïve approach: test satisfiability of $A \sqcap \neg B$ w.r.t. \mathcal{T} for all pairs A, B

- but \mathcal{T} may contain *hundreds or thousands of concept names*....

OPTIMIZATIONS FOR CLASSIFICATION

Classification: find all pairs of concept names A, B with $\mathcal{T} \models A \sqsubseteq B$

Naïve approach: test satisfiability of $A \sqcap \neg B$ w.r.t. \mathcal{T} for all pairs A, B
· but \mathcal{T} may contain *hundreds or thousands of concept names*....

Each check is costly \Rightarrow want to reduce number of checks

OPTIMIZATIONS FOR CLASSIFICATION

Classification: find all pairs of concept names A, B with $\mathcal{T} \models A \sqsubseteq B$

Naïve approach: test satisfiability of $A \sqcap \neg B$ w.r.t. \mathcal{T} for all pairs A, B

- but \mathcal{T} may contain *hundreds or thousands of concept names*....

Each check is costly \Rightarrow want to reduce number of checks

Some ideas:

- some cases are obvious
 - $A \sqsubseteq A$ and inclusions that are explicitly stated in \mathcal{T}

OPTIMIZATIONS FOR CLASSIFICATION

Classification: find all pairs of concept names A, B with $\mathcal{T} \models A \sqsubseteq B$

Naïve approach: test satisfiability of $A \sqcap \neg B$ w.r.t. \mathcal{T} for all pairs A, B

- but \mathcal{T} may contain *hundreds or thousands of concept names*....

Each check is costly \Rightarrow want to reduce number of checks

Some ideas:

- some cases are obvious
 - $A \sqsubseteq A$ and inclusions that are explicitly stated in \mathcal{T}
- use simple reasoning to obtain new (non-)entailments
 - if know $\mathcal{T} \models A \sqsubseteq B$ and $\mathcal{T} \models B \sqsubseteq D$, then $\mathcal{T} \models A \sqsubseteq D$
 - if know $\mathcal{T} \models A \sqsubseteq B$ and $\mathcal{T} \not\models A \sqsubseteq D$, then $\mathcal{T} \not\models B \sqsubseteq D$