Ontologies and Description Logics

Parcours IA - Représentation des connaissances

Meghyn Bienvenu, CNRS researcher at LaBRI

Reasoning Techniques for Expressive DLs

Tableau method

Tableau method: most popular approach to reasoning in expressive DLs

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

Tableau algorithms are used to **decide satisfiability**.

- can also be used for other reasoning tasks (e.g. instance checking) that can be reduced to satisfiability

Tableau method

Tableau method: most popular approach to reasoning in expressive DLs

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

Tableau algorithms are used to **decide satisfiability**.

- can also be used for other reasoning tasks (e.g. instance checking) that can be reduced 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

$\mathcal{ALC}$-concepts

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

$$\top \quad \bot \quad \neg \quad \sqcup \quad \sqcap \quad \forall R.C \quad \exists R.C$$

$\mathcal{ALC}$-concepts

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

$$\top \quad \bot \quad \neg \quad \sqcup \quad \sqcap \quad \forall R.C \quad \exists R.C$$

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: $\neg(A \sqcap B)$, $\exists R.\neg(\forall S.B)$, $A \sqcup \neg\forall R.B$, $\neg\top$

$\mathcal{ALC}$-concepts

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

$$\top \quad \bot \quad \neg \quad \sqcup \quad \sqcap \quad \forall R.C \quad \exists R.C$$

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 rewriting rules:

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

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

Algorithm for computing NNF

Algorithm NNF

Input: $\mathcal{ALC}$-concept C

If $C = \top$ or $C = \bot$, then output C

If $C = A$ or $C = \neg A$ (with A atomic concept), then output C

If $C = D_1 \sqcap D_2$, then output $\mathtt{NNF}(D_1) \sqcap \mathtt{NNF}(D_2)$

If $C = D_1 \sqcup D_2$, then output $\mathtt{NNF}(D_1) \sqcup \mathtt{NNF}(D_2)$

If $C = \exists R.D$, then output $\exists R.\mathtt{NNF}(D)$

If $C = \forall R.D$, then output $\forall R.\mathtt{NNF}(D)$

If $C = \neg\top$, return $\bot$; if $C = \neg\bot$, then output $\top$

If $C = \neg(D_1 \sqcap D_2)$, then output $\mathtt{NNF}(\neg D_1) \sqcup \mathtt{NNF}(\neg D_2)$

If $C = \neg(D_1 \sqcup D_2)$, then output $\mathtt{NNF}(\neg D_1) \sqcap \mathtt{NNF}(\neg D_2)$

If $C = \neg\exists R.D$, then output $\forall R.\mathtt{NNF}(\neg D)$

If $C = \neg\forall R.D$, then output $\exists R.\mathtt{NNF}(\neg D)$

If $C = \neg(\neg D)$, then output $\mathtt{NNF}(\neg D)$

(we use $\mathtt{NNF}(E)$ to denote output of NNF on input E)

Satisfiability of $\mathcal{ALC}$-concepts via tableau

We start by giving 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 $\mathcal{ALC}$-concepts via tableau

We start by giving 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 rule to some $\mathcal{A} \in S$ (*note: rules are detailed on next slide*)

Satisfiability of $\mathcal{ALC}$-concepts via tableau

We start by giving 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 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 $\mathcal{ALC}$-concepts via tableau

We start by giving 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 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 $\bot(b)$ 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 $\mathcal{ALC}$-concepts via tableau

We start by giving 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 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 $\bot(b)$ 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" if some $\mathcal{A} \in S$ is clash-free, else "no".

Tableau rules for $\mathcal{ALC}$

$\sqcap$-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 $\mathcal{ALC}$

$\sqcap$-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 $\mathcal{ALC}$

$\sqcap$-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 $\mathcal{ALC}$

$\sqcap$-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 there is no b with $\{R(a, b), C(b)\} \subseteq \mathcal{A}$
then **pick a new individual name** d and
replace $\mathcal{A}$ with $\mathcal{A} \cup \{R(a, d), C(d)\}$

Tableau example: only $\sqcap$ and $\sqcup$

Let's use the tableau procedure to test satisfiability of

$$C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$$

Tableau example: only $\sqcap$ and $\sqcup$

Let's use the tableau procedure to test satisfiability of

$$C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$$

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

Tableau example: only $\sqcap$ and $\sqcup$

Let's use the tableau procedure to test satisfiability of

$$C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$$

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

Tableau example: only $\sqcap$ and $\sqcup$

Let's use the tableau procedure to test satisfiability of

$$C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$$

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

Tableau example: only $\sqcap$ and $\sqcup$

Let's use the tableau procedure to test satisfiability of

$$C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$$

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 $\mathcal{A}_2 = \mathcal{A}'_0 \cup \{B(a_0)\}$.

Tableau example: only $\sqcap$ and $\sqcup$

Let's use the tableau procedure to test satisfiability of

$$C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$$

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 $\mathcal{A}_2 = \mathcal{A}'_0 \cup \{B(a_0)\}$.

Tableau example: only $\sqcap$ and $\sqcup$

Let's use the tableau procedure to test satisfiability of

$$C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$$

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

Tableau example: only $\sqcap$ and $\sqcup$

Let's use the tableau procedure to test satisfiability of

$$C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$$

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 $\{A(a_0), \neg A(a_0)\}$!

Tableau example: only $\sqcap$ and $\sqcup$

Let's use the tableau procedure to test satisfiability of

$$C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$$

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

Tableau example: only $\sqcap$ and $\sqcup$

Let's use the tableau procedure to test satisfiability of

$$C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$$

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

Tableau example: only $\sqcap$ and $\sqcup$

Let's use the tableau procedure to test satisfiability of

$$C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$$

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

Tableau example: only $\sqcap$ and $\sqcup$

Let's use the tableau procedure to test satisfiability of

$$C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$$

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

Tableau example: only $\sqcap$ and $\sqcup$

Let's use the tableau procedure to test satisfiability of

$$C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$$

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 $\{B(a_0), \neg B(a_0)\}$!

Tableau example: only $\sqcap$ and $\sqcup$

Let's use the tableau procedure to test satisfiability of

$$C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$$

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.

Tableau example: only $\sqcap$ and $\sqcup$

Let's use the tableau procedure to test satisfiability of

$$C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$$

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

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

Previous example in a picture

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

Previous example in a picture

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

$$((\neg B \sqcup D) \sqcap \neg A)\,(a_0) \qquad \textbf{⊓-rule}$$

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

Previous example in a picture

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

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

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

$A\,(a_0)$ | $B\,(a_0)$ **$\sqcup$-rule**

Previous example in a picture

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

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

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

$\sqcap$**-rule**

$$A\,(a_0)$$

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

$$\neg A\,(a_0)$$

$$B\,(a_0)$$

Previous example in a picture

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

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

Previous example in a picture

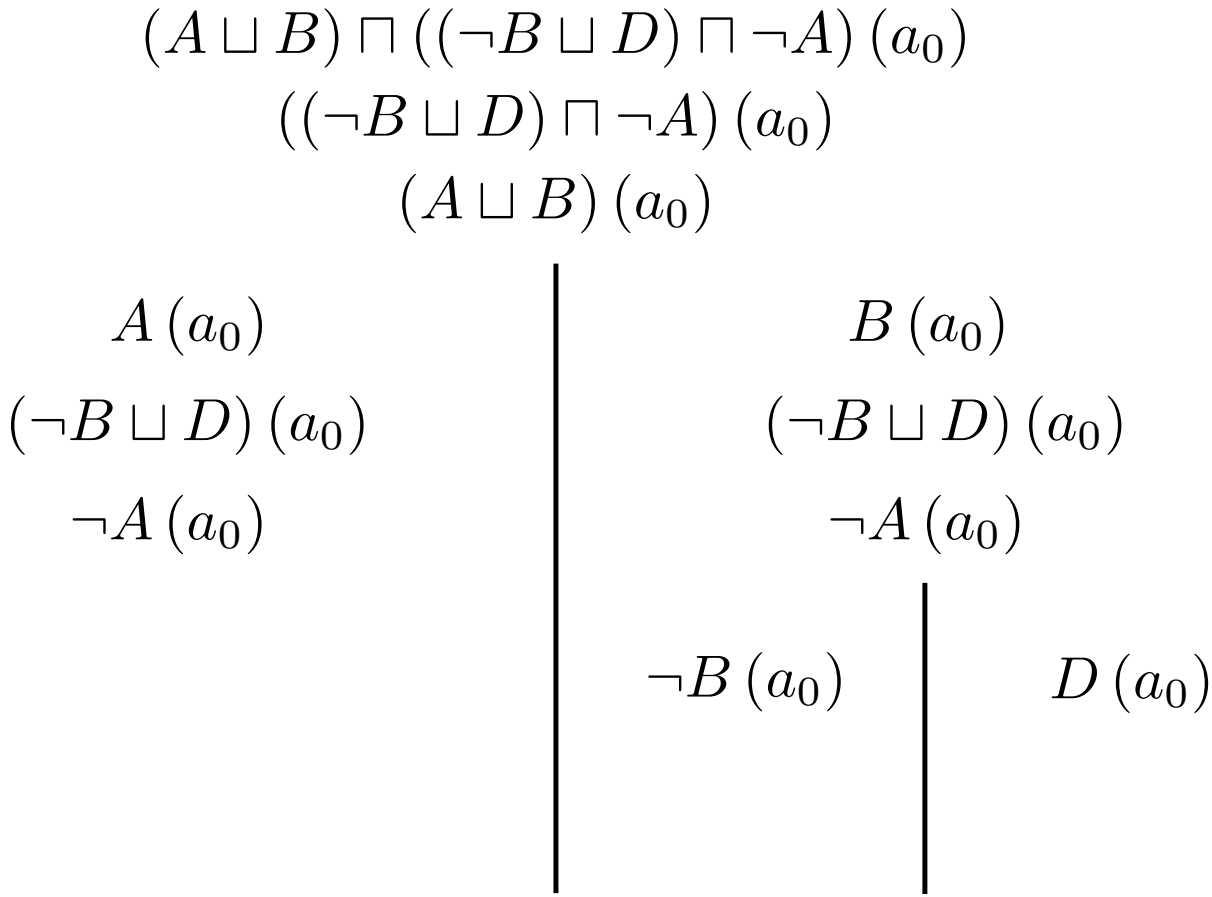
$$(A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)\,(a_0)$$
$$((\neg B \sqcup D) \sqcap \neg A)\,(a_0)$$
$$(A \sqcup B)\,(a_0)$$

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

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

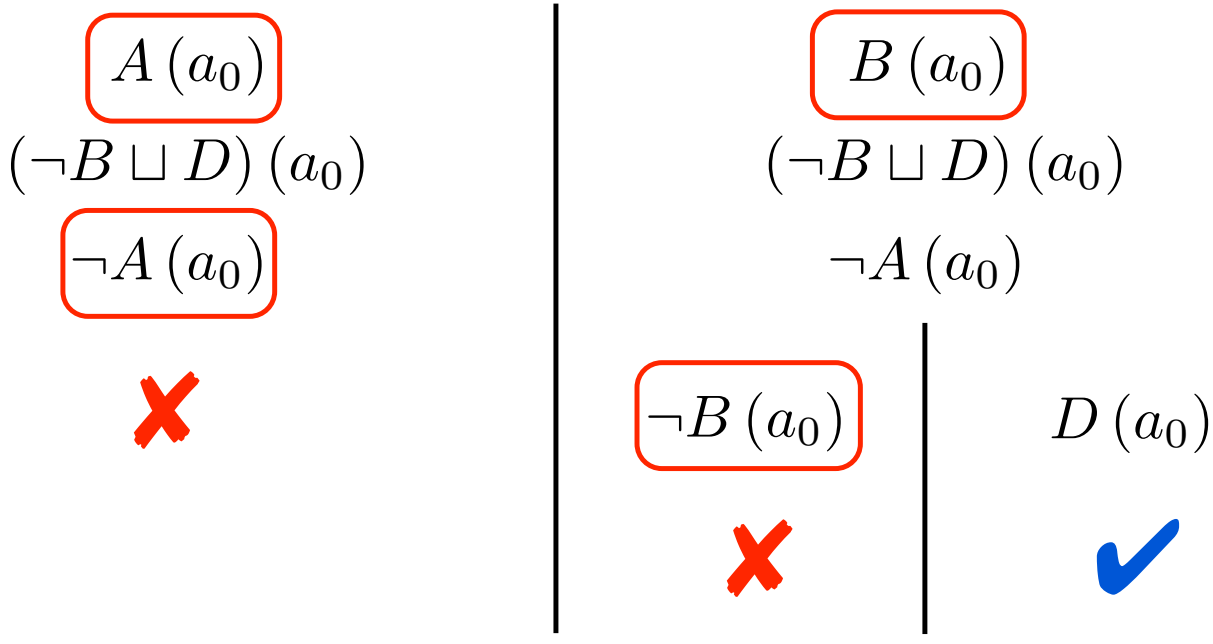
$$\neg B\,(a_0) \qquad D\,(a_0) \quad \textbf{⊔-rule}$$

Previous example in a picture

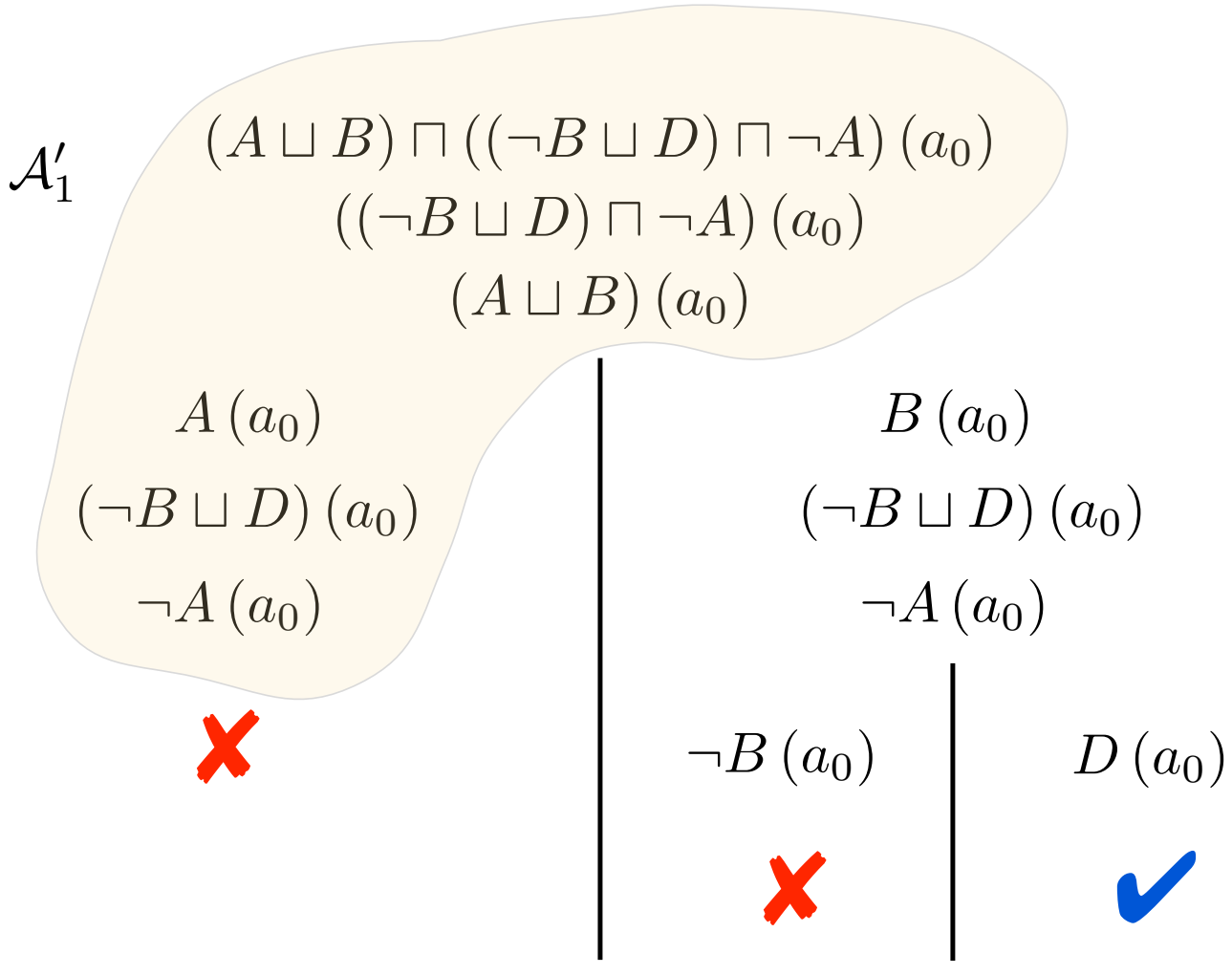


Previous example in a picture

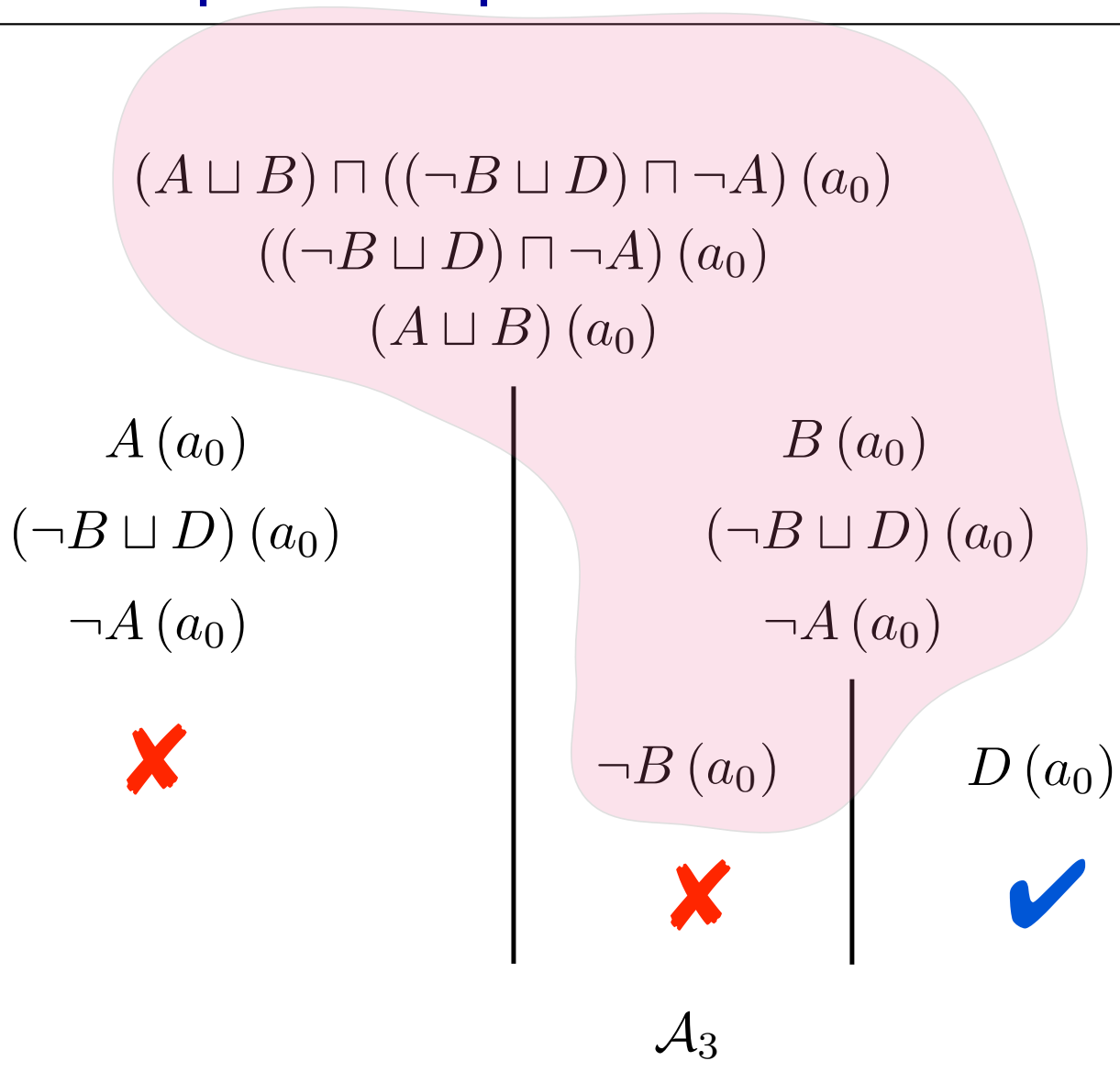
$$(A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)\,(a_0)$$
$$((\neg B \sqcup D) \sqcap \neg A)\,(a_0)$$
$$(A \sqcup B)\,(a_0)$$



Previous example in a picture



Previous example in a picture



Previous example in a picture

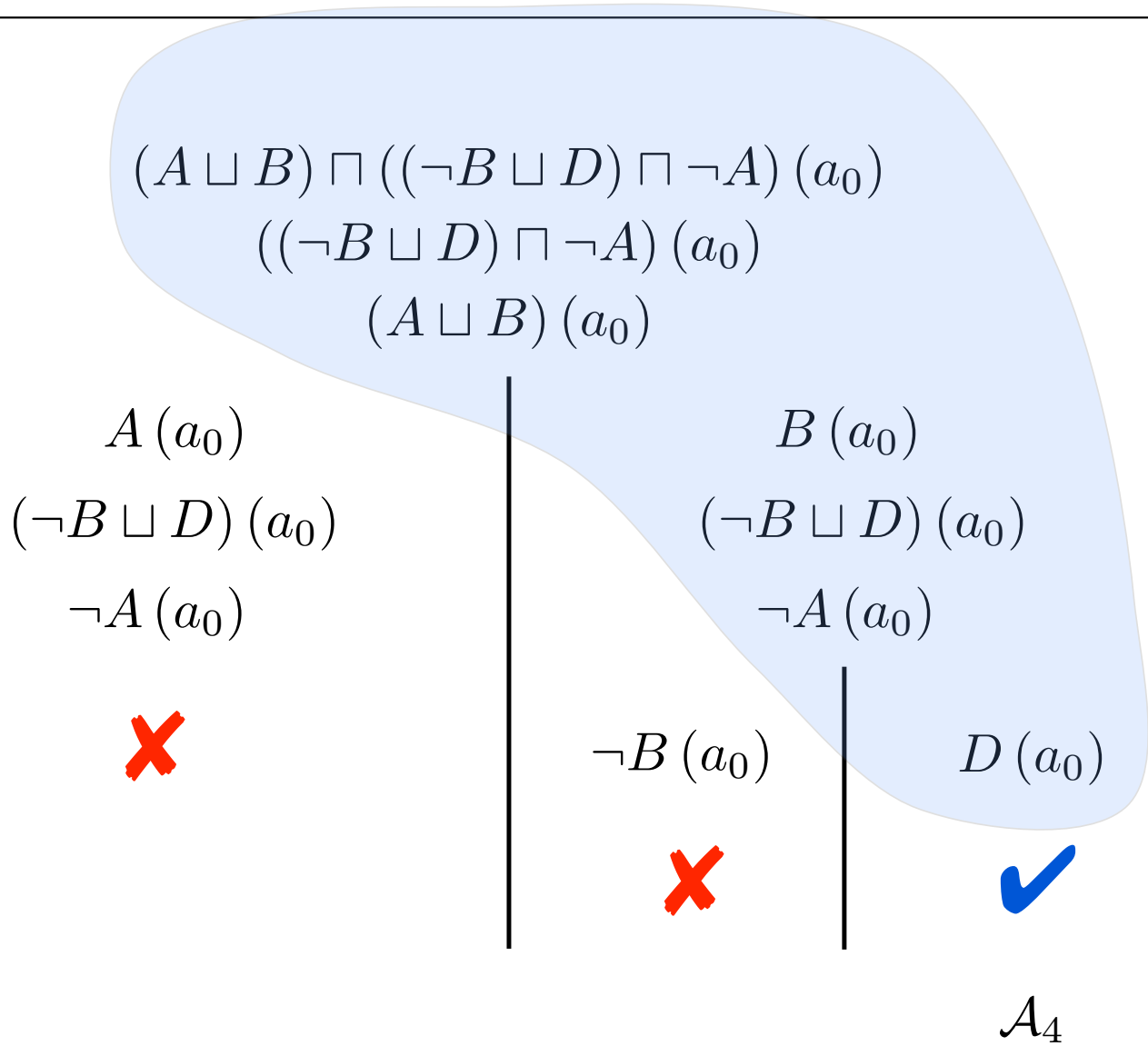


Tableau example: only $\sqcap$ and $\sqcup$

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

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

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 $C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$

Tableau example: $\forall$ and $\exists$

Let's use the tableau procedure to test satisfiability of

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

Tableau 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 $\mathcal{A}_0 = \{ (\exists R.A \sqcap \forall R.\neg A)(a_0) \}$.

Tableau 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 $\mathcal{A}'_0 = \mathcal{A}_0 \cup \{(\exists R.A)(a_0), (\forall R.\neg A)(a_0)\}$.

Tableau 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 $\mathcal{A}'_0 = \mathcal{A}_0 \cup \{(\exists R.A)(a_0), (\forall R.\neg A)(a_0)\}$.

Tableau 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 $\mathcal{A}_0'' = \mathcal{A}_0' \cup \{R(a_0, a_1), A(a_1)\}$.

Tableau 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 $\mathcal{A}_0'' = \mathcal{A}_0' \cup \{R(a_0, a_1), A(a_1)\}$.

Tableau 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 $\mathcal{A}'''_0 = \mathcal{A}''_0 \cup \{\neg A(a_1)\}$.

Tableau 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 $\{A(a_1), \neg A(a_1)\}$!

Tableau 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 a picture

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

Previous example in a picture

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

$$(\,\exists R.A\,)\,(a_0) \qquad \textbf{⊓-rule}$$

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

Previous example in a picture

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

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

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

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

$$A(a_1)$$

Previous example in a picture

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

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

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

$$R(a_0, a_1)$$

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

$$\neg A(a_1)$$

Previous example in a picture

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

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

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

$$R(a_0, a_1)$$

$$A(a_1)$$

$$\neg A(a_1)$$

Tableau example: $\forall$ and $\exists$

Suppose that we consider a slightly different concept

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

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

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

Tableau example: $\forall$ and $\exists$

Suppose that we consider a slightly different concept

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

Now the tableau 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 $C_0^{\mathcal{I}} = \{a_0\}$.

Exercise: Concept satisfiability via tableau

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

1. $C_1 = (\exists R.(A \sqcap B)) \sqcap (\forall R.(\neg A \sqcup D))$
2. $C_2 = (\exists R.\exists S.A) \sqcap (\forall R.\forall S.\neg A)$
3. $C_3 = (\exists R.B) \sqcap (\forall R.(\forall R.A)) \sqcap (\forall R.\neg A)$

If a concept is found to be satisfiable, use the result to construct an interpretation in which the concept is non-empty.

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".

Preliminary definitions

Subconcepts of a concept:

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

Preliminary definitions

Subconcepts of a concept:

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

Role depth of a concept:

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

Preliminary definitions

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

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

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

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

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 $D \in \mathsf{sub}(C_0)$

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 $D \in \mathsf{sub}(C_0)$
   - $\mathcal{A}$ 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 $D \in \mathsf{sub}(C_0)$
   - $\mathcal{A}$ 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 $D \in \mathsf{sub}(C_0)$
   - $\mathcal{A}$ 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 $\mathsf{depth}(D) \leq \mathsf{depth}(C_0) - k$

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 $D \in \mathsf{sub}(C_0)$
   - $\mathcal{A}$ 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 $\mathsf{depth}(D) \leq \mathsf{depth}(C_0) - k$
   - each individual in $\mathcal{A}$ is at distance $\leq \mathsf{depth}(C_0)$ 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 $D \in \mathsf{sub}(C_0)$
   - $\mathcal{A}$ 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 $\mathsf{depth}(D) \leq \mathsf{depth}(C_0) - k$
   - each individual in $\mathcal{A}$ is at distance $\leq \mathsf{depth}(C_0)$ 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 $D \in \mathsf{sub}(C_0)$
   - $\mathcal{A}$ 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 $\mathsf{depth}(D) \leq \mathsf{depth}(C_0) - k$
   - each individual in $\mathcal{A}$ is at distance $\leq \mathsf{depth}(C_0)$ 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
$\Rightarrow$ **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 that:

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

Soundness of CSat (2)

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

Soundness of CSat (2)

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

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 $b \in \top^{\mathcal{I}} = \Delta^{\mathcal{I}}$. We cannot have $D = \bot$ since clash-free.

Induction hypothesis (IH): suppose statement 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 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 IH, we 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 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 $\mathcal{A}_1$ 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 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 $\mathcal{A}_1$ 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.

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

$$\bigsqcap_{0 \leq i < n} \underbrace{\forall R. \ldots \forall R.}_{i \text{ times}} (\exists R.B \sqcap \exists R.\neg B)$$

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

$$\bigsqcap_{0 \leq i < n} \underbrace{\forall R. \ldots \forall R.}_{i \text{ times}} (\exists R.B \sqcap \exists R.\neg B)$$

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

- instead of a set of ABoxes, keep only one 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

Complexity of $\mathcal{ALC}$ concept satisfability

Hierarchy of complexity classes

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

(it is believed that all inclusions are strict)

Complexity of $\mathcal{ALC}$ concept satisfability

Hierarchy of complexity classes

PTIME $\subseteq$ NP $\subseteq$ ... $\subseteq$ **PSPACE** $\subseteq$ EXPTIME $\subseteq$ NEXPTIME $\subseteq$ ... $\subseteq$ EXPSPACE ...

(it is believed that all inclusions are strict)

PSPACE = class of decision problems solvable using polynomial space

PSPACE-complete problems = hardest problems in PSPACE

Complexity of $\mathcal{ALC}$ concept satisfability

Hierarchy of complexity classes

PTIME $\subseteq$ NP $\subseteq$ ... $\subseteq$ **PSPACE** $\subseteq$ EXPTIME $\subseteq$ NEXPTIME $\subseteq$ ... $\subseteq$ EXPSPACE ...

(it is believed that all inclusions are strict)

PSPACE = class of decision problems solvable using polynomial space

PSPACE-complete problems = hardest problems in PSPACE

Theorem: $\mathcal{ALC}$ 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)

Extending the tableau algorithm to KBs

Extension to KB satisfiability

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

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 satisfability.

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 we have $C \sqsubseteq D$, then every element must satisfy either $\neg C$ or D

Extension to KB satisfiability

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

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 we have $C \sqsubseteq D$, then every element must satisfy either $\neg C$ or D

Concretely, we might try adding the following rule:

TBox rule: if a is in $\mathcal{A}$, $C \sqsubseteq D \in \mathcal{T}$, & $(\mathtt{NNF}(\neg C) \sqcup \mathtt{NNF}(D))\,(a) \notin \mathcal{A}$

then replace $\mathcal{A}$ with $\mathcal{A} \cup \{(\mathtt{NNF}(\neg C) \sqcup \mathtt{NNF}(D))(a)\}$

Examples: KB satisfiability

Let's try the modified procedure on the KB $(\mathcal{T}, \{A(a)\})$ where

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

Examples: KB satisfiability

Let's try the modified procedure on the KB $(\mathcal{T}, \{A(a)\})$ where

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

Now try it on the KB $(\{F \sqsubseteq \exists S.F\}, \{F(a)\})$.

Examples: KB satisfiability

Let's try the modified procedure on the KB $(\mathcal{T}, \{A(a)\})$ where

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

Now try it on the KB $(\{F \sqsubseteq \exists S.F\}, \{F(a)\})$.

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

Blocking

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

Blocking

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

Formally, give individuals a, b in $\mathcal{A}$, 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.

Blocking

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

Formally, give individuals a, b in $\mathcal{A}$, 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 $\mathcal{A} \cup \{R(a,d), C(d)\}$

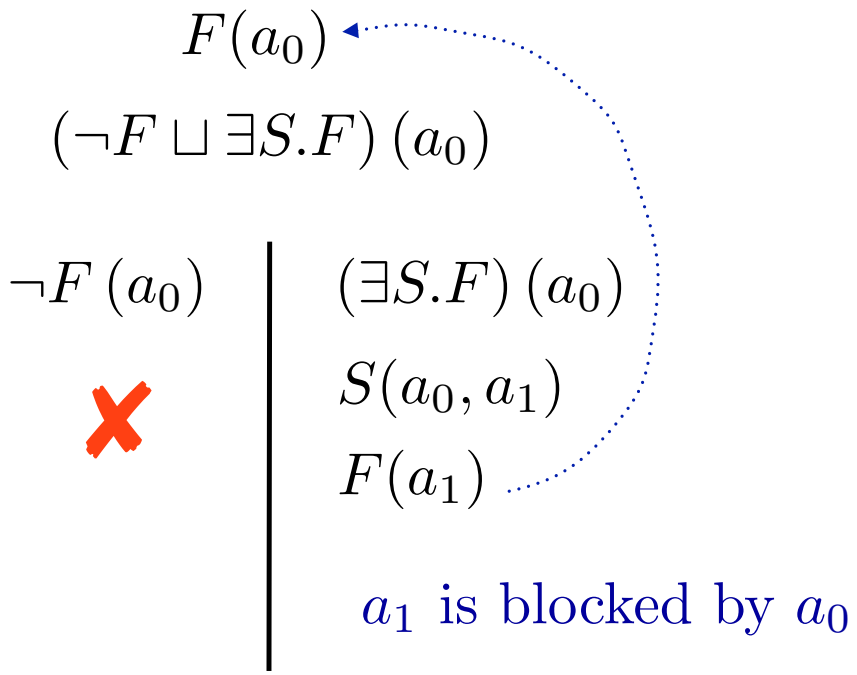
$\sqsubseteq$-rule: if a appears in $\mathcal{A}$ and a **is not blocked**, $C \sqsubseteq D \in \mathcal{T}$, and $(\mathtt{NNF}(\neg C) \sqcup \mathtt{NNF}(D))(a) \notin \mathcal{A}$, then replace $\mathcal{A}$ with $\mathcal{A} \cup \{(\mathtt{NNF}(\neg C) \sqcup \mathtt{NNF}(D))(a)\}$

Example: blocking

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

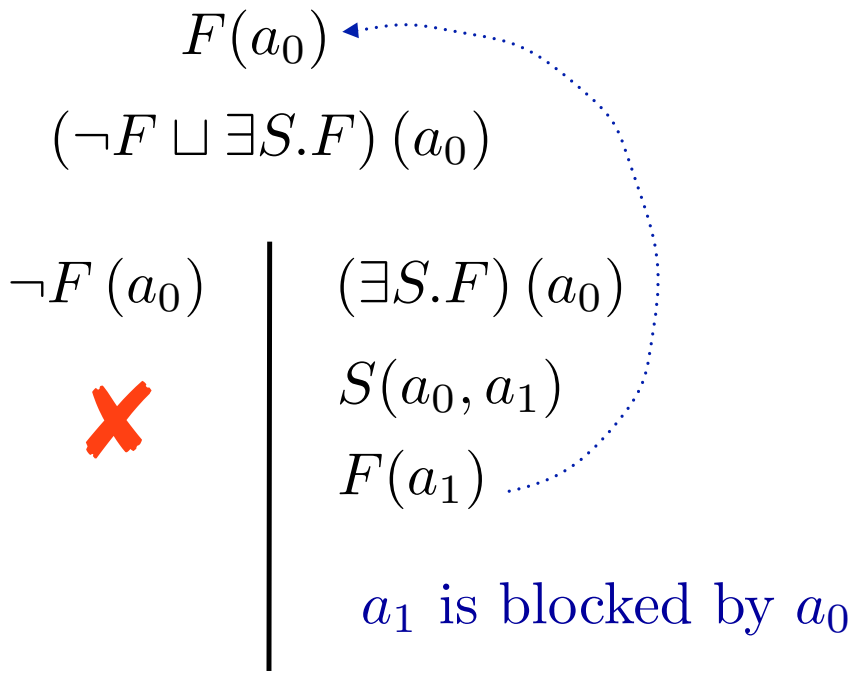
Example: blocking

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 following 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 the KB $(\mathcal{T}, \{(A \sqcap \neg D)(a_0)\})$.

Another blocking example

Consider the following 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 the KB $(\mathcal{T}, \{(A \sqcap \neg D)(a_0)\})$.

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

Another blocking example

Consider the following 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 the KB $(\mathcal{T}, \{(A \sqcap \neg D)(a_0)\})$.

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

Observation: an individual can be blocked, then later become unblocked

Properties of the tableau algorithm

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

Properties of the tableau algorithm

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

Properties of the tableau algorithm

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 the tableau algorithm

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 KB satisfiability

The KBSat algorithm generates ABoxes of at most exponential size

But even if with the tricks from earlier, need exponential space.

- *single branch may be exponentially long*

Complexity of KB satisfiability

The KBSat algorithm generates ABoxes of at most exponential size

But even if with the tricks from earlier, need exponential space.

- *single branch may be exponentially long*

This is most likely not optimal, as we can show the following:

Theorem: $\mathcal{ALC}$ KB satisfiability is EXPTIME-complete.

Complexity of KB satisfiability

The KBSat algorithm generates ABoxes of at most exponential size

But even if with the tricks from earlier, need exponential space.

- *single branch may be exponentially long*

This is most likely not optimal, as we can show the following:

Theorem: $\mathcal{ALC}$ KB satisfiability is EXPTIME-complete.

This result means **no polynomial-time algorithm can every be found**.

Optimizations

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

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!**

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

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

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

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)

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)

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

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:

$$(\mathtt{NNF}(\neg C_1) \sqcup \mathtt{NNF}(D_1))(a), \ldots, (\mathtt{NNF}(\neg C_n) \sqcup \mathtt{NNF}(D_n))(a)$$

Absorption (1)

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

$$(\mathtt{NNF}(\neg C_1) \sqcup \mathtt{NNF}(D_1))(a), \ldots, (\mathtt{NNF}(\neg C_n) \sqcup \mathtt{NNF}(D_n))(a)$$

Observation: if have inclusion $A \sqsubseteq D$ with A atomic

- 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:

$$(\mathtt{NNF}(\neg C_1) \sqcup \mathtt{NNF}(D_1))(a), \ldots, (\mathtt{NNF}(\neg C_n) \sqcup \mathtt{NNF}(D_n))(a)$$

Observation: if have inclusion $A \sqsubseteq D$ with A atomic

- 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:

$$(\mathtt{NNF}(\neg C_1) \sqcup \mathtt{NNF}(D_1))(a), \ldots, (\mathtt{NNF}(\neg C_n) \sqcup \mathtt{NNF}(D_n))(a)$$

Observation: if have inclusion $A \sqsubseteq D$ with A atomic

- 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

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

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 atomic
   - use regular $\sqsubseteq$-rule for the other TBox inclusions

Example: Absorption

Let's use absorption on the KB $(\mathcal{T}, \{A(a)\})$ from earlier 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 $(\mathcal{T}, \{A(a)\})$ from earlier 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 atomic concept 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 $(\mathcal{T}, \{A(a)\})$ from earlier 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 atomic concept 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: completely avoid disjunction, algorithm terminates much faster!

Optimizations for classification

Classification: find all pairs of atomic concepts 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 atomic concepts....*

Optimizations for classification

Classification: find all pairs of atomic concepts 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 atomic concepts....*

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

Optimizations for classification

Classification: find all pairs of atomic concepts 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 atomic concepts....*

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

Some ideas:

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

Optimizations for classification

Classification: find all pairs of atomic concepts 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 atomic concepts....*

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

Some ideas:

- some subsumptions are obvious
  - $A \sqsubseteq A$ and subsumptions that are explicitly stated in $\mathcal{T}$
- can use simple reasoning to obtain new (non-)subsumptions
  - 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 $\mathcal{T} \not\models B \sqsubseteq D$

Exercise: KB satisfiability via tableau

Consider the following TBox:

$$\mathcal{T} = \{A \sqsubseteq \forall R.B, B \sqsubseteq \neg F, E \sqsubseteq G, A \sqsubseteq D \sqcup E, D \sqsubseteq \exists R.F, \exists R.\neg B \sqsubseteq G\}$$

Use the KBSat algorithm to decide whether:

1. $\mathcal{T} \models A \sqsubseteq E$
2. $\mathcal{T} \models E \sqsubseteq G$
3. $\mathcal{T} \models E \sqsubseteq F$
4. $\mathcal{T} \models A \sqsubseteq G$
5. $\mathcal{T} \models D \sqsubseteq G$
6. $\mathcal{T} \models G \sqsubseteq F$

You are encouraged to use the optimizations introduced in the course.