Semantic Web

— to Artificial Intelligence

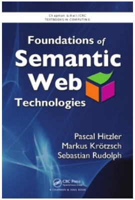
Yue Ma

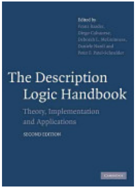
Laboratoire de Recherche en Informatique (LRI)
Université Paris Sud
ma@lri.fr

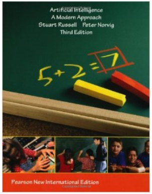
General Information

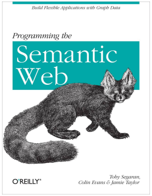
- This lecture is inspired by the following courses :
  - Knowledge Representation for the Semantic Web
    Pascal Hitzler, Wright State University, US
  - Foundations of Semantic Web Technologies
    Sebastian Rudolph, TU Dresden, Germany
  - Knowledge Graph Analysis
    Jens Lehmann, University of Bonn, Germany
  - Introduction à l'intelligence artificielle
    Laurent Simon, Polytech Paris Sud

- Reference books :









General Information

- Course materials : `www.lri.fr/~ma/M2DK`
- Organizational matters :
  - Initial plans :
    - Part 1 : Semantic Web Standards (RDF, RDFS, OWL, SPARQL)
    - Part 2 : Semantics and Reasoning for Semantic Web
    - Projects : Querying Semantic Data, Ontology Reasoning
  - Grading : average(Projects, Exam)

Reasoning Techniques for Expressive DLs

Tableau-based algorithms (1990–1995)

- first systems employing such algorithms : Kris and Crack
- Highly optimized systems : FaCT, Race, DLP

Alternative approaches (2000–2005)

- resolution-based approaches (use an optimized translation of DLs into first-order predicate logic and then apply appropriate first-order resolution provers)
- automata-based approaches (often more convenient for showing ExpTime complexity upper-bounds than tableau-based approaches)

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 $C \equiv D$.

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

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

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

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

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

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

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

Previous example in a picture

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

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

$\sqcap$-rule

Previous example in a picture

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

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

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

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

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

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

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

Previous example in a picture

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

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

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

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

Under the right branch:

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

Previous example in a picture

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

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

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

$$\boxed{A\,(a_0)}$$

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

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

✗

$$\boxed{B\,(a_0)}$$

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

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

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

✗

$$D\,(a_0)$$

✔

Previous example in a picture

$\mathcal{A}'_1$

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

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

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

$$A\,(a_0)$$

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

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

✗

$$B\,(a_0)$$

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

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

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

✗

$$D\,(a_0)$$

✔

Previous example in a picture

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

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

$\mathcal{A}_3$

Previous example in a picture

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

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

$\neg B\,(a_0)$	$D\,(a_0)$
✗	✓
	$\mathcal{A}_4$

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

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

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

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

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

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

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

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

$$R\,(a_0, a_1)$$

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

$$\neg A\,(a_1)$$

Previous example in a picture

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

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

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

$$R\,(a_0, a_1)$$

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

✘

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 $C_0^{\mathcal{I}} = \{a_0\}$.

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

The size of a concept C, denoted $|C|$, is defined as the total number of occurrences of atomic concepts, atomic roles, and constructors in C.

For example : $|A \sqcap B| = 3, |\neg(A \sqcap B)| = 4, |\neg\neg A| = 3, |\exists R.A| = 3$

Preliminary definitions

Subconcepts of a concept:

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

Preliminary definitions

Subconcepts of a concept:

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

Role depth of a concept:

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

Preliminary definitions

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

?

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

1 1 1

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

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

Termination of CSat

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

Termination of $\mathtt{CSat}$

Suppose we run $\mathtt{CSat}$ 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

So, there can only be a finite number of rule applications per individual

Next to show that the number of newly introduced individuals in a chain of rule

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 $D \in \text{sub}(C_0)$
   - $\mathcal{A}$ 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 $\leq$ depth(C_0) from a_0

Termination of $\mathtt{CSat}$

Suppose we run $\mathtt{CSat}$ 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 $\mathtt{CSat}$

Suppose we run $\mathtt{CSat}$ 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 $\mathtt{CSat}$

Suppose we run $\mathtt{CSat}$ 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.

The tableau procedure **only adds assertions** to ABoxes
$\Rightarrow$ **eventually all ABoxes will contain a clash or will be complete**

Soundness and Completeness :

$\boxed{\text{CSat returns “yes” on input } C_0. \iff C_0 \text{ is satisfiable.}}$

CSat returns “yes” on input C_0. $\Longrightarrow$ C_0 is satisfiable. (Soundness)
(I.e., C_0 is unsatisfiable. $\Longrightarrow$ CSat returns “no” on input C_0. That is, the positive answers of the algorithm are correct : if the algorithm says “consistent”, then the input concept is indeed satisfiable.)

CSat returns “yes” on input C_0. $\Longleftarrow$ C_0 is satisfiable. (Completeness)
(I.e., C_0 is unsatisfiable. $\Longleftarrow$ CSat returns “no” on input C_0. That is, the negative answers of the algorithm is correct : if the algorithm says “no”, the input concept is unsatisfiable.)

Soundness of $\texttt{CSat}$ (1)

Suppose that $\texttt{CSat}$ returns “yes” on input C_0.

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

Soundness of $\texttt{CSat}$ (1)

Suppose that $\texttt{CSat}$ 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 $\mathtt{CSat}$ (1)

Suppose that $\mathtt{CSat}$ 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 $\mathtt{CSat}$ (2)

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

Soundness of $\texttt{CSat}$ (2)

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

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

Soundness of $\mathtt{CSat}$ (2)

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

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

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

Soundness of CSat (2)

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

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

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 $b \in (E \sqcap F)^{\mathcal{I}}$

Soundness of CSat (2)

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

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

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 $b \in (\exists R.E)^{\mathcal{I}}$

Soundness of CSat (2)

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

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

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 $\mathtt{CSat}$

Bad news: our algorithm may require exponential time and space...

To see why, consider what happens if we run $\mathtt{CSat}$ on the concept

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

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

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

(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)

Homework

Exercise 1: Concept satisfiability via tableau

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

1. $\exists R.(A \sqcap B) \sqcap \forall R.(\neg A \sqcup D)$
2. $(\exists R.\exists S.A) \sqcap (\forall R.\forall S.\neg A)$
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.

Exercise 2: Negation normal form

Consider the following recursive algorithm for putting a concept in NNF.

Algorithm NNF

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

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

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

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

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

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

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

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

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

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

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

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

Exercise 2: Negation normal form

Show that the algorithm NNF has the following properties:

1. NNF terminates on every input.
2. NNF(C) is in negation normal form.
3. NNF(C) is equivalent to C.

Hint: you may want to use the size of C, which is denoted $|C|$ and defined as the total number of occurrences of atomic concepts, atomic roles, and constructors in C.

Example: $|A \sqcap B| = 3$, $|\neg(A \sqcap B)| = 4$, $|\neg\neg A| = 3$, $|\exists R.A| = 3$

Exercise 3: Adding acyclic terminologies (1)

In this exercise, we see how to test concept satisfiability w.r.t. a special kind of TBox called an acyclic terminology.

A **terminology** is a set of concept definitions

$$A \equiv C$$

where A is an atomic concept and C a (possibly complex) concept, and *no two concept definitions have the same atomic concept on the left*.

A concept definition $A \equiv C$ corresponds to the pair of inclusions $A \sqsubseteq C$ and $C \sqsubseteq A$, and so it is satisfied by $\mathcal{I}$ if $A^{\mathcal{I}} = C^{\mathcal{I}}$.

Exercise 3: Adding acyclic terminologies (2)

Given a terminology $\mathcal{T}$ and concept C, the **unfolding of C w.r.t. $\mathcal{T}$** is obtained by applying the following operation as long as possible:

- replace any atomic concept A such that $A \equiv D \in \mathcal{T}$ by D

Example: the unfolding of $A \sqcap \exists R.B$ w.r.t. the terminology

$$\{A \equiv \forall S.\forall R.B \quad B \equiv E \sqcap F \quad F \equiv \neg G\}$$

is the concept $\forall S.\forall R.(E \sqcap \neg G) \sqcap \exists R.(E \sqcap \neg G)$.

Note: for some terminologies, the process may never stop!

- take $C = A$ and $\mathcal{T} = \{A \equiv \exists R.A\}$

A terminology is called **acyclic** if the **unfolding procedure always halts**.

Solutions to the exercises 2 and 3

Exercise 2: Negation normal form

Consider the following recursive algorithm for putting a concept in NNF.

Algorithm NNF

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

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

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

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

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

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

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

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

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

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

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

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

Exercise 2: Negation normal form

Show that the algorithm `NNF` has the following properties:

1. `NNF` terminates on every input.
2. `NNF`(C) is in negation normal form.
3. `NNF`(C) is equivalent to C.

Hint: you may want to use the size of C, which is denoted $|C|$ and defined as the total number of occurrences of atomic concepts, atomic roles, and constructors in C.

Example: $|A \sqcap B| = 3$, $|\neg(A \sqcap B)| = 4$, $|\neg\neg A| = 3$, $|\exists R.A| = 3$

Solution to exercise 2: Termination

The proof is by induction on the size of the input concept.

Base case: input concept C with $|C| \leq 2$

Then we must have $C = A$ or $C = \neg A$, so `NNF` terminates on input C

Solution to exercise 2: Termination

The proof is by induction on the size of the input concept.

Base case: input concept C with $|C| \leq 2$

Then we must have $C = A$ or $C = \neg A$, so `NNF` terminates on input C

Induction hypothesis (IH): `NNF` terminates on all inputs C with $|C| \leq k$

Solution to exercise 2: Termination

The proof is by induction on the size of the input concept.

Base case: input concept C with $|C| \leq 2$

Then we must have $C = A$ or $C = \neg A$, so NNF terminates on input C

Induction hypothesis (IH): NNF terminates on all inputs C with $|C| \leq k$

Induction step: show NNF terminates on input C with $|C| = k + 1$

Let C be a concept with $|C| = k + 1$. There are several cases to consider:

- $C = D_1 \sqcap D_2$: call NNF on D_1 and D_2, both calls terminate since $|D_1| < |C|$ and $|D_2| < |C|$, hence $|D_1| \leq k$ and $|D_2| \leq k$
- $C = \neg\exists R.D$: call NNF on $\neg D$, and the call terminates since $|\neg D| \leq k$
- $C = \ldots$

Solution to exercise 2: NNF

The proof is by induction on the size of the input concept C.

Base case: input concept C with $|C| \leq 2$

We must have $C = A$ or $C = \neg A$. In both cases, `NNF` will output C, which is in NNF.

Solution to exercise 2: NNF

The proof is by induction on the size of the input concept C.

Base case: input concept C with $|C| \leq 2$

We must have $C = A$ or $C = \neg A$. In both cases, `NNF` will output C, which is in NNF.

Induction hypothesis (IH): $\texttt{NNF}(C)$ is in NNF, whenever $|C| \leq k$

Solution to exercise 2: NNF

The proof is by induction on the size of the input concept C.

Base case: input concept C with $|C| \leq 2$

We must have $C = A$ or $C = \neg A$. In both cases, $\mathtt{NNF}$ will output C, which is in NNF.

Induction hypothesis (IH): $\mathtt{NNF}(C)$ is in NNF, whenever $|C| \leq k$

Induction step: show $\mathtt{NNF}(C)$ is in NNF when $|C| = k + 1$

Let C be a concept with $|C| = k + 1$. There are several cases to consider:

- $C = D_1 \sqcap D_2$: by the IH, both $\mathtt{NNF}(D_1)$ and $\mathtt{NNF}(D_2)$ are in NNF, hence the output $\mathtt{NNF}(D_1) \sqcap \mathtt{NNF}(D_2)$ must be too.
- $C = \neg \exists R.D$: by the IH, $\mathtt{NNF}(\neg D)$ is in NNF. Thus, $\forall R.\mathtt{NNF}(\neg D)$ must also be in NNF.
- $C = \ldots$

Solution to exercise 2: Equivalence

Base case: input concept C with $|C| \leq 2$

Easy since `NNF` will output C, and C is clearly equivalent to itself.

Solution to exercise 2: Equivalence

Base case: input concept C with $|C| \leq 2$

Easy since NNF will output C, and C is clearly equivalent to itself.

Induction hypothesis (IH): $\mathtt{NNF}(C)$ is equivalent to C when $|C| \leq k$

Induction step: show $\mathtt{NNF}(C)$ is equivalent to C when $|C| = k + 1$

Let C be a concept with $|C| = k + 1$. There are several cases to consider:

- $C = \neg\exists R.D$: by the IH, $\mathtt{NNF}(\neg D)$ is equivalent to $\neg D$. Let $\mathcal{I}$ be an interpretation. We have:

$$\begin{aligned}
C^{\mathcal{I}} &= (\neg\exists R.D)^{\mathcal{I}} \\
&= \{d \in \Delta^{\mathcal{I}} \mid \text{ there is no } e \text{ with } (d, e) \in R^{\mathcal{I}} \text{ and } e \in D^{\mathcal{I}}\} \\
&= \{d \in \Delta^{\mathcal{I}} \mid \text{ for every } e \text{ with } (d, e) \in R^{\mathcal{I}} \text{ we have } e \in (\neg D)^{\mathcal{I}}\} \\
&= (\forall R.\neg D)^{\mathcal{I}} \\
&= (\forall R.\mathtt{NNF}(\neg D))^{\mathcal{I}} \\
&= (\mathtt{NNF}(C))^{\mathcal{I}}
\end{aligned}$$

- $C = \ldots$

Exercise 3: Adding acyclic terminologies (1)

In this exercise, we see how to test concept satisfiability w.r.t. a special kind of TBox called an acyclic terminology.

A **terminology** is a set of concept definitions

$$A \equiv C$$

where A is an atomic concept and C a (possibly complex) concept, and *no two concept definitions have the same atomic concept on the left*.

A concept definition $A \equiv C$ corresponds to the pair of inclusions $A \sqsubseteq C$ and $C \sqsubseteq A$, and so it is satisfied by $\mathcal{I}$ if $A^{\mathcal{I}} = C^{\mathcal{I}}$.

Exercise 3: Adding acyclic terminologies (2)

Given a terminology $\mathcal{T}$ and concept C, the **unfolding of C w.r.t. $\mathcal{T}$** is obtained by applying the following operation as long as possible:

- replace any atomic concept A such that $A \equiv D \in \mathcal{T}$ by D

Example: the unfolding of $A \sqcap \exists R.B$ w.r.t. the terminology

$$\{A \equiv \forall S.\forall R.B \quad B \equiv E \sqcap F \quad F \equiv \neg G\}$$

is the concept $\forall S.\forall R.(E \sqcap \neg G) \sqcap \exists R.(E \sqcap \neg G)$.

Note: for some terminologies, the process may never stop!

- take $C = A$ and $\mathcal{T} = \{A \equiv \exists R.A\}$

A terminology is called **acyclic** if the **unfolding procedure always halts**.

Exercise 3: Adding acyclic terminologies (3)

We can test satisfiability of C w.r.t. acyclic terminology $\mathcal{T}$ as follows:

Step 1 Compute the unfolding U of C w.r.t. $\mathcal{T}$.

Step 2 Put U into negation normal form.

Step 3 Run `CSat` on U, and output the same answer as `CSat`.

First task: Use the above procedure to test satisfiability of the concepts:

1. $B \sqcap \forall R.\neg G$
2. $A \sqcap \forall R.\exists R.G$

with respect to the following acyclic terminology:

$$\{A \equiv \exists R.B \sqcap \exists S.\neg B \quad B \equiv \exists R.D \sqcap \exists R.E \quad G \equiv D \sqcap E\}$$

Second task: explain why the above procedure is a decision procedure for $\mathcal{ALC}$ concept satisfiability w.r.t. acyclic terminologies.

(note: I am not asking for a formal proof, just a clear explanation)

Solution to exercise 3: first task (1)

For the first concept, we first unfold w.r.t. $\mathcal{T}$ to get:

$$(\exists R.D \sqcap \exists R.E) \sqcap \forall R.\neg(D \sqcap E)$$

and then put the latter concept into NNF:

$$(\exists R.D \sqcap \exists R.E) \sqcap \forall R.(\neg D \sqcup \neg E)$$

The tableau algorithm returns "yes" on the preceding concept, so the **first concept is satisfiable w.r.t.** $\mathcal{T}$.

Solution to exercise 3: first task (2)

For the second concept, we first unfold w.r.t. $\mathcal{T}$ to get:

$$\exists R.(\exists R.D \sqcap \exists R.E) \sqcap \exists S.(\neg(\exists R.D \sqcap \exists R.E)) \sqcap \forall R.\exists R.(D \sqcap E)$$

and then put the latter concept into NNF:

$$\exists R.(\exists R.D \sqcap \exists R.E) \sqcap \exists S.(\forall R.\neg D \sqcup \forall R.\neg E) \sqcap \forall R.\exists R.(D \sqcap E)$$

The tableau algorithm returns "yes" on the preceding concept, so the **second concept is also satisfiable w.r.t.** $\mathcal{T}$.

Solution to exercise 3: second task

Suppose we run the procedure on concept C and acyclic terminology $\mathcal{T}$.

Termination: Step 1 terminates because $\mathcal{T}$ is acyclic. Steps 2 and 3 terminate because we know that $\mathtt{NNF}$ and $\mathtt{CSat}$ both terminate.

Solution to exercise 3: second task

Suppose we run the procedure on concept C and acyclic terminology $\mathcal{T}$.

Termination: Step 1 terminates because $\mathcal{T}$ is acyclic. Steps 2 and 3 terminate because we know that $\mathtt{NNF}$ and $\mathtt{CSat}$ both terminate.

Soundness and Completeness: Let C_1 be the concept after Step 1 and C_2 the concept after Step 2. Then we have:

Solution to exercise 3: second task

Suppose we run the procedure on concept C and acyclic terminology $\mathcal{T}$.

Termination: Step 1 terminates because $\mathcal{T}$ is acyclic. Steps 2 and 3 terminate because we know that $\mathtt{NNF}$ and $\mathtt{CSat}$ both terminate.

Soundness and Completeness: Let C_1 be the concept after Step 1 and C_2 the concept after Step 2. Then we have:

$$\mathtt{CSat} \text{ returns "yes" on } C_2 \quad \Leftrightarrow \quad C_2 \text{ is satisfiable}$$

Soundness and completeness of $\mathtt{CSat}$ (previous lecture)

Solution to exercise 3: second task

Suppose we run the procedure on concept C and acyclic terminology $\mathcal{T}$.

Termination: Step 1 terminates because $\mathcal{T}$ is acyclic. Steps 2 and 3 terminate because we know that $\mathtt{NNF}$ and $\mathtt{CS}a\mathtt{t}$ both terminate.

Soundness and Completeness: Let C_1 be the concept after Step 1 and C_2 the concept after Step 2. Then we have:

$$\begin{aligned}\mathtt{CS}a\mathtt{t}\text{ returns "yes" on } C_2 \quad &\Leftrightarrow \quad C_2 \text{ is satisfiable}\\ &\Leftrightarrow \quad C_1 \text{ is satisfiable}\end{aligned}$$

NNF preserves equivalence (previous lecture)

Solution to exercise 3: second task

Suppose we run the procedure on concept C and acyclic terminology $\mathcal{T}$.

Termination: Step 1 terminates because $\mathcal{T}$ is acyclic. Steps 2 and 3 terminate because we know that $\mathtt{NNF}$ and $\mathtt{CSat}$ both terminate.

Soundness and Completeness: Let C_1 be the concept after Step 1 and C_2 the concept after Step 2. Then we have:

$$\begin{aligned} \mathtt{CSat} \text{ returns "yes" on } C_2 \quad &\Leftrightarrow \quad C_2 \text{ is satisfiable} \\ &\Leftrightarrow \quad C_1 \text{ is satisfiable} \\ &\Leftrightarrow \quad C \text{ is satisfiable w.r.t. } \mathcal{T} \end{aligned}$$

C_1 satisfiable $\Rightarrow$ take $\mathcal{I}$ with $C_1^{\mathcal{I}} \neq \emptyset$, then reinterpret the defined concepts according to $\mathcal{T}$ in order to get model of $\mathcal{T}$ with C non-empty

Solution to exercise 3:   second task

Suppose we run the procedure on concept C and acyclic terminology $\mathcal{T}$.

Termination: Step 1 terminates because $\mathcal{T}$ is acyclic. Steps 2 and 3 terminate because we know that NNF and CSat both terminate.

Soundness and Completeness: Let C_1 be the concept after Step 1 and C_2 the concept after Step 2. Then we have:

$$\begin{array}{lcl} \texttt{CSat} \text{ returns "yes" on } C_2 & \Leftrightarrow & C_2 \text{ is satisfiable} \\ & \Leftrightarrow & C_1 \text{ is satisfiable} \\ & \Leftrightarrow & C \text{ is satisfiable w.r.t. } \mathcal{T} \end{array}$$

C satisfiable w.r.t. $\mathcal{T} \Rightarrow$ take model $\mathcal{I}$ of $\mathcal{T}$ with $C^{\mathcal{I}} \neq \emptyset$, then because of the definitions in $\mathcal{T}$, must have $C_1^{\mathcal{I}} \neq \emptyset$

Back to tableau algorithms

Extension to KB satisfiability

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

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 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 following KBs

1. $(\mathcal{T}, \{A(a)\})$
2. $(\mathcal{T}, \{R(c, a), B(a)\})$

where $\mathcal{T}$ is the following TBox:

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

Examples: KB satisfiability

Let's try the modified procedure on the following KBs

1. $(\mathcal{T}, \{A(a)\})$
2. $(\mathcal{T}, \{R(c, a), B(a)\})$

where $\mathcal{T}$ is the following TBox:

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

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

Examples: KB satisfiability

Let's try the modified procedure on the following KBs

1. $(\mathcal{T}, \{A(a)\})$
2. $(\mathcal{T}, \{R(c,a), B(a)\})$

where $\mathcal{T}$ is the following TBox:

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

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 $(\{F \sqsubseteq \exists S.F, \{F(a)\}\}$.

$$F(a_0)$$

$$(\neg F \sqcup \exists S.F)\,(a_0)$$

$$\begin{array}{c|l}
\neg F\,(a_0) & (\exists S.F)\,(a_0) \\
 & S(a_0, a_1) \\
✗ & F(a_1) \\
 & \\
 & a_1 \text{ is blocked by } a_0
\end{array}$$

Example: blocking

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

$$F(a_0)$$

$$(\neg F \sqcup \exists S.F)(a_0)$$

$\neg F(a_0)$	$(\exists S.F)(a_0)$
✘	$S(a_0, a_1)$
	$F(a_1)$
	a_1 is blocked by a_0

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 $\mathtt{KBSat}$ (for KB satisfiability).

Termination: The algorithm $\mathtt{KBSat}$ always terminates.

- similar to before: bound the size of generated ABoxes

Soundness: $\mathtt{KBSat}$ 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 $\texttt{KBSat}$ (for KB satisfiability).

Termination: The algorithm $\texttt{KBSat}$ always terminates.

- similar to before: bound the size of generated ABoxes

Soundness: $\texttt{KBSat}$ 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 $\Rightarrow$ $\texttt{KBSat}$ 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:

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

Absorption (1)

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

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

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:

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

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:

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

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

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

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 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 $\mathcal{T} \not\models B \sqsubseteq D$

Homework

Exercise 4: KB satisfiability via tableau

Consider the following TBox (from a previous homework):

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

Use the $\mathtt{KBSat}$ 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. $\mathcal{T} \models G \sqsubseteq F$

Please explain your steps. You are encouraged to use the optimizations introduced in the course.