ONTLOGIES & DESCRIPTION LOGICS

Part of Logic and Languages Module

Meghyn Bienvenu (LaBRI - CNRS & Université de Bordeaux)
REASONING IN EXPRESSIVE DLS
Tableau method: popular approach for reasoning in expressive DLs
- implemented in state-of-the-art DL reasoners

Tableau algorithms are used to decide satisfiability
- solve other tasks (e.g. entailment) by reducing them to satisfiability
**Tableau method**: popular approach for reasoning in expressive DLs
- implemented in *state-of-the-art DL reasoners*

Tableau algorithms are used to **decide satisfiability**
- solve other tasks (e.g. entailment) by reducing them to satisfiability

Idea: to determine whether a given (concept or KB) $\Psi$ is satisfiable, try to construct a (representation of a) model of $\Psi$
- if we succeed, then we have shown that $\Psi$ is satisfiable
- if we fail despite having *considered all possibilities*, then we have proven that $\Psi$ is unsatisfiable
Recall that $\mathcal{ALC}$ concepts are built using the following constructors:

$$\top \quad \bot \quad \neg \quad \sqcup \quad \sqcap \quad \forall r.C \quad \exists r.C$$
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 \textit{negation normal form (NNF)} if the symbol \( \neg \) only appears directly in front of atomic concepts.

- \textbf{in NNF}: \( A \sqcap \neg B, \exists r. \neg A, \neg A \sqcup \neg B \)
- \textbf{not in NNF}: \( \neg(A \sqcap B), \exists r. \neg(\forall s. B), A \sqcup \neg \forall r. B, \neg \top \)
Recall that \textit{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 \textit{ALC} concept \(C\) is in \textbf{negation normal form (NNF)} if the symbol \(\neg\) only appears directly in front of atomic concepts.

\begin{itemize}
  \item \textbf{in NNF}: \(A \sqcap \neg B, \exists r. \neg A, \neg A \sqcup \neg B\)
  \item \textbf{not in NNF}: \(\neg(A \sqcap B), \exists r. \neg(\forall s. B), A \sqcup \neg \forall r. B, \neg \top\)
\end{itemize}

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

\begin{align*}
\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{align*}

Note: say \(C\) and \(D\) are equivalent if the empty TBox entails \(C \equiv D\).
Algorithm NNF, takes as 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 $\text{NNF}(D_1) \sqcap \text{NNF}(D_2)$
- If $C = D_1 \sqcup D_2$, then output $\text{NNF}(D_1) \sqcup \text{NNF}(D_2)$
- If $C = \exists r.D$, then output $\exists r.\text{NNF}(D)$
- If $C = \forall r.D$, then output $\forall r.\text{NNF}(D)$
- If $C = \neg \top$, return $\bot$; if $C = \neg \bot$, then output $\top$
- If $C = \neg(D_1 \sqcap D_2)$, then output $\text{NNF}(\neg D_1) \sqcup \text{NNF}(\neg D_2)$
- If $C = \neg(D_1 \sqcup D_2)$, then output $\text{NNF}(\neg D_1) \sqcap \text{NNF}(\neg D_2)$
- If $C = \neg \exists r.D$, then output $\forall r.\text{NNF}(\neg D)$
- If $C = \neg \forall r.D$, then output $\exists r.\text{NNF}(\neg D)$
- If $C = \neg(\neg D)$, then output $\text{NNF}(D)$

Use $\text{NNF}(E)$ to denote output of $\text{NNF}$ on input $E$
We begin by presenting a tableau algorithm for deciding satisfiability of $\mathcal{ALC}$-concepts (in NNF) w.r.t. the empty TBox

Procedure for testing satisfiability of $C_0$:

- We work with a set $S$ of ABoxes
- Initially, $S$ contains a single ABox $\{C_0(a_0)\}$
We begin by presenting a tableau algorithm for deciding satisfiability of $\mathcal{ALC}$-concepts (in NNF) w.r.t. the empty TBox.

**Procedure for testing satisfiability of $C_0$:**
- We work with a set $S$ of ABoxes.
- Initially, $S$ contains a single ABox $\{C_0(a_0)\}$.
- At each stage, we **apply a tableau rule** to some $A \in S$.
  (note: rules are detailed on next slide)
We begin by presenting a tableau algorithm for deciding
satisfiability of \textit{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 \textbf{apply a tableau rule} to some $A \in S$
  \textit{(note: rules are detailed on next slide)}
- A rule application involves replacing $A$ by one or two ABoxes that extend $A$ with new assertions
We begin by presenting a tableau algorithm for deciding satisfiability of ALC-concepts (in NNF) w.r.t. the empty TBox.

Procedure for testing satisfiability of \( C_0 \):

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

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

Procedure for testing satisfiability of $C_0$:

- We work with a set $S$ of ABoxes
- Initially, $S$ contains a single ABox $\{C_0(a_0)\}$
- At each stage, we apply a tableau rule to some $\mathcal{A} \in S$ (note: rules are detailed on next slide)
- A rule application involves replacing $\mathcal{A}$ by one or two ABoxes that extend $\mathcal{A}$ with new assertions
- Stop applying rules when either:
  - every $\mathcal{A} \in S$ contains a clash, i.e. an assertion $\bot(b)$ 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, satisfiable’ if some $\mathcal{A} \in S$ is clash-free, else “no”.
\textbf{\textit{\(\square\)-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 \(\mathcal{A} \cup \{C_1(a), C_2(a)\}\)
\(\Box\)-rule: \ if \ (C_1 \cap 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)\}\)

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

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

\( \forall \)-rule: if \(\{\forall r. C(a), r(a, b)\} \in A\) and \(C(b) \not\in A\)
then replace \(A\) with \(A \cup \{C(b)\}\)
TABLEAU RULES FOR ALC

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

\(\sqcap\)-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 \(C(b) \not\in \mathcal{A}\)
then replace \(\mathcal{A}\) with \(\mathcal{A} \cup \{C(b)\}\)

\(\exists\)-rule:
if \(\{\exists r.C(a)\} \in \mathcal{A}\) and no \(b\) with \(\{r(a, b), C(b)\} \subseteq \mathcal{A}\),
then pick a new individual name \(d\) and
replace \(\mathcal{A}\) with \(\mathcal{A} \cup \{r(a, d), C(d)\}\)
Test satisfiability of concept \( C = (A \sqcup B) \cap ((\neg B \sqcup D) \cap \neg A) \)
FIRST EXAMPLE: \( \sqcap \) AND \( \sqcup \)

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

Start with \( S = \{ A_0 \} \) where \( A_0 = \{( (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A) )(a_0) \} \).
FIRST EXAMPLE: \( \sqcap \) AND \( \sqcup \)

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

Start with \( S = \{ A_0 \} \) where \( A_0 = \{ ( (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)) \}(a_0) \).

Apply \( \sqcap \)-rule to \( A_0 \):
get \( S = \{ A'_0 \} \) where \( A'_0 = A_0 \cup \{ (A \sqcup B)(a_0), ((\neg B \sqcup D) \sqcap \neg A)(a_0) \} \).
FIRST EXAMPLE: ⊓ AND ⊔

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

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 $\mathcal{A}_0'$:
Test satisfiability of concept \( C = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A) \)

Start with \( S = \{ A_0 \} \) where \( A_0 = \{ ((A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A))(a_0) \} \).

**Apply \( \sqcap \)-rule to \( A_0 \):**
get \( S = \{ A'_0 \} \) where \( A'_0 = A_0 \cup \{ (A \sqcup B)(a_0), ((\neg B \sqcup D) \sqcap \neg A)(a_0) \} \).

**Apply \( \sqcup \)-rule to \( A'_0 \):**
get \( S = \{ A_1, A_2 \} \) where \( A_1 = A'_0 \cup \{ A(a_0) \} \) and \( A_2 = A'_0 \cup \{ B(a_0) \} \).
Test satisfiability of concept \( C = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A) \)

Start with \( S = \{ A_0 \} \) where \( A_0 = \{ ((A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A))(a_0) \} \).

Apply \( \sqcap \)-rule to \( A_0 \):
get \( S = \{ A'_0 \} \) where \( A'_0 = A_0 \cup \{ (A \sqcup B)(a_0), ((\neg B \sqcup D) \sqcap \neg A)(a_0) \} \).

Apply \( \sqcup \)-rule to \( A'_0 \):
get \( S = \{ A_1, A_2 \} \) where \( A_1 = A'_0 \cup \{ A(a_0) \} \) and \( A_2 = A'_0 \cup \{ B(a_0) \} \).
Test satisfiability of concept $C = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$

Start with $S = \{ A_0 \}$ where $A_0 = \{ ((A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A))(a_0) \}$.

Apply $\sqcap$-rule to $A_0$:
get $S = \{ A'_0 \}$ where $A'_0 = A_0 \cup \{ (A \sqcup B)(a_0), ((\neg B \sqcup D) \sqcap \neg A)(a_0) \}$.

Apply $\sqcup$-rule to $A'_0$:
get $S = \{ A_1, A_2 \}$ where $A_1 = A'_0 \cup \{ A(a_0) \}$ and $A_2 = A'_0 \cup \{ B(a_0) \}$.

Apply $\sqcap$-rule to $A_1$:
get $S = \{ A'_1, A_2 \}$ where $A'_1 = A_1 \cup \{ ((\neg B \sqcup D)(a_0), \neg A(a_0) \}$
Test satisfiability of concept \[ C = (A \sqcup B) \cap ((\neg B \sqcup D) \cap \neg A) \]

Start with \( S = \{ A_0 \} \) where \( A_0 = \{ ( (A \sqcup B) \cap ((\neg B \sqcup D) \cap \neg A ) ) (a_0) \} \).

**Apply □-rule to \( A_0 \):**
get \( S = \{ A'_0 \} \) where \( A'_0 = A_0 \cup \{ (A \sqcup B)(a_0), (\neg B \sqcup D) \cap \neg A)(a_0) \} \).

**Apply ⊓-rule to \( A'_0 \):**
get \( S = \{ A_1, A_2 \} \) where \( A_1 = A'_0 \cup \{ A(a_0) \} \) and \( A_2 = A'_0 \cup \{ B(a_0) \} \).

**Apply □-rule to \( A_1 \):**
get \( S = \{ A'_1, A_2 \} \) where \( A'_1 = A_1 \cup \{ (\neg B \sqcup D)(a_0), \neg A(a_0) \} \)

\( A'_1 \) contains clash \( \{ A(a_0), \neg A(a_0) \} \)!
FIRST EXAMPLE: □ AND ⊓

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

Start with \( S = \{ A_0 \} \) where \( A_0 = \{ ((A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A))(a_0) \} \).

Apply □-rule to \( A_0 \):  
get \( S = \{ A'_0 \} \) where \( A'_0 = A_0 \cup \{ (A \sqcup B)(a_0), ((\neg B \sqcup D) \sqcap \neg A)(a_0) \} \).

Apply ⊓-rule to \( A'_0 \):  
get \( S = \{ A_1, A_2 \} \) where \( A_1 = A'_0 \cup \{ A(a_0) \} \) and \( A_2 = A'_0 \cup \{ B(a_0) \} \).

Apply □-rule to \( A_1 \):  
get \( S = \{ A'_1, A_2 \} \) where \( A'_1 = A_1 \cup \{ ((\neg B \sqcup D)(a_0), \neg A(a_0) \} \)
Test satisfiability of concept $C = (A \sqcup B) \cap ((\neg B \sqcup D) \cap \neg A)$

Start with $S = \{ A_0 \}$ where $A_0 = \{ ((A \sqcup B) \cap ((\neg B \sqcup D) \cap \neg A))(a_0) \}$.

Apply $\sqcap$-rule to $A_0$: 
get $S = \{ A'_0 \}$ where $A'_0 = A_0 \cup \{ (A \sqcup B)(a_0), ((\neg B \sqcup D) \cap \neg A)(a_0) \}$.

Apply $\sqcup$-rule to $A'_0$: 
get $S = \{ A_1, A_2 \}$ where $A_1 = A'_0 \cup \{ A(a_0) \}$ and $A_2 = A'_0 \cup \{ B(a_0) \}$.

Apply $\sqcap$-rule to $A_1$: 
get $S = \{ A'_1, A_2 \}$ where $A'_1 = A_1 \cup \{ (\neg B \sqcup D)(a_0), \neg A(a_0) \}$

Apply $\sqcap$-rule to $A_2$: 
get $S = \{ A'_1, A'_2 \}$ where $A'_2 = A_2 \cup \{ (\neg B \sqcup D)(a_0), \neg A(a_0) \}$
Test satisfiability of concept \( C = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A) \)

Start with \( S = \{ A_0 \} \) where \( A_0 = \{ ((A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A))(a_0) \} \).

**Apply \( \sqcap \)-rule to \( A_0 \):**
get \( S = \{ A_0' \} \) where \( A_0' = A_0 \cup \{(A \sqcup B)(a_0), ((\neg B \sqcup D) \sqcap \neg A)(a_0)\} \).

**Apply \( \sqcup \)-rule to \( A_0' \):**
get \( S = \{ A_1, A_2 \} \) where \( A_1 = A_0' \cup \{A(a_0)\} \) and \( A_2 = A_0' \cup \{B(a_0)\} \).

**Apply \( \sqcap \)-rule to \( A_1 \):**
get \( S = \{ A_1', A_2 \} \) where \( A_1' = A_1 \cup \{((\neg B \sqcup D)(a_0), \neg A(a_0))\} \)

**Apply \( \sqcap \)-rule to \( A_2 \):**
get \( S = \{ A_1', A_2' \} \) where \( A_2' = A_2 \cup \{((\neg B \sqcup D)(a_0), \neg A(a_0))\} \)
Test satisfiability of concept \[ C = (A \sqcup B) \cap ((\neg B \sqcup D) \cap \neg A) \]

Start with \( S = \{ A_0 \} \) where \( A_0 = \{ ((A \sqcup B) \cap ((\neg B \sqcup D) \cap \neg A))(a_0) \} \).

**Apply \( \sqcap \)-rule to \( A_0 \):**
get \( S = \{ A'_0 \} \) where \( A'_0 = A_0 \cup \{ (A \sqcup B)(a_0), ((\neg B \sqcup D) \cap \neg A)(a_0) \} \).

**Apply \( \sqcup \)-rule to \( A'_0 \):**
get \( S = \{ A_1, A_2 \} \) where \( A_1 = A'_0 \cup \{ A(a_0) \} \) and \( A_2 = A'_0 \cup \{ B(a_0) \} \).

**Apply \( \sqcap \)-rule to \( A_1 \):**
get \( S = \{ A'_1, A_2 \} \) where \( A'_1 = A_1 \cup \{ (\neg B \sqcup D)(a_0), \neg A(a_0) \} \)

**Apply \( \sqcap \)-rule to \( A_2 \):**
get \( S = \{ A'_1, A'_2 \} \) where \( A'_2 = A_2 \cup \{ (\neg B \sqcup D)(a_0), \neg A(a_0) \} \)

**Apply \( \sqcup \)-rule to \( A'_2 \):**
get \( S = \{ A'_1, A_3, A_4 \} \) where \( A_3 = A'_2 \cup \{ \neg B(a_0) \}, A_4 = A'_2 \cup \{ D(a_0) \} \)
Test satisfiability of concept $C = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$

Start with $S = \{ A_0 \}$ where $A_0 = \{ ((A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A))(a_0) \}$.

Apply $\sqcap$-rule to $A_0$:
get $S = \{ A'_0 \}$ where $A'_0 = A_0 \cup \{(A \sqcup B)(a_0), ((\neg B \sqcup D) \sqcap \neg A)(a_0)\}$.

Apply $\sqcup$-rule to $A'_0$:
get $S = \{ A_1, A_2 \}$ where $A_1 = A'_0 \cup \{A(a_0)\}$ and $A_2 = A'_0 \cup \{B(a_0)\}$.

Apply $\sqcap$-rule to $A_1$:
get $S = \{ A'_1, A_2 \}$ where $A'_1 = A_1 \cup \{((\neg B \sqcup D)(a_0), \neg A(a_0)\}$

Apply $\sqcap$-rule to $A_2$:
get $S = \{ A'_1, A'_2 \}$ where $A'_2 = A_2 \cup \{(\neg B \sqcup D)(a_0), \neg A(a_0)\}$

Apply $\sqcup$-rule to $A'_2$:
get $S = \{ A'_1, A_3, A_4 \}$ where $A_3 = A'_2 \cup \{\neg B(a_0)\}$, $A_4 = A'_2 \cup \{D(a_0)\}$

$A_3$ contains clash $\{B(a_0), \neg B(a_0)\}$!
First Example: □ AND □

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

Start with \( S = \{ A_0 \} \) where \( A_0 = \{ ((A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A))(a_0) \} \).

Apply □-rule to \( A_0 \):
get \( S = \{ A'_0 \} \) where \( A'_0 = A_0 \cup \{ (A \sqcup B)(a_0), ((\neg B \sqcup D) \sqcap \neg A)(a_0) \} \).

Apply □-rule to \( A'_0 \):
get \( S = \{ A_1, A_2 \} \) where \( A_1 = A'_0 \cup \{ A(a_0) \} \) and \( A_2 = A'_0 \cup \{ B(a_0) \} \).

Apply □-rule to \( A_1 \):
get \( S = \{ A'_1, A_2 \} \) where \( A'_1 = A_1 \cup \{ (\neg B \sqcup D)(a_0), \neg A(a_0) \} \)

Apply □-rule to \( A_2 \):
get \( S = \{ A'_1, A'_2 \} \) where \( A'_2 = A_2 \cup \{ (\neg B \sqcup D)(a_0), \neg A(a_0) \} \)

Apply □-rule to \( A'_2 \):
get \( S = \{ A'_1, A_3, A_4 \} \) where \( A_3 = A'_2 \cup \{ \neg B(a_0) \}, A_4 = A'_2 \cup \{ D(a_0) \} \)

\( A_4 \) is complete, so we can stop.
Test satisfiability of concept $C = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$

Start with $S = \{ A_0 \}$ where $A_0 = \{ ( (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)) (a_0) \}$.

Apply $\sqcap$-rule to $A_0$:
get $S = \{ A_0' \}$ where $A_0' = A_0 \cup \{ (A \sqcup B)(a_0), ((\neg B \sqcup D) \sqcap \neg A)(a_0) \}$.

Apply $\sqcup$-rule to $A_0'$:
get $S = \{ A_1, A_2 \}$ where $A_1 = A_0' \cup \{ A(a_0) \}$ and $A_2 = A_0' \cup \{ B(a_0) \}$.

Apply $\sqcap$-rule to $A_1$:
get $S = \{ A_1', A_2 \}$ where $A_1' = A_1 \cup \{ (\neg B \sqcup D)(a_0), \neg A(a_0) \}$

Apply $\sqcap$-rule to $A_2$:
get $S = \{ A_1', A_2' \}$ where $A_2' = A_2 \cup \{ (\neg B \sqcup D)(a_0), \neg A(a_0) \}$

Apply $\sqcup$-rule to $A_2'$:
get $S = \{ A_1', A_3, A_4 \}$ where $A_3 = A_2' \cup \{ \neg B(a_0) \}$, $A_4 = A_2' \cup \{ D(a_0) \}$

$A_4$ is complete and contains no clash $\Rightarrow C_0$ is satisfiable
Test satisfiability of concept

\[ C = (A \sqcup B) \cap ((\neg B \sqcup D) \cap \neg A) \]

\[ (A \sqcup B) \cap ((\neg B \sqcup D) \cap \neg A) (a_0) \]
Test satisfiability of concept \( C = (A \sqcup B) \cap ((\neg B \sqcup D) \cap \neg A) \)

\[
(A \sqcup B) \cap ((\neg B \sqcup D) \cap \neg A) (a_0)
\]

\[
((\neg B \sqcup D) \cap \neg A) (a_0)
\]

\[
(A \sqcup B) (a_0)
\]

**\( \cap \)-rule**
Test satisfiability of concept \( C = (A \sqcup B) \cap ((\neg B \sqcup D) \cap \neg A) \)

\[
\begin{align*}
(A \sqcup B) \cap ((\neg B \sqcup D) \cap \neg A) (a_0) \\
((\neg B \sqcup D) \cap \neg A) (a_0) \\
(A \sqcup B) (a_0) \\
A (a_0) & \quad | \quad \text{□-rule} & \quad B (a_0)
\end{align*}
\]
Test satisfiability of concept \( C = (A \sqcup B) \cap ((\neg B \sqcup D) \cap \neg A) \)
Test satisfiability of concept $C = (A \sqcup B) \cap ((\neg B \sqcup D) \cap \neg A)$

\[(A \sqcup B) \cap ((\neg B \sqcup D) \cap \neg A) (a_0)\]
\[((\neg B \sqcup D) \cap \neg A) (a_0)\]
\[(A \sqcup B) (a_0)\]

\[A (a_0)\]
\[\neg B \sqcup D) (a_0)\]
\[\neg A (a_0)\]

\[B (a_0)\]
\[\neg B \sqcup D) (a_0)\]
\[\neg A (a_0)\]

$\sqcap$-rule
Test satisfiability of concept $C = (A \sqcup B) \cap ((\neg B \sqcup D) \cap \neg A)$

\[
\begin{align*}
(A \sqcup B) \cap ((\neg B \sqcup D) \cap \neg A) & (a_0) \\
((\neg B \sqcup D) \cap \neg A) & (a_0) \\
(A \sqcup B) & (a_0) \\
A & (a_0) \\
(\neg B \sqcup D) & (a_0) \\
\neg A & (a_0) \\
\neg B & (a_0) \\
\neg A & (a_0) \\
\neg B & (a_0) \\
D & (a_0) \quad \text{\textbox{-rule}}
\end{align*}
\]
Test satisfiability of concept $C = (A \sqcup B) \cap ((\neg B \sqcup D) \cap \neg A)$

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

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

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

$\neg B \ (a_0)$

$D \ (a_0)$
Test satisfiability of concept $C = (A \sqcup B) \cap ((\neg B \sqcup D) \cap \neg A)$

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

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

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

$D (a_0)$
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 $C_0 = (A \sqcup B) \sqcap ((\neg B \sqcup D) \sqcap \neg A)$
ANOTHER EXAMPLE: ∀ AND ∃

Let’s use the tableau procedure to test satisfiability of

\[ C = \exists r. A \land \forall r. \neg A \]
Let’s use the tableau procedure to test satisfiability of

\[ C = \exists r. A \cap \forall r. \neg A \]

Start with \( S = \{ A_0 \} \) where \( A_0 = \{ (\exists r. A \cap \forall r. \neg A)(a_0) \} \).
Let’s use the tableau procedure to test satisfiability of

\[ C = \exists r. A \sqcap \forall r. \neg A \]

Start with \( S = \{ A_0 \} \) where \( A_0 = \{ (\exists r. A \sqcap \forall r. \neg A)(a_0) \} \).

**Apply \( \sqcap \)-rule to \( A_0 \):**

get \( S = \{ A'_0 \} \) where \( A'_0 = A_0 \cup \{(\exists r. A)(a_0), (\forall r. \neg A)(a_0)\} \).
Let’s use the tableau procedure to test satisfiability of

\[ C = \exists r. A \sqcap \forall r. \neg A \]

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 \( \mathcal{A}_0' = \mathcal{A}_0 \cup \{(\exists r. A)(a_0), (\forall r. \neg A)(a_0)\}\).
Let’s use the tableau procedure to test satisfiability of

\[ C = \exists r. A \land \forall r. \neg A \]

Start with \( S = \{ A_0 \} \) where \( A_0 = \{ (\exists r. A \land \forall r. \neg A)(a_0) \} \).

**Apply \( \sqcap \)-rule to \( A_0 \):**

get \( S = \{ A'_0 \} \) where \( A'_0 = A_0 \cup \{(\exists r. A)(a_0), (\forall r. \neg A)(a_0)\} \).

**Apply \( \exists \)-rule to \( A'_0 \):**

get \( S = \{ A''_0 \} \) where \( A''_0 = A'_0 \cup \{r(a_0, a_1), A(a_1)\} \).
Let’s use the tableau procedure to test satisfiability of

\[ C = \exists r. A \cap \forall r. \neg A \]

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

Apply \( \cap \)-rule to \( \mathcal{A}_0 \):
get \( S = \{ \mathcal{A}' \} \) where \( \mathcal{A}' = \mathcal{A}_0 \cup \{ (\exists r. A)(a_0), (\forall r. \neg A)(a_0) \} \).

Apply \( \exists \)-rule to \( \mathcal{A}' \):
get \( S = \{ \mathcal{A}'' \} \) where \( \mathcal{A}'' = \mathcal{A}' \cup \{ r(a_0, a_1), A(a_1) \} \).
Let’s use the tableau procedure to test satisfiability of

\[ C = \exists r. A \land \forall r. \neg A \]

Start with \( S = \{ A_0 \} \) where \( A_0 = \{ (\exists r. A \land \forall r. \neg A)(a_0) \} \).

**Apply \( \cap \)-rule to \( A_0 \):**
get \( S = \{ A'_0 \} \) where \( A'_0 = A_0 \cup \{ (\exists r. A)(a_0), (\forall r. \neg A)(a_0) \} \).

**Apply \( \exists \)-rule to \( A'_0 \):**
get \( S = \{ A''_0 \} \) where \( A''_0 = A'_0 \cup \{ r(a_0, a_1), A(a_1) \} \).

**Apply \( \forall \)-rule to \( A''_0 \):**
get \( S = \{ A'''_0 \} \) where \( A'''_0 = A''_0 \cup \{ \neg A(a_1) \} \).
Let’s use the tableau procedure to test satisfiability of

\[ C = \exists r. A \sqcap \forall r. \neg A \]

Start with \( S = \{ A_0 \} \) where \( A_0 = \{ (\exists r. A \sqcap \forall r. \neg A)(a_0) \} \).

Apply \( \sqcap \)-rule to \( A_0 \):
get \( S = \{ A'_0 \} \) where \( A'_0 = A_0 \cup \{(\exists r. A)(a_0), (\forall r. \neg A)(a_0)\} \).

Apply \( \exists \)-rule to \( A'_0 \):
get \( S = \{ A''_0 \} \) where \( A''_0 = A'_0 \cup \{ r(a_0, a_1), A(a_1) \} \).

Apply \( \forall \)-rule to \( A''_0 \):
get \( S = \{ A'''_0 \} \) where \( A'''_0 = A''_0 \cup \{ \neg A(a_1) \} \).

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

"ANOTHER EXAMPLE: \( \forall \) AND \( \exists \)"
ANOTHER EXAMPLE: \( \forall \) AND \( \exists \)

Let’s use the tableau procedure to test satisfiability of

\[ C = \exists r. A \cap \forall r. \neg A \]

Start with \( S = \{ A_0 \} \) where \( A_0 = \{ (\exists r. A \cap \forall r. \neg A)(a_0) \} \).

Apply \( \cap \)-rule to \( A_0 \):
get \( S = \{ A'_0 \} \) where \( A'_0 = A_0 \cup \{(\exists r. A)(a_0), (\forall r. \neg A)(a_0)\} \).

Apply \( \exists \)-rule to \( A'_0 \):
get \( S = \{ A''_0 \} \) where \( A''_0 = A'_0 \cup \{r(a_0, a_1), A(a_1)\} \).

Apply \( \forall \)-rule to \( A''_0 \):
get \( S = \{ A'''_0 \} \) where \( A'''_0 = A''_0 \cup \{\neg A(a_1)\} \).

The only ABox in \( S \) contains a clash \( \Rightarrow C_0 \) is unsatisfiable
Test satisfiability of concept $C = \exists r. A \land \forall r. \neg A$

$$(\exists r. A \land \forall r. \neg A)(a_0)$$
Test satisfiability of concept \( C = \exists r. A \land \forall r. \neg A \)

\[
(\exists r. A \land \forall r. \neg A)(a_0) \\
(\exists r. A)(a_0) \quad \square\text{-rule} \\
(\forall r. \neg A)(a_0)
\]
Test satisfiability of concept \( C = \exists r. A \land \forall r. \neg A \)

\[
(\exists r. A \land \forall r. \neg A)(a_0) \\
(\exists r. A)(a_0) \\
(\forall r. \neg A)(a_0) \\
r(a_0, a_1) \quad \exists\text{-rule} \\
A(a_1)
\]
Test satisfiability of concept $C = \exists r. A \sqcap \forall r. \neg A$

\[
(\exists r. A \sqcap \forall r. \neg A)(a_0)
\]

\[
(\exists r. A)(a_0)
\]

\[
(\forall r. \neg A)(a_0)
\]

\[
r(a_0, a_1)
\]

\[
A(a_1)
\]

\[
\neg A(a_1) \quad \forall\text{-rule}
\]
Test satisfiability of concept $C = \exists r.A \land \forall r.\neg A$

\[
(\exists r.A \land \forall r.\neg A)(a_0)
\]
\[
(\exists r.A)(a_0)
\]
\[
(\forall r.\neg A)(a_0)
\]
\[
r(a_0, a_1)
\]
\[
A(a_1)
\]
\[
\neg A(a_1)
\]

\[
\times
\]
Test satisfiability of concept $C = \exists r.A \sqcap \forall r.\neg A$

$(\exists r.A \sqcap \forall r.\neg A)(a_0)\n(\exists r.A)(a_0)\n(\forall r.\neg A)(a_0)\n\neg A(a_1)\n\neg A(a_1)\n\times$

Conclude that $C$ is unsatisfiable
Suppose that we consider a slightly different concept

\[ C_0 = \exists r. A \sqcap \forall r. \neg B \]

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

\[
(\exists r. A \sqcap \forall r. \neg B)(a_0) \quad (\exists r. A)(a_0) \quad (\forall r. \neg B)(a_0) \quad r(a_0, a_1) \quad A(a_1) \quad \neg B(a_1)
\]
Suppose that we consider a slightly different concept

\[ C_0 = \exists r. A \land \forall r. \neg B \]

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

\[(\exists r. A \land \forall r. \neg B)(a_0) (\exists r. A)(a_0) (\forall r. \neg B)(a_0) r(a_0, a_1) A(a_1) \neg B(a_1)\]

**Corresponding interpretation \( I \):**

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

Can check that \( I \) is such that \( C^I_0 = \{a_0\} \).
Let’s call our tableau algorithm \texttt{CSat} (for concept satisfiability).

To show that \texttt{CSat} is a decision procedure, we must show:

- **Termination:** The algorithm \texttt{CSat} always terminates.

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

- **Completeness:** $C_0$ satisfiable $\Rightarrow \texttt{CSat}$ will output “yes”.
Subconcepts of a concept:

\[
\begin{align*}
\text{sub}(A) &= \{A\} \\
\text{sub}(\neg C) &= \{\neg C\} \cup \text{sub}(C) \\
\text{sub}(\exists r. C) &= \{\exists r. C\} \cup \text{sub}(C) \\
\text{sub}(\forall r. C) &= \{\forall r. C\} \cup \text{sub}(C) \\
\text{sub}(C_1 \sqcup C_2) &= \{C_1 \sqcup C_2\} \cup \text{sub}(C_1) \cup \text{sub}(C_2) \\
\text{sub}(C_1 \sqcap C_2) &= \{C_1 \sqcap C_2\} \cup \text{sub}(C_1) \cup \text{sub}(C_2)
\end{align*}
\]
Subconcepts of a concept:

\[
\begin{align*}
\text{sub}(A) &= \{A\} \\
\text{sub}(\neg C) &= \{\neg C\} \cup \text{sub}(C) \\
\text{sub}(\exists r. C) &= \{\exists r. C\} \cup \text{sub}(C) \\
\text{sub}(\forall r. C) &= \{\forall r. C\} \cup \text{sub}(C) \\
\text{sub}(C_1 \sqcup C_2) &= \{C_1 \sqcup C_2\} \cup \text{sub}(C_1) \cup \text{sub}(C_2) \\
\text{sub}(C_1 \sqcap C_2) &= \{C_1 \sqcap C_2\} \cup \text{sub}(C_1) \cup \text{sub}(C_2)
\end{align*}
\]

Role depth of a concept:

\[
\begin{align*}
\text{depth}(A) &= \text{depth}(\top) = \text{depth}(\bot) = 0 \\
\text{depth}(\neg C) &= \text{depth}(C) \\
\text{depth}(\exists r. C) &= \text{depth}(\forall r. C) = \text{depth}(C) + 1 \\
\text{depth}(C_1 \sqcup C_2) &= \text{depth}(C_1 \sqcap C_2) = \max(\text{depth}(C_1), \text{depth}(C_2))
\end{align*}
\]
Preliminary Definitions

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

\[
\begin{align*}
\text{sub}(A) &= \{A\} \\
\text{sub}(\neg C) &= \{\neg C\} \cup \text{sub}(C) \\
\text{sub}(\exists r. C) &= \{\exists r. C\} \cup \text{sub}(C) \\
\text{sub}(\forall r. C) &= \{\forall r. C\} \cup \text{sub}(C) \\
\text{sub}(C_1 \sqcup C_2) &= \{C_1 \sqcup C_2\} \cup \text{sub}(C_1) \cup \text{sub}(C_2) \\
\text{sub}(C_1 \sqcap C_2) &= \{C_1 \sqcap C_2\} \cup \text{sub}(C_1) \cup \text{sub}(C_2)
\end{align*}
\]

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

\[
\begin{align*}
\text{depth}(A) &= \text{depth}(\top) = \text{depth}(\bot) = 0 \\
\text{depth}(\neg C) &= \text{depth}(C) \\
\text{depth}(\exists r. C) &= \text{depth}(\forall r. C) = \text{depth}(C) + 1 \\
\text{depth}(C_1 \sqcup C_2) &= \text{depth}(C_1 \sqcap C_2) = \max(\text{depth}(C_1), \text{depth}(C_2))
\end{align*}
\]
Suppose we run \texttt{CSat} starting from $S = \{\{C_0(a_0)\}\}$. 

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

We observe that for every ABox $\mathcal{A}$ generated by the procedure:
Suppose we run \texttt{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 \( D \in \text{sub}(C_0) \)
Suppose we run $\text{CSat}$ 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
Suppose we run \textbf{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
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 $\text{depth}(D) \leq \text{depth}(C_0) - k$
Suppose we run \textbf{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 \( D \in \text{sub}(C_0) \)
   
   \( \cdot \) \( \mathcal{A} \) 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 \( \text{depth}(D) \leq \text{depth}(C_0) - k \)
   
   \( \cdot \) each individual in \( \mathcal{A} \) is at distance \( \leq \text{depth}(C_0) \) from \( a_0 \)
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)
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
Suppose we run \texttt{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 \( D \in \text{sub}(C_0) \)
   
   \( \cdot \) \( \mathcal{A} \) 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 \( \text{depth}(D) \leq \text{depth}(C_0) - k \)
   
   \( \cdot \) 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: \textbf{bound on the size of ABoxes} generated by the procedure

The tableau procedure \textbf{only adds assertions} to ABoxes

\( \Rightarrow \) \textbf{eventually all ABoxes will contain a clash or will be complete}
Suppose that $\text{CSat}$ returns “yes” on input $C_0$.

Then $S$ must contain a complete and clash-free ABox $\mathcal{A}$. 
Suppose that \( \text{CSat} \) 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 \( C_0^\mathcal{I} \neq \emptyset \).
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 $C_0^\mathcal{I} \neq \emptyset$

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

\[
D(b) \in \mathcal{A} \quad \Rightarrow \quad b \in D^\mathcal{I}
\]
Base case: $D = A$ or $D = \neg A$ or $D = \top$ or $D = \bot$
**Base case:** $D = A$ or $D = \neg A$ or $D = \top$ or $D = \bot$

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

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

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

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

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

Again, many cases to consider:

- $D = E \cap F$: since $A$ is complete, it must contain both $E(b)$ and $F(b)$. Applying the IH, we get $b \in E^I$ and $b \in F^I$, hence $b \in (E \cap F)^I$

- $D = \exists r.E$: since $A$ is complete, there exists $c$ such that $r(b, c) \in A$ and $E(c) \in A$. Then $(b, c) \in r^I$. From IH, get $c \in E^I$, so $b \in (\exists r.E)^I$

- $D = E \cup F$: **left as practice**

- $D = \forall R.E$: **left as practice**
Suppose that the concept $C_0$ is satisfiable.

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

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

We observe that the tableau rules are satisfiability-preserving:

- If an ABox $\mathcal{A}$ is satisfiable and $\mathcal{A}'$ is the result of applying a rule to $\mathcal{A}$, then $\mathcal{A}'$ is also satisfiable.

- If $\mathcal{A}$ is satisfiable and $\mathcal{A}_1$ and $\mathcal{A}_2$ are obtained when applying a rule to $\mathcal{A}$, then either $\mathcal{A}_1$ is satisfiable or $\mathcal{A}_2$ is satisfiable.
Suppose that the concept $C_0$ is satisfiable.

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

We observe that the tableau rules are satisfiability-preserving:

- If an ABox $\mathcal{A}$ is satisfiable and $\mathcal{A}'$ is the result of applying a rule to $\mathcal{A}$, then $\mathcal{A}'$ is also satisfiable.

- If $\mathcal{A}$ is satisfiable and $\mathcal{A}_1$ and $\mathcal{A}_2$ are obtained when applying a rule to $\mathcal{A}$, then either $\mathcal{A}_1$ is satisfiable or $\mathcal{A}_2$ is satisfiable.

We start with a satisfiable ABox and the rules are satisfiability-preserving, so eventually we will reach a complete, satisfiable (thus: clash-free) ABox and output ‘yes’.
Bad news: our algorithm may require exponential time and space...

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

\[
\bigwedge_{0 \leq i < n} \forall r \ldots \forall r. (\exists r.B \sqcap \exists r.\neg B)
\]

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

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

\[
\bigwedge_{0 \leq i < n} \forall r. \ldots \forall r. (\exists r. B \sqcap \exists r. \neg B)
\]

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

- instead of set of ABoxes, keep only 1 ABox in memory at a time
  - when apply the $\sqcup$-rule, first examine $A_1$, then afterwards examine $A_2$
  - remember that second disjunct stills needs to be checked
- explore the children of an individual one at a time
  - possible because no interaction between the different “branches”
  - store which $\exists r. C$ concepts have been tested, which are left to do
- this allows us to keep at most $|C_0|$ individuals in memory at a time
Hierarchy of complexity classes

\[ \text{PTIME} \subseteq \text{NP} \subseteq \cdots \subseteq \text{PSPACE} \subseteq \text{EXPTIME} \subseteq \text{NEXPTIME} \subseteq \cdots \subseteq \text{EXPSPACE} \cdots \]

(it is believed that all inclusions are strict)
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)

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

PSPACE-complete problems = hardest problems in PSPACE
Hierarchy of complexity classes

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

(it is believed that all inclusions are strict)

\text{PSPACE} = \text{class of decision problems solvable in polynomial space}

\text{PSPACE}-complete problems = hardest problems in PSPACE

**Theorem**: \text{ALC} concept satisfiability (no TBox) is \text{PSPACE}-complete.

- Membership in \text{PSPACE} shown using modified tableau procedure
- Hardness for \text{PSPACE} shown by giving a reduction from some known \text{PSPACE}-hard problem (e.g. QBF validity)
Now we want to modify the algorithm to handle KB satisfiability.

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

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

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

Adding a TBox is a bit more tricky...

Idea: if \( C \sqsubseteq D \), then every element must satisfy either \( \neg C \) or \( D \)
Now we want to modify the algorithm to handle KB satisfiability.

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

Adding a TBox is a bit more tricky...

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

Concretely, we might try adding the following rule:

**TBox rule**

if \( a \) is in \( A \), \( C \sqsubseteq D \in \mathcal{T} \), & \( (\text{NNF}(\neg C) \sqcup \text{NNF}(D))(a) \notin A \)
then replace \( A \) with \( A \cup \{(\text{NNF}(\neg C) \sqcup \text{NNF}(D))(a)\} \)
Let’s try the modified procedure on the KB \( (\{F \sqsubseteq \exists s. F\}, \{F(a)\}) \).
Let’s try the modified procedure on the KB \( (\{F \sqsubseteq \exists s. F\}, \{F(a)\}) \).

Seems we have a problem... How can we ensure termination?
Basic idea: if two individuals “look the same”, then it is unnecessary to explore both of them.
**Basic idea:** if two individuals “look the same”, then it is unnecessary to explore both of them.

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

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

Say that individual $a$ **is blocked** (in $\mathcal{A}$) if some $b$ blocks $a$. 
Basic idea: if two individuals “look the same”, then it is unnecessary to explore both of them

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

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

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

Modify rules so that they only apply to unblocked individuals.
TABLEAU RULES FOR KBS

\(\square\)-rule: if \((C_1 \cap C_2)(a) \in A\), \textbf{a is not blocked}, and \(\{C_1(a), C_2(a)\} \not\subseteq A\), then replace \(A\) with \(A \cup \{C_1(a), C_2(a)\}\)

\(\square\)-rule: if \((C_1 \cup C_2)(a) \in A\), \textbf{a is not blocked}, and \(\{C_1(a), C_2(a)\} \cap A = \emptyset\), then replace \(A\) with \(A \cup \{C_1(a)\}\) and \(A \cup \{C_2(a)\}\)

\(\forall\)-rule: if \(\{\forall r. C(a), r(a, b)\} \in A\), \textbf{a is not blocked}, and \(C(b) \not\in A\), then replace \(A\) with \(A \cup \{C(b)\}\)

\(\exists\)-rule: if \(\{\exists r. C(a)\} \in A\), \textbf{a is not blocked}, and no \(\{r(a, b), C(b)\} \subseteq A\), then \textbf{pick a new individual name} \(d\) and replace \(A\) with \(A \cup \{r(a, d), C(d)\}\)

\(\square\)-rule: if \(a\) appears in \(A\) and \textbf{a is not blocked}, \(C \sqsubseteq D \in \mathcal{T}\), and \((\text{NNF}(\neg C) \sqcup \text{NNF}(D))(a) \not\in A\), then replace \(A\) with \(A \cup \{(\text{NNF}(\neg C) \sqcup \text{NNF}(D))(a)\}\)
Let’s try blocking on the problematic KB \(\{F \subseteq \exists s.F\}, \{F(a)\}\)
Let’s try blocking on the problematic KB \( \{F \sqsubseteq \exists s.F\}, \{F(a)\} \)

\[
\begin{align*}
F(a_0) \\
(\neg F \sqcup \exists s.F)(a_0) \\
\neg F(a_0) & \quad (\exists s.F)(a_0) \\
\times & \quad s(a_0, a_1) \\
& \quad F(a_1) \\
\end{align*}
\]

\( a_1 \) is blocked by \( a_0 \)
EXAMPLE: BLOCKING

Let’s try blocking on the problematic KB \(
\{F \subseteq \exists s.F\}, \{F(a)\}\)

\[
\begin{align*}
F(a_0) & \\
(\neg F \sqcup \exists s.F)(a_0) & \\
\neg F(a_0) & (\exists s.F)(a_0) \\
\text{✗} & s(a_0, a_1) \\
& F(a_1)
\end{align*}
\]

\(a_1\) is blocked by \(a_0\)

We obtain a complete and clash-free ABox \(\Rightarrow\) the KB is satisfiable
ANOTHER BLOCKING EXAMPLE

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

We can do this by running the algorithm on $(\mathcal{T}, \{(A \sqcap \neg D)(a_0)\})$. Result: the KB is unsatisfiable $\Rightarrow$ $\mathcal{T} \models A \sqsubseteq D$.

Observe: individual can be blocked, then later become unblocked.
Consider the TBox $\mathcal{T} = \{A \sqsubseteq \exists r.A, A \sqsubseteq B, \exists r.B \sqsubseteq D\}$ and suppose we want to test whether $\mathcal{T} \models A \sqsubseteq D$.

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

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

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

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

Observe: individual can be blocked, then later become unblocked
Let’s call our new tableau algorithm **KBSat** (for KB satisfiability).

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

- similar to before: bound the size of generated ABoxes
Let’s call our new tableau algorithm **KBSat** (for KB satisfiability).

**Termination**: The algorithm **KBSat** always terminates.
- similar to before: bound the size of generated ABoxes

**Soundness**: **KBSat** outputs “yes” on \((\mathcal{T}, \mathcal{A}) \Rightarrow (\mathcal{T}, \mathcal{A})\) is satisfiable.
- again, we use complete, clash-free ABox to build a model
- tricky part: need to handle the blocked individuals
Let’s call our new tableau algorithm **KBSat** (for KB satisfiability).

**Termination**: The algorithm **KBSat** always terminates.
- similar to before: bound the size of generated ABoxes

**Soundness**: **KBSat** outputs “yes” on \((\mathcal{T}, \mathcal{A}) \Rightarrow (\mathcal{T}, \mathcal{A})\) is satisfiable.
- again, we use complete, clash-free ABox to build a model
- tricky part: need to handle the blocked individuals

**Completeness**: \((\mathcal{T}, \mathcal{A})\) satisfiable \(\Rightarrow\) **KBSat** will output “yes”.
- again, show rules satisfiability-preserving
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

So: \texttt{KBSat} is a **decision procedure for KB satisfiability**.
Tableau procedure takes **exponential time and space**
- can have **exponentially long ‘branches’** to explore
Tableau procedure takes exponential time and space
  · can have exponentially long ‘branches’ to explore

Complexity results tell us this is unavoidable in worst case:

Theorem: In $\mathcal{ALC}$, KB satisfiability is EXPTIME-complete
  · for highly expressive DLs ($\leadsto$ OWL 2): complexity even higher

This result means no polynomial-time algorithm can every be found.
Despite high worst-case complexity, tableau algorithms for \textit{ALC} and other expressive DLs can work well in practice.

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

However, good performance crucially depends on optimizations!
Despite high worst-case complexity, tableau algorithms for \( \mathcal{ALC} \) and other expressive DLs can work well in practice.

However, good performance crucially depends on optimizations!

Many types of optimizations:

- explore only one branch of one ABox at a time
Despite high worst-case complexity, tableau algorithms for $\mathcal{ALC}$ and other expressive DLs can work well in practice.

However, good performance crucially depends on optimizations!

Many types of optimizations:
- explore only one branch of one ABox at a time
- strategies / heuristics for choosing next rule to apply
Despite high worst-case complexity, tableau algorithms for $\mathcal{ALC}$ and other expressive DLs can work well in practice.

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

Many types of optimizations:
- explore only one branch of one ABox at a time
- strategies / heuristics for choosing next rule to apply
- caching of results to reduce redundant computation
Despite high worst-case complexity, \textbf{tableau algorithms} for \textit{ALC} and other expressive DLs \textbf{can work well in practice}.

However, \textbf{good performance crucially depends on optimizations}!

Many types of optimizations:

- explore only one branch of one ABox at a time
- strategies / heuristics for choosing next rule to apply
- caching of results to reduce redundant computation
- examine source of conflicts to prune search space (backjumping)
Despite high worst-case complexity, tableau algorithms for $\mathcal{ALC}$ and other expressive DLs can work well in practice.

However, good performance crucially depends on optimizations!

Many types of optimizations:

- explore only one branch of one ABox at a time
- strategies / heuristics for choosing next rule to apply
- caching of results to reduce redundant computation
- examine source of conflicts to prune search space (backjumping)
- reduce number of $\sqcap$’s created by TBox inclusions (absorption)
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 $\sqcap$’s created by TBox inclusions (absorption)
- reduce number of satisfiability checks during classification
When \( \mathcal{T} = \{C_i \sqsubseteq D_i \mid 1 \leq i \leq n\} \), we get \( n \) disjunctions per individual:

\[
(\text{NNF}(\neg C_1) \sqcup \text{NNF}(D_1))(a), \ldots, (\text{NNF}(\neg C_n) \sqcup \text{NNF}(D_n))(a)
\]
When \( \mathcal{T} = \{ C_i \subseteq D_i \mid 1 \leq i \leq n \} \), we get \( n \) disjunctions per individual:

\[
(\text{NNF}(\neg C_1) \sqcup \text{NNF}(D_1))(a), \ldots, (\text{NNF}(\neg C_n) \sqcup \text{NNF}(D_n))(a)
\]

**Observation:** if have \( A \sqsubseteq D \) with \( A \) a concept name

- if don’t have \( A(a) \), can satisfy the inclusion by choosing \( \neg A(a) \)
- if have \( A(a) \), then must have \( D(a) \)
When $T = \{C_i \sqsubseteq D_i \mid 1 \leq i \leq n\}$, we get $n$ disjunctions per individual:

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

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

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

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

$\sqsubseteq^\text{at}$-rule: if $A(a) \in \mathcal{A}$, $a$ is not blocked, $A \sqsubseteq D \in T$ (with $A$ atomic), and $D(a) \not\in \mathcal{A}$, then replace $\mathcal{A}$ with $\mathcal{A} \cup \{D(a)\}$
When $\mathcal{T} = \{ C_i \sqsubseteq D_i \mid 1 \leq i \leq n \}$, we get $n$ disjunctions per individual:

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

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

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

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

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

**Good news:** we’ve lowered the number of disjunctions!
Second observation: can transform some inclusions with complex concept on left into equivalent inclusions with atomic left-hand side.
**Second observation**: can **transform** some inclusions with complex concept on left into **equivalent inclusions with atomic left-hand side**

\[(A \sqcap C) \sqsubseteq D \quad \leadsto \quad A \sqsubseteq (\neg C \sqcup D)\]
Second observation: can transform some inclusions with complex concept on left into equivalent inclusions with atomic left-hand side

\[(A \cap C) \subseteq D \quad \leadsto \quad A \subseteq (\neg C \sqcup D)\]

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 $\sqsubseteq^\text{at}$-rule for inclusions $A \subseteq D$ with $A$ a concept name
   - use regular $\sqsubseteq$-rule for the other TBox inclusions
Let’s use absorption on the KB \((\mathcal{T}, \{A(a)\})\) with:

\[
\{ \quad A \sqsubseteq \exists r. B \quad B \sqsubseteq D \quad \exists r. D \sqsubseteq \neg A \quad \}
\]
Let’s use absorption on the KB \((\mathcal{T}, \{A(a)\})\) with:

\[
\left\{ \begin{array}{c}
A \sqsubseteq \exists r. B \\
B \sqsubseteq D \\
\exists r. D \sqsubseteq \neg A
\end{array} \right. 
\]

· first two inclusions in \(\mathcal{T}\) already have concept name on left
· third inclusion in \(\mathcal{T}\) can be equivalently written as \(A \sqsubseteq \forall r. \neg D\)
· so: only need to use \(\sqsubseteq^{at}\)-rule
Let’s use absorption on the KB $(\mathcal{T}, \{A(a)\})$ with:

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

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

**Result:** avoid disjunction, algorithm terminates much faster!
Classification: find all pairs of concept names $A, B$ with $\mathcal{T} \models A \sqsubseteq B$

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

- but $\mathcal{T}$ may contain hundreds or thousands of concept names....
Classification: find all pairs of concept names $A, B$ with $\mathcal{T} \models A \sqsubseteq B$

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

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

Each check is costly $\Rightarrow$ want to reduce number of checks
Classification: find all pairs of concept names $A, B$ with $\mathcal{T} \models A \sqsubseteq B$

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

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

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

Some ideas:

- some cases are obvious
  - $A \sqsubseteq A$ and inclusions that are explicitly stated in $\mathcal{T}$
Classification: **find all pairs of concept names** $A, B$ with $\mathcal{T} \models A \sqsubseteq B$

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

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

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

Some ideas:

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