Tableau Techniques for DLs and an introduction to OWL2

M. Koubarakis, G. Santipantakis

December 16, 2024

Table of Contents

DL - Revision

- $\mathcal{ALC}$ is the Attribute Language with general Complement.

DL - Revision

- $\mathcal{ALC}$ is the Attribute Language with general Complement.
- The $\mathcal{C}$ denotes an extension (with complement) of a more restrictive language $\mathcal{AL}$.

DL - Revision

- $\mathcal{ALC}$ is the Attribute Language with general Complement.
- The $\mathcal{C}$ denotes an extension (with complement) of a more restrictive language $\mathcal{AL}$.
- Other possible extensions of DL include:

DL - Revision

- $\mathcal{ALC}$ is the Attribute Language with general Complement.
- The $\mathcal{C}$ denotes an extension (with complement) of a more restrictive language $\mathcal{AL}$.
- Other possible extensions of DL include:
  - $\mathcal{H}$: Role hierarchies;

DL - Revision

- $\mathcal{ALC}$ is the Attribute Language with general Complement.
- The $\mathcal{C}$ denotes an extension (with complement) of a more restrictive language $\mathcal{AL}$.
- Other possible extensions of DL include:
  - $\mathcal{H}$: Role hierarchies;
  - $\mathcal{R}$: Complex role hierarchies;

DL - Revision

- $\mathcal{ALC}$ is the Attribute Language with general Complement.
- The $\mathcal{C}$ denotes an extension (with complement) of a more restrictive language $\mathcal{AL}$.
- Other possible extensions of DL include:
  - $\mathcal{H}$: Role hierarchies;
  - $\mathcal{R}$: Complex role hierarchies;
  - $\mathcal{N}$: Cardinality restrictions;

DL - Revision

- $\mathcal{ALC}$ is the Attribute Language with general Complement.
- The $\mathcal{C}$ denotes an extension (with complement) of a more restrictive language $\mathcal{AL}$.
- Other possible extensions of DL include:
  - $\mathcal{H}$: Role hierarchies;
  - $\mathcal{R}$: Complex role hierarchies;
  - $\mathcal{N}$: Cardinality restrictions;
  - $\mathcal{Q}$: Qualified cardinality restrictions;

DL - Revision

- $\mathcal{ALC}$ is the Attribute Language with general Complement.
- The $\mathcal{C}$ denotes an extension (with complement) of a more restrictive language $\mathcal{AL}$.
- Other possible extensions of DL include:
  - $\mathcal{H}$: Role hierarchies;
  - $\mathcal{R}$: Complex role hierarchies;
  - $\mathcal{N}$: Cardinality restrictions;
  - $\mathcal{Q}$: Qualified cardinality restrictions;
  - $\mathcal{O}$: Closed classes;

DL - Revision

- $\mathcal{ALC}$ is the Attribute Language with general Complement.
- The $\mathcal{C}$ denotes an extension (with complement) of a more restrictive language $\mathcal{AL}$.
- Other possible extensions of DL include:
  - $\mathcal{H}$: Role hierarchies;
  - $\mathcal{R}$: Complex role hierarchies;
  - $\mathcal{N}$: Cardinality restrictions;
  - $\mathcal{Q}$: Qualified cardinality restrictions;
  - $\mathcal{O}$: Closed classes;
  - $\mathcal{I}$: Inverse roles;

DL - Revision

- $\mathcal{ALC}$ is the Attribute Language with general Complement.
- The $\mathcal{C}$ denotes an extension (with complement) of a more restrictive language $\mathcal{AL}$.
- Other possible extensions of DL include:
  - $\mathcal{H}$: Role hierarchies;
  - $\mathcal{R}$: Complex role hierarchies;
  - $\mathcal{N}$: Cardinality restrictions;
  - $\mathcal{Q}$: Qualified cardinality restrictions;
  - $\mathcal{O}$: Closed classes;
  - $\mathcal{I}$: Inverse roles;
  - $\mathcal{D}$: Datatypes;
  - ...

DL - Revision

- $\mathcal{ALC}$ is the Attribute Language with general Complement.
- The $\mathcal{C}$ denotes an extension (with complement) of a more restrictive language $\mathcal{AL}$.
- Other possible extensions of DL include:
  - $\mathcal{H}$: Role hierarchies;
  - $\mathcal{R}$: Complex role hierarchies;
  - $\mathcal{N}$: Cardinality restrictions;
  - $\mathcal{Q}$: Qualified cardinality restrictions;
  - $\mathcal{O}$: Closed classes;
  - $\mathcal{I}$: Inverse roles;
  - $\mathcal{D}$: Datatypes;
  - ...
- Often we shorten $\mathcal{ALC}^+$ ($\mathcal{ALC}$ extended with transitive roles) to just $\mathcal{S}$ for more advanced languages, so e.g. $\mathcal{SHOIN}$ is $\mathcal{ALC}^+ + \mathcal{H} + \mathcal{O} + \mathcal{I} + \mathcal{N}$.

$\mathcal{ALC}$ - Revision

Syntax	Semantics	Terminology
A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$	atomic concept
R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$	atomic role
$\top$	$\Delta^{\mathcal{I}}$	top (universal) concept
$\bot$	$\emptyset$	bottom concept
$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$	concept complement
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	concept conjunction
$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	concept disjunction
$\forall R.C$	$\{x \mid (\forall y)((x,y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}})\}$	universal restriction
$\exists R.C$	$\{x \mid (\exists y)((x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}})\}$	existential restriction

$\mathcal{ALC}$ - Examples

- $\texttt{Person} \sqcap \neg \texttt{Female}$
- $\texttt{Female} \sqcup \texttt{Male}$
- $\forall \texttt{hasChild.Person}$
- $\exists \texttt{hasChild.Person}$
- $\exists \texttt{hasChild.Person} \sqcap \forall \texttt{hasChild.Person}$
- $(\texttt{Female} \sqcap \forall \texttt{hasChild.Person})(\texttt{ANNA})$
- $\texttt{hasChild}(\texttt{BOB}, \texttt{ANNA})$

Typical Reasoning Tasks

- Concept satisfiability (i.e. $\mathcal{K} \not\models C \equiv \bot$ for some concept C)

Typical Reasoning Tasks

- Concept satisfiability (i.e. $\mathcal{K} \not\models C \equiv \bot$ for some concept C)
- Subsumption (i.e. $\mathcal{K} \models C \sqsubseteq D$, for some concepts C, D)

Typical Reasoning Tasks

- Concept satisfiability (i.e. $\mathcal{K} \not\models C \equiv \bot$ for some concept C)
- Subsumption (i.e. $\mathcal{K} \models C \sqsubseteq D$, for some concepts C, D)
- Classification (given some concept C, for all D in TBox, determine if $C \sqsubseteq D$)

Typical Reasoning Tasks

- Concept satisfiability (i.e. $\mathcal{K} \not\models C \equiv \bot$ for some concept C)
- Subsumption (i.e. $\mathcal{K} \models C \sqsubseteq D$, for some concepts C, D)
- Classification (given some concept C, for all D in TBox, determine if $C \sqsubseteq D$)
- Knowledge base satisfiability (i.e. determine if $\mathcal{K}$ has a model)

Typical Reasoning Tasks

- Concept satisfiability (i.e. $\mathcal{K} \not\models C \equiv \bot$ for some concept C)
- Subsumption (i.e. $\mathcal{K} \models C \sqsubseteq D$, for some concepts C, D)
- Classification (given some concept C, for all D in TBox, determine if $C \sqsubseteq D$)
- Knowledge base satisfiability (i.e. determine if $\mathcal{K}$ has a model)
- Instance checking (i.e. $\mathcal{K} \models C(a)$, for some concept C and an instance a)

Typical Reasoning Tasks

- Concept satisfiability (i.e. $\mathcal{K} \not\models C \equiv \bot$ for some concept C)
- Subsumption (i.e. $\mathcal{K} \models C \sqsubseteq D$, for some concepts C, D)
- Classification (given some concept C, for all D in TBox, determine if $C \sqsubseteq D$)
- Knowledge base satisfiability (i.e. determine if $\mathcal{K}$ has a model)
- Instance checking (i.e. $\mathcal{K} \models C(a)$, for some concept C and an instance a)
- Answering (DL-)queries (find all a s.t. $\{a \mid \mathcal{K} \models C(a)\}$)

Typical Reasoning Tasks

- Concept satisfiability (i.e. $\mathcal{K} \not\models C \equiv \bot$ for some concept C)
- Subsumption (i.e. $\mathcal{K} \models C \sqsubseteq D$, for some concepts C, D)
- Classification (given some concept C, for all D in TBox, determine if $C \sqsubseteq D$)
- Knowledge base satisfiability (i.e. determine if $\mathcal{K}$ has a model)
- Instance checking (i.e. $\mathcal{K} \models C(a)$, for some concept C and an instance a)
- Answering (DL-)queries (find all a s.t. $\{a \mid \mathcal{K} \models C(a)\}$)
- Realization (given an individual a, find the most specific concept C s.t. $\mathcal{K} \models C(a)$)

Reduction to Satisfiability

The reasoning problems can be solved by reducing them to the problem of knowledge base satisfiability:

- Concept Satisfiability

  $\mathcal{K} \not\models C \equiv \bot$ iff there exists an x such that $\mathcal{K} \cup \{C(x)\}$ is satisfiable

- Subsumption

  $\mathcal{K} \models C \sqsubseteq D$ iff there exists an x s.t. $\mathcal{K} \cup \{(C \sqcap \neg D)(x)\}$ is not satisfiable

- Instance Checking

  $\mathcal{K} \models C(a)$ iff $\mathcal{K} \cup \{\neg C(a)\}$ is not satisfiable

Reasoning Algorithms for DLs

- Terminating, complete and efficient algorithms for deciding **satisfiability** – and all the other reasoning problems we presented earlier – are available for various DLs.
- Most of these algorithms are based on **tableau proof techniques**.
- The tableau algorithm is a proof algorithm to check the consistency of a logical formula, by inferring that its negation is a contradiction (**proof by refutation**)

For a list of reasoners see also:
http://owl.cs.manchester.ac.uk/tools/list-of-reasoners/

Table of Contents

Tableau Proof Techniques

We will give a short introduction of tableau proof techniques for

- Propositional logic (PL)
- First-order logic (FOL)

before we move to the case of description logics.
What we want to demonstrate is that tableau techniques have been standard proof techniques in other logics before they were used by DL researchers. Regarding DLs, there are also close connections to tableau techniques for modal logics but we will not introduce them here in any detail.
In the literature, the term **semantic tableau** is also used.

Tableau Proof Techniques - PL

Tableau are **refutation systems** for PL (like **resolution**). To prove that a formula P is a **tautology** (or **valid**), we start with $\neg P$ and produce a **contradiction**.

The procedure for doing this involves **expanding** $\neg P$ so that inessential details of its logical structure are cleared away.

In tableau proofs, such an expansion takes the form of a **tree**, where nodes are labeled with formulas.

Each **branch** of this tree should be thought of as representing the **conjunction** of the formulas appearing on it, and the tree itself as representing the **disjunction** of its branches.

Uniform Notation for PL

Theorem. (Unique Parsing) Every propositional formula is in exactly one of the following categories:

1. atomic (propositional symbol, $\top$ or $\bot$).
2. $\neg X$, for a unique propositional formula X.
3. $(X \circ Y)$ for a unique binary symbol $\circ$ and unique propositional formulas X and Y.

Uniform Notation for PL (cont'd)

Based on the unique parsing theorem, we can group all propositional formulas of the forms $(X \circ Y)$ and $\neg(X \circ Y)$ into two categories, those that act **conjunctively**, which we call α-**formulas**, and those that act **disjunctively**, which we call β-**formulas**:

α	α_1	α_2
$X \wedge Y$	X	Y
$\neg(X \vee Y)$	$\neg X$	$\neg Y$
$\neg(X \supset Y)$	X	$\neg Y$

β	β_1	β_2
$\neg(X \wedge Y)$	$\neg X$	$\neg Y$
$X \vee Y$	X	Y
$X \supset Y$	$\neg X$	Y

Uniform notation allows us to have a large number of basic connectives, and still not do unnecessary work in proofs.

Tableau Expansion Rules

The following **tableau expansion rules** are used to manipulate trees (transform a tree into another) in tableau proofs:

$$\frac{\neg\neg P}{P} \qquad \frac{\neg\top}{\bot} \qquad \frac{\neg\bot}{\top} \qquad \frac{\alpha}{\begin{array}{l}\alpha_1\\ \alpha_2\end{array}} \qquad \frac{\beta}{\beta_1 \mid \beta_2}$$

How are these rules used?

Example

Let us assume that we want to show that the formula

$$(P \supset (Q \supset R)) \supset ((P \vee S) \supset ((Q \supset R) \vee S)).$$

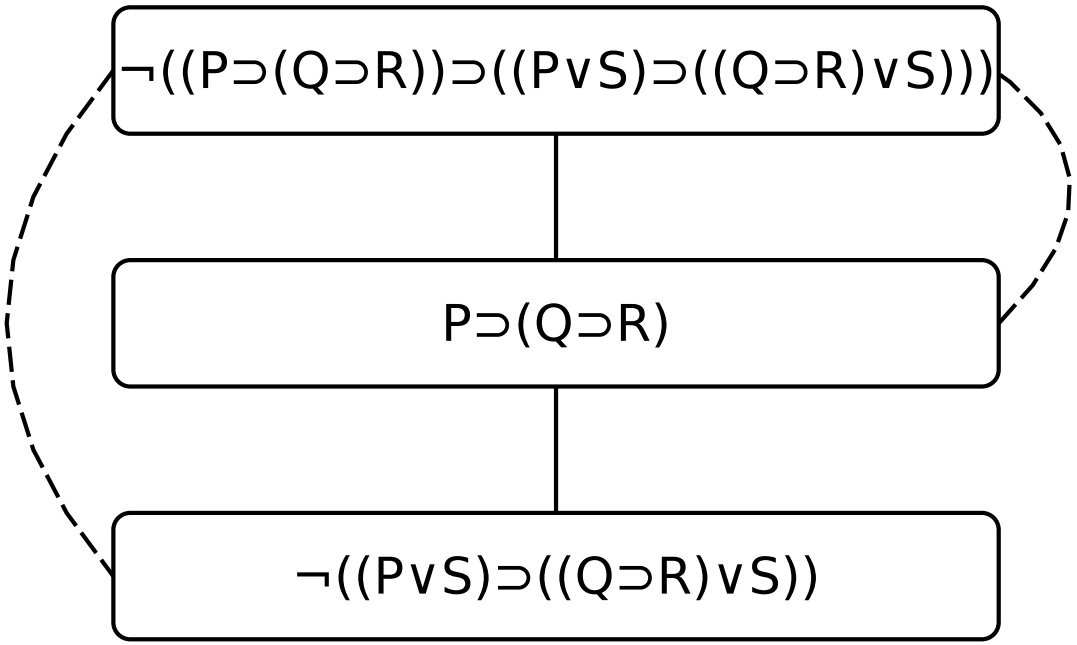
is a tautology. The following tree is a tableau proof of this formula. Notice that the proof starts with **the negation of the given formula** to be shown to be a tautology.

Example (cont'd)

$\neg((P \supset (Q \supset R)) \supset ((P \vee S) \supset ((Q \supset R) \vee S)))$

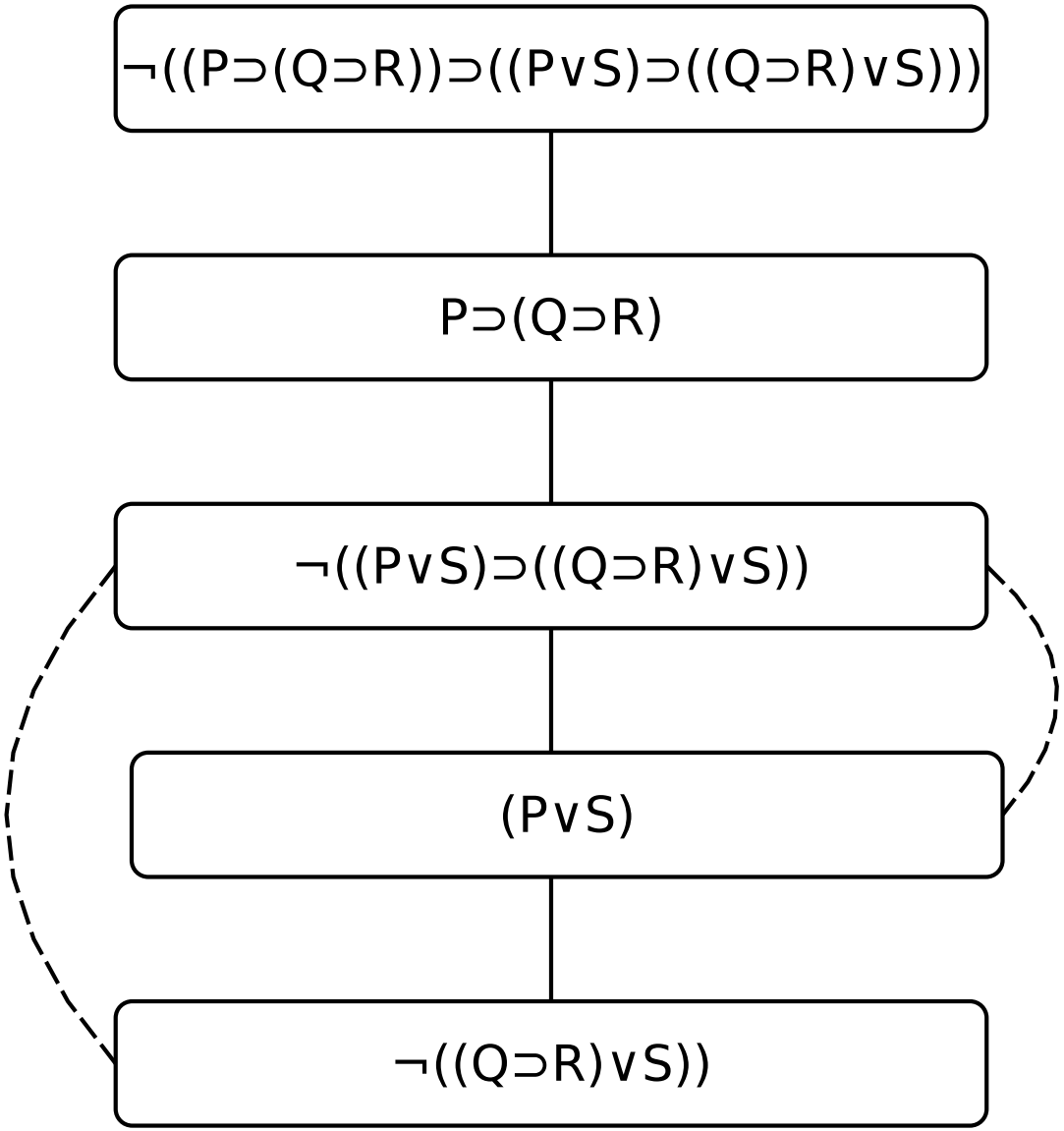
negation of formula

Example (cont'd)



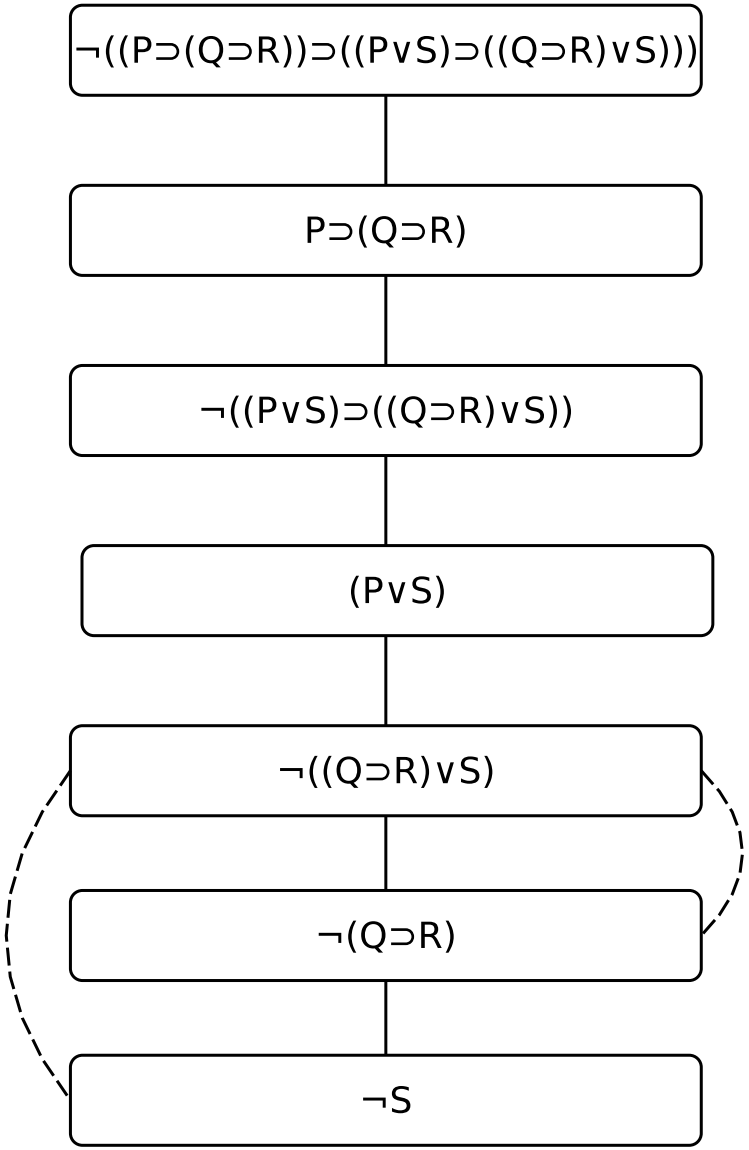
α	α1	α2
¬(X⊃Y)	X	¬Y

Example (cont'd)



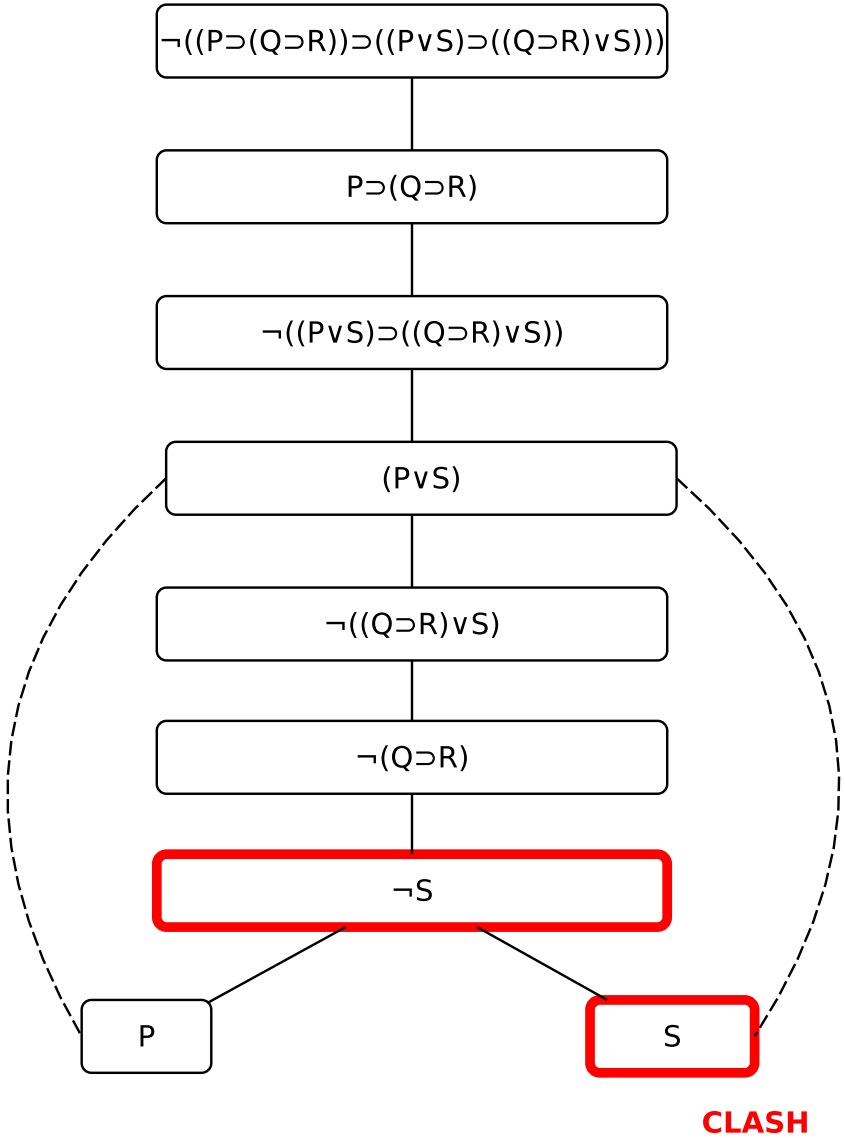
α	α1	α2
¬(X⊃Y)	X	¬Y

Example (cont'd)



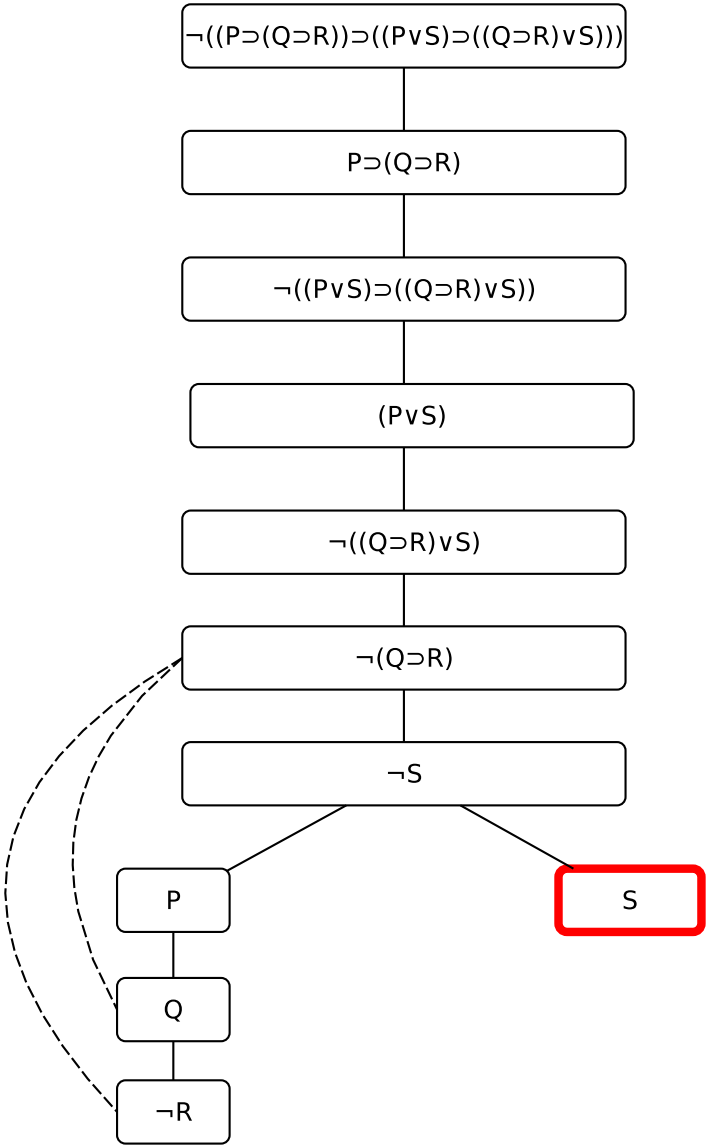
α	α1	α2
¬(X∨Y)	¬X	¬Y

Example (cont'd)



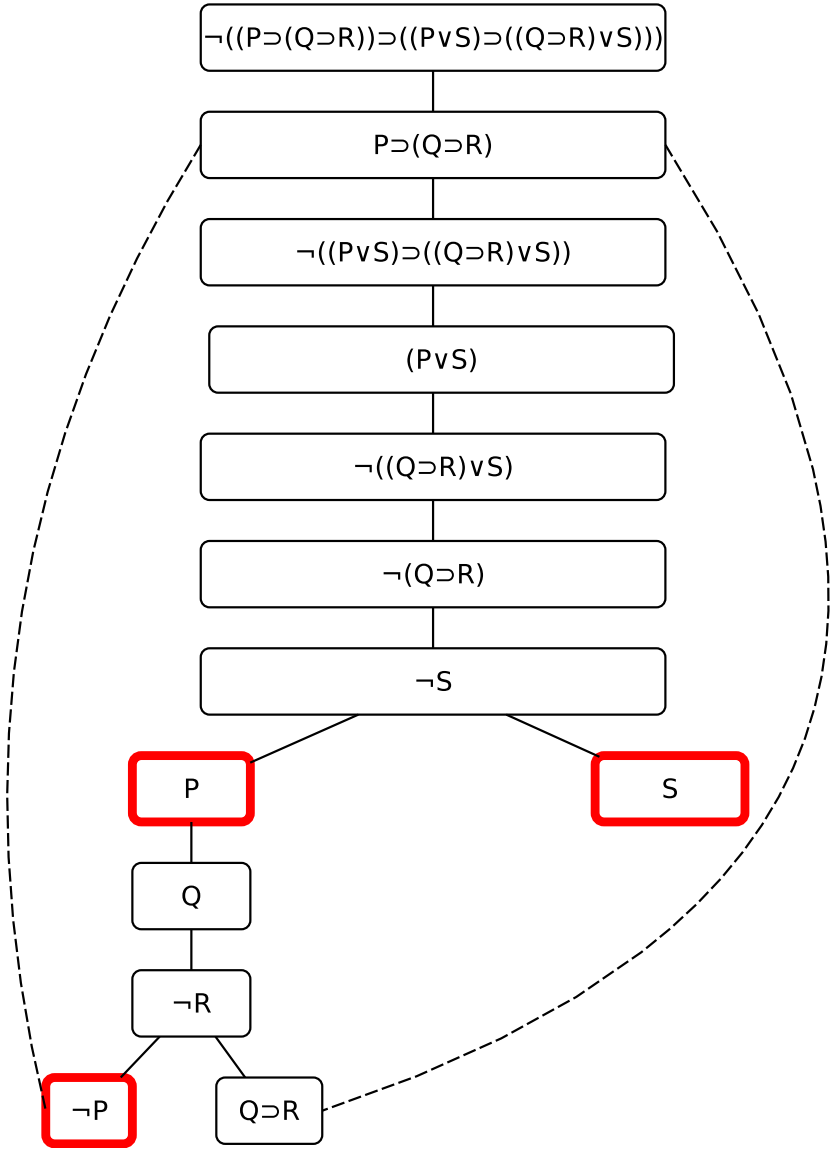
β	β1	β2
X∨Y	X	Y

Example (cont'd)



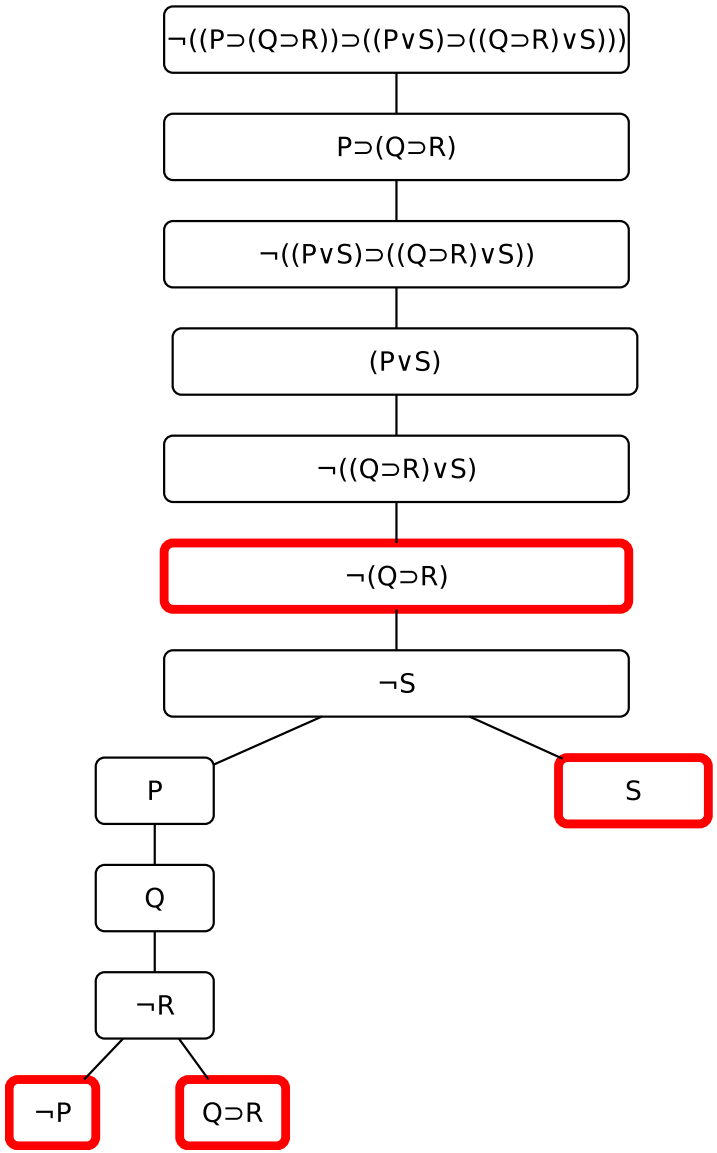
α	α1	α2
¬(X⊃Y)	X	¬Y

Example (cont'd)



β	β1	β2
X⊃Y	¬X	Y

Example (cont'd)



β	β1	β2
X⊃Y	¬X	Y

Closed Tableau

A branch θ of a tableau is called **closed** if both X and $\neg X$ occur on θ for some propositional formula X, or if $\perp$ occurs on θ.
If A and $\neg A$ occur on θ where A is atomic, or if $\perp$ occurs, θ is said to be **atomically closed**.
A tableau is **(atomically) closed** if every branch is (atomically) closed.
A **tableau proof** of X is a closed tableau for $\{\neg X\}$.

Soundness and Completeness

Theorem. (Soundness) If a sentence ϕ of PL has a tableaux proof then ϕ is a tautology.

Theorem. (Completeness) If a sentence ϕ of PL is a tautology then ϕ has a tableau proof.

Example (cont'd)

Notice that all branches of the tableau in this example are **closed**. Thus, the tableau is **closed** and the given formula

$$(P \supset (Q \supset R)) \supset ((P \vee S) \supset ((Q \supset R) \vee S))$$

is a **tautology**.

Uniform Notation for FOL

The uniform notation we introduced for PL can be extended to FOL. The additional machinery is that all **quantified** formulas and their negations are grouped into two categories, those that act **universally**, which are called γ-**formulas**, and those that act **existentially**, which are called δ-**formulas**. For each variety and for each term t, an **instance** $\gamma(t)$ or $\delta(t)$ is defined.

γ	$\gamma(t)$
$(\forall x)\Phi$	$\Phi\{x/t\}$
$\neg(\exists x)\Phi$	$\neg\Phi\{x/t\}$

δ	$\delta(t)$
$(\exists x)\Phi$	$\Phi\{x/t\}$
$\neg(\forall x)\Phi$	$\neg\Phi\{x/t\}$

Tableau Proofs for FOL

In informal proofs, **new constant symbols** are routinely used.
The formal counterpart is **parameters**, constant symbols not part of our original language.
In tableau proofs, we will use sentences of L^{par}, the extension of the given language L by the addition of a countable list of **new parameters**.

Tableau Expansion Rules for FOL

In the case of FOL, we have the PL **tableau expansion rules** plus the following two:

$$\frac{\gamma}{\gamma(t)}$$

(for any closed term t of L^{par})

$$\frac{\delta}{\delta(p)}$$

(for a **new** parameter p of L^{par})

Example

Let us assume that we want to prove that the FOL formula

$$(\forall x)(P(x) \vee Q(x)) \supset ((\exists x)P(x) \vee (\forall x)Q(x))$$

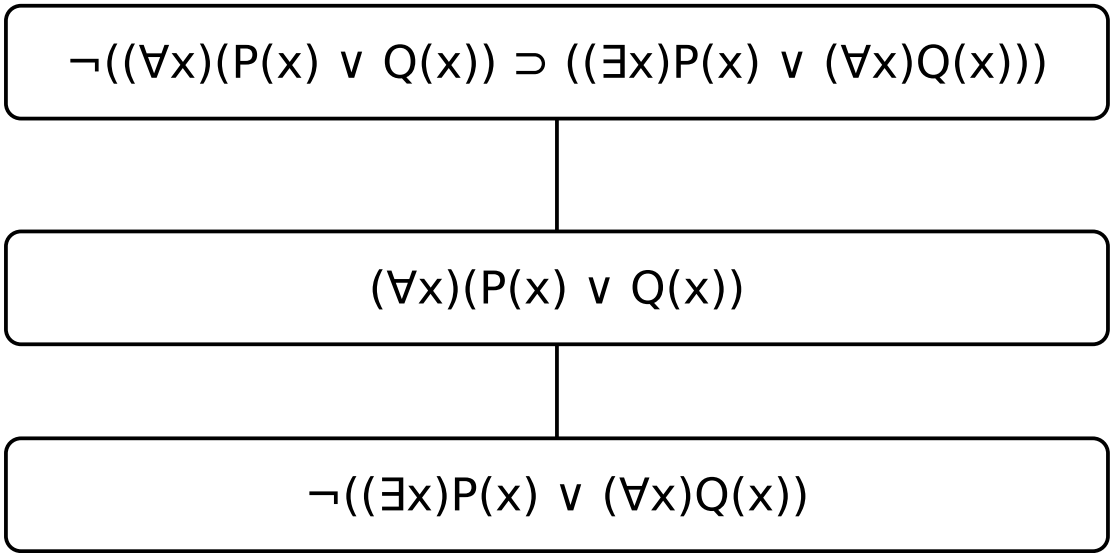
is valid. The following tree is a tableau proof of this formula. The resulting tableau is closed.
Notice that the proof starts with **the negation of the given formula** to be shown to be valid.

Example (cont'd)

$\neg((\forall x)(P(x) \vee Q(x)) \supset ((\exists x)P(x) \vee (\forall x)Q(x)))$

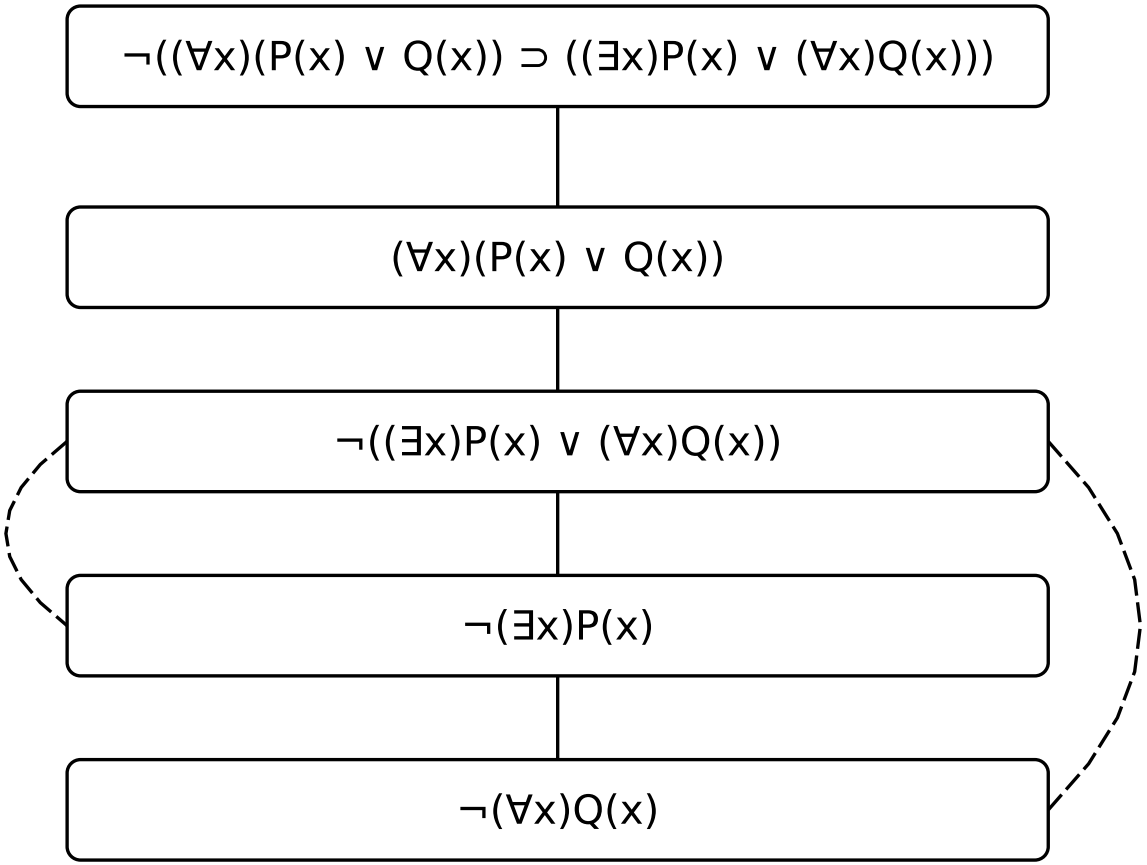
negation of formula

Example (cont'd)



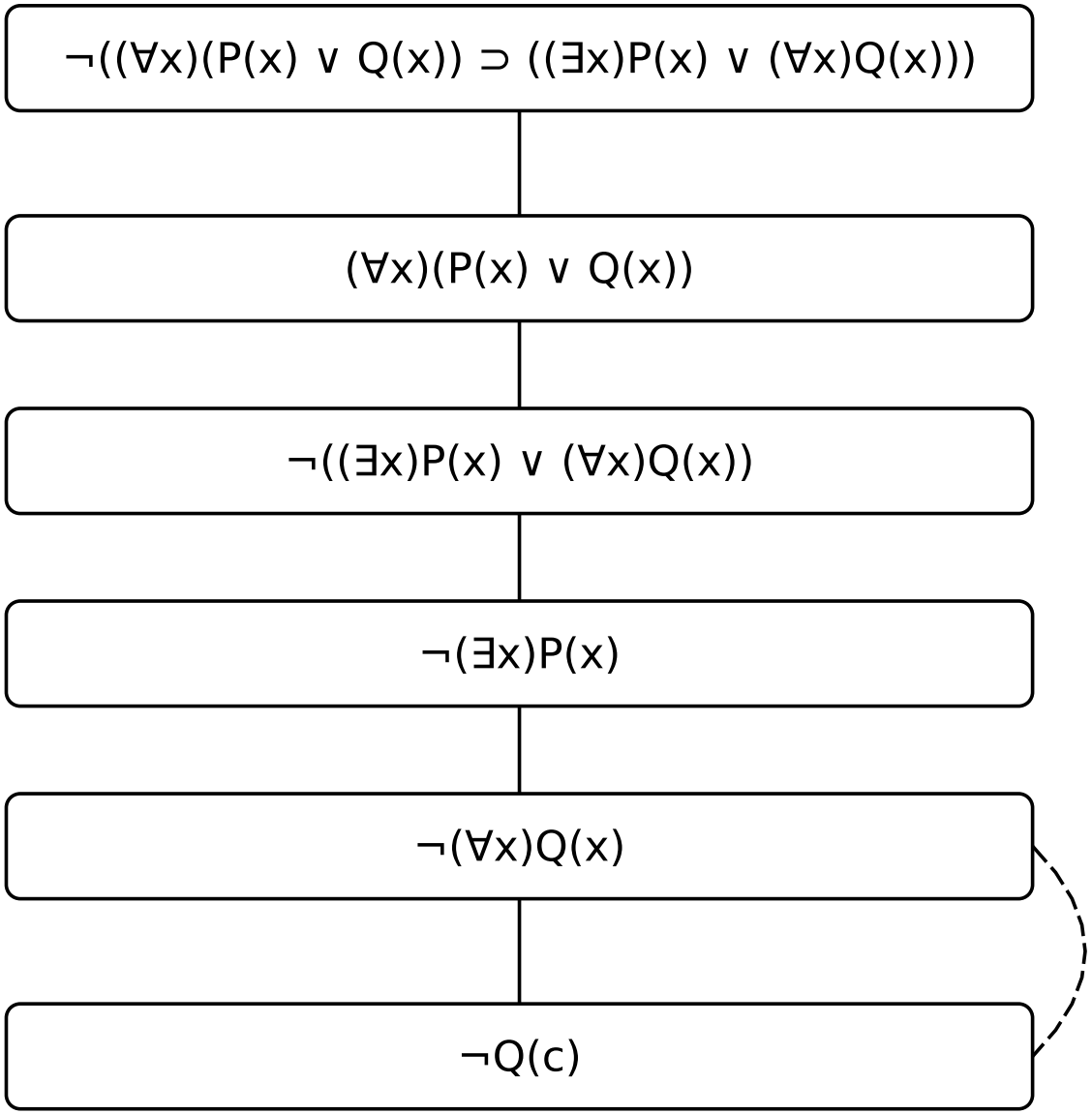
α	α1	α2
¬(X⊃Y)	X	¬Y

Example (cont'd)



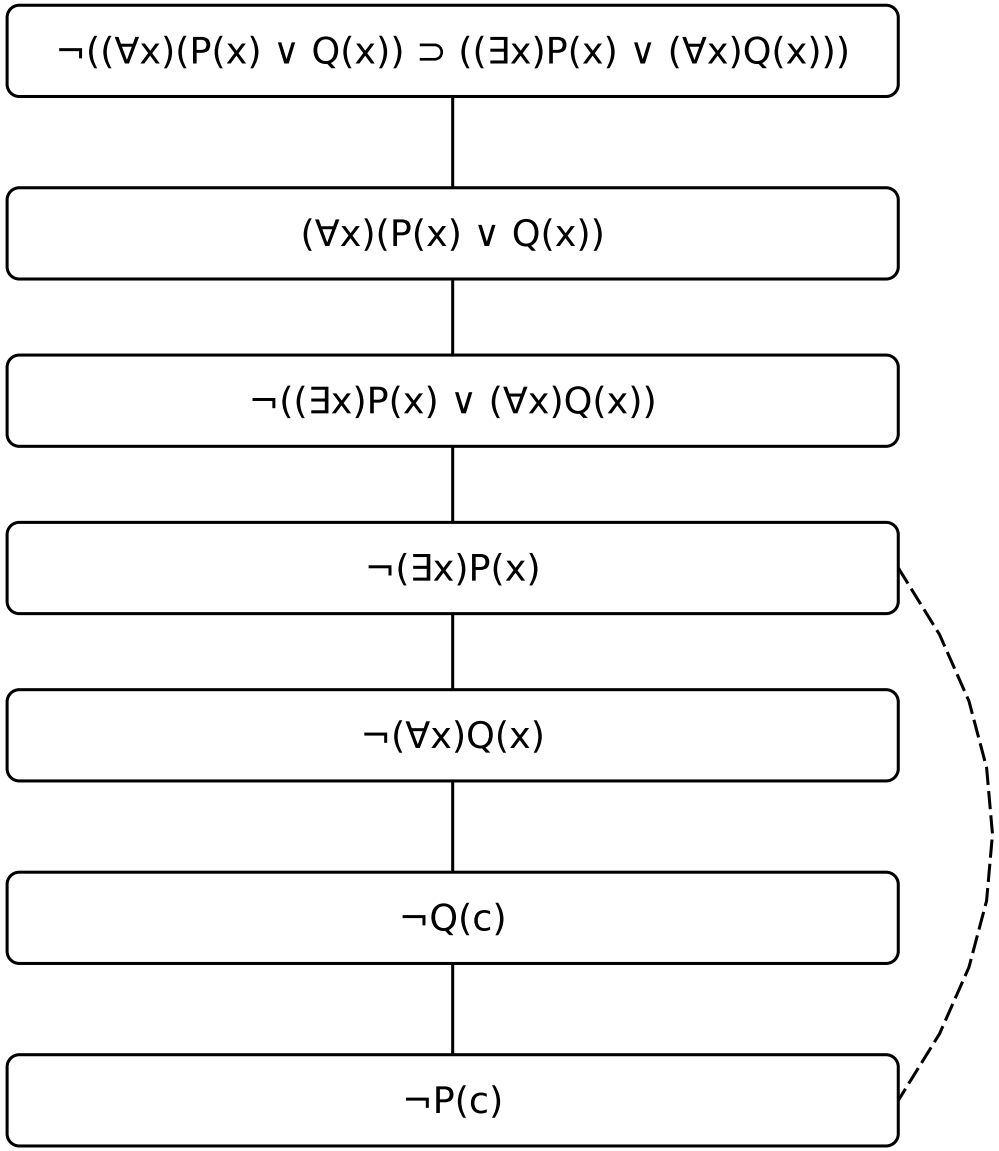
α	α1	α2
¬(X∨Y)	¬X	¬Y

Example (cont'd)



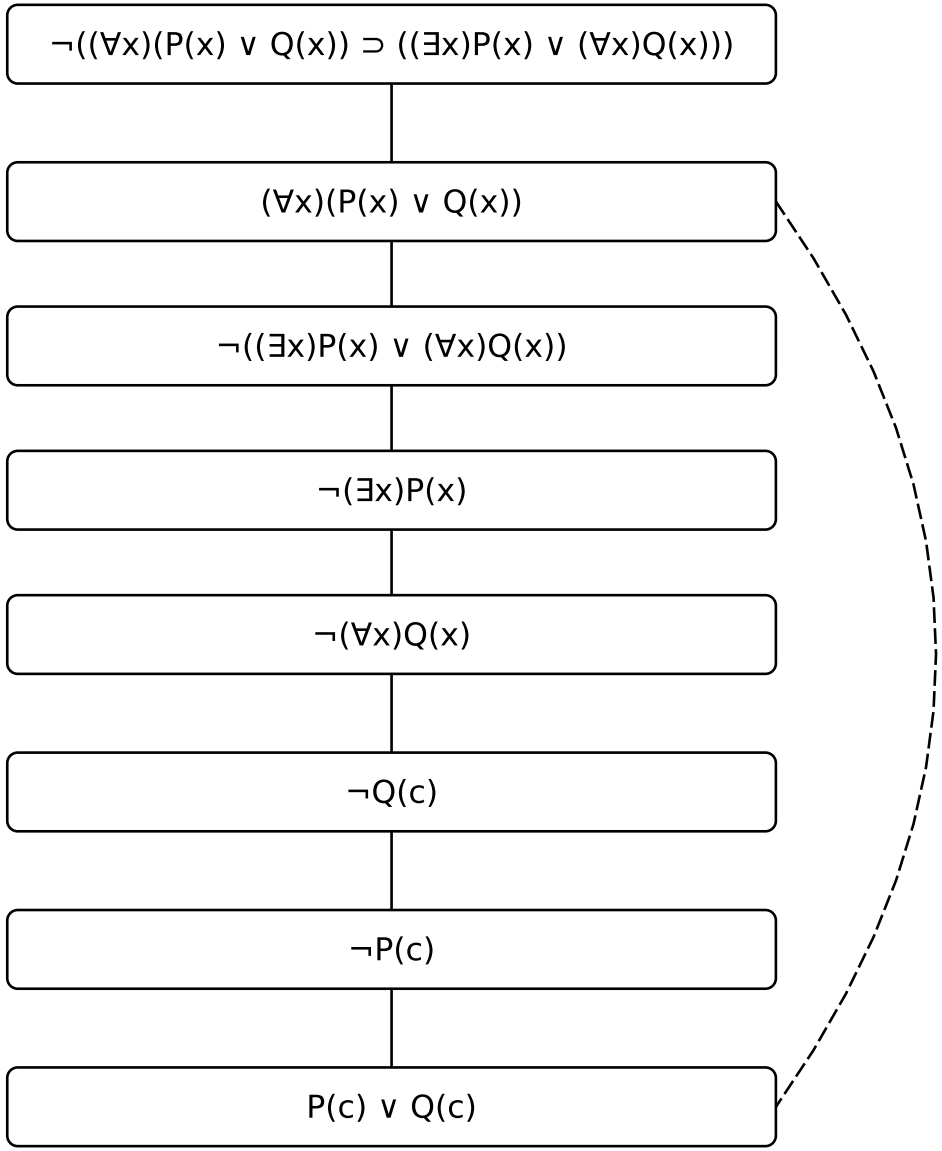
δ	δ(t)
¬(∀x)Φ	¬Φ{x/t}

Example (cont'd)



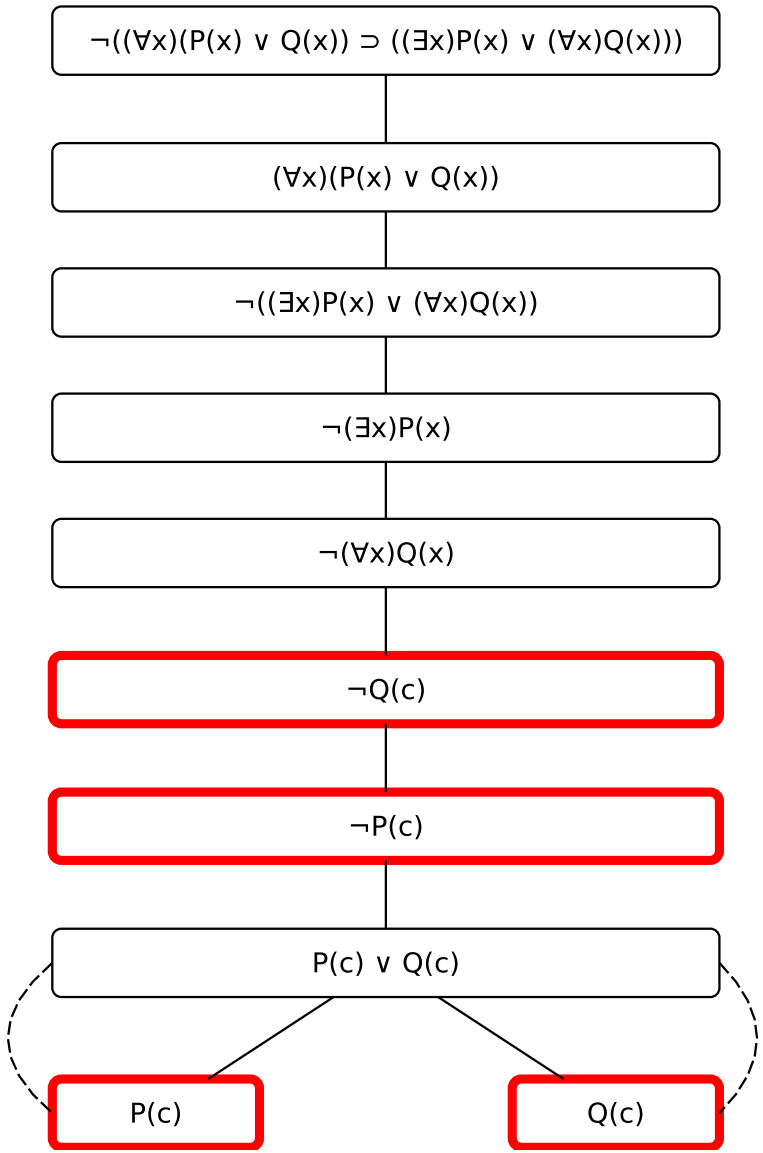
γ	γ(t)
¬(∃x)Φ	¬Φ{x/t}

Example (cont'd)



γ
γ(t) for any closed term t of Lpar)

Example (cont'd)



β	β1	β2
X∨Y	X	Y

Soundness and Completeness

Theorem. (Soundness) If a FOL sentence ϕ has a tableau proof then ϕ is valid.

Theorem. (Completeness) If a sentence ϕ of FOL is valid, then ϕ has a tableau proof.

Because FOL is not decidable, tableau proofs may not always terminate. The source of this difficulty is the γ rule.

Trivial example: Suppose we have a tableau branch containing both $(\exists x)\neg P(x)$ and $(\forall y)P(y)$. We might apply the δ-rule to the first formula, adding $\neg P(c)$, where c is a new parameter. But then using the γ-rule on the second, we might add one after the other $P(t_1), P(t_2), \ldots$ where $t_1, t_2, \ldots$ are all distinct closed terms different from c. In this way, we never produce the obvious closure.

Deciding Satisfiability in DL Using Tableau

Tableau proofs are decision procedures for solving the problem of satisfiability in a DL.
If a formula is **satisfiable**, the procedure will constructively exhibit a **model** of the formula.
The basic idea (as in PL and FOL) is to incrementally build such a model by looking at the formula and decomposing it in a top/down fashion. The procedure exhaustively looks at all the possibilities.
If a formula is **unsatisfiable**, the procedure can eventually prove that **no model** could be found.

Tableau Proofs for $\mathcal{ALC}$ Concept Satisfiability

Given an $\mathcal{ALC}$ concept C, the tableau algorithm for **concept satisfiability** tries to construct a finite interpretation $\mathcal{I}$ that satisfies C i.e., it contains an element a such that $a \in C^{\mathcal{I}}$.
We follow the paper by Baader and Sattler (2001) given in the readings, and use an ABox assertion $C(a)$ to encode this.
In some papers of the literature, a **constraint system** is used to implement the tableau (the two approaches are equivalent).

Tableau Proofs: the High-Level Algorithm

1. We start with the ABox assertion $C(a)$.
2. We add formulas to the tableau by applying certain **transformation rules**. Transformation rules are either **deterministic** or **nondeterministic** (result in **branches**).
3. We apply the transformation rules until either a **contradiction** is generated in **every branch**, or **there is a branch where no more rule is applicable**.
   In the former case C is unsatisfiable. In the latter case, C is satisfiable and this branch gives a **non-empty model** of C.

Negation Normal Form

For the tableau techniques to work, the formula in question has to be transformed into negation normal form.

Definition. A formula is in **negation normal form** if negation appears only in front of atomic concepts.

Negation Normal Form (cont'd)

Applying the following equivalences, we can transform any $\mathcal{ALC}$ formula into an equivalent one in negation normal form:

- $\neg\top \equiv \bot$
- $\neg\bot \equiv \top$
- $\neg\neg C \equiv C$
- $\neg(C \sqcap D) \equiv \neg C \sqcup \neg D$
- $\neg(C \sqcup D) \equiv \neg C \sqcap \neg D$
- $\neg(\forall R.C) \equiv \exists R.\neg C$
- $\neg(\exists R.C) \equiv \forall R.\neg C$

Transformation Rules: the AND rule

The transformation rules come straightforwardly from the semantics of constructors.

If in an **arbitrary interpretation** $\mathcal{I}$, whose domain contains an **arbitrary** element a, we have that $a \in (C \sqcap D)^{\mathcal{I}}$, then from the semantics we know that a should be in the intersection of $C^{\mathcal{I}}$ and $D^{\mathcal{I}}$, i.e. it should be in both $C^{\mathcal{I}}$ and $D^{\mathcal{I}}$.

We can use **ABox assertions** to encode this in a transformation rule as follows.

If

- $(C \sqcap D)(a)$ is in $\mathcal{A}$, but
- $C(a)$ and $D(a)$ are not both in $\mathcal{A}$

then

$$\mathcal{A} := \mathcal{A} \cup \{C(a), D(a)\}$$

The OR Rule (or $\sqcup$-rule)

Similarly, we have the following rule.

If

- $(C \sqcup D)(a)$ is in $\mathcal{A}$, but
- neither $C(a)$ nor $D(a)$ is in $\mathcal{A}$

then

$$\mathcal{A} := \mathcal{A} \cup \{C(a)\}$$

or

$$\mathcal{A} := \mathcal{A} \cup \{D(a)\}$$

This rule forces us to introduce **sets of ABoxes** as a formal tool to represent tableau proofs.

The SOME rule (or $\exists$-rule)

From the semantics, we have the following. If in an arbitrary interpretation $\mathcal{I}$, whose domain contains an arbitrary element a, we have that $a \in (\exists R.C)^{\mathcal{I}}$, then there must be an element b (not necessarily distinct from a) such that $(a, b) \in R^{\mathcal{I}}$ and $b \in C^{\mathcal{I}}$.
We can use **ABox assertions** to encode this in a transformation rule as follows. **If**

- $(\exists R.C)(a)$ is in $\mathcal{A}$ and
- there is no individual c such that both $R(a, c)$ and $C(c)$ are in $\mathcal{A}$

then

$$\mathcal{A} := \mathcal{A} \cup \{R(a, b), C(b)\}$$

where b is a **new individual** not occurring in $\mathcal{A}$.

The FORALL Rule (or $\forall$-rule)

Similarly, we have the following rule.

If

- $(\forall R.C)(a)$ is in $\mathcal{A}$
- $R(a, b)$ is in $\mathcal{A}$, and
- $C(b)$ is not in $\mathcal{A}$

then

$$\mathcal{A} := \mathcal{A} \cup \{C(b)\}$$

Some Definitions

Definition. An ABox is called **complete** if none of the above transformation rules applies to it.

While building a tableau proof, we can look for evident **contradictions** to see if the tableau is not satisfiable. We call these contradictions **clashes**.

Definition. An ABox $\mathcal{A}$ contains a **clash** if

- $\{\bot(a)\} \subseteq \mathcal{A}$, or
- $\{C(a), (\neg C)(a)\} \subseteq \mathcal{A}$

for some individual a and concept C.

Definition. An ABox is called **closed** if it contains a clash, and **open** otherwise.

Note: ABoxes correspond to **branches** in a tableau so the definitions can be given for tableaux too.

Tableau Proofs: the Algorithm Revisited

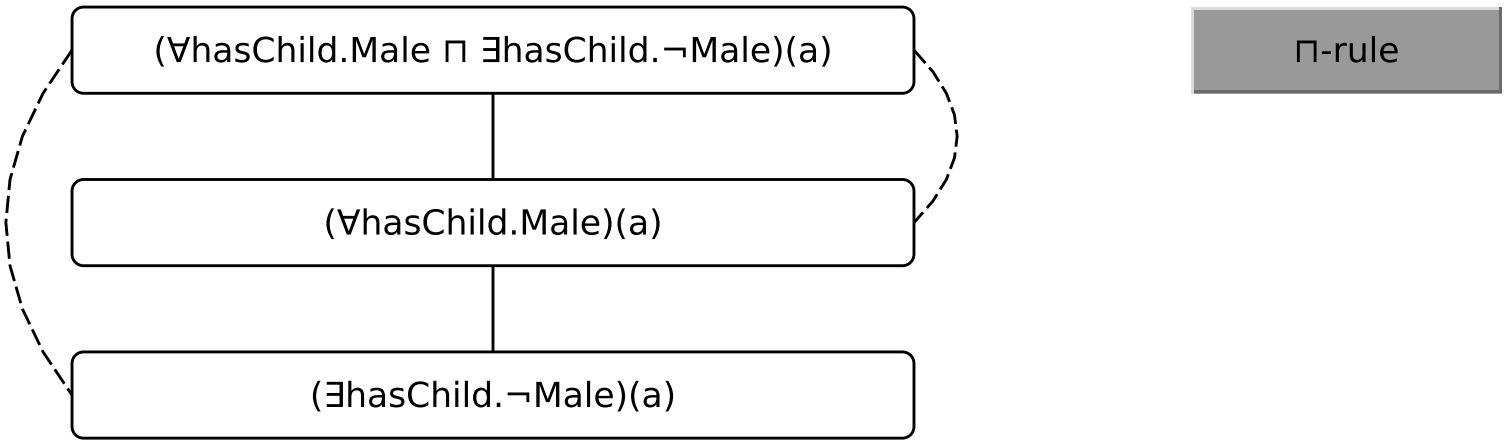
1. We start with the ABox assertion $C(a)$.
2. We add formulas to the tableau by applying the previous rules.
3. We apply the rules until either a **contradiction** is generated in **every branch** (all branches are closed ABoxes), or **there is a branch where no contradiction appears and no rule is applicable** (this branch is an open and complete ABox).
   In the former case C is **unsatisfiable**. In the latter case, C is **satisfiable** and this branch gives a **non-empty model** of C.

Examples

Check satisfiability of the concept

$$\forall \texttt{hasChild.Male} \sqcap \exists \texttt{hasChild.} \neg \texttt{Male}$$

The tableau method will proceed as follows:

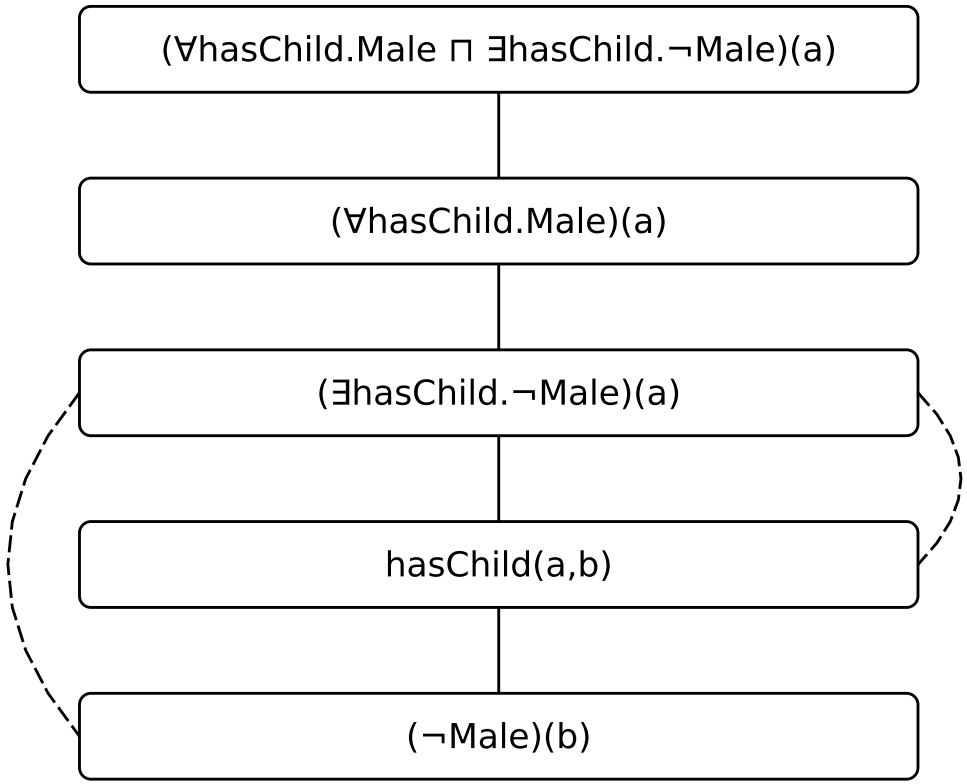


Examples

Check satisfiability of the concept

$$\forall \texttt{hasChild.Male} \sqcap \exists \texttt{hasChild.}\neg\texttt{Male}$$

The tableau method will proceed as follows:

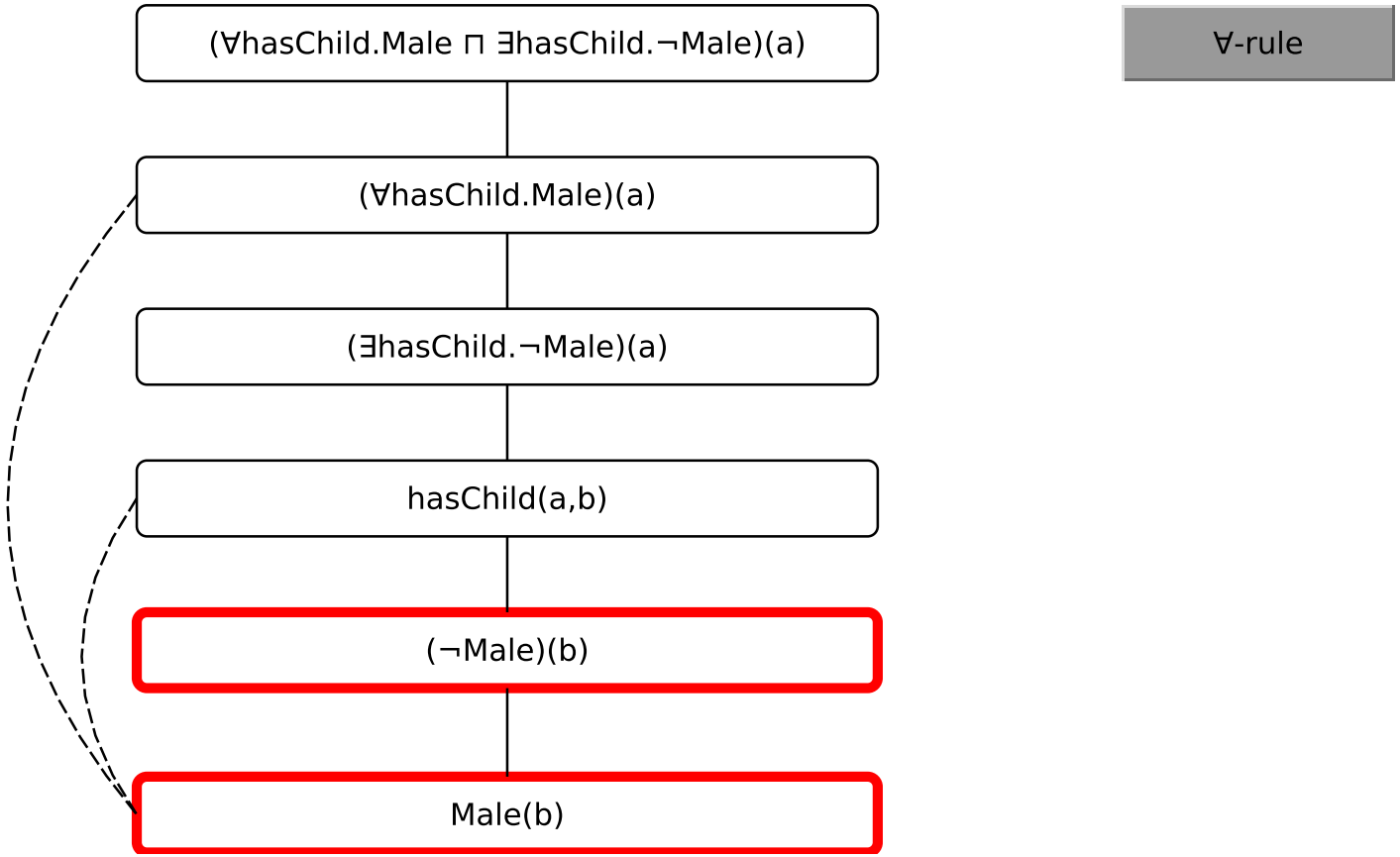


Examples

Check satisfiability of the concept

$$\forall \texttt{hasChild.Male} \sqcap \exists \texttt{hasChild.}\neg\texttt{Male}$$

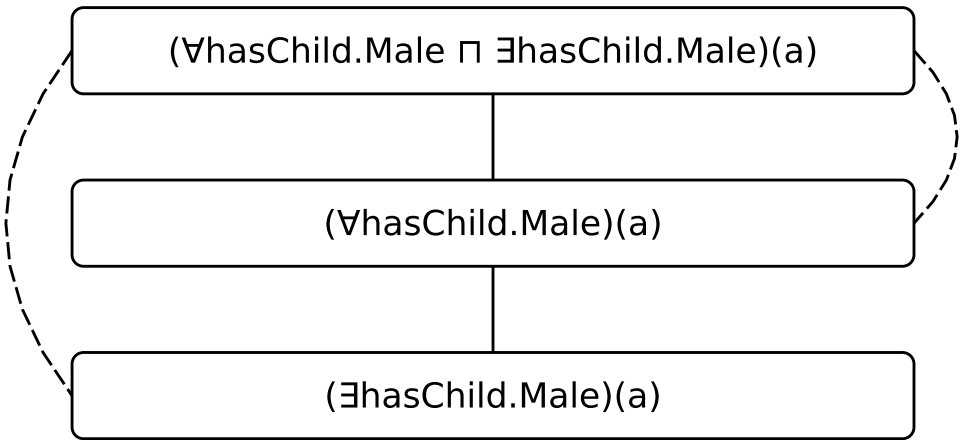
The tableau method will proceed as follows:



Examples (cont'd)

Check satisfiability of the concept:

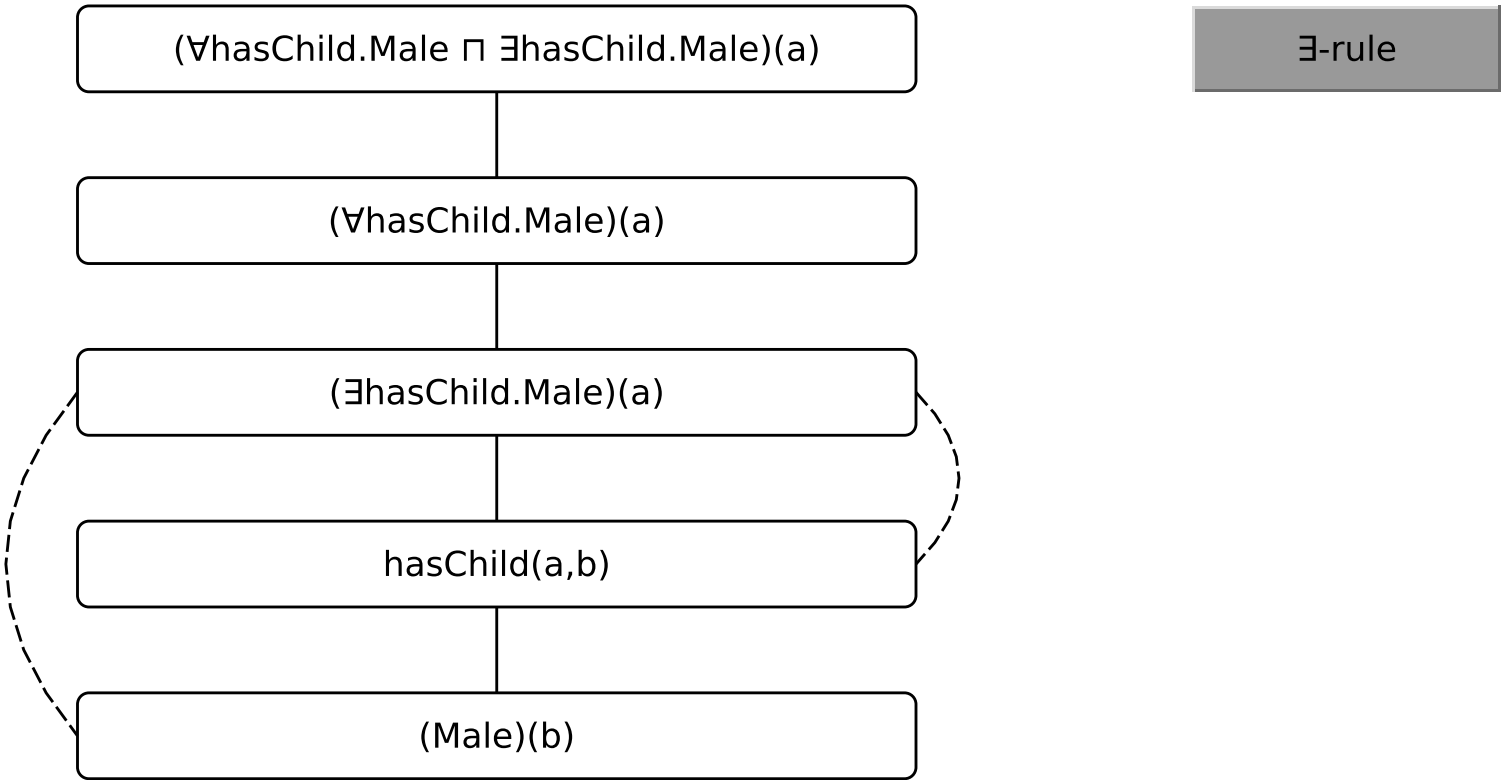
$$\forall \texttt{hasChild.Male} \sqcap \exists \texttt{hasChild.Male}$$



Examples (cont'd)

Check satisfiability of the concept:

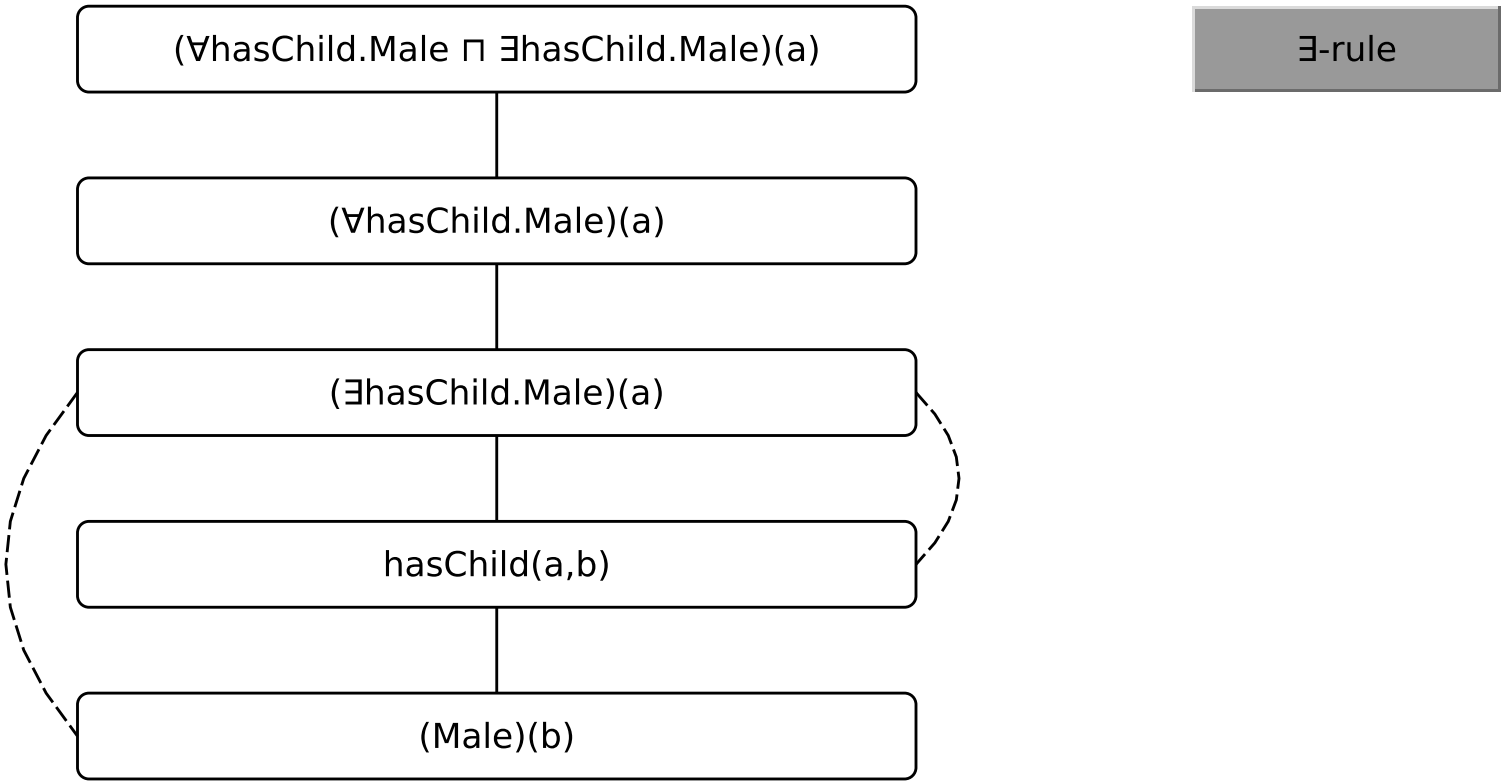
$$\forall \texttt{hasChild.Male} \sqcap \exists \texttt{hasChild.Male}$$



Examples (cont'd)

Check satisfiability of the concept:

$$\forall \texttt{hasChild.Male} \sqcap \exists \texttt{hasChild.Male}$$



The above tableau with one branch (ABox) is complete and open, thus the given formula is **satisfiable.**

Satisfiability of ABoxes

Naturally, we can also check the satisfiability of ABoxes using tableau techniques.

Example: Consider the ABox consisting of the following formulas:

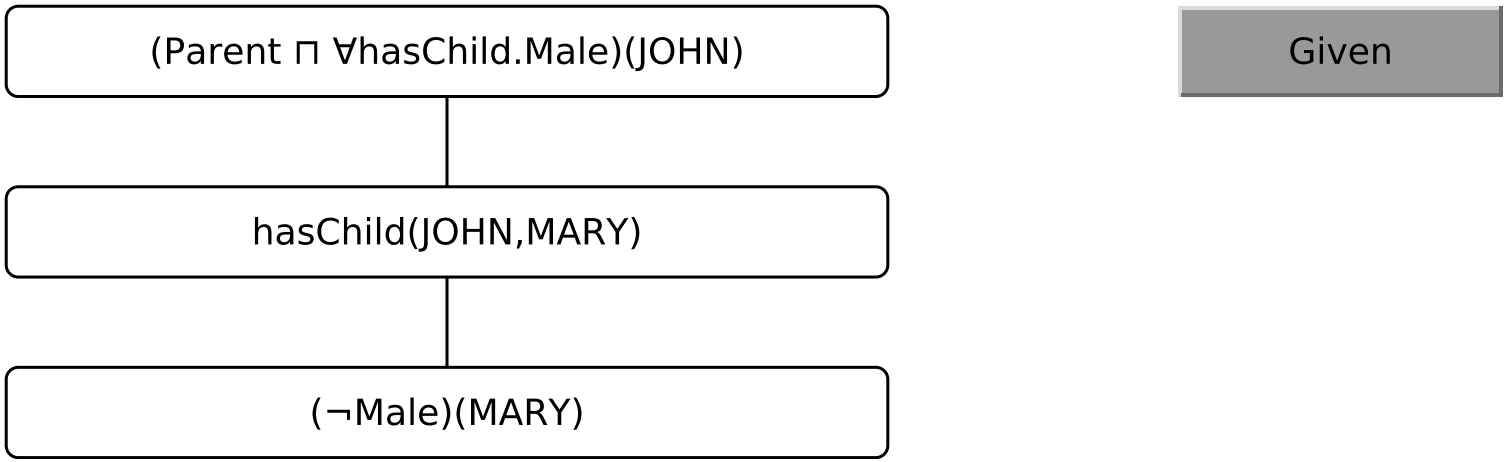
$$(\texttt{Parent} \sqcap \forall \texttt{hasChild}.\texttt{Male})(\texttt{JOHN})$$

$$\neg \texttt{Male}(\texttt{MARY})$$

$$\texttt{hasChild}(\texttt{JOHN}, \texttt{MARY})$$

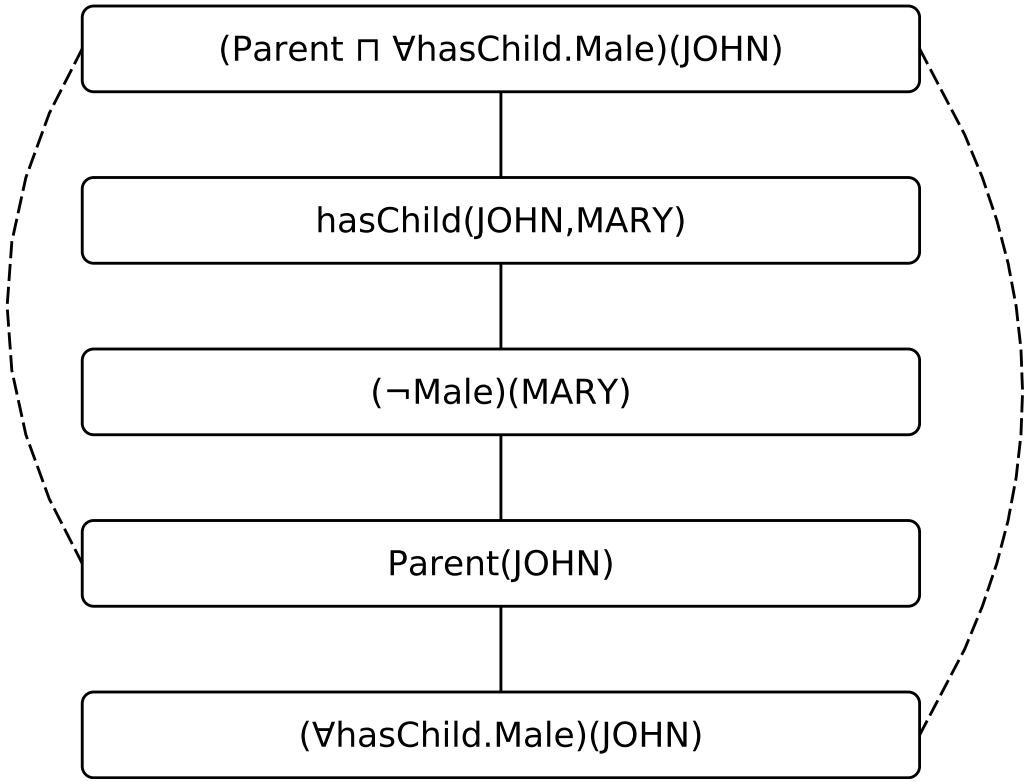
Satisfiability of ABoxes (cont'd)

The tableau technique in this case gives us:



Satisfiability of ABoxes (cont'd)

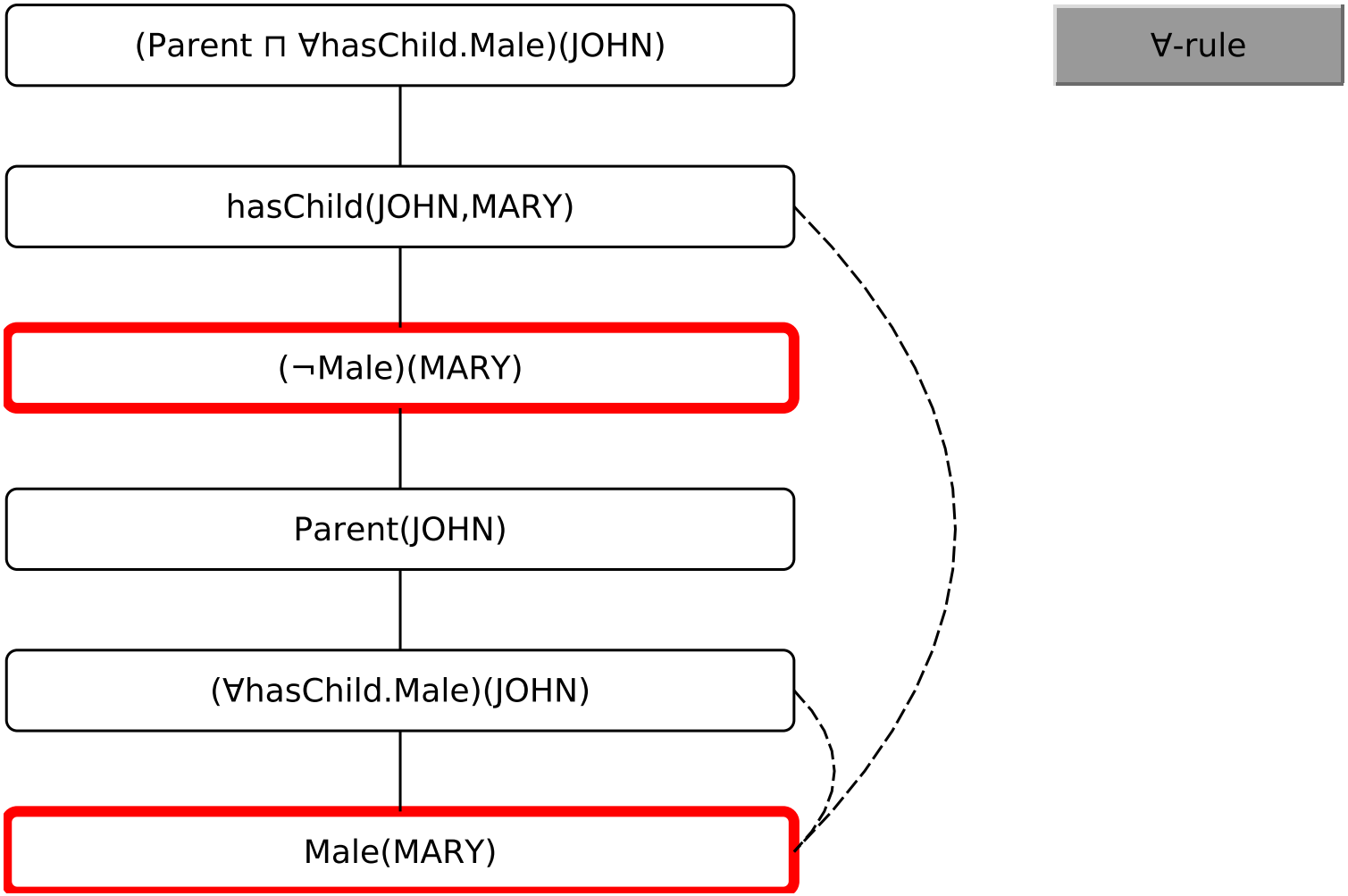
The tableau technique in this case gives us:



⊓-rule

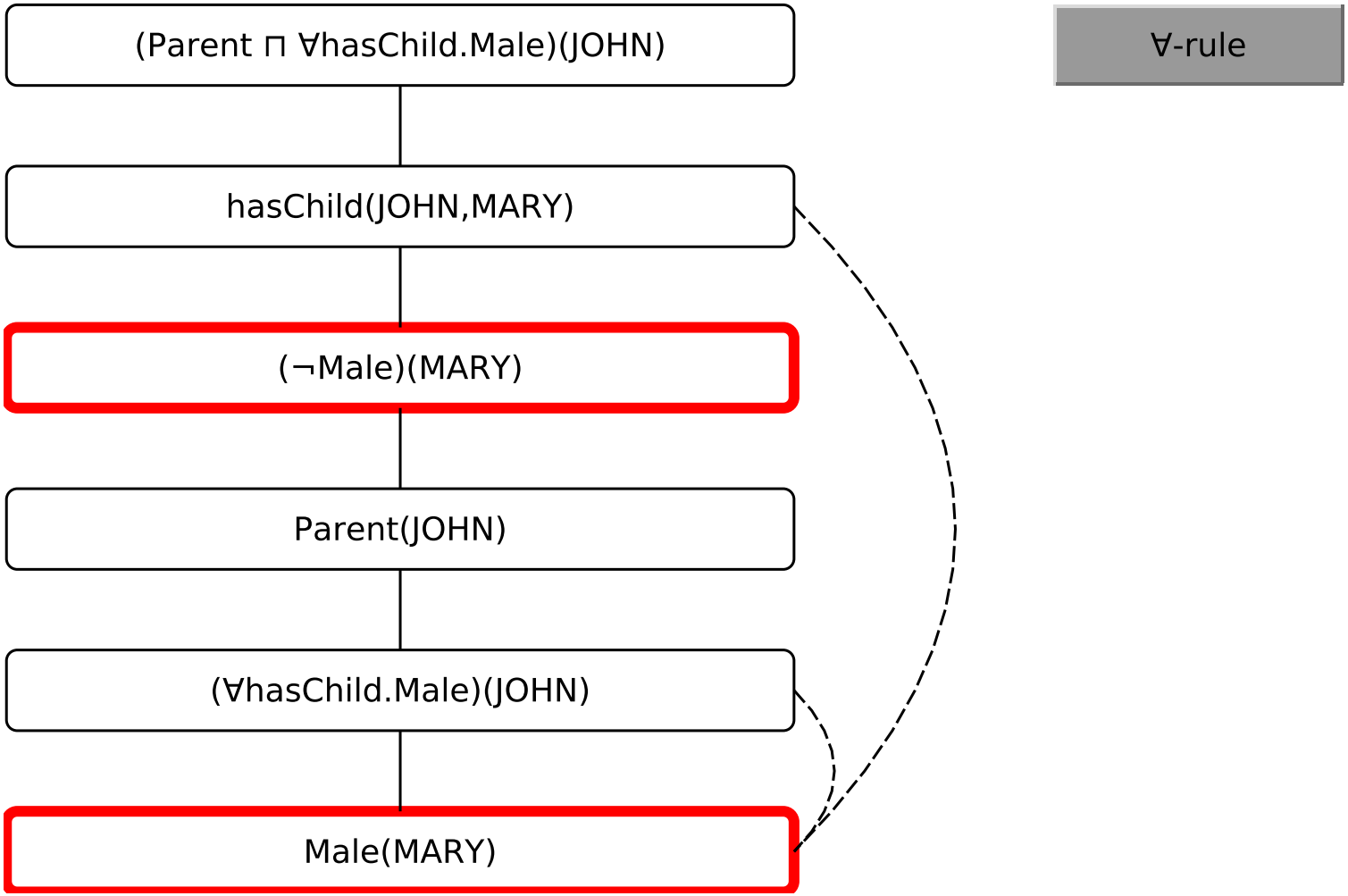
Satisfiability of ABoxes (cont'd)

The tableau technique in this case gives us:



Satisfiability of ABoxes (cont'd)

The tableau technique in this case gives us:



Thus the ABox is **unsatisfiable.**

Soundness of the Tableau Method for $\mathcal{ALC}$

The tableau method does not add unnecessary contradictions. Deterministic rules always **preserve the satisfiability** of any ABox involved in the proof, and nondeterministic rules allow always a choice of application that preserves satisfiability.

Termination of the Tableau Method for $\mathcal{ALC}$

Termination can be proved by using the following arguments:

- All rules but $\forall$ are never applied twice on the same ABox assertion.

Termination of the Tableau Method for $\mathcal{ALC}$

Termination can be proved by using the following arguments:

- All rules but $\forall$ are never applied twice on the same ABox assertion.
- The $\forall$-rule is never applied to an individual a more times than the number of the **direct successors** of a, which is bounded by the length of a concept (for a definition of the concept of direct successors see the formal proof).

Termination of the Tableau Method for $\mathcal{ALC}$

Termination can be proved by using the following arguments:

- All rules but $\forall$ are never applied twice on the same ABox assertion.
- The $\forall$-rule is never applied to an individual a more times than the number of the **direct successors** of a, which is bounded by the length of a concept (for a definition of the concept of direct successors see the formal proof).
- Finally, each rule application to a constraint $C(a)$ adds constraints $D(b)$ such that D is a strict subexpression of C.

Completeness of the Tableau Method for $\mathcal{ALC}$

If $\mathcal{A}$ is a **complete** and **open** ABox (i.e., a branch) in a tableau proof of $C(a)$ then $\mathcal{A}$ is **satisfiable**.
The following is a **canonical interpretation** $\mathcal{I}$ of $\mathcal{A}$ that can be obtained from the tableau:

- The domain $\Delta^{\mathcal{I}}$ of $\mathcal{I}$ consists of the individuals occurring in $\mathcal{A}$.
- For each atomic concept P, we define $P^{\mathcal{I}}$ to be $\{x|\ P(x) \in \mathcal{A}\}$.
- For each atomic role R, we define $R^{\mathcal{I}}$ to be $\{(x, y)|\ R(x, y) \in \mathcal{A}\}$.

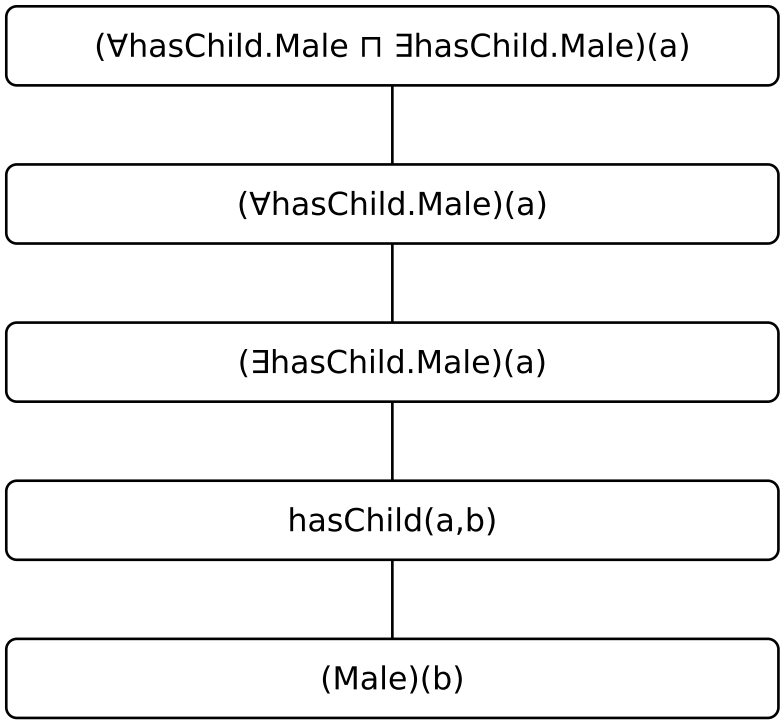
Using this interpretation, it is possible to construct an interpretation for C such that $C^{\mathcal{I}}$ is nonempty. In other words, C is satisfiable.

Example

We have shown that the concept:

$$\forall \texttt{hasChild.Male} \sqcap \exists \texttt{hasChild.Male}$$

is satisfiable:



Example (cont'd)

A model $\mathcal{I}$ of the Abox has domain $\Delta^{\mathcal{I}} = \{\mathtt{a}, \mathtt{b}\}$ and

$$\mathtt{Male}^{\mathcal{I}} = \{\mathtt{b}\}, \quad \mathtt{hasChild}^{\mathcal{I}} = \{(\mathtt{a}, \mathtt{b})\}.$$

The concept

$$\forall \mathtt{hasChild}.\mathtt{Male} \sqcap \exists \mathtt{hasChild}.\mathtt{Male}$$

is satisfiable because

$$(\forall \mathtt{hasChild}.\mathtt{Male} \sqcap \exists \mathtt{hasChild}.\mathtt{Male})^{\mathcal{I}} = \{\mathtt{a}\} \neq \emptyset$$

Tableau Techniques for TBoxes

So far we have used tableau techniques to deal with **concept satisfiability** and **ABox satisfiability**.
The literature gives us tableaux techniques for dealing with TBoxes as well. So eventually, we can use tableaus to decide whether a knowledge base is satisfiable or not.
The easiest case for TBoxes is when we have **acyclic terminologies.**

Acyclic Terminologies

A TBox is called an **acyclic terminology** if it is a set of concept definitions that do not contain multiple or cyclic definitions.
Multiple definitions are terminological axioms of the form

$$A \equiv B_1, \ldots, A \equiv B_n$$

for distinct concept expressions $B_1, \ldots, B_n$.
Cyclic definitions are terminological axioms of the form

$$A_1 \equiv C_1, \ldots, A_n \equiv C_n$$

where A_i occurs in C_{i-1} $(1 < i \leq n)$ and A_1 occurs in C_n.

Acyclic Terminologies (cont'd)

If the acyclic terminology $\mathcal{T}$ contains a concept definition $A \equiv C$ then A is called its **defined name** and C its **defining concept**.

Reasoning with acyclic terminologies can be reduced to reasoning without TBoxes by **unfolding the definitions**: this is achieved by repeatedly replacing defined names by their defining concepts until no more defined names exist.
Unfolding might lead to an **exponential blow-up** in the size of the produced ABox.

Example

TBox:

$$\texttt{MixedTeam} \equiv \texttt{Team} \sqcap \exists \texttt{hasMember.Male} \sqcap \exists \texttt{hasMember.Female}$$

$$\texttt{Male} \equiv \neg \texttt{Female}$$

ABox:

$$\texttt{MixedTeam}(\texttt{FC})$$

$$(\forall \texttt{hasMember.Male})(\texttt{FC})$$

The above knowledge base is unsatisfiable. How can we prove it using tableau?

Example (cont'd)

After unfolding the definition of $\texttt{MixedTeam}$, we have:

$$(\texttt{Team} \sqcap \exists \texttt{hasMember}.\texttt{Male} \sqcap \exists \texttt{hasMember}.\texttt{Female})(\texttt{FC})$$

$$(\forall \texttt{hasMember}.\texttt{Male})(\texttt{FC})$$

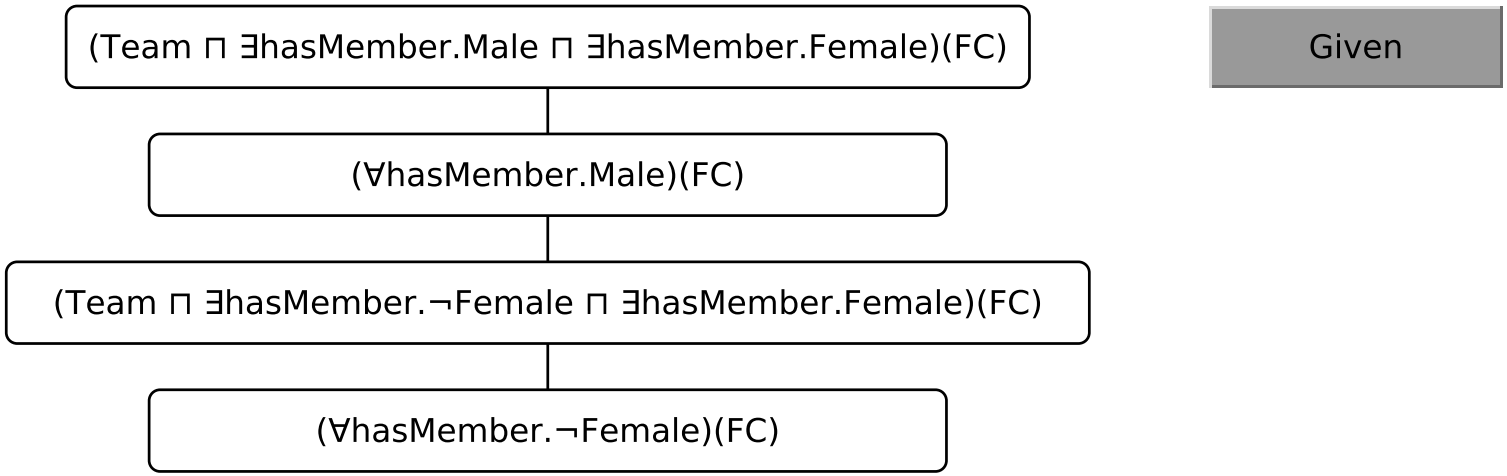
After unfolding the definition of $\texttt{Male}$, we have:

$$(\texttt{Team} \sqcap \exists \texttt{hasMember}.\neg\texttt{Female} \sqcap \exists \texttt{hasMember}.\texttt{Female})(\texttt{FC})$$

$$(\forall \texttt{hasMember}.\neg\texttt{Female})(\texttt{FC})$$

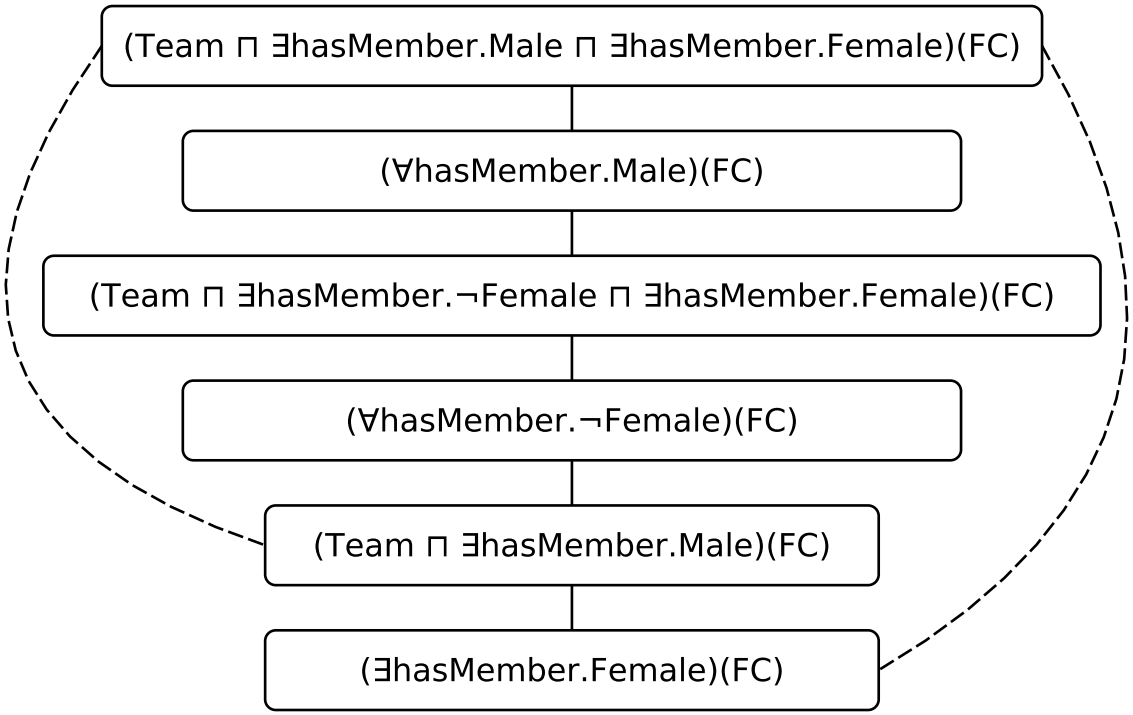
Example (cont'd)

The closed tableau showing unsatisfiability is as follows:



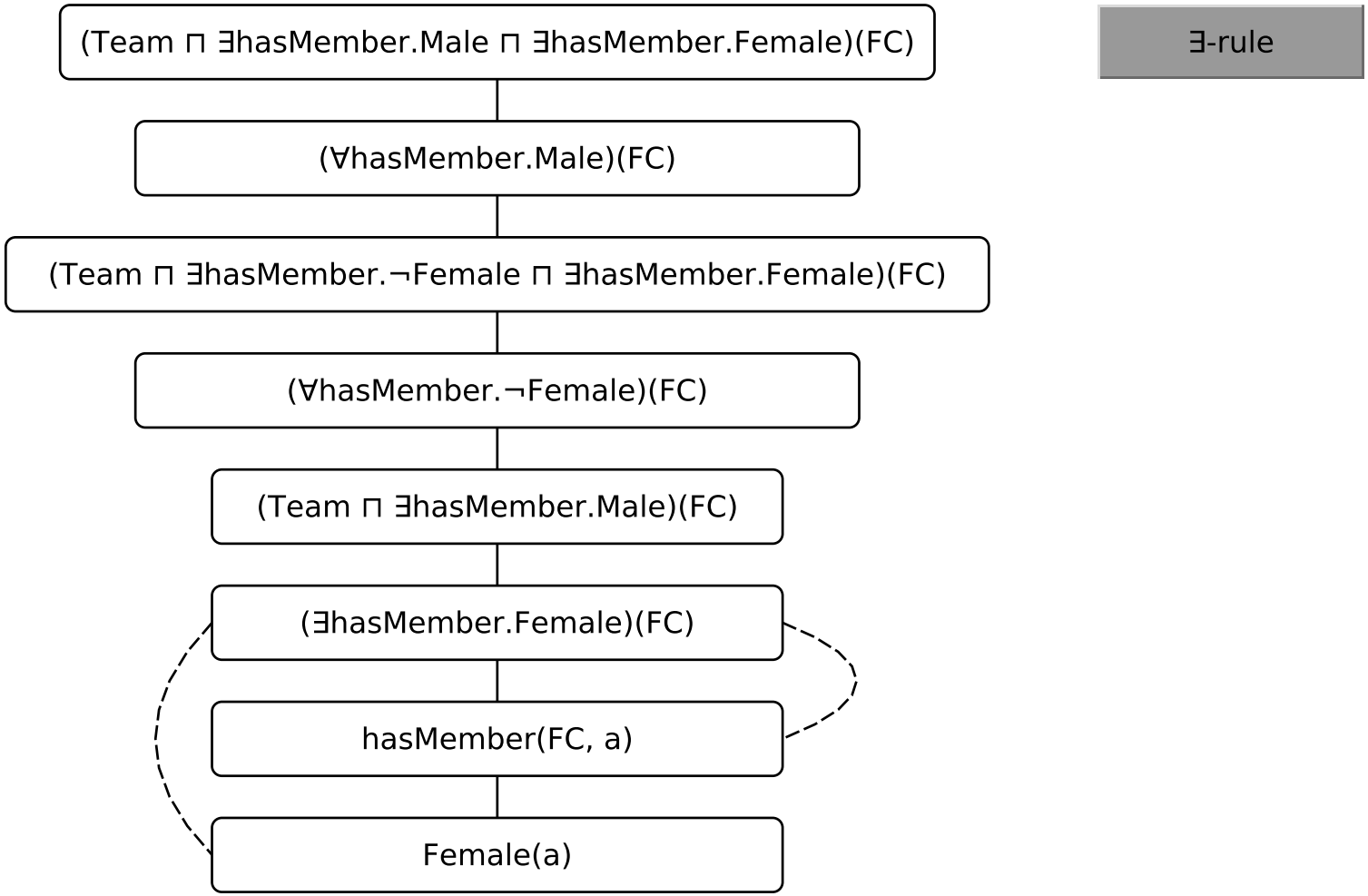
Example (cont'd)

The closed tableau showing unsatisfiability is as follows:



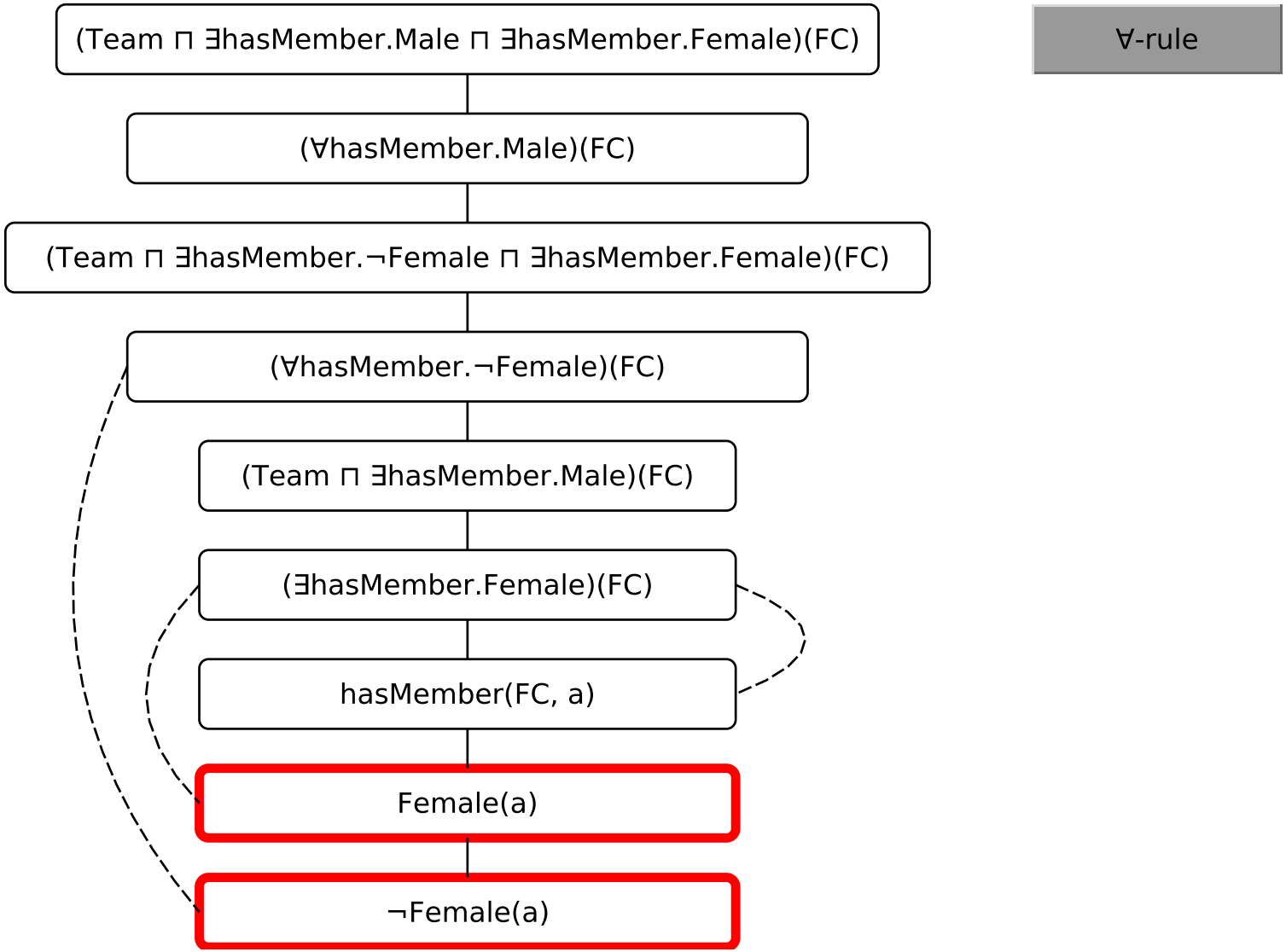
Example (cont'd)

The closed tableau showing unsatisfiability is as follows:



Example (cont'd)

The closed tableau showing unsatisfiability is as follows:



General TBoxes

General TBoxes include concept definitions and concept inclusions.

Example:

$$\texttt{Woman} \equiv \texttt{Person} \sqcap \texttt{Female}$$

$$\texttt{Person} \sqsubseteq \exists \texttt{hasParent.Person}$$

Tableau Techniques for General TBoxes

To construct a tableau proof for general TBoxes, we proceed as follows:

1. Given a of TBox $\mathcal{T}$, we construct a set of concepts $\widehat{\mathcal{T}}$ as follows:
   - Each concept definition is equivalently re-written as two concept inclusions.
   - Each concept inclusion $C \sqsubseteq D$ is rewritten as $\neg C \sqcup D$.
2. We compute the negation normal form $nnf(\widehat{\mathcal{T}})$ of $\widehat{\mathcal{T}}$ as the set of the negation normal forms of its members.

Example

$$\mathcal{T} = \{\ \texttt{Woman} \equiv \texttt{Person} \sqcap \texttt{Female},\ \texttt{Person} \sqsubseteq \exists \texttt{hasParent}.\texttt{Person}\ \}$$

$\mathcal{T}$ can be equivalently rewritten as follows:

$$\widehat{\mathcal{T}} = \{\ \texttt{Woman} \sqsubseteq \texttt{Person} \sqcap \texttt{Female},\ \texttt{Person} \sqcap \texttt{Female} \sqsubseteq \texttt{Woman},$$

$$\texttt{Person} \sqsubseteq \exists \texttt{hasParent}.\texttt{Person}\ \}$$

Then

$$\widehat{\mathcal{T}} = \{\ \neg\texttt{Woman} \sqcup (\texttt{Person} \sqcap \texttt{Female}),\ \neg(\texttt{Person} \sqcap \texttt{Female}) \sqcup \texttt{Woman},$$

$$\neg\texttt{Person} \sqcup \exists \texttt{hasParent}.\texttt{Person}\ \}$$

Example (cont'd)

Then

$$\mathtt{nnf}(\widehat{\mathcal{T}}) = \{\ \neg\mathtt{Woman} \sqcup (\mathtt{Person} \sqcap \mathtt{Female}),\ \neg\mathtt{Person} \sqcup \neg\mathtt{Female} \sqcup \mathtt{Woman},$$

$$\neg\mathtt{Person} \sqcup \exists\mathtt{hasParent}.\mathtt{Person}\ \}$$

Rationale

The rationale behind the construction of $\widehat{\mathcal{T}}$ is the following.
Given any Tbox $\mathcal{T}$ such that $\widehat{\mathcal{T}} = \{C_1, \ldots, C_n\}$, it is easy to see that $\mathcal{T}$ is equivalent to

$$\top \sqsubseteq C_1 \sqcap \cdots \sqcap C_n.$$

How do we prove this?
We have to prove that for every interpretation $\mathcal{I}$:

$$\mathcal{I} \models \mathcal{T} \text{ iff } \mathcal{I} \models \top \sqsubseteq C$$

where:

$$C = \bigsqcap_{(A_i \equiv B_i) \in \mathcal{T}} ((A_i \sqcup \neg B_i) \sqcap (\neg A_i \sqcup B_i)) \sqcap \bigsqcap_{(A_i \sqsubseteq B_i) \in \mathcal{T}} (\neg A_i \sqcup B_i)$$

The $\sqsubseteq$-rule

We now introduce a new inference rule.
If a is an individual that appears in $\mathcal{A}$ and C is a concept in $\widehat{\mathcal{T}}$ **then**

$$\mathcal{A} := \mathcal{A} \cup \{C(a)\}.$$

Example

Let us assume we are given the following knowledge base:

$$\texttt{Woman} \equiv \texttt{Person} \sqcap \texttt{Female}$$

$$\texttt{Person(ANN)},\ \texttt{Female(ANN)},\ \neg\texttt{Woman(ANN)}$$

Can you use tableau to prove that it is unsatisfiable?

Example of Non Terminating Proof

Let us consider the following knowledge base:

$$\mathtt{C} \sqsubseteq \exists \mathtt{R}.\mathtt{C},\ \mathtt{C}(\mathtt{a})$$

If we apply tableau techniques, we can have a non-terminating proof.

Example of Non Terminating Proof (cont'd)

1	$\mathtt{C}(\mathtt{a})$	
2	$(\neg\mathtt{C} \sqcup \exists\mathtt{R}.\mathtt{C})(\mathtt{a})$	by $\sqsubseteq$
3	$(\neg\mathtt{C})(\mathtt{a})$	
4	**Clash**	

4	$(\exists\mathtt{R}.\mathtt{C})(\mathtt{a})$	by 2, $\sqcup$
5	$\mathtt{R}(\mathtt{a},\mathtt{b})$	by 3b, $\exists$
6	$\mathtt{C}(\mathtt{b})$	by 3b, $\exists$
7	$(\neg\mathtt{C} \sqcup \exists\mathtt{R}.\mathtt{C})(\mathtt{b})$	by $\sqsubseteq$
8	$(\neg\mathtt{C})(\mathtt{b})$	
10	**Clash**	

9	$(\exists\mathtt{R}.\mathtt{C})(\mathtt{b})$	by 7, $\sqcup$
11	...	

At Step 11, we can continue as in Step 5 by introducing a new individual c and so on ...

Blocking

In order to guarantee terminating proofs even in the presence of concept inclusions, we can introduce the concept of **blocking.**

Intuition: Blocking prevents application of of the same rule again and again i.e., when it is clear that the subtree rooted in some node x is similar to the subtree rooted in some predecessor node y of x.

The tableau expansion rules given previously can then be modified so that they apply **only to individuals** a **that are not blocked.**

In this way, the tableau techniques can be seen to be **sound and complete decision procedures for** $\mathcal{ALC}$**.**

Details of blocking can be found in the papers in the Readings.

Computational Complexity

Even for a simple DL like $\mathcal{ALC}$, the satisfiability and the consistency problem (without TBoxes) are PSPACE-complete.
With general TBoxes, the satisfiability and the consistency problem become EXPTIME-complete.
In practice there are reasoning algorithms (and implemented reasoners) that do much better than what the above worst-case complexity results tell us to expect.

Efficiency of Implemented Reasoners

See the paper

> *Ian Horrocks. Semantics ⊓ scalability = ⊥? Journal of Zhejiang University - Science C. 13(4):241-244, 2012.*

available from `http://www.cs.ox.ac.uk/people/ian.horrocks/Publications/download/2012/Horr12a.pdf` which discusses briefly the performance of current scalable DL reasoners and contains pointers to other relevant papers.

Table of Contents

OWL in brief

OWL (Web Ontology Language) is an ontology language with semantics based on Description Logics, towards an intelligent web of data.

- An OWL ontology consists of classes, properties and instances identified by IRIs.

OWL in brief

OWL (Web Ontology Language) is an ontology language with semantics based on Description Logics, towards an intelligent web of data.

- An OWL ontology consists of classes, properties and instances identified by IRIs.
- An OWL ontology can also be identified by an IRI, an OWL ontology can import other OWL ontologies by their IRI.

OWL in brief

OWL (Web Ontology Language) is an ontology language with semantics based on Description Logics, towards an intelligent web of data.

- An OWL ontology consists of classes, properties and instances identified by IRIs.
- An OWL ontology can also be identified by an IRI, an OWL ontology can import other OWL ontologies by their IRI.
- OWL is based on DLs and inherits their expressive power:

OWL in brief

OWL (Web Ontology Language) is an ontology language with semantics based on Description Logics, towards an intelligent web of data.

- An OWL ontology consists of classes, properties and instances identified by IRIs.
- An OWL ontology can also be identified by an IRI, an OWL ontology can import other OWL ontologies by their IRI.
- OWL is based on DLs and inherits their expressive power:
  - OWL [$\mathcal{SHOIN}(\mathcal{D})$] is W3C Recommendation since 2004.

OWL in brief

OWL (Web Ontology Language) is an ontology language with semantics based on Description Logics, towards an intelligent web of data.

- An OWL ontology consists of classes, properties and instances identified by IRIs.
- An OWL ontology can also be identified by an IRI, an OWL ontology can import other OWL ontologies by their IRI.
- OWL is based on DLs and inherits their expressive power:
  - OWL [$\mathcal{SHOIN}(\mathcal{D})$] is W3C Recommendation since 2004.
  - OWL2 [$\mathcal{SROIQ}(\mathcal{D})$] is W3C Recommendation since 2009.

OWL in brief

OWL (Web Ontology Language) is an ontology language with semantics based on Description Logics, towards an intelligent web of data.

- An OWL ontology consists of classes, properties and instances identified by IRIs.
- An OWL ontology can also be identified by an IRI, an OWL ontology can import other OWL ontologies by their IRI.
- OWL is based on DLs and inherits their expressive power:
  - OWL [$\mathcal{SHOIN}(\mathcal{D})$] is W3C Recommendation since 2004.
  - OWL2 [$\mathcal{SROIQ}(\mathcal{D})$] is W3C Recommendation since 2009.
- OWL makes the Open World Assumption (OWA).

OWL in brief

OWL (Web Ontology Language) is an ontology language with semantics based on Description Logics, towards an intelligent web of data.

- An OWL ontology consists of classes, properties and instances identified by IRIs.
- An OWL ontology can also be identified by an IRI, an OWL ontology can import other OWL ontologies by their IRI.
- OWL is based on DLs and inherits their expressive power:
  - OWL [$\mathcal{SHOIN}(\mathcal{D})$] is W3C Recommendation since 2004.
  - OWL2 [$\mathcal{SROIQ}(\mathcal{D})$] is W3C Recommendation since 2009.
- OWL makes the Open World Assumption (OWA).
- OWL does not make the Unique Name Assumption (UNA).

OWL in brief

OWL (Web Ontology Language) is an ontology language with semantics based on Description Logics, towards an intelligent web of data.

- An OWL ontology consists of classes, properties and instances identified by IRIs.
- An OWL ontology can also be identified by an IRI, an OWL ontology can import other OWL ontologies by their IRI.
- OWL is based on DLs and inherits their expressive power:
  - OWL [$\mathcal{SHOIN}(\mathcal{D})$] is W3C Recommendation since 2004.
  - OWL2 [$\mathcal{SROIQ}(\mathcal{D})$] is W3C Recommendation since 2009.
- OWL makes the Open World Assumption (OWA).
- OWL does not make the Unique Name Assumption (UNA).
- As in DLs, constructors are used to describe complex classes (concepts).

OWL Syntax Variants

Various serializations exist for OWL:

- Functional syntax (see also: https://www.w3.org/TR/owl2-primer/)

OWL Syntax Variants

Various serializations exist for OWL:

- Functional syntax (see also: https://www.w3.org/TR/owl2-primer/)
- An extension of existing RDF/XML

OWL Syntax Variants

Various serializations exist for OWL:

- Functional syntax (see also: https://www.w3.org/TR/owl2-primer/)
- An extension of existing RDF/XML
- An independent XML serialization (OWL/XML)

OWL Syntax Variants

Various serializations exist for OWL:

- Functional syntax (see also: https://www.w3.org/TR/owl2-primer/)
- An extension of existing RDF/XML
- An independent XML serialization (OWL/XML)
- Manchester syntax, also used in Protege (see also: https://www.w3.org/TR/owl2-manchester-syntax/)

OWL Syntax Variants

Various serializations exist for OWL:

- Functional syntax (see also: https://www.w3.org/TR/owl2-primer/)
- An extension of existing RDF/XML
- An independent XML serialization (OWL/XML)
- Manchester syntax, also used in Protege (see also: https://www.w3.org/TR/owl2-manchester-syntax/)
- Turtle

OWL in a nutshell

OWL 2 ontologies include the following:

- classes: representing sets of elements in the domain,

OWL in a nutshell

OWL 2 ontologies include the following:

- classes: representing sets of elements in the domain,
- properties: distinguished to ObjectProperty and DataProperty (the former relates instances to each other, the latter relates instances to data values),

OWL in a nutshell

OWL 2 ontologies include the following:

- classes: representing sets of elements in the domain,
- properties: distinguished to ObjectProperty and DataProperty (the former relates instances to each other, the latter relates instances to data values),
- instances (or individuals) representing entities in the domain.

OWL in a nutshell

OWL 2 ontologies include the following:

- classes: representing sets of elements in the domain,
- properties: distinguished to ObjectProperty and DataProperty (the former relates instances to each other, the latter relates instances to data values),
- instances (or individuals) representing entities in the domain.
- Expressions: describing complex classes of elements in the domain (i.e. complex concepts or roles in DL terms).

OWL in a nutshell

OWL 2 ontologies include the following:

- classes: representing sets of elements in the domain,
- properties: distinguished to ObjectProperty and DataProperty (the former relates instances to each other, the latter relates instances to data values),
- instances (or individuals) representing entities in the domain.
- Expressions: describing complex classes of elements in the domain (i.e. complex concepts or roles in DL terms).
- Axioms are statements that are asserted to be true (e.g., a subclass axiom)

OWL in a nutshell

OWL 2 ontologies include the following:

- classes: representing sets of elements in the domain,
- properties: distinguished to ObjectProperty and DataProperty (the former relates instances to each other, the latter relates instances to data values),
- instances (or individuals) representing entities in the domain.
- Expressions: describing complex classes of elements in the domain (i.e. complex concepts or roles in DL terms).
- Axioms are statements that are asserted to be true (e.g., a subclass axiom)
- DL reasoners can be employed to draw inferences from axioms

OWL in a nutshell

OWL 2 ontologies include the following:

- classes: representing sets of elements in the domain,
- properties: distinguished to ObjectProperty and DataProperty (the former relates instances to each other, the latter relates instances to data values),
- instances (or individuals) representing entities in the domain.
- Expressions: describing complex classes of elements in the domain (i.e. complex concepts or roles in DL terms).
- Axioms are statements that are asserted to be true (e.g., a subclass axiom)
- DL reasoners can be employed to draw inferences from axioms
- More about OWL in the next lecture.

Readings

- F. Baader. Description Logics. In Reasoning Web: Semantic Technologies for Information Systems, 5th International Summer School 2009, volume 5689 of Lecture Notes in Computer Science, pages 1-39. Springer-Verlag, 2009.
  Available from
  `http://lat.inf.tu-dresden.de/research/papers.html`.
- F. Baader and U. Sattler. An Overview of Tableau Algorithms for Description Logics. Studia Logica, 69:5-40, 2001.
  Available from
  `http://www.cs.man.ac.uk/~sattler/ulis-ps.html`.

Readings (cont'd)

- Franz Baader, Ian Horrocks, and Ulrike Sattler. Description Logics. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, Handbook of Knowledge Representation. Elsevier, 2007.
  Available from `http://www.comlab.ox.ac.uk/people/ian.horrocks/Publications/complete.html#2007`
- (Optional). Melvin Fitting. First-Order Logic and Automated Theorem Proving. 2nd edition. Springer, 1996.
  This is a good introduction to tableau proofs for PL and FOL.

Acknowledgements

The slides on tableau techniques for DLs have been prepared by modifying slides by Enrico Franconi, University of Bolzano-Bozen, Italy.
See `http://www.inf.unibz.it/~franconi/dl/course/` for Enrico's course on DLs.
Some other courses on DLs are listed on `http://dl.kr.org/courses.html`.