Predicate logic - Satisfiability

Dr. Son P. Nguyen

UEL
VNU-HCMC

March 27, 2016

Satisfiability and Tautologies

Definition

Let S be a signature, and φ an S-sentence.

- φ is satisfiable if there is an S-structure $\mathcal{M}$ with $\mathcal{M} \models \varphi$.
- φ is a tautology if for all S-structures $\mathcal{M}$ we have $\mathcal{M} \models \varphi$.

Proposition

Let S be a signature, and let $\varphi, \psi_1, \ldots, \psi_n$ be S-sentences.

1. $\{\psi_1, \ldots, \psi_n\} \models \varphi \iff \psi_1 \wedge \cdots \wedge \psi_n \wedge \neg\varphi$ *is not satisfiable.*
2. $\{\psi_1, \ldots, \psi_n\} \models \varphi \iff (\psi_1 \wedge \cdots \wedge \psi_n) \rightarrow \varphi$ *is a tautology.*

Proof of Part 1

"$\Longleftarrow$" Suppose $\mathcal{M} \not\models \psi_1 \wedge \cdots \wedge \psi_n \wedge \neg\varphi$ for all S-structures $\mathcal{M}$. We have to show that $\{\psi_1, \ldots, \psi_n\} \models \varphi$.

To this end, let $\mathcal{M}$ be an S-structure such that $\mathcal{M} \models \psi_i$ for all $i \in \{1, \ldots, n\}$. In other words, $\mathcal{M} \models \psi_1 \wedge \cdots \wedge \psi_n$. Since $\mathcal{M} \models \psi_1 \wedge \cdots \wedge \psi_n$, but $\mathcal{M} \not\models \psi_1 \wedge \cdots \wedge \psi_n \wedge \neg\varphi$, we must have $\mathcal{M} \not\models \neg\varphi$. The latter is equivalent to $\mathcal{M} \models \varphi$.

We have thus shown that for all S-structures $\mathcal{M}$, if $\mathcal{M} \models \psi_i$ for all $i \in \{1, \ldots, n\}$, then $\mathcal{M} \models \varphi$. This proves $\{\psi_1, \ldots, \psi_n\} \models \varphi$. $\square$

Satisfiability: Examples

1. $\exists x \big(P(x) \land \neg P(x)\big)$ is not satisfiable.
2. $\forall x \, \exists y \, R(x,y) \land \neg \forall u \, \exists v \, R(v,u)$ is satisfiable.
3. $\forall x \, P(x) \land \exists x \, \neg P(x)$ is not satisfiable.

Proof of 2

Claim: $\forall x\, \exists y\, R(x,y) \land \neg \forall u\, \exists v\, R(v,u)$ is satisfiable.

Proof: We have to show that there is an $\{R\}$-structure $\mathcal{M}$ such that the sentence is satisfied in $\mathcal{M}$.

To this end, let $\mathcal{M}$ be the $\{R\}$-structure with:

- domain $\{1,2\}$,
- $R^{\mathcal{M}} = \{(1,1),(2,1)\}$.

Then,

- $\mathcal{M} \models \forall x\, \exists y\, R(x,y)$: For all $d \in \{1,2\}$, there exists an element $d' \in \{1,2\}$ with $(d,d') \in R^{\mathcal{M}}$;
- $\mathcal{M} \not\models \forall u\, \exists v\, R(v,u)$: For $d' = 2$, there does not exist an element $d \in \{1,2\}$ with $(d,d') \in R^{\mathcal{M}}$.

Hence, $\mathcal{M} \models \forall x\, \exists y\, R(x,y) \land \neg \forall u\, \exists v\, R(v,u)$. $\square$

Proof of 3

Claim: $\forall x\, P(x) \land \exists x\, \neg P(x)$ is not satisfiable.

Proof: By contradiction. Suppose that the sentence is satisfiable. Then, there is a $\{P\}$-structure $\mathcal{M}$ with $\mathcal{M} \models \forall x\, P(x) \land \exists x\, \neg P(x)$. Recall that the latter denotes:

$$(\mathcal{M}, a) \models \forall x\, P(x) \land \exists x\, \neg P(x) \tag{$\star$}$$

for an arbitrary assignment a in $\mathcal{M}$. Now, $(\star)$ implies:

1. $(\mathcal{M}, a) \models \forall x\, P(x)$ and
2. $(\mathcal{M}, a) \models \exists x\, \neg P(x)$.

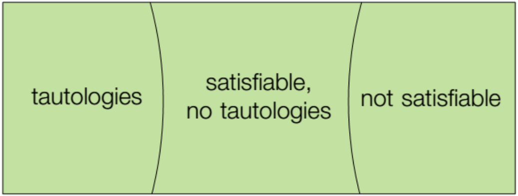
The second item states that there is a $d \in \mathrm{dom}(\mathcal{M})$ with $(\mathcal{M}, a[x \mapsto d]) \not\models P(x)$. This implies $(\mathcal{M}, a) \not\models \forall x\, P(x)$, and hence contradicts the first item above. $\square$

Satisfiability vs Tautologies

Proposition

Let S be a signature, and let φ be an S-sentence. Then, the following are equivalent:

1. *φ is a tautology.*
2. *$\neg\varphi$ is not satisfiable.*



Semantic Consequence vs Satisfiability

Proposition

Let S be a signature, and let $\varphi, \psi_1, \ldots, \psi_n$ be S-sentences.

1. $\{\psi_1, \ldots, \psi_n\} \models \varphi \iff \psi_1 \wedge \cdots \wedge \psi_n \wedge \neg\varphi$ *is not satisfiable.*
2. $\{\psi_1, \ldots, \psi_n\} \models \varphi \iff (\psi_1 \wedge \cdots \wedge \psi_n) \to \varphi$ *is a tautology.*

Note: There is a similar connection between semantic equivalence and satisfiability/tautologies – e.g., use the connection between semantic equivalence and semantic consequence.

Satisfiability Problem for Predicate Logic

Satisfiability Problem (for Predicate Logic)

Input: a signature S, and an S-sentence φ

Task: If φ is satisfiable, output "yes", otherwise "no"

Recall: The analogous problem for propositional logic can be solved, e.g., via truth tables or the tableau algorithm.

Theorem

The satisfiability problem for predicate logic is undecidable:

There is no algorithm that solves the satisfiability problem for predicate logic.

Corollaries

The following problems are also undecidable:

Input:	a signature S, and an S-sentence φ
Task:	If φ is a tautology, output "yes", otherwise "no"

Input:	a signature S, and S-sentences $\psi_1, \ldots, \psi_n$, φ
Task:	If $\{\psi_1, \ldots, \psi_n\} \models \varphi$, output "yes", otherwise "no"

Proof: An algorithm for any of these problems could be used to solve the satisfiability problem (for predicate logic):

- φ is satisfiable $\iff$ $\neg\varphi$ is no tautology;
- φ is satisfiable $\iff$ $\emptyset \not\models \neg\varphi$.

But the satisfiability problem is undecidable. $\square$

Undecidability of Satisfiability – Overview

Theorem

The satisfiability problem for predicate logic is undecidable:

There is no algorithm that solves the satisfiability problem for predicate logic.

Proof Steps:

1. Introduce an auxiliary problem, the Halting problem, and prove that it is undecidable.
2. Show that if there was an algorithm for the satisfiability problem, then it could be used to solve the Halting problem (which is impossible by step 1).

Algorithms and Computations

For the proof, we need a precise definition of the concept of algorithm (or computation).

1. Usually: based on Turing machines

Alan Turing
1912–1954

2. Here (due to time-constraints):

   An algorithm is a program written in some standard programming language. For definiteness, we choose Java.

Decidability

- A decision problem is a problem where the task is to decide if a given input has a certain property.

  Example: Satisfiability problem

- A decision problem is undecidable if there is no algorithm that solves it (and terminates on every input).

  Example: To show that the satisfiability problem is undecidable, we have to rule out the existence of an algorithm that takes as input

  - a signature S and
  - an S-sentence φ

  and outputs "yes" or "no" depending on whether φ is satisfiable.

The Halting Problem

Halting Problem
Input: a Java program P and an input x for P
Task: output "yes" if P terminates when it is run with x as input; otherwise output "no"

- The Halting Problem is a decision problem.
- We assume that every Java program takes only one input (say, the string of command line arguments).

Theorem

The Halting Problem is undecidable.

Proof Sketch

1. Assume the Halting Problem is *not* undecidable. Then, there is a Java program P_{Halt} that solves it.
2. Let P_{new} be a new Java program working as follows:

   Input: a Java program Q
   - out $=$ the output of P_{Halt} when it is started with $P := Q$ and $x := Q$
   - **if** out==“no” **then** output “yes” **else** run forever

3. P_{new} cannot be a Java program:

P_{new} terminates when it is started with input P_{new}

$\iff$ P_{Halt} outputs “no” when started with $P := P_{\text{new}}$ & $x := P_{\text{new}}$

$\iff$ P_{new} does not terminate when it is started with input P_{new}.

This is a contradiction, so the assumption in 1 is wrong.

Undecidability of the Satisfiability Problem

- For any Java program P and input x, we can construct an S-sentence φ (for a suitable signature S) such that

  $$\varphi \text{ is satisfiable } \iff P \text{ terminates when given } x \text{ as input.}$$

  Intuition: Construct φ in such a way that it is satisfied in an S-structure $\mathcal{M}$ precisely if $\mathcal{M}$ describes an execution of P on input x that finishes after a finite number of steps (i.e., calls `System.exit(...)` at some point).

- Since the Halting Problem is undecidable, we conclude that the satisfiability problem is undecidable.

The Tableau Method for Predicate Logic, or How to Enumerate Tautologies

Proof Procedures

- Checking whether a sentence is satisfiable or a tautology can often be done by special proof procedures.
- Proof procedures:
  - Resolution (Robinson ’65)

    Foundation of automated theorem proving, in particular, of the programming language Prolog
  - Tableau method (here)
  - Natural deduction, sequent and Hilbert calculi, …

Simplification

In the following, we assume that:

Signatures contain only predicate and constant symbols.

Note: The tableau method can be extended to the general case.

Tableau Method – Basic Idea

Input: sentence φ

- Derive sentences that have to be satisfied if φ is satisfied.

$\forall x\ P(x)$

$\boxed{\neg P(c)} \vee \boxed{\neg P(d)}$

$\forall x\ P(x)$	$\forall x\ P(x)$
$\neg P(c) \vee \neg P(d)$	$\neg P(c) \vee \neg P(d)$
$\neg P(c)$	$\neg P(d)$

Tableau Method – Basic Idea

Input: sentence φ

- Derive sentences that have to be satisfied if φ is satisfied.

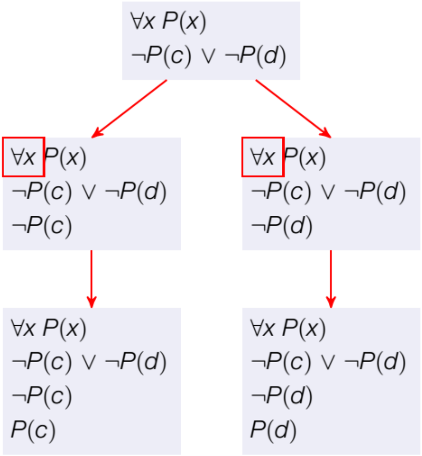
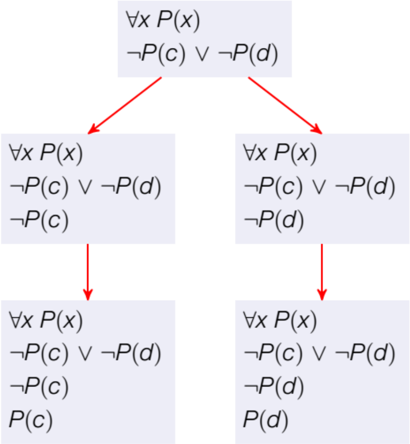


Tableau Method – Basic Idea

Input: sentence φ

- Derive sentences that have to be satisfied if φ is satisfied.
- If we derive an inconsistent pair of sentences on every path, φ is unsatisfiable.



Constraints

Definition

Let S be a signature. An S-constraint $\mathcal{C}$ (or just constraint) is a finite set of S-sentences.

- It is satisfiable if there is an S-structure $\mathcal{M}$ such that $\mathcal{M} \models \varphi$ for all $\varphi \in \mathcal{C}$.
- It contains a clash if there is a sentence $\varphi \in \mathcal{C}$ with $\neg\varphi \in \mathcal{C}$.

Example

- $\mathcal{C}_1 = \{ \forall x P(x) \}$ is satisfiable
- $\mathcal{C}_2 = \{ \forall x P(x), \neg P(c) \}$ is *not* satisfiable
- $\mathcal{C}_3 = \{ \forall x P(x), \neg P(c), P(c) \}$ contains a clash

Completion Rules 1/3

Constraints $\mathcal{C}$ without clashes can be extended via the following completion rules:

$\wedge$-rule: Applicable to $\mathcal{C}$ if $\varphi \wedge \psi \in \mathcal{C}$ and $\{\varphi, \psi\} \not\subseteq \mathcal{C}$.
Result: $\mathcal{C} \cup \{\varphi, \psi\}$.

$\neg\neg$-rule: Applicable to $\mathcal{C}$ if $\neg\neg\varphi \in \mathcal{C}$ and $\varphi \notin \mathcal{C}$.
Result: $\mathcal{C} \cup \{\varphi\}$.

$\neg\vee$-rule: Applicable to $\mathcal{C}$ if $\neg(\varphi \vee \psi) \in \mathcal{C}$ and $\{\neg\varphi, \neg\psi\} \not\subseteq \mathcal{C}$.
Result: $\mathcal{C} \cup \{\neg\varphi, \neg\psi\}$.

$\vee$-rule: Applicable to $\mathcal{C}$ if $\varphi \vee \psi \in \mathcal{C}$, $\varphi \notin \mathcal{C}$, and $\psi \notin \mathcal{C}$.
Result: $\mathcal{C} \cup \{\varphi\}$ **or** $\mathcal{C} \cup \{\psi\}$.

$\neg\wedge$-rule: Applicable to $\mathcal{C}$ if $\neg(\varphi \wedge \psi) \in \mathcal{C}$, $\neg\varphi \notin \mathcal{C}$, and $\neg\psi \notin \mathcal{C}$.
Result: $\mathcal{C} \cup \{\neg\varphi\}$ **or** $\mathcal{C} \cup \{\neg\psi\}$.

Substitution

Definition

Let S be a signature (without function symbols).

Given an S-formula φ, a variable x, and a constant symbol $c \in S$, let $\varphi[x/c]$ be the S-formula obtained from φ by replacing every free occurrence of x in φ by c.

Example:

Let $\varphi = P(x) \wedge \exists x\, R(x, y)$. Then:

- $\varphi[x/c] = P(c) \wedge \exists x\, R(x, y)$
- $\varphi[y/c] = P(x) \wedge \exists x\, R(x, c)$

Completion Rules 2/3

Constraints $\mathcal{C}$ without clashes can also be extended via the following additional completion rules:

$\exists$-rule: Applicable to $\mathcal{C}$ if $\exists x\, \varphi \in \mathcal{C}$ and $\varphi[x/c] \notin \mathcal{C}$ for all constant symbols c.

Result: $\mathcal{C} \cup \{\varphi[x/c]\}$, where c is a constant symbol that does not occur in $\mathcal{C}$.

$\forall$-rule: Applicable to $\mathcal{C}$ if $\forall x\, \varphi \in \mathcal{C}$ and $\varphi[x/c] \notin \mathcal{C}$ for some constant symbol c.

Result: $\mathcal{C} \cup \{\varphi[x/c]\}$, where c is a constant symbol with $\varphi[x/c] \notin \mathcal{C}$.

(If c does not occur in $\mathcal{C}$, then the application of the $\forall$-rule is said to create a new constant.)

Completion Rules 3/3

Constraints $\mathcal{C}$ without clashes can also be extended via the following additional completion rules:

$\neg\exists$-rule: Applicable to $\mathcal{C}$ if $\neg\exists x\, \varphi \in \mathcal{C}$ and $\forall x\, \neg\varphi \notin \mathcal{C}$.
Result: $\mathcal{C} \cup \{\forall x\, \neg\varphi\}$

$\neg\forall$-rule: Applicable to $\mathcal{C}$ if $\neg\forall x\, \varphi \in \mathcal{C}$ and $\exists x\, \neg\varphi \notin \mathcal{C}$.
Result: $\mathcal{C} \cup \{\exists x\, \neg\varphi\}$

Tableau Paths

Definition

A tableau path is a sequence $\mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2, \ldots$ of constraints such that for every constraint $\mathcal{C}_i$ ($i \geq 1$) in the sequence:

- $\mathcal{C}_{i-1}$ does not contain a clash,
- $\mathcal{C}_i$ is the result of applying a completion rule to $\mathcal{C}_{i-1}$.

Example 1

$$\begin{tikzcd}
\mathcal{C}_0 \quad \{\forall x \forall y\, R(x,y) \wedge \neg R(c,c)\} \arrow[d, "\wedge\text{-rule}"] \\
\mathcal{C}_1 \quad \{\, \forall x \forall y\, R(x,y) \wedge \neg R(c,c),\ \forall x \forall y\, R(x,y),\ \neg R(c,c) \,\} \arrow[d, "\forall\text{-rule}"] \\
\mathcal{C}_2 \quad \{\, \forall x \forall y\, R(x,y) \wedge \neg R(c,c),\ \forall x \forall y\, R(x,y),\ \neg R(c,c),\ \forall y\, R(c,y) \,\} \arrow[d, "\forall\text{-rule}"] \\
\mathcal{C}_3 \quad \{\, \forall x \forall y\, R(x,y) \wedge \neg R(c,c),\ \forall x \forall y\, R(x,y),\ \neg R(c,c),\ \forall y\, R(c,y),\ R(c,c) \,\}
\end{tikzcd}$$

Properties of Tableau Paths

A tableau path…

- is complete if it cannot be extended to a longer tableau path.
- contains a clash if it has the form $\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_n$, and $\mathcal{C}_n$ contains a clash.

Note: Every tableau path containing a clash is also complete.

Example 1 Revisited

$$\mathcal{C}_0 \quad \{\forall x \forall y\, R(x,y) \land \neg R(c,c)\}$$

$$\downarrow \ \land\text{-rule}$$

$$\mathcal{C}_1 \quad \{\, \forall x \forall y\, R(x,y) \land \neg R(c,c),\ \forall x \forall y\, R(x,y),\ \neg R(c,c) \,\}$$

$$\downarrow \ \forall\text{-rule}$$

$$\mathcal{C}_2 \quad \{\, \forall x \forall y\, R(x,y) \land \neg R(c,c),\ \forall x \forall y\, R(x,y),\ \neg R(c,c),\ \forall y\, R(c,y) \,\}$$

$$\downarrow \ \forall\text{-rule}$$

$$\mathcal{C}_3 \quad \{\, \forall x \forall y\, R(x,y) \land \neg R(c,c),\ \forall x \forall y\, R(x,y),\ \neg R(c,c),\ \forall y\, R(c,y),\ R(c,c) \,\}$$

contains a clash
is complete

Example 2

$\mathcal{C}_0$ $\{\forall x\, P(x) \wedge \neg\exists x\, P(x)\}$

$\downarrow$ $\wedge$-rule

$\mathcal{C}_1$ $\{\ \forall x\, P(x) \wedge \neg\exists x\, P(x),\ \forall x\, P(x),\ \neg\exists x\, P(x)\ \}$

$\downarrow$ $\neg\exists$-rule

$\mathcal{C}_2$ $\{\ \forall x\, P(x) \wedge \neg\exists x\, P(x),\ \forall x\, P(x),\ \neg\exists x\, P(x),\ \forall x\, \neg P(x)\ \}$

$\downarrow$ $\forall$-rule

$\mathcal{C}_3$ $\{\ \forall x\, P(x) \wedge \neg\exists x\, P(x),\ \forall x\, P(x),\ \neg\exists x\, P(x),\ \forall x\, \neg P(x),\ P(c)\ \}$

$\downarrow$ $\forall$-rule

$\mathcal{C}_4$ $\{\ \forall x\, P(x) \wedge \neg\exists x\, P(x),\ \forall x\, P(x),\ \neg\exists x\, P(x),\ \forall x\, \neg P(x),\ P(c),\ \neg P(c)\ \}$

contains a clash

Note:
The first application of the $\forall$-rule creates a new constant (the second one does not).

Heuristic:
Apply $\forall$-rules that create new constants only if no other rule is applicable.

Fairness

Definition

A tableau path $\mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2, \ldots$ is fair if $\forall$-rules that create new constants are only applied if no other rule can be applied.

Example: The tableau paths from Example 1 and 2 are fair.

Applying the Tableau Method

Theorem

Let S be a signature (without function symbols). The following are equivalent for every S-sentence φ:

1. *φ is unsatisfiable.*
2. *Every complete fair tableau path starting in the constraint $\{\varphi\}$ contains a clash.*

Corollary:

An S-sentence φ is a tautology if and only if every complete fair tableau path starting in $\{\neg\varphi\}$ contains a clash.

(Recall: φ is a tautology $\iff$ $\neg\varphi$ is unsatisfiable.)

Proof of the Theorem

We prove the converses:

1. Soundness:
   If there is a complete fair tableau path that starts in $\{\varphi\}$ and does not contain a clash, then φ is satisfiable.

2. Completeness:
   If φ is satisfiable, then there is a complete fair tableau path that starts in $\{\varphi\}$ and does not contain a clash.

The proofs are similar to those of soundness and completeness of the tableau algorithm for propositional logic.

To prove soundness, one defines an S-structure $\mathcal{M}$ in which all sentences of all constraints of the tableau path are satisfied. Such a structure can be extracted from the tableau path.

Unsatisfiability Using the Tableau Method

- Given:
  a signature S (without function symbols), an S-sentence φ
- Question:
  Is φ unsatisfiable?
- Method:
  Show that all complete fair tableau paths that start in $\{\varphi\}$ contain a clash.

Tautologies Using the Tableau Method

- Given:
  a signature S (without function symbols), an S-sentence φ
- Question:
  Is φ a tautology?
- Method:
  Show that all complete fair tableau paths that start in $\{\neg\varphi\}$ contain a clash.

Example 1 and 2 Revisited

- $\forall x \forall y\, R(x, y) \wedge \neg R(c, c)$ is unsatisfiable, as witnessed by the tableau path on slide 272
- $\forall x\, P(x) \wedge \neg\exists x\, P(x)$ is unsatisfiable, as witnessed by the tableau path on slide 273

Example 3

$\exists x\, P(x) \wedge \forall x\, \neg(P(x) \vee Q(x))$ is unsatisfiable

Example 4

$\exists x \exists y\, R(x, y) \wedge \neg R(c, d)$ is satisfiable

Enumerating Tautologies

Theorem

There is an algorithm that, given a signature S (without function symbols), outputs exactly all unsatisfiable S-sentences.

Corollory:
There is an algorithm that, given a signature S (without function symbols), outputs exactly all S-sentences that are tautologies.

Remark:
There is no such algorithm for satisfiable S-sentences.