Non-classical logics

Lecture 2: Classical logic, Part 3

12.11.2014

Viorica Sofronie-Stokkermans

sofronie@uni-koblenz.de

Winter Semester 2014/2015

Last time

- Propositional logic (Syntax, Semantics)
- Problems: Checking unsatisfiability

  NP complete

  PTIME for certain fragments of propositional logic
- Normal forms (CNF/DNF)
- Translations to CNF/DNF

Decision Procedures for Satisfiability

- Simple Decision Procedures

  truth table method

- The Resolution Procedure

- Tableaux

...

Today

- Methods for checking satisfiability

  Semantic Tableaux

- First-order logic

1.6 Semantic Tableaux

Literature:

M. Fitting: First-Order Logic and Automated Theorem Proving, Springer-Verlag, New York, 1996, chapters 3, 6, 7.

R. M. Smullyan: First-Order Logic, Dover Publ., New York, 1968, revised 1995.

Like resolution, semantic tableaux were developed in the sixties, by R. M. Smullyan on the basis of work by Gentzen in the 30s and of Beth in the 50s.
(According to Fitting, semantic tableaux were first proposed by the Polish scientist Z. Lis in a paper in Studia Logica 10, 1960 that was only recently rediscovered.)

Idea

Idea (for the propositional case):

A set $\{F \wedge G\} \cup N$ of formulas has a model if and only if $\{F \wedge G, F, G\} \cup N$ has a model.

A set $\{F \vee G\} \cup N$ of formulas has a model if and only if $\{F \vee G, F\} \cup N$ or $\{F \vee G, G\} \cup N$ has a model.

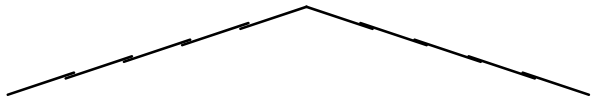
(and similarly for other connectives).

To avoid duplication, represent sets as paths of a tree.

Continue splitting until two complementary formulas are found $\Rightarrow$ inconsistency detected.

A Tableau for $\{P \wedge \neg(Q \vee \neg R),\ \neg Q \vee \neg R\}$

1. $P \wedge \neg(Q \vee \neg R)$
2. $\neg Q \vee \neg R$



3. $\neg Q$	4. $\neg R$
5. P	10. P
6. $\neg(Q \vee \neg R)$	11. $\neg(Q \vee \neg R)$
7. $\neg Q$	
8. $\neg\neg R$	
9. R	

This tableau is not "maximal", however the first "path" is. This path is not "closed", hence the set $\{1, 2\}$ is satisfiable. (These notions will all be defined below.)

Properties

Properties of tableau calculi:

analytic: inferences according to the logical content of the symbols.

goal oriented: inferences operate directly on the goal to be proved (unlike, e. g., resolution).

global: some inferences affect the entire proof state (set of formulas), as we will see later.

Propositional Expansion Rules

Expansion rules are applied to the formulas in a tableau and expand the tableau at a leaf. We append the conclusions of a rule (horizontally or vertically) at a *leaf*, whenever the premise of the expansion rule matches a formula appearing *anywhere* on the path from the root to that leaf.

Negation Elimination

$$\frac{\neg\neg F}{F} \qquad \frac{\neg\top}{\bot} \qquad \frac{\neg\bot}{\top}$$

Propositional Expansion Rules

α-Expansion

(for formulas that are essentially conjunctions: append subformulas α_1 and α_2 one on top of the other)

$$\frac{\alpha}{\begin{array}{l}\alpha_1\\ \alpha_2\end{array}}$$

β-Expansion

(for formulas that are essentially disjunctions: append β_1 and β_2 horizontally, i. e., branch into β_1 and β_2)

$$\frac{\beta}{\beta_1 \mid \beta_2}$$

Classification of Formulas

conjunctive			disjunctive		
α	α_1	α_2	β	β_1	β_2
$X \wedge Y$	X	Y	$\neg(X \wedge Y)$	$\neg X$	$\neg Y$
$\neg(X \vee Y)$	$\neg X$	$\neg Y$	$X \vee Y$	X	Y
$\neg(X \rightarrow Y)$	X	$\neg Y$	$X \rightarrow Y$	$\neg X$	Y

We assume that the binary connective $\leftrightarrow$ has been eliminated in advance.

Tableaux: Notions

A semantic tableau is a marked (by formulas), finite, unordered tree and inductively defined as follows: Let $\{F_1, \ldots, F_n\}$ be a set of formulas.

(i) The tree consisting of a single path

$$F_1$$
$$\vdots$$
$$F_n$$

is a tableau for $\{F_1, \ldots, F_n\}$.
(We do not draw edges if nodes have only one successor.)

Tableaux: Notions

(ii) If T is a tableau for $\{F_1, \ldots, F_n\}$ and if T' results from T by applying an expansion rule then T' is also a tableau for $\{F_1, \ldots, F_n\}$.

A path (from the root to a leaf) in a tableau is called closed, if it either contains $\bot$, or else it contains both some formula F and its negation $\neg F$. Otherwise the path is called open.

A tableau is called closed, if all paths are closed.

A tableau proof for F is a closed tableau for $\{\neg F\}$.

Tableaux: Notions

A path P in a tableau is called maximal, if for each non-atomic formula F on P there exists a node in P at which the expansion rule for F has been applied.

In that case, if F is a formula on P, P also contains:

(i) F_1 and F_2, if F is a α-formula,

(ii) F_1 or F_2, if F is a β-formula, and

(iii) F', if F is a negation formula, and F' the conclusion of the corresponding elimination rule.

A tableau is called maximal, if each path is closed or maximal.

Tableaux: Notions

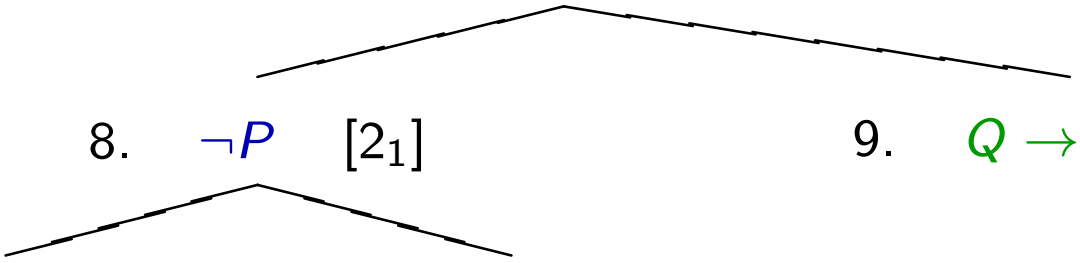
A tableau is called strict, if for each formula the corresponding expansion rule has been applied at most once on each path containing that formula.

A tableau is called clausal, if each of its formulas is a clause.

A Sample Proof

One starts out from the negation of the formula to be proved.

1. $\neg[(P \to (Q \to R)) \to ((P \vee S) \to ((Q \to R) \vee S))]$
2. $(P \to (Q \to R))$ $[1_1]$
3. $\neg((P \vee S) \to ((Q \to R) \vee S))$ $[1_2]$
4. $P \vee S$ $[3_1]$
5. $\neg((Q \to R) \vee S))$ $[3_2]$
6. $\neg(Q \to R)$ $[5_1]$
7. $\neg S$ $[5_2]$



8. $\neg P$ $[2_1]$
9. $Q \to R$ $[2_2]$

10. P $[4_1]$
11. S $[4_2]$

There are three paths, each of them closed.

Properties of Propositional Tableaux

We assume that T is a tableau for $\{F_1, \ldots, F_n\}$.

Theorem 1.8:

$\{F_1, \ldots, F_n\}$ satisfiable $\Leftrightarrow$ some path (i.e., the set of its formulas) in T is satisfiable.

(Proof by induction over the structure of T.)

Corollary 1.9:

T closed $\Rightarrow$ $\{F_1, \ldots, F_n\}$ unsatisfiable

Properties of Propositional Tableaux

Theorem 1.10:
Let T be a strict propositional tableau. Then T is finite.

Proof:
New formulas resulting from expansion are either $\bot$, $\top$ or subformulas of the expanded formula. By strictness, on each path a formula can be expanded at most once. Therefore, each path is finite, and a finitely branching tree with finite paths is finite (König's Lemma).

Conclusion: Strict and maximal tableaux can be effectively constructed.

Refutational Completeness

Theorem 1.11:

Let P be a maximal, open path in a tableau. Then set of formulas on P is satisfiable.

Theorem 1.12:

$\{F_1, \ldots, F_n\}$ satisfiable $\Leftrightarrow$ there exists no closed strict tableau for $\{F_1, \ldots, F_n\}$.

Consequences

The validity of a propositional formula F can be established by constructing a strict, maximal tableau T for $\{\neg F\}$:

- T closed $\Leftrightarrow$ F valid.
- It suffices to test complementarity of paths wrt. atomic formulas.
- Which of the potentially many strict, maximal tableaux one computes does not matter. In other words, tableau expansion rules can be applied don't-care non-deterministically ("proof confluence").

Checking validity of formulae

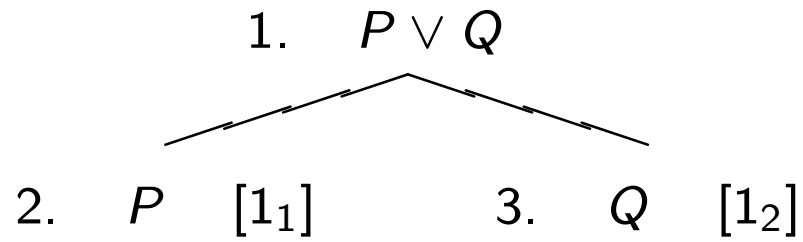
Nota bene: We cannot check the validity of a formula F by constructing a strict, maximal tableau for F.

Checking validity of formulae

Nota bene: We cannot check the validity of a formula F by constructing a strict, maximal tableau for F.

Example: Let $F \quad := \quad (P \vee Q)$

A strict, maximal tableau for F is:



This shows that F is satisfiable. Nothing can be inferred about the validity of F this way.

To check whether F is valid, we construct a strict, maximal tableau T for $\neg F$. If T is closed, then $\neg F$ is unsatisfiable, hence F is valid; otherwise F is not valid.

(In the example below, we can construct a strict, maximal tableau for $\neg F$ which is not closed, so F is not valid.)

Part 2: First-Order Logic

First-order logic

- formalizes fundamental mathematical concepts
- is expressive (Turing-complete)
- is not too expressive
  (e. g. not axiomatizable: natural numbers, uncountable sets)
- has a rich structure of decidable fragments
- has a rich model and proof theory

First-order logic is also called (first-order) predicate logic.

2.1 Syntax

Syntax:

- non-logical symbols (domain-specific)
  $\Rightarrow$ terms, atomic formulas
- logical symbols (domain-independent)
  $\Rightarrow$ Boolean combinations, quantifiers

Signature

A signature

$$\Sigma = (\Omega, \Pi),$$

fixes an alphabet of non-logical symbols, where

- Ω is a set of function symbols f with arity $n \geq 0$, written f/n,
- Π is a set of predicate symbols p with arity $m \geq 0$, written p/m.

If $n = 0$ then f is also called a constant (symbol).
If $m = 0$ then p is also called a propositional variable.
We use letters P, Q, R, S, to denote propositional variables.

Variables

Predicate logic admits the formulation of abstract, schematic assertions.
(Object) variables are the technical tool for schematization.

We assume that

$$X$$

is a given countably infinite set of symbols which we use for (the denotation of) variables.

Terms

Terms over Σ (resp., Σ-terms) are formed according to these syntactic rules:

$$\begin{array}{lcll} s, t, u, v & ::= & x & , x \in X \quad \text{(variable)} \\ & | & f(s_1, ..., s_n) & , f/n \in \Omega \quad \text{(functional term)} \end{array}$$

By $\mathsf{T}_\Sigma(X)$ we denote the set of Σ-terms (over X).

A term not containing any variable is called a ground term.

By T_Σ we denote the set of Σ-ground terms.

Terms

In other words, terms are formal expressions with well-balanced brackets which we may also view as marked, ordered trees.

The markings are function symbols or variables.

The nodes correspond to the subterms of the term.

A node v that is marked with a function symbol f of arity n has exactly n subtrees representing the n immediate subterms of v.

Atoms

Atoms (also called atomic formulas) over Σ are formed according to this syntax:

$$\begin{array}{lcll} A, B & ::= & p(s_1, ..., s_m) & , \; p/m \in \Pi \\ \left[\right. & | & (s \approx t) & \text{(equation)} \quad \left. \right] \end{array}$$

Whenever we admit equations as atomic formulas we are in the realm of first-order logic with equality. Admitting equality does not really increase the expressiveness of first-order logic. But deductive systems where equality is treated specifically can be much more efficient.

Literals

$$
\begin{array}{lcll}
L & ::= & A & \text{(positive literal)} \\
 & | & \neg A & \text{(negative literal)}
\end{array}
$$

Clauses

$$
\begin{array}{lcll}
C, D & ::= & \bot & \text{(empty clause)} \\
 & | & L_1 \vee \ldots \vee L_k, \;\; k \geq 1 & \text{(non-empty clause)}
\end{array}
$$

General First-Order Formulas

$\mathsf{F}_\Sigma(X)$ is the set of first-order formulas over Σ defined as follows:

$$
\begin{array}{lcll}
F, G, H & ::= & \bot & \text{(falsum)} \\
& | & \top & \text{(verum)} \\
& | & A & \text{(atomic formula)} \\
& | & \neg F & \text{(negation)} \\
& | & (F \wedge G) & \text{(conjunction)} \\
& | & (F \vee G) & \text{(disjunction)} \\
& | & (F \to G) & \text{(implication)} \\
& | & (F \leftrightarrow G) & \text{(equivalence)} \\
& | & \forall x\, F & \text{(universal quantification)} \\
& | & \exists x\, F & \text{(existential quantification)}
\end{array}
$$

Notational Conventions

We omit brackets according to the following rules:

- $\neg \quad >_p \quad \wedge \quad >_p \quad \vee \quad >_p \quad \rightarrow \quad >_p \quad \leftrightarrow$
  (binding precedences)
- $\vee$ and $\wedge$ are associative and commutative
- $\rightarrow$ is right-associative

$Qx_1, \ldots, x_n\, F$ abbreviates $Qx_1 \ldots Qx_n\, F$.

Notational Conventions

We use infix-, prefix-, postfix-, or mixfix-notation with the usual operator precedences.

Examples:

$$
\begin{array}{rcl}
s + t * u & \text{for} & +(s, *(t, u)) \\
s * u \leq t + v & \text{for} & \leq (*(s, u), +(t, v)) \\
-s & \text{for} & -(s) \\
0 & \text{for} & 0()
\end{array}
$$

Bound and Free Variables

In QxF, $Q \in \{\exists, \forall\}$, we call F the scope of the quantifier Qx.
An *occurrence* of a variable x is called bound, if it is inside the scope of a quantifier Qx.
Any other occurrence of a variable is called free.

Formulas without free variables are also called closed formulas or sentential forms.

Formulas without variables are called ground.

Bound and Free Variables

Example:

$$\overbrace{\forall y \quad (\overbrace{\forall x \quad p(x)}^{\text{scope}} \quad \rightarrow \quad q(x, y))}^{\text{scope}}$$

The occurrence of y is bound, as is the first occurrence of x.

The second occurrence of x is a free occurrence.

Substitutions

Substitution is a fundamental operation on terms and formulas that occurs in all inference systems for first-order logic.

In general, substitutions are mappings

$$\sigma : X \to \mathsf{T}_\Sigma(X)$$

such that the domain of σ, that is, the set

$$\mathit{dom}(\sigma) = \{x \in X \mid \sigma(x) \neq x\},$$

is finite. The set of variables introduced by σ, that is, the set of variables occurring in one of the terms $\sigma(x)$, with $x \in \mathit{dom}(\sigma)$, is denoted by $\mathit{codom}(\sigma)$.

Substitutions

Substitutions are often written as $[s_1/x_1, \ldots, s_n/x_n]$, with x_i pairwise distinct, and then denote the mapping

$$[s_1/x_1, \ldots, s_n/x_n](y) = \begin{cases} s_i, & \text{if } y = x_i \\ y, & \text{otherwise} \end{cases}$$

We also write $x\sigma$ for $\sigma(x)$.

The modification of a substitution σ at x is defined as follows:

$$\sigma[x \mapsto t](y) = \begin{cases} t, & \text{if } y = x \\ \sigma(y), & \text{otherwise} \end{cases}$$

Application of a Substitution

"Homomorphic" extension of σ to terms and formulas:

$$
\begin{aligned}
f(s_1, \ldots, s_n)\sigma &= f(s_1\sigma, \ldots, s_n\sigma) \\
\bot\sigma &= \bot \\
\top\sigma &= \top \\
p(s_1, \ldots, s_n)\sigma &= p(s_1\sigma, \ldots, s_n\sigma) \\
(u \approx v)\sigma &= (u\sigma \approx v\sigma) \\
\neg F\sigma &= \neg(F\sigma) \\
(F \rho G)\sigma &= (F\sigma \,\rho\, G\sigma)\ ; \quad \text{for each binary connective } \rho \\
(Qx\, F)\sigma &= Qz\,(F\,\sigma[x \mapsto z])\ ; \quad \text{with } z \text{ a fresh variable}
\end{aligned}
$$

2.2 Semantics

To give semantics to a logical system means to define a notion of truth for the formulas. The concept of truth that we will now define for first-order logic goes back to Tarski.

As in the propositional case, we use a two-valued logic with truth values "true" and "false" denoted by 1 and 0, respectively.

Structures

A Σ-structure (also called Σ-interpretation or sometimes Σ-algebra) is a triple

$$\mathcal{A} = (U,\ (f_{\mathcal{A}} : U^n \to U)_{f/n \in \Omega},\ (p_{\mathcal{A}} \subseteq U^m)_{p/m \in \Pi})$$

where $U \neq \emptyset$ is a set, called the universe of $\mathcal{A}$.

Remark: Instead of writing $p_{\mathcal{A}} \subseteq U^m$ we can also us the characteristic function and write:

$$p_{\mathcal{A}} : U^m \to \{0, 1\}.$$

Normally, by abuse of notation, we will have $\mathcal{A}$ denote both the structure and its universe.

By Σ-Str we denote the class of all Σ-structures.

Assignments

A variable has no intrinsic meaning. The meaning of a variable has to be defined externally (explicitly or implicitly in a given context) by an assignment.

A (variable) assignment, also called a valuation (over a given Σ-structure $\mathcal{A}$), is a map $\beta : X \to \mathcal{A}$.

Variable assignments are the semantic counterparts of substitutions.

Value of a Term in $\mathcal{A}$ with Respect to β

By structural induction we define

$$\mathcal{A}(\beta) : \mathsf{T}_\Sigma(X) \to \mathcal{A}$$

as follows:

$$\mathcal{A}(\beta)(x) = \beta(x), \qquad x \in X$$

$$\mathcal{A}(\beta)(f(s_1, \ldots, s_n)) = f_{\mathcal{A}}(\mathcal{A}(\beta)(s_1), \ldots, \mathcal{A}(\beta)(s_n)), \qquad f/n \in \Omega$$

Value of a Term in $\mathcal{A}$ with Respect to β

In the scope of a quantifier we need to evaluate terms with respect to modified assignments. To that end, let $\beta[x \mapsto a] : X \to \mathcal{A}$, for $x \in X$ and $a \in \mathcal{A}$, denote the assignment

$$\beta[x \mapsto a](y) := \begin{cases} a & \text{if } x = y \\ \beta(y) & \text{otherwise} \end{cases}$$

Truth Value of a Formula in $\mathcal{A}$ with Respect to β

$\mathcal{A}(\beta) : \mathsf{F}_\Sigma(X) \to \{0,1\}$ is defined inductively as follows:

$$
\begin{aligned}
\mathcal{A}(\beta)(\bot) &= 0 \\
\mathcal{A}(\beta)(\top) &= 1 \\
\mathcal{A}(\beta)(p(s_1,\ldots,s_n)) = 1 &\quad\Leftrightarrow\quad (\mathcal{A}(\beta)(s_1),\ldots,\mathcal{A}(\beta)(s_n)) \in p_{\mathcal{A}} \\
\mathcal{A}(\beta)(s \approx t) = 1 &\quad\Leftrightarrow\quad \mathcal{A}(\beta)(s) = \mathcal{A}(\beta)(t) \\
\mathcal{A}(\beta)(\neg F) = 1 &\quad\Leftrightarrow\quad \mathcal{A}(\beta)(F) = 0 \\
\mathcal{A}(\beta)(F \rho G) &= \mathsf{B}_\rho(\mathcal{A}(\beta)(F), \mathcal{A}(\beta)(G)) \\
&\quad \text{with } \mathsf{B}_\rho \text{ the Boolean function associated with } \rho \\
\mathcal{A}(\beta)(\forall x F) &= \min_{a \in U}\{\mathcal{A}(\beta[x \mapsto a])(F)\} \\
\mathcal{A}(\beta)(\exists x F) &= \max_{a \in U}\{\mathcal{A}(\beta[x \mapsto a])(F)\}
\end{aligned}
$$

Example

The "Standard" Interpretation for Peano Arithmetic:

$$
\begin{aligned}
U_{\mathbb{N}} &= \{0, 1, 2, \ldots\} \\
0_{\mathbb{N}} &= 0 \\
s_{\mathbb{N}} &: n \mapsto n + 1 \\
+_{\mathbb{N}} &: (n, m) \mapsto n + m \\
*_{\mathbb{N}} &: (n, m) \mapsto n * m \\
\leq_{\mathbb{N}} &= \{(n, m) \mid n \text{ less than or equal to } m\} \\
<_{\mathbb{N}} &= \{(n, m) \mid n \text{ less than } m\}
\end{aligned}
$$

Note that $\mathbb{N}$ is just one out of many possible Σ_{PA}-interpretations.

Example

Values over $\mathbb{N}$ for Sample Terms and Formulas:

Under the assignment $\beta : x \mapsto 1, y \mapsto 3$ we obtain

$$\begin{aligned}
\mathbb{N}(\beta)(s(x) + s(0)) &= 3 \\
\mathbb{N}(\beta)(x + y \approx s(y)) &= 1 \\
\mathbb{N}(\beta)(\forall x, y(x + y \approx y + x)) &= 1 \\
\mathbb{N}(\beta)(\forall z\ z \leq y) &= 0 \\
\mathbb{N}(\beta)(\forall x \exists y\ x < y) &= 1
\end{aligned}$$