Elements of Mathematical Logic

November 10, 2005

- Propositional calculus
- Predicate calculus
- Decision procedures
- Resolution theorem proving

Syntax of propositional logic

Symbols of propositional logic: *atomic propositions* $p, q, r, \cdots$,
logical connectors $\neg$ and $\rightarrow$, and parantheses ( ).

Formulas of propositional logic:
– any atomic proposition is a formula
– if α is a formula, then $(\neg\alpha)$ is a formula.
– if α and β are formulas, then $(\alpha \rightarrow \beta)$ is a formula.

Other known operators can be introduced as shorthands:
– $(\alpha \wedge \beta) \stackrel{\text{def}}{=} (\neg(\alpha \rightarrow (\neg\beta)))$
– $(\alpha \vee \beta) \stackrel{\text{def}}{=} ((\neg\alpha) \rightarrow \beta)$
– $(\alpha \leftrightarrow \beta) \stackrel{\text{def}}{=} ((\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha))$

Simplified notation: without redundant parantheses;
precedence order defined as: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$; $\rightarrow$ is right-associative

Valuation functions (truth assignments)

A valuation v is a function defined for all propositional formulas, with values in $\{T, F\}$ such that:
– $v(p)$ is defined for any atomic proposition p.
– $v(\neg\alpha) = \begin{cases} T & \text{if } v(\alpha) = F \\ F & \text{if } v(\alpha) = T \end{cases}$
– $v(\alpha \to \beta) = \begin{cases} F & \text{if } v(\alpha) = T \text{și } v(\beta) = F \\ T & \text{otherwise} \end{cases}$

An *interpretation* = a valuation for the atomic propositions of a formula
An intrepretation *satisfies* a formula if the latter is evaluated to T
(we say that the interpretation is a *model* for that formula).

valid formula (*tautology*): true in all interpretations
satisfiable formula: true in at least one interpretation
unsatisfiable formula (*contradiction*): false in any interpretation

Syntactic and semantic approach

Semantic approach: based on *logical implication* (logical truth)

$$H \models \varphi$$

A set of formulas H *implies* a formula φ if any truth function that satisfies H (i.e., all formulas in H) also satisfies φ.

Syntactic approach: *logical proof*
– based on syntactic manipulation of formulas:

Is a theorem provable from a set of *axioms*, using *deduction rules* ?

Axioms and deduction rules

Axion schemes for propositional logic:

A1: $(\alpha \rightarrow (\beta \rightarrow \alpha))$

A2: $((\alpha \rightarrow (\beta \rightarrow \gamma)) \rightarrow ((\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \gamma)))$

A3: $(((\neg\beta) \rightarrow (\neg\alpha)) \rightarrow (((\neg\beta) \rightarrow \alpha) \rightarrow \beta))$

(called *schemes* (schemata) because axioms are obtained substituting particular formulas of propositional logic)

We introduce a single deduction rule (*modus ponens, MP*):
From the formulas φ and $\varphi \rightarrow \psi$ we can deduce ψ.

Deduction

Let H be a set of formulas. We call *deduction* from H a sequence of formulas $A_1, A_2, \cdots, A_n$, such that:

1. A_i is an axiom, or
2. A_i is a formula from H, or
3. A_i follows by MP from two previous sequence items A_j, A_k where $j < i$, $k < i$.

We say that A_n follows from H (is deducible, is a consequence): $H \vdash A_n$

Example: we prove that $(\varphi \to \varphi)$

(1) $\varphi \to ((\varphi \to \varphi) \to \varphi))$	**A1**
(2) $\varphi \to ((\varphi \to \varphi) \to \varphi)) \to ((\varphi \to (\varphi \to \varphi) \to (\varphi \to \varphi))$	**A2**
(3) $(\varphi \to (\varphi \to \varphi) \to (\varphi \to \varphi)$	MP(1,2)
(4) $\varphi \to (\varphi \to \varphi)$	**A1**
(5) $\varphi \to \varphi$	MP(3,4)

Deduction theorem

Let H be a set of formulas and α, β two formulas.
Then $H \vdash \alpha \rightarrow \beta$ if and only if $H \cup \{\alpha\} \vdash \beta$.
- used as additional inference rule to simplify proofs

Other corollaries:
- if $H \vdash \alpha$ and $H \vdash \alpha \rightarrow \beta$, then $H \vdash \beta$
- if $G \subseteq H$ and $G \vdash \alpha$, then $H \vdash \alpha$
- if $H \vdash G$ and $G \vdash \alpha$, then $H \vdash \alpha$

Soundness and Completeness

establish the correspondence between the syntactic approach, based on deduction, and the semantic approach, based on truth values.

Soundness: If H is a set of formulas, and α is a formula such that $H \vdash \alpha$, then $H \models \alpha$.
(Any thorem in propositional logic is a tautology)

Completeness: If H is a set of formulas, and α is a formula such that $H \models \alpha$, then $H \vdash \alpha$. (Any tautology is a theorem).

Proof: based on the following notions and auxiliary results:

A set of formulas H is *inconsistent* if there is a formula α such that $H \vdash \alpha$ and $H \vdash \neg\alpha$.

Any consistent set of formulas can be extended to a *maximally consistent* set (adding any other formula makes it inconsistent)

A set of formulas is *consistent* if and only if it is *satisfiable*.

First-order languages

The symbols of a first-order language are:

- parantheses ( )
- logical connectors $\neg$ and $\rightarrow$
- the quantifier $\forall$ (universal quantifier)
- a set of identifiers $v_0, v_1, \cdots$ for *variables*
- a (possibly empty) set of symbols for *constants*
- for any $n \geq 1$ a set of n-ary *function* symbols (of n arguments)
- for any $n \geq 1$ a set of n-ary *predicates* (relations)

First-order languages with equality: contain $=$ as special symbol in addition to the above.

First-order terms and formulas

Terms of a first-order language (defined by structural induction)
- any variable symbol v_n
- any constant symbol c
- $f(t_1, \cdots, t_n)$, if f is an n-ary function symbol and $t_1, \cdots, t_n$ are terms

(Well-formed) Formulas of a first-order language:
- $P(t_1, \cdots, t_n)$, where P is an n-ary predicate and $t_1, \cdots, t_n$ are terms
- $t_1 = t_2$, where t_1 and t_2 are terms (for languages with equality)
- $\neg\alpha$, where α is a formula
- $\alpha \rightarrow \beta$, where α, β are formulas
- $\forall v_n \varphi$ where v_n is a variable and φ is a formula

Interpretations and valuations

An *interpretation* (*structure*) I for the predicate language $\mathcal{L}$ consists of:
- a nonempty set U called the *universe* or the *domain* of I
(the set of values which the variables can take)
- for any constant symbol c, a value $c_I \in U$
- for any n-ary function symbol f, a function $f : U^n \to U$
- for any n-ary predicate symbol P, a subset $P_I \subseteq U^n$.

Let I be an interpretation with universe U for $\mathcal{L}$, and let V be the set of all variable symbols from $\mathcal{L}$. A *valuation* is a function $s : V \to U$.
Extending the valuation s to terms and formulas we obtain a truth function (valuation) for all formulas in $\mathcal{L}$. We write $I \models s(\varphi)$ or $I \models \varphi[s]$ if the valuation s evaluates formula φ to true in the interpretation I.
Define: $I \models s(\forall x \varphi)$ if $I \models s_{x \leftarrow d}(\varphi)$ for any $d \in U$, where
$s_{x \leftarrow d}$ is the valuation $s_{x \leftarrow d}(v) = \begin{cases} d & \text{if variable } v \text{ is } x \\ \text{s(v)} & \text{for any other variable } v \end{cases}$

Denote $I \models \varphi$ (I is a *model* for φ) if $I \models s(\varphi)$ for any valuation s.

Axioms of predicate calculus

Define: variable x can be *substituted* with term t in $\forall y \varphi$ if:

- x does not appear free in φ or
- y does not appear in t and x can be substituted with t in φ

A1: $(\alpha \rightarrow (\beta \rightarrow \alpha))$

A2: $((\alpha \rightarrow (\beta \rightarrow \gamma)) \rightarrow ((\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \gamma)))$

A3: $(((\neg\beta) \rightarrow (\neg\alpha)) \rightarrow (((\neg\beta) \rightarrow \alpha) \rightarrow \beta))$

A4: $(\forall x(\alpha \rightarrow \beta) \rightarrow (\forall x\alpha \rightarrow \forall x\beta))$

A5: $(\forall x\alpha \rightarrow \alpha[x \leftarrow t])$, if x can be substituted with t in α

A6: $(\alpha \rightarrow \forall x\alpha)$ if x does not appear free in α

For equality, we also add

A7: $x = x$

A8: $x = y \rightarrow \alpha = \beta$

where β is obtained from α by replacing arbitrarily many occurrences of x with y.

Soundness and completeness

Let H be a set of formulas and φ a formula. We say that H implies φ ($H \models \varphi$) if for any interpretation I, $I \models H$ implies $I \models \varphi$.

First-order predicate calculus is sound and complete
(like propositional logic).

For any hypothesis set H, and any formula φ, $H \vdash \varphi$ iff $H \models \varphi$.

Note: The notion of completeness above is *different* from the notion of completeness which asks whether a set of axioms is sufficient for deducing any formula or its negation.

The question whether $H \vdash \varphi$ is undecidable in general.

Satisfiability. Applications

Problem: Determine whether a propositional formula is satisfiable.

Context: generally, complex formulas with hundreds or thousands of variables.

The problem appears:
- in determining the equivalence of two circuits or models
- as basic step in theorem proving
- used instead of BDDs for symbolic model checking

Representation: conjunctive normal (canonical) form

efficient *decision procedures* based on the Davis-Putnam algorithm

Terminology: *unit clause*: composed of a single literal
pure literal: appears only positively (or only negated)

Davis-Putnam Algorithm

function Satisfiable (listă de clauze S)
repeat
  for each unit clause or pure literal L from S **do**
    eliminate all clauses containing L
    eliminate ¬ L from all clauses
  if S is empty **return** TRUE
  elsif S contains the empty clause **return** FALSE
until no more changes
choose a literal L from S for decomposition (true/false)
if Satisfiable (S ∪ {L}) **return** TRUE
elsif Satisfiable (S ∪ {¬L}) **return** TRUE
else return FALSE

Theorem provers

Great variety:
- for proving results from mathematics
- for system verification (especially programs)

Generally, implemented for higher-order logics
- allow types described by means of predicates
- have inductive capabilities

Basic approaches to proving:
- forward chaining (derive theorems getting closer to the goal)
- or backwards chaining (generate intermediate conclusions for the given goal)
- application of inference rules: controlled by *tactics*

the resolution method. Clausal form

Any formula without free variables in predicate calculus can be written in clausal form in a sequence of 8 steps

Example: start with
$\forall x[\neg P(x) \rightarrow \exists y(D(x,y) \wedge \neg(E(f(x),y) \vee E(x,y))] \wedge \neg\forall x P(x)$

(1) Eliminate all connectors except $\wedge$, $\vee$, $\neg$:
$\forall x[\neg\neg P(x) \vee \exists y(D(x,y) \wedge \neg(E(f(x),y) \vee E(x,y)))] \wedge \neg\forall x P(x)$

(2) Translate all negation inwards until they reach predicates:
$\forall x[P(x) \vee \exists y(D(x,y) \wedge \neg E(f(x),y) \wedge \neg E(x,y))] \wedge \exists x \neg P(x)$

(3) Rename variables, with unique name for each quantifier:
$\forall x[P(x) \vee \exists y(D(x,y) \wedge \neg E(f(x),y) \wedge \neg E(x,y))] \wedge \exists z \neg P(z)$

Transforming to clausal form (cont.)

(4) Eliminate existential quantifiers (skolemize)
For $\exists y$ within a quantifier $\forall x$, create a *Skolem function* $y = g(x)$
(the value of y depends in general on the value of x).
Otherwise, choose a new *Skolem constant*.
$\forall x[P(x) \vee (D(x, g(x)) \wedge \neg E(f(x), g(x)) \wedge \neg E(x, g(x)))] \wedge \neg P(a)$

(5) Bring to prenex normal form (all $\forall$ quantifiers in front)
$\forall x([P(x) \vee (D(x, g(x)) \wedge \neg E(f(x), g(x)) \wedge \neg E(x, g(x)))] \wedge \neg P(a))$

(6) Eliminate prefix with universal quantifiers
$[P(x) \vee (D(x, g(x)) \wedge \neg E(f(x), g(x)) \wedge \neg E(x, g(x)))] \wedge \neg P(a)$

(7) convert to conjunctive normal form
$(P(x) \vee D(x, g(x))) \wedge (P(x) \vee \neg E(f(x), g(x))) \wedge (P(x) \vee \neg E(x, g(x))) \wedge \neg P(a)$
(8) Eliminate $\wedge$ and write disjunctions as separate clauses

Resolution principle

Consider two clauses, written as sets of disjunctive terms.

Consider first the case of propositional formulas

Call *resolvent* of two clauses C_1, C_2 with respect to literal l
(for which $l \in C_1, (\neg l) \in C_2$): $\text{rez}_l(C_1, C_2) = (C_1 \setminus \{l\}) \cup (C_2 \setminus \{\neg l\})$.

Example: $\text{rez}_p(\{p, q, r\}, \{\neg p, s\}) = \{q, r, s\}$.

$(p \vee q \vee r) \wedge (\neg p \vee s) \rightarrow (q \vee r \vee s)$

Proposition: $C_1, C_2 \models \text{rez}_l(C_1, C_2)$.

Corollary: $C_1 \wedge C_2$ is satisfiable iff $\text{rez}_l(C_1, C_2)$ is satisfiable.

We determine the satisfiability of a formula in conjunctive normal form by repeatedly adding resolvents, and trying to derive the empty clause.

Term unification

For predicate calculus, proceed likewise; but instead of a literal l and its negation, $\neg l$ consider the negation $\neg l'$ of a literal l' that can be *unified* with it.

Two literals can be unified if there is a term substitution for the occurring variables that makes the literals identical.

Example: $P(a, x, y)$ and $P(z, f(z), b)$ can be unified to $P(a, f(a), b)$.

To unify two literals: successively unify terms on same argument position (for functions and predicates) until the same literal is obtained, or unification becomes impossible (symbols of different functions, or unification of x with a term containing x).