Automated Reasoning*

Christoph Weidenbach

Summer Term 2008

Topics of the Course I

Propositional logic

> language: syntax, semantics
> calculi: DPLL-procedure
> implementation: 2-watched literal, clause learning

Linear arithmetic

> language: syntax, semantics
> calculi: Fourier-Motzkin

Propositional logic modulo a theory T

> syntax, semantics
> calculi: DPLL(T)-procedure, ...
> implementation: coupling

*This document contains the text of the lecture slides (almost verbatim) plus some additional information, mostly proofs of theorems that are presented on the blackboard during the course. It is not a full script and does not contain the examples and additional explanations given during the lecture. Moreover it should not be taken as an example how to write a research paper – neither stylistically nor typographically.

Topics of the Course II

First-order predicate logic with equality

> syntax, semantics, model theory, . . .
> calculi: superposition (SUP)
> implementation: sharing, indexing

First-order predicate logic with equality modulo a theory T

> syntax, semantics, model theory, . . .
> calculi: SUP(T)
> implementation: coupling

1 Propositional Logic

Propositional logic

- logic of truth values
- decidable (but NP-complete)
- can be used to describe functions over a finite domain
- important for hardware applications (e. g., model checking)

1.1 Syntax

- propositional variables
- logical symbols
  $\Rightarrow$ Boolean combinations

Propositional Variables

Let Π be a set of *propositional variables*.

We use letters P, Q, R, S, to denote propositional variables.

Propositional Formulas

F_Π is the set of propositional formulas over Π defined as follows:

$$
\begin{array}{rcll}
F, G, H & ::= & \bot & \text{(falsum)} \\
 & \mid & \top & \text{(verum)} \\
 & \mid & P, \quad P \in \Pi & \text{(atomic formula)} \\
 & \mid & \neg F & \text{(negation)} \\
 & \mid & (F \wedge G) & \text{(conjunction)} \\
 & \mid & (F \vee G) & \text{(disjunction)} \\
 & \mid & (F \rightarrow G) & \text{(implication)} \\
 & \mid & (F \leftrightarrow G) & \text{(equivalence)}
\end{array}
$$

Notational Conventions

- We omit brackets according to the following rules:
  - $\neg \;>_p\; \vee \;>_p\; \wedge \;>_p\; \rightarrow \;>_p\; \leftrightarrow$ (binding precedences)
  - $\vee$ and $\wedge$ are associative
  - $\rightarrow$ is right-associative,
    i. e., $F \rightarrow G \rightarrow H$ means $F \rightarrow (G \rightarrow H)$.

1.2 Semantics

In *classical logic* (dating back to Aristoteles) there are "only" two truth values "true" and "false" which we shall denote, respectively, by 1 and 0.

There are *multi-valued logics* having more than two truth values.

Valuations

A propositional variable has no intrinsic meaning. The meaning of a propositional variable has to be defined by a valuation.

A Π-*valuation* is a map

$$\mathcal{A} : \Pi \rightarrow \{0, 1\}.$$

where $\{0, 1\}$ is the set of *truth values*.

Truth Value of a Formula in $\mathcal{A}$

Given a Π-valuation $\mathcal{A}$, the function $\mathcal{A}^* : \Sigma$-formulas $\to \{0,1\}$ is defined inductively over the structure of F as follows:

$$\begin{aligned}
\mathcal{A}^*(\bot) &= 0\\
\mathcal{A}^*(\top) &= 1\\
\mathcal{A}^*(P) &= \mathcal{A}(P)\\
\mathcal{A}^*(\neg F) &= \mathsf{B}_\neg(\mathcal{A}^*(F))\\
\mathcal{A}^*(F \rho G) &= \mathsf{B}_\rho(\mathcal{A}^*(F), \mathcal{A}^*(G))
\end{aligned}$$

where B_ρ is the Boolean function associated with ρ defined by the usual truth table.

For simplicity, we write $\mathcal{A}$ instead of $\mathcal{A}^*$.

We also write ρ instead of B_ρ, i. e., we use the same notation for a logical symbol and for its meaning (but remember that formally these are different things.)

1.3 Models, Validity, and Satisfiability

F is *valid* in $\mathcal{A}$ ($\mathcal{A}$ is a *model* of F; F holds under $\mathcal{A}$):

$$\mathcal{A} \models F \;:\Leftrightarrow\; \mathcal{A}(F) = 1$$

F is *valid* (or is a *tautology*):

$$\models F \;:\Leftrightarrow\; \mathcal{A} \models F \text{ for all } \Pi\text{-valuations } \mathcal{A}$$

F is called *satisfiable* if there exists an $\mathcal{A}$ such that $\mathcal{A} \models F$. Otherwise F is called *unsatisfiable* (or *contradictory*).

Entailment and Equivalence

F *entails* (*implies*) G (or G *is a consequence of* F), written $F \models G$, if for all Π-valuations $\mathcal{A}$, whenever $\mathcal{A} \models F$ then $\mathcal{A} \models G$.

F and G are called *equivalent*, written $F \models\!\!\!\!\mid G$, if for all Π-valuations $\mathcal{A}$ we have $\mathcal{A} \models F \Leftrightarrow \mathcal{A} \models G$.

Proposition 1.1 *$F \models G$ if and only if $\models (F \to G)$.(Proof follows)*

Proof. ($\Rightarrow$) Suppose that F entails G. Let $\mathcal{A}$ be an arbitrary Π-valuation. We have to show that $\mathcal{A} \models F \rightarrow G$. If $\mathcal{A}(F) = 1$, then $\mathcal{A}(G) = 1$ (since $F \models G$), and hence $\mathcal{A}(F \rightarrow G) = 1$. Otherwise $\mathcal{A}(F) = 0$, then $\mathcal{A}(F \rightarrow G) = \mathsf{B}_{\rightarrow}(0, \mathcal{A}(G)) = 1$ independently of $\mathcal{A}(G)$. In both cases, $\mathcal{A} \models F \rightarrow G$.

($\Leftarrow$) Suppose that F does not entail G. Then there exists a Π-valuation $\mathcal{A}$ such that $\mathcal{A} \models F$, but not $\mathcal{A} \models G$. Consequently, $\mathcal{A}(F \rightarrow G) = \mathsf{B}_{\rightarrow}(\mathcal{A}(F), \mathcal{A}(G)) = \mathsf{B}_{\rightarrow}(1, 0) = 0$, so $(F \rightarrow G)$ does not hold in $\mathcal{A}$. $\square$

Proposition 1.2 *$F \models\!\mid G$ if and only if $\models (F \leftrightarrow G)$.*

Proof. Follows from Prop. 1.1. $\square$

Extension to sets of formulas N in the "natural way":

$N \models F$ if for all Π-valuations $\mathcal{A}$:
if $\mathcal{A} \models G$ for all $G \in N$, then $\mathcal{A} \models F$.

Validity vs. Unsatisfiability

Validity and unsatisfiability are just two sides of the same medal as explained by the following proposition.

Proposition 1.3 *F is valid if and only if $\neg F$ is unsatisfiable.(Proof follows)*

Proof. ($\Rightarrow$) If F is valid, then $\mathcal{A}(F) = 1$ for every valuation $\mathcal{A}$. Hence $\mathcal{A}(\neg F) = \mathsf{B}_{\neg}(\mathcal{A}(F)) = \mathsf{B}_{\neg}(1) = 0$ for every valuation $\mathcal{A}$, so $\neg F$ is unsatisfiable.

($\Leftarrow$) Analogously. $\square$

Hence in order to design a theorem prover (validity checker) it is sufficient to design a checker for unsatisfiability.

In a similar way, entailment $N \models F$ can be reduced to unsatisfiability:

Proposition 1.4 *$N \models F$ if and only if $N \cup \{\neg F\}$ is unsatisfiable.*

Checking Unsatisfiability

Every formula F contains only finitely many propositional variables. Obviously, $\mathcal{A}(F)$ depends only on the values of those finitely many variables in F under $\mathcal{A}$.

If F contains n distinct propositional variables, then it is sufficient to check 2^n valuations to see whether F is satisfiable or not.
$\Rightarrow$ truth table.

So the satisfiability problem is clearly deciadable (but, by Cook's Theorem, NP-complete).

Nevertheless, in practice, there are (much) better methods than truth tables to check the satisfiability of a formula. (later more)

Substitution Theorem

Proposition 1.5 *Let F and G be equivalent formulas, let H be a formula in which F occurs as a subformula.*

Then H is equivalent to H' where H' is obtained from H by replacing the occurrence of the subformula F by G. (Notation: $H = H[F]$, $H' = H[G]$. Proof follows)

Proof. The proof proceeds by induction over the formula structure of H.

Each of the formulas $\bot$, $\top$, and P for $P \in \Pi$ contains only one subformula, namely itself. Hence, if $H = H[F]$ equals $\bot$, $\top$, or P, then $H = F$, $H' = G$, and H and H' are equivalent by assumption.

If $H = H_1 \wedge H_2$, then either F equals H (this case is treated as above), or F is a subformula of H_1 or H_2. Without loss of generality, assume that F is a subformula of H_1, so $H = H_1[F] \wedge H_2$. By the induction hypothesis, $H_1[F]$ and $H_1[G]$ are equivalent. Hence, for every valuation $\mathcal{A}$, $\mathcal{A}(H') = \mathcal{A}(H_1[G] \wedge H_2) = \mathcal{A}(H_1[G]) \wedge \mathcal{A}(H_2) = \mathcal{A}(H_1[F]) \wedge \mathcal{A}(H_2) = \mathcal{A}(H_1[F] \wedge H_2) = \mathcal{A}(H)$.

The other boolean connectives are handled analogously. $\square$

Some Important Equivalences

Proposition 1.6 *The following equivalences are valid for all formulas F, G, H:*

$$
\begin{aligned}
(F \wedge F) &\leftrightarrow F \\
(F \vee F) &\leftrightarrow F && \textit{(Idempotency)} \\
(F \wedge G) &\leftrightarrow (G \wedge F) \\
(F \vee G) &\leftrightarrow (G \vee F) && \textit{(Commutativity)} \\
(F \wedge (G \wedge H)) &\leftrightarrow ((F \wedge G) \wedge H) \\
(F \vee (G \vee H)) &\leftrightarrow ((F \vee G) \vee H) && \textit{(Associativity)} \\
(F \wedge (G \vee H)) &\leftrightarrow ((F \wedge G) \vee (F \wedge H)) \\
(F \vee (G \wedge H)) &\leftrightarrow ((F \vee G) \wedge (F \vee H)) && \textit{(Distributivity)}
\end{aligned}
$$

$$
\begin{aligned}
(F \wedge (F \vee G)) &\leftrightarrow F \\
(F \vee (F \wedge G)) &\leftrightarrow F && \textit{(Absorption)} \\
(\neg\neg F) &\leftrightarrow F && \textit{(Double Negation)} \\
\neg(F \wedge G) &\leftrightarrow (\neg F \vee \neg G) \\
\neg(F \vee G) &\leftrightarrow (\neg F \wedge \neg G) && \textit{(De Morgan's Laws)} \\
(F \wedge G) &\leftrightarrow F, \textit{ if } G \textit{ is a tautology} \\
(F \vee G) &\leftrightarrow \top, \textit{ if } G \textit{ is a tautology} \\
(F \wedge G) &\leftrightarrow \bot, \textit{ if } G \textit{ is unsatisfiable} \\
(F \vee G) &\leftrightarrow F, \textit{ if } G \textit{ is unsatisfiable} && \textit{(Tautology Laws)}
\end{aligned}
$$

$$
\begin{aligned}
(F \leftrightarrow G) &\leftrightarrow ((F \rightarrow G) \wedge (G \rightarrow F)) && \textit{(Equivalence)} \\
(F \rightarrow G) &\leftrightarrow (\neg F \vee G) && \textit{(Implication)}
\end{aligned}
$$

1.4 Normal Forms

We define *conjunctions* of formulas as follows:

$\bigwedge_{i=1}^{0} F_i = \top.$

$\bigwedge_{i=1}^{1} F_i = F_1.$

$\bigwedge_{i=1}^{n+1} F_i = \bigwedge_{i=1}^{n} F_i \wedge F_{n+1}.$

and analogously *disjunctions:*

$\bigvee_{i=1}^{0} F_i = \bot.$

$\bigvee_{i=1}^{1} F_i = F_1.$

$\bigvee_{i=1}^{n+1} F_i = \bigvee_{i=1}^{n} F_i \vee F_{n+1}.$

Literals and Clauses

A *literal* is either a propositional variable P or a negated propositional variable $\neg P$.

A *clause* is a (possibly empty) disjunction of literals.

CNF and DNF

A formula is in *conjunctive normal form (CNF, clause normal form),* if it is a conjunction of disjunctions of literals (or in other words, a conjunction of clauses).

A formula is in *disjunctive normal form (DNF),* if it is a disjunction of conjunctions of literals.

Warning: definitions in the literature differ:

are complementary literals permitted?
are duplicated literals permitted?
are empty disjunctions/conjunctions permitted?

Checking the validity of CNF formulas or the unsatisfiability of DNF formulas is easy:

A formula in CNF is valid, if and only if each of its disjunctions contains a pair of complementary literals P and $\neg P$.

Conversely, a formula in DNF is unsatisfiable, if and only if each of its conjunctions contains a pair of complementary literals P and $\neg P$.

On the other hand, checking the unsatisfiability of CNF formulas or the validity of DNF formulas is known to be coNP-complete.

Conversion to CNF/DNF

Proposition 1.7 *For every formula there is an equivalent formula in CNF (and also an equivalent formula in DNF).*

Proof. We consider the case of CNF.

Apply the following rules as long as possible (modulo associativity and commutativity of $\wedge$ and $\vee$):

Step 1: Eliminate equivalences:

$$(F \leftrightarrow G) \;\Rightarrow_K\; (F \rightarrow G) \wedge (G \rightarrow F)$$

Step 2: Eliminate implications:

$$(F \rightarrow G) \;\Rightarrow_K\; (\neg F \vee G)$$

Step 3: Push negations downward:

$$\begin{aligned} \neg(F \vee G) &\;\Rightarrow_K\; (\neg F \wedge \neg G) \\ \neg(F \wedge G) &\;\Rightarrow_K\; (\neg F \vee \neg G) \end{aligned}$$

Step 4: Eliminate multiple negations:

$$\neg\neg F \;\Rightarrow_K\; F$$

Step 5: Push disjunctions downward:

$$(F \wedge G) \vee H \;\Rightarrow_K\; (F \vee H) \wedge (G \vee H)$$

Step 6: Eliminate $\top$ and $\bot$:

$$\begin{aligned} (F \wedge \top) &\;\Rightarrow_K\; F \\ (F \wedge \bot) &\;\Rightarrow_K\; \bot \\ (F \vee \top) &\;\Rightarrow_K\; \top \\ (F \vee \bot) &\;\Rightarrow_K\; F \\ \neg\bot &\;\Rightarrow_K\; \top \\ \neg\top &\;\Rightarrow_K\; \bot \end{aligned}$$

Proving termination is easy for most of the steps; only step 3 and step 5 are a bit more complicated.

For step 3, we can prove termination in the following way: We define a function μ from formulas to positive integers such that $\mu(\bot) = \mu(\top) = \mu(P) = 1$, $\mu(\neg F) = 2\mu(F)$, $\mu(F \wedge G) = \mu(F \vee G) = \mu(F \rightarrow G) = \mu(F \leftrightarrow G) = \mu(F) + \mu(G) + 1$. Whenever a formula H' is the result of applying a rule of step 3 to a formula H, then $\mu(H) > \mu(H')$. Since μ takes only integer values and $\mu(H) \geq 1$ for all formulas H, step 3 must terminate.

Termination of step 5 is proved similarly using a function ν from formulas to positive integers such that $\nu(\bot) = \nu(\top) = \nu(P) = 1$, $\nu(\neg F) = \nu(F) + 1$, $\nu(F \wedge G) = \nu(F \rightarrow G) = \nu(F \leftrightarrow G) = \nu(F) + \nu(G) + 1$, and $\nu(F \vee G) = 2\nu(F)\nu(G)$. Again, if a formula H' is the result of applying a rule of step 5 to a formula H, then $\nu(H) > \nu(H')$. Since ν takes only integer values and Since $\nu(H) \geq 1$ for all formulas H, step 5 terminates, too.

The resulting formula is equivalent to the original one and in CNF.

The conversion of a formula to DNF works in the same way, except that disjunctions have to be pushed downward in step 5. □

Complexity

Conversion to CNF (or DNF) may produce a formula whose size is *exponential* in the size of the original one.

Satisfiability-preserving Transformations

The goal

"find a formula G in CNF such that $F \models\!\mid G$"

is unpractical.

But if we relax the requirement to

"find a formula G in CNF such that $F \models \bot \Leftrightarrow G \models \bot$"

we can get an efficient transformation.

Idea: A formula $F[F']$ is satisfiable if and only if $F[P] \wedge (P \leftrightarrow F')$ is satisfiable (where P is a new propositional variable that works as an abbreviation for F').

We can use this rule recursively for all subformulas in the original formula (this introduces a linear number of new propositional variables).

Conversion of the resulting formula to CNF increases the size only by an additional factor (each formula $P \leftrightarrow F'$ gives rise to at most one application of the distributivity law).

Optimized Transformations

A further improvement is possible by taking the *polarity* of the subformula F into account.

Assume that F contains neither $\rightarrow$ nor $\leftrightarrow$. A subformula F' of F has *positive polarity* in F, if it occurs below an even number of negation signs; it has *negative polarity* in F, if it occurs below an odd number of negation signs.

Proposition 1.8 *Let $F[F']$ be a formula containing neither $\rightarrow$ nor $\leftrightarrow$; let P be a propositional variable not occurring in $F[F']$.*

If F' has positive polarity in F, then $F[F']$ is satisfiable if and only if $F[P] \wedge (P \rightarrow F')$ is satisfiable.

If F' has negative polarity in F, then $F[F']$ is satisfiable if and only if $F[P] \wedge (F' \rightarrow P)$ is satisfiable.

Proof. Exercise. □