Part 1: Propositional Logic

Literature (also for first-order logic)

Schöning: Logik für Informatiker, Spektrum

Fitting: First-Order Logic and Automated Theorem Proving, Springer

Last time

1.1 Syntax

- Language
  - propositional variables
  - logical symbols
    $\Rightarrow$ Boolean combinations
- Propositional Formulae

1.2 Semantics

- Valuations
- Truth value of a formula in a valuation
- Models, Validity, and Satisfiability

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 iff 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 if for all Π-valuations $\mathcal{A}$ we have $\mathcal{A} \models F \Leftrightarrow \mathcal{A} \models G$.

Proposition 1.1:

F entails G iff $(F \rightarrow G)$ is valid

Proposition 1.2:

F and G are equivalent iff $(F \leftrightarrow G)$ is valid.

Entailment and Equivalence

Extension to sets of formulas N in the "natural way", e.g., $N \models F$ 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:

$$F \text{ valid} \Leftrightarrow \neg F \text{ unsatisfiable}$$

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

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

Validity vs. Unsatisfiability

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

Proposition 1.4:

$$N \models F \Leftrightarrow N \cup \neg F \text{ unsatisfiable}$$

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

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 decidable
(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)

Checking Unsatisfiability

The satisfiability problem is clearly decidable
(but, by Cook's Theorem, NP-complete).

For sets of propositional formulae of a certain type, satisfiability can be checked in polynomial time:

Examples: 2SAT, Horn-SAT (will be discussed in the exercises)

Dichotomy theorem. Schaefer [Schaefer, STOC 1978] identified six classes of sets S of Boolean formulae for which $SAT(S)$ is in PTIME. He proved that all other types of sets of formulae yield an NP-complete problem.

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: By induction over the formula structure of H.

Structural Induction

Goal: Prove a property P of propositional formulae

Prove that for every formula F, $P(F)$ holds.

Induction basis: Show that $P(F)$ holds for all $F \in \Pi \cup \{\top, \bot\}$

Let F be a formula (not in $\Pi \cup \{\top, \bot\}$).

Induction hypothesis: We assume that $P(G)$ holds for all strict subformulae G of F.

Induction step: Using the induction hypothesis, we show that $P(F)$ holds as well.
In order to prove that $P(F)$ holds we usually need to consider various cases (reflecting the way the formula F is built):

Case 1: $F = \neg G$
Case 2: $F = G_1 \wedge G_2$
Case 3: $F = G_1 \vee G_2$
Case 4: $F = G_1 \rightarrow G_2$
Case 5: $F = G_1 \leftrightarrow G_2$

Some Important Equivalences

Proposition 1.6:

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

$$(F \wedge F) \leftrightarrow F$$

$$(F \vee F) \leftrightarrow F \qquad \text{(Idempotency)}$$

$$(F \wedge G) \leftrightarrow (G \wedge F)$$

$$(F \vee G) \leftrightarrow (G \vee F) \qquad \text{(Commutativity)}$$

$$(F \wedge (G \wedge H)) \leftrightarrow ((F \wedge G) \wedge H)$$

$$(F \vee (G \vee H)) \leftrightarrow ((F \vee G) \vee H) \qquad \text{(Associativity)}$$

$$(F \wedge (G \vee H)) \leftrightarrow ((F \wedge G) \vee (F \wedge H))$$

$$(F \vee (G \wedge H)) \leftrightarrow ((F \vee G) \wedge (F \vee H)) \qquad \text{(Distributivity)}$$

Some Important Equivalences

Proposition 1.7:

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

$$(F \wedge (F \vee G)) \leftrightarrow F$$

$$(F \vee (F \wedge G)) \leftrightarrow F \qquad \text{(Absorption)}$$

$$(\neg\neg F) \leftrightarrow F \qquad \text{(Double Negation)}$$

$$\neg(F \wedge G) \leftrightarrow (\neg F \vee \neg G)$$

$$\neg(F \vee G) \leftrightarrow (\neg F \wedge \neg G) \qquad \text{(De Morgan's Laws)}$$

$(F \wedge G) \leftrightarrow F$, if G is a tautology

$(F \vee G) \leftrightarrow \top$, if G is a tautology (Tautology Laws)

$(F \wedge G) \leftrightarrow \bot$, if G is unsatisfiable

$(F \vee G) \leftrightarrow F$, if G is unsatisfiable (Tautology Laws)

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.

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.

Example of clauses:

$\bot$	the empty clause
P	positive unit clause
$\neg P$	negative unit clause
$P \vee Q \vee R$	positive clause
$P \vee \neg Q \vee \neg R$	clause
$P \vee P \vee \neg Q \vee \neg R \vee R$	allow repetitions/complementary 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?

CNF and DNF

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 1.8:

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:

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

Conversion to CNF/DNF

Step 2: Eliminate implications:

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

Step 3: Push negations downward:

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

Step 4: Eliminate multiple negations:

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

The formula obtained from a formula F after applying steps 1-4 is called the negation normal form (NNF) of F

Conversion to CNF/DNF

Step 5: Push disjunctions downward:

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

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

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

Conversion to CNF/DNF

Proving termination is easy for most of the steps; only steps 1, 3 and 5 are a bit more complicated.

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.