Propositional Logic, Part II

CS 3234: Logic and Formal Systems

Martin Henz and Aquinas Hobor

September 2, 2010

Generated on Thursday 2 September, 2010, 18:46

1 Remaining Questions

Formulas in propositional logic can be categorized into the following three classes:

- valid (hold under all valuations),
- unsatisfiable (hold under no valuation),
- invalid but satisfiable (holds under some, but not all valuations).

The question remains, however, how to *decide* whether a given formula is valid or satisfiable. Another important question asks whether two formulas are equivalent, i.e. whether one evaluates to T under exactly the same valuations under which the other evaluates to T.

Definition 1. *A* decision problem *is a question in some formal system with a yes-or-no answer.*

Example 1. *The question whether a given propositional formula is satisisfiable (unsatisfiable, valid, invalid) is a decision problem.*

Example 2. *The question whether two given propositional formulas are equivalent is also a decision problem.*

Clearly, the truth table method allows us to answer these questions. If the last column of the truth table of a formula has only T, then the formula is valid, if it has only F, it is unsatisfiable. If the truth tables of two formulas have the same last column (provided the order of valuations of propositional atoms is the same), then the two formulas are equivalent.

Thus, using a truth table, we can implement an *algorithm* that returns *"yes"* if the formula is satisifiable, and that returns *"no"* if the formula is unsatisfiable.

Definition 2. *Decision problems for which there is an algorithm computing "yes" whenever the answer is "yes", and "no" whenever the answer is "no", are called* decidable*.*

So the good news is that the questions of validity, satisfiability and equivalence of formulas of propositional logic are decidable. However, propositional formulas can be large, so the practical question remains whether we can answer these questions *efficiently*.

Proving satisfiability/validity using truth tables or natural deduction is impractical for large formulas, since they always require an effort that grows exponentially with the number of variables that occur in the formula.

Is there a *practical* way of deciding satisfiability? Is there an *efficient* algorithm that decides whether a given formula is satisfiable? More precisely, is there a *polynomial-time* algorithm that decides whether a given formula is satisfiable?

2 Complexity of Satisfiability

The unfortunate answer to this question is: We do not know (as of 2010)! However, what we do know is that a satisfiable formula can be quickly demonstrated to be satisfiable. We only need to produce a valuation that makes the formula true. The evaluation of the formula with respect to the valuation can be done in a time proportional to the size of the formula.

Such a proof for the affirmative answer of a decision problem is called a *witness*.

Definition 3. *Decision problems for which the "yes" answer has a proof that can be checked in polynomial time, are called* NP[1]*.*

The class NP was introduced by Stephen Cook in 1971 at the 3rd Annual ACM Symposium on Theory of Computing. At the conference, there was a fierce debate whether there could be a polynomial time algorithm to solve such problems. John Hopcroft convinced the delegates that the problem should be put off to be solved at some later date. In 1972, Richard Karp presented 21 mutually equivalent problems, for which no polynomial time algorithms was known. Cook and Leonid Levin proved independently that propositional satisifiability is in this class, which was dubbed *NP-complete*.

The class of problems for which a polynomial algorithm exists, is called P. The question naturally arises then, whether there is a polynomial algorithm for an NP-complete problem. All NP-complete problems are equivalent (more precisely they can be translated to each other in polynomial time), and thus if there was a polynomial time algorithm for any NP-complete problem, they would all be in P. So the question can be stated as: "$P = \mathrm{NP}$?"

[1]NP stands for "**N**on-deterministic **P**olynomial time", a terminology that stems from an alternative definition of the class, which defines NP to be the set of decision problems solvable in polynomial time by a non-deterministic Turing machine.

To date no proof has been found for either $P = \text{NP}$ or $P \neq \text{NP}$. Many computer scientists assume $P \neq \text{NP}$, and therefore consider the NP-complete problems as "intractable". Over the years, many "proofs" for one or the other answer have been proposed, and subsequently rejected, most recently by Vinay Deolalikar (a researcher at HP), in August 2010.

3 Conjunctive Normal Form

Definition 4. *A literal L is either an atom p or the negation of an atom $\neg p$. A formula C is in* conjunctive normal form *(CNF) if it is a conjunction of clauses, where each clause is a disjunction of literals:*

$$\begin{array}{rcl} L & ::= & p | \neg p \\ D & ::= & L | L \vee D \\ C & ::= & D | D \wedge C \end{array}$$

Example 3. $(\neg p \vee q \vee r) \wedge (\neg q \vee r) \wedge (\neg r)$ *is in CNF.*

Example 4. $(\neg p \vee q \vee r) \wedge ((p \wedge \neg q) \vee r) \wedge (\neg r)$ *is not in CNF.*

Example 5. $(\neg p \vee q \vee r) \wedge \neg(\neg q \vee r) \wedge (\neg r)$ *is not in CNF.*

Lemma 1. *A disjunction of literals $L_1 \vee L_2 \vee \cdots \vee L_m$ is valid iff there are $1 \leq i, j \leq m$ such that L_i is $\neg L_j$.*

Example 6. *In order to disprove*

$$\models (\neg q \vee p \vee r) \wedge (\neg p \vee r) \wedge q$$

we only need to disprove any of:

$$\models (\neg q \vee p \vee r) \qquad \models (\neg p \vee r) \qquad \models q$$

Example 7. *In order to prove*

$$\models (\neg q \vee p \vee q) \wedge (p \vee r \vee \neg p) \wedge (r \vee \neg r)$$

we need to prove all of:

$$\models (\neg q \vee p \vee q) \qquad \models (p \vee r \neg p) \qquad \models (r \vee \neg r)$$

Clearly, the definitions of satisfiability and validity imply that a formula ϕ is satisfiable iff $\neg\phi$ is not valid. Thus, we can test satisfiability of ϕ by transforming $\neg\phi$ into CNF, and show that some clause is not valid.

4 Transformation to CNF

Theorem 1. *Every formula in the propositional calculus can be transformed into an equivalent formula in CNF.*

Proof. The proof of this theorem consists of an algorithm that transforms arbitrary formulas into CNF. It proceeds as follows:

1. Eliminate implication using:
   $A \rightarrow B \equiv \neg A \vee B$
2. Push all negations inward using De Morgan's laws:
   $\neg(A \wedge B) \equiv (\neg A \vee \neg B)$
   $\neg(A \vee B) \equiv (\neg A \wedge \neg B)$
3. Eliminate double negations using the equivalence $\neg\neg A \equiv A$
4. The formula now consists of disjunctions and conjunctions of literals. Use the distributive laws
   $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$
   $(A \wedge B) \vee C \equiv (A \vee C) \wedge (B \vee C)$
   to eliminate conjunctions within disjunctions.

$\square$

Example 8.

$$\begin{aligned}
(\neg p \rightarrow \neg q) \rightarrow (p \rightarrow q) &\equiv \neg(\neg\neg p \vee \neg q) \vee (\neg p \vee q) \\
&\equiv (\neg\neg\neg p \wedge q) \vee (\neg p \vee q) \\
&\equiv (\neg p \wedge q) \vee (\neg p \vee q) \\
&\equiv (\neg p \vee \neg p \vee q) \wedge (q \vee \neg p \vee q)
\end{aligned}$$

5 Overview of Algorithms for Satisfiability

The algorithms for proving satisfiability of a formula ψ can be broadly categorized into the following classes:

- CNF-based: Transform $\neg\psi$ into Conjunctive Normal Form *ncnf* and prove validity (non-validity) of *ncnf*
- Transform ψ into Conjunctive Normal Form *cnf* and search for a satisfying valuation
- Search-based algorithms: Exhaustively search a tree of valuations, until a satisfying valuation is found. One class of such algorithms is called DPLL (Davis-Putnam-Logemann-Loveland), instances of which are among the most efficient solvers for the satisfiability problem.

- Incomplete algorithms: Such algorithms return “yes” for some satisfiable formulas, and run forever for other satisfiable formulas and all unsatisfiable formulas. Often, these algorithms perform *local search*, repeatedly trying to improve a satisfying valuation, starting from a computed or randomly generated initial valuation. an example of such an algorithm is WalkSAT.

- Propagation-based algorithms: Use propagation rules to make implicit logic information explicit. The outcome of such an algorithm can be “yes” for proving satisfiability “no” for proving unsatisfiability, and “don’t know” for cases where this method does not find a proof.