Formal Verification Methods 1: Propositional Logic

John Harrison

Intel Corporation

- Course overview
- Propositional logic
- A resurgence of interest
- Logic and circuits
- Normal forms
- The Davis-Putnam procedure
- Stålmarck's method
- Conclusions

Overview

We aim to give a broad overview of the current verification methods employed in the hardware industry.

1. Propositional Logic
2. Symbolic Simulation
3. Model Checking
4. General Theorem Proving
5. Floating Point Verification

We start with the 'simplest' logic (propositional logic) and work our way up to higher order logic.

The last lecture focuses on our own work, verifying floating-point algorithms using the HOL higher order logic theorem prover.

Propositional Logic

By the second week of this summer school, we probably all know what propositional logic is.

English	Standard	Boolean	Other
false	$\bot$	0	F
true	$\top$	1	T
not p	$\neg p$	$\overline{p}$	$-p$, $\sim p$
p and q	$p \wedge q$	pq	$p\&q$, $p \cdot q$
p or q	$p \vee q$	$p + q$	$p \mid q$, p *or* q
p implies q	$p \Rightarrow q$	$p \leq q$	$p \rightarrow q$, $p \supset q$
p iff q	$p \Leftrightarrow q$	$p = q$	$p \equiv q$, $p \sim q$

In the context of circuits, it's often referred to as 'Boolean algebra', and many designers use the Boolean notation.

A resurgence of interest!

Traditionally, propositional logic has been regarded as fairly boring, and is usually regarded as a stepping-stone on the way to first order logic (and beyond).

- There are severe limitations to what can be said with propositional logic.
- Propositional logic is trivially decidable in theory ...
- ... but the usual methods aren't efficient enough for interesting problems.

However, the last decade has seen a remarkable upsurge of interest in propositional logic.

In fact, it's arguably the hottest topic in automated theorem proving!

Why?

Why the resurgence?

- There *are* many interesting problems that can be expressed in propositional logic
- Efficient algorithms *can* often decide large, interesting problems

Propositional satisfiability was the original NP-complete problem.

The theory of NP completeness shows that many difficult combinatorial problems can in principle be reduced to propositional satisfiability checking.

Recently it has become clear that reducing problems to propositional logic can often be a good way to solve them in practice!

Logic and circuits

The correspondence between digital logic circuits and propositional logic has been known for a long time.

Digital design	Propositional Logic
circuit	formula
logic gate	propositional connective
input wire	atom
internal wire	subexpression
voltage level	truth value

Many problems in circuit design and verification can be reduced to propositional tautology or satisfiability checking.

For example optimization correctess: $\phi \Leftrightarrow \phi'$ is a tautology.

Encoding as SAT

Many other apparently difficult combinatorial problems can be encoded as Boolean satisfiability (SAT), e.g. scheduling, planning.

Using circuit representations for multipliers, we can encode factorization problems as Boolean satisfiability. Here's '6 is a prime number':

$$
\begin{aligned}
\neg(\quad & (out_0 \Leftrightarrow x_0 \wedge y_0) \wedge \\
& (out_1 \Leftrightarrow (x_0 \wedge y_1 \Leftrightarrow \neg(x_1 \wedge y_0))) \wedge \\
& (v_2^2 \Leftrightarrow (x_0 \wedge y_1) \wedge x_1 \wedge y_0) \wedge \\
& (u_2^0 \Leftrightarrow ((x_1 \wedge y_1) \Leftrightarrow \neg v_2^2)) \wedge \\
& (u_2^1 \Leftrightarrow (x_1 \wedge y_1) \wedge v_2^2) \wedge \\
& (out_2 \Leftrightarrow u_2^0) \wedge (out_3 \Leftrightarrow u_2^1) \wedge \\
& \neg out_0 \wedge out_1 \wedge out_2 \wedge \neg out_3)
\end{aligned}
$$

We can read off the factorization $6 = 2 \times 3$ from a refuting assignment.

Efficient methods

The naive truth table method is quite impractical for formulas with more than a dozen primitive propositions.

Practical use of propositional logic mostly relies on one of the following algorithms for deciding tautology or satisfiability:

- Binary decision diagrams (BDDs)
- The Davis-Putnam method (DP, DPLL)
- Stålmarck's method

BDDs will be discussed in the next lecture. This time we focus on Davis-Putnam, while also explaining the basic idea of Stålmarck's method.

DP and DPLL

Actually, the original Davis-Putnam procedure is not much used now.

What is usually called the Davis-Putnam method is actually a later refinement due to Davis, Loveland and Logemann (hence DPLL).

We formulate it as a test for *satisfiability.* It has three main components:

- Transformation to conjunctive normal form (CNF)
- Application of simplification rules
- Splitting

Normal forms

In ordinary algebra we can reach a 'sum of products' form of an expression by:

- Eliminating operations other than addition, multiplication and negation, e.g. $x - y \mapsto x + -y$.
- Pushing negations inwards, e.g. $-(-x) \mapsto x$ and $-(x + y) \mapsto -x + -y$.
- Distributing multiplication over addition, e.g. $x(y + z) \mapsto xy + xz$.

In logic we can do exactly the same, e.g. $p \Rightarrow q \mapsto \neg p \vee q$, $\neg(p \wedge q) \mapsto \neg p \vee \neg q$ and $p \wedge (q \vee r) \mapsto (p \wedge q) \vee (p \wedge r)$.

The first two steps give 'negation normal form' (NNF).

Following with the last (distribution) step gives 'disjunctive normal form' (DNF), analogous to a sum-of-products.

Conjunctive normal form

Conjunctive normal form (CNF) is the dual of DNF, where we reverse the roles of 'and' and 'or' in the distribution step to reach a 'product of sums':

$$\begin{array}{rcl} p \lor (q \land r) & \mapsto & (p \lor q) \land (p \lor r) \\ (p \land q) \lor r & \mapsto & (p \lor r) \land (q \lor r) \end{array}$$

Reaching such a CNF is the first step of the Davis-Putnam procedure.

Unfortunately the naive distribution algorithm can cause the size of the formula to grow exponentially — not a good start. Consider for example:

$$(p_1 \land p_2 \land \cdots \land p_n) \lor (q_1 \land p_2 \land \cdots \land q_n)$$

Definitional CNF

A cleverer approach is to introduce new variables to stand for subformulas.

Although this isn't logically equivalent, it does preserve satisfiability. For example, we can go from:

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

introduce new variables for subformulas:

$$(p_1 \Leftrightarrow q \wedge \neg r) \wedge$$

$$(p_2 \Leftrightarrow p \vee p_1) \wedge$$

$$(p_3 \Leftrightarrow p_2 \wedge s) \wedge$$

$$p_3$$

then transform to (3-)CNF in the usual way:

$$(\neg p_1 \vee q) \wedge (\neg p_1 \vee \neg r) \wedge (p_1 \vee \neg q \vee r) \wedge$$

$$(\neg p_2 \vee p \vee p_1) \wedge (p_2 \vee \neg p) \wedge (p_2 \vee \neg p_1) \wedge$$

$$(\neg p_3 \vee p_2) \wedge (\neg p_3 \vee s) \wedge (p_3 \vee \neg p_2 \vee \neg s) \wedge$$

$$p_3$$

Clausal form

It's convenient to think of the CNF form as a set of sets:

- Each disjunction $p_1 \vee \cdots \vee p_n$ is thought of as the set $\{p_1, \ldots, p_n\}$, called a *clause*.
- The overall formula, a conjunction of clauses $C_1 \wedge \cdots \wedge C_m$ is though of as a set $\{C_1, \ldots, C_m\}$.

Since 'and' and 'or' are associative, commutative and idempotent, nothing of logical significance is lost in this interpretation.

Special cases: an empty clause means $\bot$ (and is hence unsatisfiable) and an empty set of clauses means $\top$ (and is hence satisfiable).

Simplification rules

At the core of the Davis-Putnam method are two transformations on the set of clauses:

I The 1-literal rule: if a unit clause p appears, remove $\neg p$ from other clauses and remove all clauses including p.

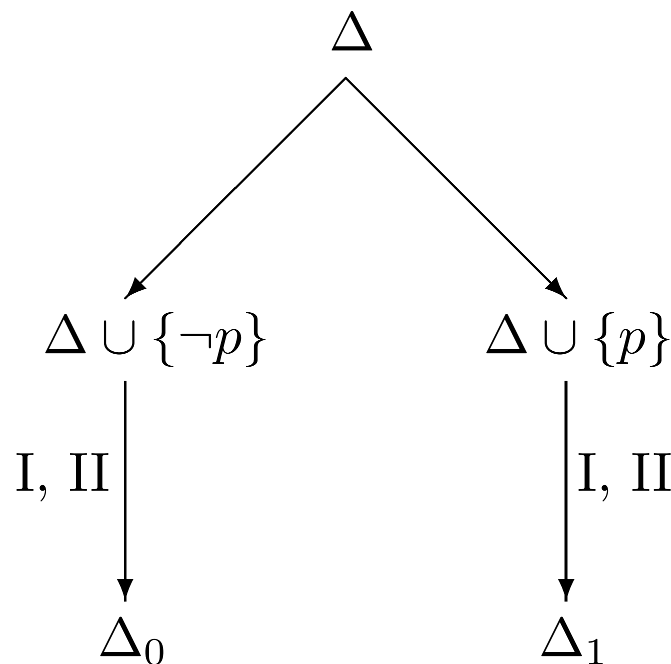
II The affirmative-negative rule: if p occurs *only* negated, or *only* unnegated, delete all clauses involving p.

These both preserve satisfiability of the set of clause sets.

Splitting

In general, the simplification rules will not lead to a conclusion. We need to perform case splits.

Given a clause set Δ, simply choose a variable p, and consider the two new sets $\Delta \cup \{p\}$ and $\Delta \cup \{\neg p\}$.



In general, these case-splits need to be nested, and in the worst case, behaviour is exponential.

But usually, performing the intermediate simplifications between case splits makes performance much better than with truth tables.

Industrial strength SAT solvers

For big applications, there are several important tweaks to the basic DPLL algorithm:

- Highly efficient data structures
- Good heuristics for picking 'split' variables
- Intelligent non-chronological backtracking / conflict clauses

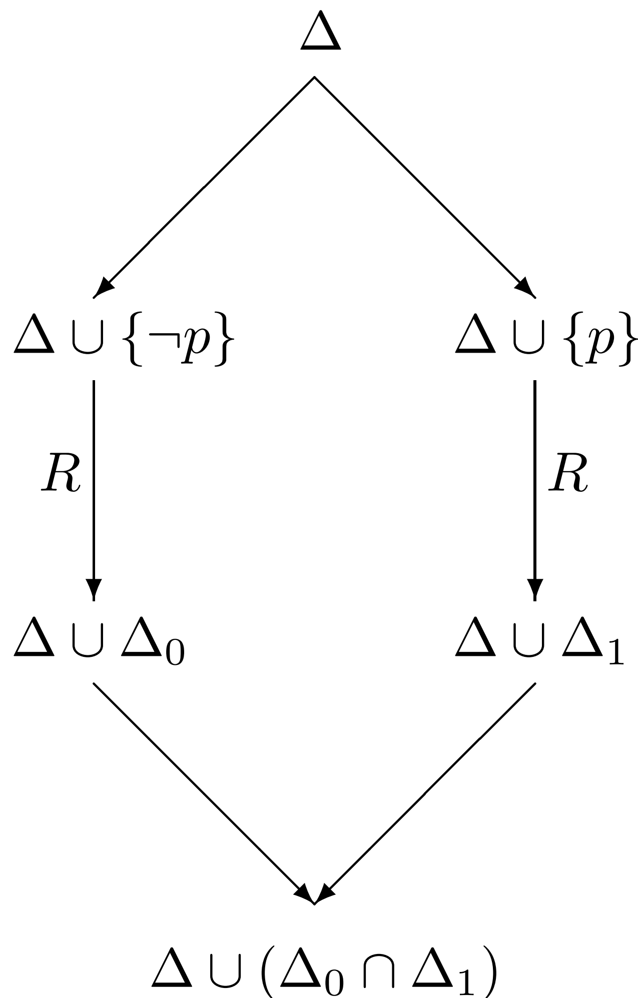
Some well-known provers are

- GRASP
- SATO
- Chaff

Chaff pays close attention to low-level details like memory hierarchy, and seems to be the current favourite.

Stålmarck's algorithm

Stålmarck's 'dilemma' rule attempts to avoid nested case splits by feeding back common information from both branches.



This and other algorithms are used in a successful commercial tool suite marketed by Prover Technology.

Summary

- Propositional logic is no longer the ugly sister of theorem proving
- A wide variety of practical problems can usefully be encoded in SAT
- There is intense interest in efficient algorithms for SAT
- Many of the most successful systems are still based on minor refinements of the ancient Davis-Putnam procedure
- Can we invent a better SAT algorithm?