Prace Naukowe Wyższej Szkoły Pedagogicznej w Częstochowie
Matematyka VIII, Częstochowa 2000 - 2001

Fast Automatic Deduction

Lidia Stępień

Abstract

In this note we present algorithm of fast checking tautology and/or satisfiability for the classical propositional logic, which is due to Martin Davis and Hilary Putnam. A new wave of interest in classical logic has been stimulated by recent impressive technological advances, particularly the significant increase of computers' capacity and speed. It opens entirely new perspectives of applications of classical logic on industrial scale, because many important technical and hardware problems can be expressed in the form of propositional formulas, often of length well exceeding human potential to deal with them. One of the most important tasks is to check quickly if a given (sometimes excessively long) formula is a tautology. Fast tautology/satisfiability verification relies not only on hardware, but first of all on good choice of feasible cases and fast algorithms.

I Introduction.

Testing tautology/satisfiability for classical propositional logic is one of the well known *NP - complete problem (i.e., ,,nondeterministic polynomial time").* The complexity of the classical methods, like the truth tables, is exponential in the size of an input formula. There is no known general algorithm that can verify the tautology/satisfiability problem much faster. (The interested reader is referred to [6]) But in certain simple cases, fast algorithm for testing tautology/satisfiability give a derivation in linear or polynomial time.

Some effective algorithms for testing tautology/satisfiability are: the Davis-Putnam procedure [5][4], SATO [14], the Stålmarck method [11] and GRASP [9] (Generic seaRch Algorithm for the Satisfiability Problem). In this paper we present the Davis-Putnam procedure, because which has long been a major practical tool for solving satisfiability problems. The other algorithms mentioned above are recent modifications of that one. Those methods are used in many situations in mathematics, computer science and in verification of large scale industrial systems. (e. g., see, Birnbaum and Lozinskii [2], Urbie and Stickel [13], SYRF [12], Borälv and Stålmarck [3],

Silva and Sakallah [10])

II Propositional Logic.

First we recall some notions of propositional logic which are used in the next section. After syntax and semantic we present the axiom system (following A. Grzegorczyk [7]) with Modus Ponens rule as the only rule of derivation. For this system we formulate the completness theorem. We don't supply a proof of this theorem because it is not a goal of this paper. Reader can find it in Bell and Slomson [1].

1. Syntax:
   - $p, q, r, \ldots$ is a infinite list of propositional letters
   - $\neg, \vee, \wedge, \rightarrow, \equiv$ are logical connectives
   - $0, 1$ two constans.

   The propositional formulas form the smallest set such that:
   - All atomic formulas are the formulas.
   - If α is a formula then $\neg\alpha$ is a formula.
   - If α and β are formulas then $\alpha \vee \beta, \alpha \wedge \beta, \alpha \rightarrow \beta, \alpha \equiv \beta$ are formulas.
2. Truth values:
   $$Tr = \{f, t\}$$
3. Semantics:

   A valuation is a mapping $v : F \rightarrow Tr$ meeting the conditions:
   - $v(0) = f; \qquad v(1) = t$
   - $v(\neg\alpha) = \neg v(\alpha)$
   - $v(\alpha \bullet \beta) = v(\alpha) \bullet v(\beta)$, where „$\bullet$" is a one of the binary connectives.

A propositional formula α is a *tautology* if $v(\alpha) = t$, for every valuation v. $\models \alpha$ denote that a propositional formula α is a *tautology*.

A propositional formula α is *satisfiable* if $v(\alpha) = t$, for some valuation v.

The axioms of Classical Propositional Logic

(A1) $p \rightarrow (q \rightarrow p)$

(A2) $(p \rightarrow (q \rightarrow r)) \rightarrow ((p \rightarrow q) \rightarrow (p \rightarrow r))$

(A3) $(p \equiv q) \rightarrow (p \rightarrow q)$

(A4) $(p \equiv q) \rightarrow (q \rightarrow p)$

(A5) $(p \rightarrow q) \rightarrow ((q \rightarrow p) \rightarrow (p \equiv q))$

(A6) $(p \vee q) \rightarrow (q \vee p)$

(A7) $(p \wedge q) \rightarrow (q \wedge p)$

(A8) $(p \wedge q) \rightarrow p$

(A9) $p \rightarrow (p \vee q)$

(A10) $p \rightarrow (q \rightarrow (p \wedge q))$

(A11) $((p \rightarrow r) \wedge (q \rightarrow r)) \rightarrow ((p \vee q) \rightarrow r)$

(A12) $(p \rightarrow (q \wedge \neg q)) \rightarrow \neg p$

(A13) $(p \wedge \neg p) \rightarrow q$

Modus Ponens

$$\text{(MP)} \quad \frac{\alpha, \alpha \rightarrow \beta}{\beta}$$

A *proof* of a formula α is a finite sequence $\alpha_1, \ldots, \alpha_n$, of formulas such that $\alpha_n = \alpha$ and for each $j \leq n$:

1) α_j is either an axiom or

2) for some $i < j$ and $k < j, \alpha_j$ is an immediate consequence of α_i and α_k according to the rule modus ponens (MP).

$\vdash \alpha$ denote that exists a proof of α.

The Completeness Theorem $\models \alpha \Leftrightarrow \vdash \alpha$.

4. Normal Forms

Def.1

$\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n$ is the *disjunction* of propositional formulas $\alpha_1, \alpha_2, \ldots, \alpha_n$,

$\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n$ is the *conjunction* of propositional formulas $\alpha_1, \alpha_2, \ldots, \alpha_n$.

Remarks:

1. We denote a disjunction $\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n$ by $[\alpha_1, \alpha_2, \ldots, \alpha_n]$
2. We denote a conjunction $\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n$ by $< \alpha_1, \alpha_2, \ldots, \alpha_n >$
3. If v is a valuation we get:

$v(\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n) = t$ if v maps some member of the list $\alpha_1, \alpha_2, \ldots, \alpha_n$ to t,

$v(\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n) = f$ otherwise;

$v(\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n) = t$ if v maps every member of the list $\alpha_1, \alpha_2, \ldots, \alpha_n$ to t,

$v(\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n) = f$ otherwise.

4. Empty Lists

$v([]) = f$ and $v(<>) = t$ so that $[] \equiv 0$ and $<> \equiv 1$ are tautologies.

Def.2 A *literal* L is a propositional letter or the negation of a propositional letter, or a constant 0 or 1.

Def.3 A propositional formula is in *the conjunctive normal form (CNF)* if it is a conjunction $< D_1, D_2, \ldots, D_n >$ in which each member is a disjunction of literals.

A propositional formula is in *the disjunctive normal form (DNF)* if it is a disjunction $[C_1, C_2, \ldots, C_n]$ in which each member is a conjunction of literals.

II The Davis - Putnam Procedure.

The Davis - Putnam Procedure was introduced in 1960 [5]. The propositional version is still among the fastest of the automatic theorem-proving technique. The Davis - Putnam procedure is a refutation method. (following [6]) To prove α, start with $\neg\alpha$ and derive a contradiction. The first phase is: a conversion to *the conjunctive normal form* and then test the unsatisfiability of a CNF.

First we introduce some basic notions.

Def. 1 A *clause* is a disjunction of literals.

Def. 2 A *block* is a disjunction of CNF.

Def. 3 For a propositional letter $\mathbf{p}$, we set $\overline{\mathbf{p}} = \neg\mathbf{p}$ and $\overline{\neg\mathbf{p}} = \mathbf{p}$. The literals $\mathbf{L}$ and $\overline{\mathbf{L}}$ are *complementary literals.*

Def.4 A clause $\mathbf{C_1}$ subsumes a clause $\mathbf{C_2}$ if every literal in $\mathbf{C_1}$ also occurs in $\mathbf{C_2}$.

There are two families of rules. The first are preliminary rules that are not strictly necessary but that can considerably speed up later steps. Then there are the primary rules that are the essence of the Davis-Putnam procedure.

Preliminary Rules:

Preliminary Step 1 (PS1) Remove any repetitions from the clauses in a block, and (maybe) arrange the literals into some standard order.

Preliminary Step 2 (PS2) Delete any clause that contains both a literal and its complement. Delete any clause that contains 1. Delete every occurence of 0.

Primary Rules:

One-Literal Rule (OLR) Suppose B is a block containing a formula S in the CNF, and S in turn contains the one -literal clause $[\mathbf{L}]$. Modify B by changing S as follows: Remove from S all clauses containing $\mathbf{L}$, and delete all occurrences of $\overline{\mathbf{L}}$ from the remaining clauses of S.

Affirmative-Negative Rule (ANR) Suppose B is a block containing a formula S in the CNF, some clauses in S contain the literal $\mathbf{L}$, and no clauses in S contain $\overline{\mathbf{L}}$. Modify B by removing from S all clauses containing $\mathbf{L}$.

Subsumption Rule (SR) Suppose B is a block containing a formula S in the CNF, and S contains clauses $\mathbf{C}_1$ and $\mathbf{C}_2$ where $\mathbf{C}_1$ subsumes $\mathbf{C}_2$. Modify B by removing clause $\mathbf{C}_2$ from S.

Splitting Rule (SpR) Suppose B is a block containing a formula S in the CNF, some clauses in S contain the literal $\mathbf{L}$ while others contain $\overline{\mathbf{L}}$. (There may also be clauses with neither.) Let $\mathbf{S_L}$ be the formula in the CNF that results when all clauses in S containing $\mathbf{L}$ are removed, and all occurrences of $\overline{\mathbf{L}}$ are deleted. Likewise, $\mathbf{S}_{\overline{\mathbf{L}}}$ be the formula in the CNF that results when all clauses in S containing $\overline{\mathbf{L}}$ are removed, and all occurrences of $\mathbf{L}$ are deleted. Modify B by replacing the formula S in the CNF by the two formulas $\mathbf{S_L}$ and $\mathbf{S}_{\overline{\mathbf{L}}}$ in the CNF.

We define a Davis-Putnam derivation:

Def.4 Let B be a block. A *Davis-Putnam derivation* for B is a finite sequence of blocks $B_1, B_2, \ldots, B_n$, where $B_1 = B$, and otherwise each block in the sequence comes from its predecessor using one of the four rules. A derivation *succeeds* if it ends with a block in which each formula in the CNF contains the empty clause. A derivation *fails* if it ends with a blocks in which some formula in the CNF itself is empty.

To *prove* a formula α in this system, begin with $\neg\alpha$, convert this to a formula S in the CNF, form the bloc $[S]$, perhaps simplify this using the two preliminary rules, then show there is a Davis-Putnam derivation that succeeds.

Now we prove that an application one of primary rules will not affect the satisfiability of a block.

Lemma 1 S is satisfiable if and only if S^* is satisfiable, where S^* is obtained from S by a single application one of the primary rules.

Proof:

Case 1: *One-Literal Rule*

Let S be the formula in the CNF.

$$S =< [L], [L|D_1], \ldots, [L|D_n], [\neg L|E_1], \ldots, [\neg L|E_j], F_1, \ldots, F_k >$$

Let S^* be the formula after application the One-Literal Rule to S.

$$S^* =< E_1, \ldots, E_j, F_1, \ldots, F_k > .$$

„⇒" Suppose that S is satisfiable. Then exists a valuation v such that $v : S \to \{t\}$. Then in particular: $v(L) = t, v([L]) = t, v([L|D_1]) = t, \ldots, v([L|D_n]) = t, v([\neg L|E_1]) = t$, but $v(\neg L) = f$, it must be that $v(E_1) = t, \ldots$ similarly $v([\neg L|E_j]) = t$ hence $v(E_j) = t$, $v(F_1) = t, \ldots, v(F_k) = t$.

Hence, each member of S^* maps to t under v, so S^* is satisfiable.

„⇐" Suppose that S^* is satisfiable. Then exists a valuation v such that $v : S^* \to \{t\}$. We define a new valuation w.

$$w = \begin{cases} v(\mathbf{p}) & \text{for every propositional letter } \mathbf{p} \neq \mathbf{L} \\ \mathbf{t} & \text{for } \mathbf{L} \end{cases}$$

Hence, $w(F_1) = v(F_1) = t, \ldots, w(F_k) = v(F_k) = t$. Since $w(L) = t, w([L]) = t$ and hence $w([L|D_1]) = t, \ldots, w([L|D_n]) = t$. Since $w(E_1) = v(E_1) = t$ and hence $w([\neg L|E_1]) = t, \ldots$, similarly $w([\neg L|E_j]) = t$ because $w(E_j) = v(E_j) = t$.

Thus, w maps every member of S to t, so S is satisfiable.

Case 2: *The Affirmative - Negative Rule*

Let S be the formula in the CNF.

$$S =< [L], [L|D_1], \ldots, [L|D_n], E_1, \ldots, E_k >$$

Let S^* be the formula after application the Affirmative-Negative Rule to S.

$$S^* =< E_1, \ldots, E_k > .$$

„⇒" Suppose that S is satisfiable. Then exists a valuation v such that $v : S \to \{t\}$. Then in particular: $v(L) = t, v([L]) = t$, $v([L|D_1]) = t, \ldots, v([L|D_n]) = t, v(E_1) = t, \ldots, v(E_k) = t$.

Hence, each member of S^* maps to t under v, so S^* is satisfiable.

„⇐" Suppose that S^* is satisfiable. Then exists a valuation v such that $v : S^* \to \{t\}$. We define a new valuation w.

$$w = \begin{cases} v(\mathbf{p}) & \text{for every propositional letter } p \neq L \\ t & \text{for } L \end{cases}$$

Hence, $w(E_1) = v(E_1) = t, \ldots, w(E_k) = v(E_k) = t$. Since $w(L) = t, w([L]) = t$ and hence $w([L|D_1]) = t, \ldots, w([L|D_n]) = t$.

Thus, w maps every member of S to t, so S is satisfiable.

Case 3: *Subsumption Rule*

Let S be the formula in the CNF.

$$S =< [L|D], [L|D|E], F_1, \ldots, F_k >$$

Let S^* be the formula after application the Subsumption Rule to S.

$$S^* =< [L|D], F_1, \ldots, F_k > .$$

„⇒" Suppose that S is satisfiable. Then exists a valuation v such that $v : S \to \{t\}$. Then in particular: $v([L|D]) = t, v([L|D|E]) = t, v(F_1) = t$, $\ldots, v(F_k) = t$.

Hence, each member of S^* maps to t under v, so S^* is satisfiable.

„⇐" Suppose that S^* is satisfiable. Then exists a valuation v such that $v : S^* \to \{t\}$. We define a new valuation w.

$$w = \begin{cases} v(\mathbf{p}) & \text{for every propositional letter } p \neq L \\ t & \text{for } L \end{cases}$$

Since $w(L) = t$ and $w([L]) = t$ and hence $w([L|D]) = t$ and $w([L|D|E]) = t, w(F_1) = v(F_1) = t, \ldots, w(F_k) = v(F_k) = t$.

Thus, w maps every member of S to t, so S is satisfiable.

Case 4: Splitting Rule

Let S be the formula in the CNF.

$$S =< [L|D_1], \ldots, [L|D_n], [\neg L|E_1], \ldots, [\neg L|E_j], F_1, \ldots, F_k >$$

Let S^* be the formula after application the Splitting Rule to S.

$$S^* =< E_1, \ldots, E_j, F_1, \ldots, F_k > \vee < D_1, \ldots, D_n, F_1, \ldots, F_k > .$$

„⇒" Suppose that S is satisfiable. Then exists a valuation v such that $v : S \to \{t\}$. Then in particular: $v([L|D_1]) = t, \ldots, v([L|D_n]) = t$, $v([\neg L|E_1]) = t, \ldots, v([\neg L|E_j]) = t, v(F_1) = t, \ldots, v(F_k) = t$.

Suppose that $v(L) = t$ then $v(\neg L) = f$ and hence $v(E_1) = t, \ldots,$ $v(E_j) = t$.

Since v is the valuation we have:

$$v(S^*) = v(< E_1, \ldots, E_j, F_1, \ldots, F_k > \vee < D_1, \ldots, D_n, F_1, \ldots, F_k >) =$$

$$= v(< E_1, \ldots, E_j, F_1, \ldots, F_k > \underline{\vee} v(< D_1, \ldots, D_n, F_1, \ldots, F_k >),$$

but $v(< E_1, \ldots, E_j, F_1, \ldots, F_k >) = t$ and hence $v(S^*) = t$, so S^* is satisfiable.

„⇐" Suppose that S^* is satisfiable (sufficed that one of the member of S^* is satisfiable). Then exists a valuation v such that $v : S^* \to \{t\}$. We define a new valuation w.

$$w = \begin{cases} v(\mathbf{p}) & \text{for every propositional letter } p \neq L \\ t & \text{for } L \end{cases}$$

Since $w(L) = t$ and $w(E_1) = v(E_1) = t, \ldots, w(E_j) = v(E_j) = t$ then $w([L|D_1]) = t, \ldots, w([L|D_n]) = t$ and $w([\neg L|E_1]) = t, \ldots, w([\neg L|E_j]) = t, w(F_1) = v(F_1) = t, \ldots, w(F_k) = v(F_k) = t$.

Thus, w maps every member of S to t, so S is satisfiable. □

Lemma 2

If $[S]$ has a Davis-Putnam derivation that succeeds then $[S]$ is not satisfiable.

Proof:

Suppose that $[S]$ is satisfiable. We show a contradiction with the hypothesis that $[S]$ has a Davis-Putnam derivation that succeeds.

Every proof attempt must terminate, because we began with a finite number of distinct literals that occur in S and these rules can be applied only a finite number of times. Applying any rule to member S of block $[S]$ gives S^*. By lemma 1, S^* is satisfiable, since S is satisfiable. Hence, a final block of any proof attempt is satisfiable. But a final block containing the empty formula in the CNF is satisfiable and hence a derivation fails.

We show that exists the proof, which fails, so it is a contradiction with the hypothesis. □

Theorem 1 (soundness and completeness for the procedure)

α is a tautology $\Longrightarrow$ α has a Davis-Putnam derivation that succeeds.

Proof:

„$\Leftarrow$" Suppose that α has a Davis-Putnam derivation that succeeds. We start with $\{\neg\alpha\}$, convert this to a CNF S and form the block $[S]$. By lemma 2, $[S]$ is not satisfiable, hence $\{\neg\alpha\}$ is not satisfiable and hence α is a tautology.

„$\Rightarrow$" Suppose that α is a tautology and every Davis-Putnam derivation for α fails. First, we convert $\{\neg\alpha\}$ to the formula S in the CNF and form block $[S]$. Consider arbitrary Davis-Putnam derivation of $[S]$. It fails. Hence, its final block is satisfiable. Thus, by lemma 1, $[S]$ is satisfiable too. Hence, α cannot be a tautology. Contradiction. □

For speed of derivation in the Davis-Putnam procedure important is which rule is applied. Why ? The Splitting Rule multiplies the number of cases to be considered, so its use should be postponed as far as possible. In contradistinction to the Splitting Rule, the One-Literal Rule simply cuts down on the number of clauses we need to consider without introducing any other complications. So it should be applied earlier than any other rule. The order in which we presented the rules is also a good order in which to apply them

Example:

I $p \to ((\neg q \wedge q) \to r)$

1. Negate the formula:

$$\neg(p \to ((\neg q \wedge q) \to r))$$

2. Convert to a clause set, and form corresponding block:

$$[< [p], [\neg q], [q], [\neg r] >]$$

3. Apply the One-Literal Rule on the literal q :

$$[< [p], [], [\neg r] >]$$

4. Apply the One-Literal Rule on the literal p :

$$[< [], [\neg r] >]$$

5. Apply the One-Literal Rule on the literal $\neg r$:

$$[< [] >]$$

The formula in the CNF in final block contains the empty clause, so the derivation has succeeded.

II $((p \vee q) \to (p \vee \neg q)) \to (\neg p \vee q)$

1. Negate the formula:

$$\neg(((p \vee q) \to (p \vee \neg q)) \to (\neg p \vee q))$$

2. Convert to a clause set, and form corresponding block:

$$[< [\neg p, p, \neg q], [\neg q, p, \neg q], [p], [\neg q] >]$$

3. Apply Preliminary Rule 1 :

$$[< [p, \neg p, \neg q], [p, \neg q], [p], [\neg q] >]$$

6. Apply Preliminary Rule 2 :

$$[< [p, \neg q], [p], [\neg q] >]$$

7. Apply the One-Literal Rule on the literal $\neg q$:

$$[< [p] >]$$

4. Apply the One-Literal Rule on the literal p :

$$[<>]$$

Some formula in the CNF in the final block is empty, so the derivation has failed.

III $(p \equiv q) \vee (p \equiv \neg q)$

1. Negate the formula:

$$\neg((p \equiv q) \vee (p \equiv \neg q))$$

2. Convert to a clause set, and form corresponding block:

$$[< [p, \neg p], [\neg p, \neg q], [p, q], [q, \neg q], [p, \neg p], [p, \neg q], [\neg p, q], [q, \neg q] >]$$

3. Apply the Preeliminary Rule 2 (4 times):

$$[< [\neg p, \neg q], [p, q], [p, \neg q], [\neg p, q] >]$$

4. Use the Splitting Rule, splitting on the literal p :

$$[< [\neg q], [q] >, < [q], [\neg q] >]$$

5. Apply the One-Literal Rule to each clause set in the block:

$$[< [] >, < [] >]$$

Each formula in the CNF in final block contains the empty clause, so the derivation has succeeded.

References:

[1] J. L. Bell, A. B. Slomson, Models and Ultraproducts: an introduction, North-Holland Publishing Company-Amsterdam-London, 1971.

[2] E. Birnbaum, E. L. Lozinskii, The Good Old Davis-Putnam Procedure Helps Counting Models, Journal of Artificial Intelligence Research 10 (1999) 457-477.

[3] A. Borälv and G. Stålmarck, Automated Verification in Railways, In Indastrial-Strength Formal Methods in Practice, eds. M. G. Hinchey and J. P. Bowen, Springer-Verlag, London, 1999.

[4] M. Davis, G. Logemann, D. Loveland. A machine program for theorem-proving. Communications of the ACM, 5(7): 394-397, July 1962.

[5] M. Davis and H. Putnam: A computing procedure for quantification theory. Journal of the ACM, 7: 201-215, 1960. Reprinted in [Siekman and Wrightson, 1983].

[6] M. C. Fitting: First-Order Logic and Automated Theorem Proving. Springer-Verlag. New York, 1990. Second Edition, Springer-Verlag, New York, 1996.

[7] A. Grzegorczyk, Zarys logiki matematycznej, PWN, Warszawa, 1981.

[8] J. Siekman and G. Wrightson (editors): Automation of Reasoning. Springer-Verlag, New York, 1983.

[9] J. P. M. Silva and K. A. Sakallah, GRASP - A New Search Algorithm for Satisfiability, Transactions on Computers, vol. 48, (no 5), pp. 506-21, 1999.

[10] J. P. M. Silva and K. A. Sakallah, Conflict Analysis in Search Algorithms for Satisfiability, Technical Report RT-4-96, INESC, May 1996.

[11] G. Stålmarck, A system for determining propositional logic theorems by applying values and rules to triplets that are generated from a formula, 1989. Swedish Patent No. 467 076 (approved 1992), U. S. Patent No. 5 276 897 (approved 1994), European Patent No. 0 403 454 (approved 1995)

[12] Esprit project 22703. SYnchronous Reactive Formalisms SYRF, Project Programme, 1996.
http://www.imag.fr/VERIMAG/SYNCHRONE/SYFR/

[13] T. E. Uribe, M. E. Stickel. Ordered Binary Decision Diagrams and the Davis-Putnam Procedure. Lecture Notes in Computer Science, 845: 34-49, Springer-Verlag, September, 1994.

[14] H. Zhang, SATO: A decision procedure for propositional logic, Association for Automated Reasoning Newsletter, 22, 1-3, March 1993,

Pedagogical University
Institute of Mathematics and Computer Science
Al. Armii Krajowej 13/15
42-200 Częstochowa
l.szyper@mp.wsp.czest.pl