Formal Methods in Software Engineering

DPLL — Spring 2025

Konstantin Chukharev

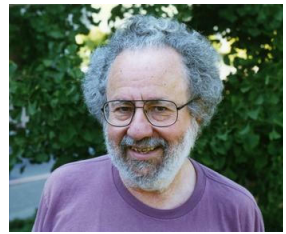
§1 Algorithms for SAT

Davis–Putnam Algorithm

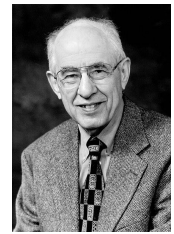
The first algorithm for solving the SAT problem was proposed by Martin Davis and Hilary Putnam in 1960 [1].

Satisfiability-preserving transformations:

- The 1-literal rule (**unit propagation**).
- The affirmative-negative rule (**pure literal**).
- The atomic formula elimination rule (**resolution**).



Martin Davis



Hilary Putnam

The first two rules reduce the total number of literals in the formula. The third rule reduces the number of variables in the formula. By repeatedly applying these rules, we can simplify the formula until it becomes *trivially* satisfiable (formula without clauses) or unsatisfiable (formula containing an empty clause).

Hereinafter, we assume that the formulas are given in CNF form.

Unit Propagation Rule

Definition 1 (Unit clause): A *unit clause* is a clause with a single literal.

Suppose (p) is a unit clause. Recall that $\overline{p}$ denotes the complement literal: $\overline{p} = \begin{cases} \neg p \text{ if } p \text{ is positive} \\ p \text{ if } p \text{ is negative} \end{cases}$

Then, the unit propagation rule is defined as follows:

- Assign the value of p to true.
- Remove all instances of $\overline{p}$ from clauses in the formula (shortening the corresponding clauses).
- Remove all clauses containing p (including the unit clause itself).

Example: Consider the formula $(A \vee B) \wedge (A \vee \neg B) \wedge (\neg A \vee B) \wedge (\neg A \vee \neg B) \wedge (A)$. The unit clause (A) is present in the formula. Applying the unit propagation rule, we remove all clauses containing A (positive literal), and remove $\neg A$ (negative literal) from the remaining clauses: ~~$(A \vee B)$~~ $\wedge$ ~~$(A \vee \neg B)$~~ $\wedge$ ~~$(\neg A \vee B)$~~ $\wedge$ ~~$(\neg A \vee B)$~~ $\wedge$ ~~$(\neg A \vee \neg B)$~~ $\wedge$ ~~(A)~~, which simplifies to $(B) \wedge (\neg B)$.

Pure Literal Rule

Definition 2 (Pure literal): A literal p is *pure* if it appears in the formula only positively or only negatively.

The pure literal rule is defined as follows:

- Assign the value of p to true.
- Remove all clauses containing a pure literal.

Example: Consider the formula $(A \vee B) \wedge (A \vee C) \wedge (B \vee C)$. The literal A is pure, as it appears only positively. Applying the pure literal rule, we assign $A = 1$ and remove all clauses containing A, which simplifies the formula to $(B \vee C)$.

Resolution Rule

1. Select a propositional variable p that appears both positively and negatively in the formula.
2. Partition the relevant clauses:
   - Let P be the set of all clauses that contain p.
   - Let N be the set of all clauses that contain $\neg p$.
3. Perform the resolution step:
   - For each pair of clauses $C_P \in P$ and $C_N \in N$, construct the *resolvent* by removing p and $\neg p$, then merging the remaining literals:

$$C_P \otimes_p C_N = (C_P \setminus \{p\}) \cup (C_N \setminus \{\neg p\})$$

   Example: $(a \vee b \vee \neg c) \otimes_b (a \vee \neg b \vee d \vee \neg e) = (a \vee \neg c \vee d \vee \neg e)$
4. Update the formula:
   - Remove all clauses in P and N.
   - Add the newly derived resolvents to the formula.

Davis–Putnam–Logemann–Loveland (DPLL) Algorithm

The DPLL algorithm [2] is a complete, backtracking search algorithm for deciding the satisfiability of propositional logic formulas in CNF, that is, for solving the CNF-SAT problem.

Introduced by Martin Davis, George Logemann, and Donald Loveland in 1961, the algorithm is a refinement of the Davis–Putnam algorithm.

In DPLL, the resolution rule is replaced with a *splitting* rule.

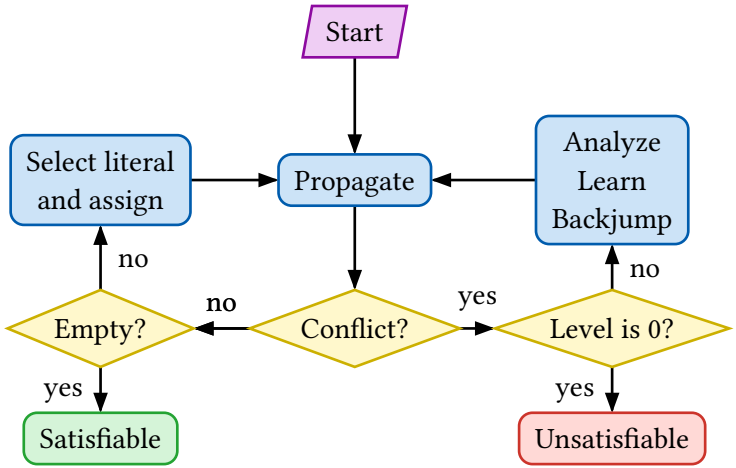
1. Let Δ be the current set of clauses.
2. Choose a propositional variable p occuring in the formula.
3. Recursively check the satisfiability of $\Delta \cup \{(p)\}$:
   - If satisfiable, return *satisfiable*.
   - If unsatisfiable, recursively check the satisfiability of $\Delta \cup \{(\neg p)\}$ and return that result.

The DPLL algorithm is a *complete* algorithm: it will eventually find a satisfying assignment iff one exists.

DPLL Pseudocode

Input: set of clauses S
Output: *satisfiable* or *unsatisfiable*

1 $S := \text{propagate}(S)$
2 **if** S is empty **then**
  return *satisfiable*
3 **if** S contains the empty clause **then**
  return *unsatisfiable*
4 $L := \text{select\_literal}(S)$
5 **if** $\text{DPLL}(S \cup \{L\}) = \textit{satisfiable}$ **then**
  return *satisfiable*
6 **else**
  return $\text{DPLL}(S \cup \{\neg L\})$
7 **end**



§2 Advanced Topics

Abstract DPLL

Definition 3: Abstract DPLL is a high-level framework for a general and simple abstract rule-based formulation of the DPLL procedure. [3], [4]

DPLL procedure is being modelled by a *transition system*: a set of *states* and a *transition relation*.

- States are denoted by S.
- We write $S \Longrightarrow S'$ when the pair (S, S') is in the transition relation, meaning that S' is *reachable* from S in one *transition step*.
- We denote by $\Longrightarrow^*$ the reflexive-transitive closure of $\Longrightarrow$.
- We write $S \Longrightarrow^! S'$ if $S \Longrightarrow^* S'$ and S' is a *final* state, i.e., there is no S'' such that $S' \Longrightarrow S''$.
- A state is either *fail* or a pair $M \parallel F$, where M is a *model* (a sequence of *annotated literals*) and F is a finite set of clauses.
- An empty sequence of literals is denoted by $\emptyset$.
- A literal can be annotated as *decision literal*, which is denoted by l^d.
- We write F, C to denote the set $F \cup \{C\}$.

DPLL

The *basic DPLL system* consists of the following transition rules:

- *UnitPropagate*:
  $M \parallel F, (C \vee l) \Longrightarrow Ml \parallel F, (C \vee l)$ **if** $\begin{cases} M \vDash \neg C \\ l \text{ is undefined in } M \end{cases}$
- *PureLiteral*:
  $M \parallel F \Longrightarrow M \parallel F$ **if** $\begin{cases} l \text{ occurs in some clause of } F \\ \neg l \text{ does not occur in any clause of } F \\ l \text{ is undefined in } M \end{cases}$
- *Decide*:
  $M \parallel F, C \Longrightarrow Ml^d \parallel F, C$ **if** $\begin{cases} l \text{ or } \neg l \text{ occurs in a clause of } F \\ l \text{ is undefined in } M \end{cases}$
- *Fail*:
  $M \parallel F, C \Longrightarrow \mathit{fail}$ **if** $\begin{cases} M \vDash \neg C \\ M \text{ contains no decision literals} \end{cases}$
- *Backtrack*:
  $Ml^d N \parallel F, C \Longrightarrow M\neg l \parallel F, C$ **if** $\begin{cases} Ml^d N \vDash \neg C \\ N \text{ contains no decision literals} \end{cases}$

CDCL

Extended rules:

- *Learn*:
  $M \parallel F \Longrightarrow M \parallel F, C \quad \textbf{if} \begin{cases} \text{all atoms of } C \text{ occur in } F \\ F \vDash C \end{cases}$
- *Backjump*:
  $$Ml^d N \parallel F, C \Longrightarrow Ml' \parallel F, C \quad \textbf{if} \begin{cases} Ml^d N \vDash \neg C \\ \text{there is asome clause } C' \vee l' \text{ such that:} \\ F, C \vDash C' \vee l' \\ l' \text{ is undefined in } M \\ l' \vee \neg l' \text{ occurs in } F \text{ or in } Ml^d N \end{cases}$$
- *Forget*:
  $M \parallel F, C \Longrightarrow M \parallel F \quad \textbf{if} \begin{cases} F \vDash C \end{cases}$
- *Restart*:
  $M \parallel F \Longrightarrow \emptyset \parallel F$

TODO: discuss

TODO

- [ ] CDCL
- [◧] Abstract DPLL

Bibliography

[1] M. Davis and H. Putnam, "A computing procedure for quantification theory," *Journal of the ACM*, vol. 7, no. 3, pp. 201–215, 1960, doi: 10.1145/321033.321034.

[2] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem-proving," *Communications of the ACM*, vol. 5, no. 7, pp. 394–397, 1962, doi: 10.1145/368273.368557.

[3] R. Nieuwenhuis, A. Oliveras, and C. Tinelli, "Abstract DPLL and Abstract DPLL Modulo Theories," *Logic for Programming, Artificial Intelligence, and Reasoning*, vol. 3452. pp. 36–50, 2005. doi: 10.1007/978-3-540-32275-7_3.

[4] R. Nieuwenhuis, A. Oliveras, and C. Tinelli, "Solving SAT and SAT Modulo Theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T)," *Journal of the ACM*, vol. 53, no. 6, pp. 937–977, 2006, doi: 10.1145/1217856.1217859.