

# On Clause Learning Algorithms for Satisfiability

Sam Buss

SDF@60  
Kurt Gödel Research Center  
Vienna  
July 13, 2013

## Satisfiability (SAT)

An instance of SATISFIABILITY (SAT) is a set  $\Gamma$  of clauses, interpreted as a conjunction of disjunction of literals, i.e. a CNF formula in propositional logic.

The SAT problem is the question of whether  $\Gamma$  is satisfiable.

SAT is well-known to be NP-complete. Indeed, most canonical NP-complete problems, including the “ $k$ -step NTM acceptance problem” are efficiently many-one reducible to SAT, namely by quasilinear time reductions. Therefore SAT is in some sense the *hardest* NP-complete problem.

So it is not surprising that SAT can be used to express many types of problems from industrial applications. What is surprising is how efficiently these kinds of problems can be solved.

**Conflict Driven Clause Learning SAT algorithms** are remarkably successful:

- ▶ Routinely solve industrial problems with  $\geq 100,000$ 's of variables. Find satisfying assignment or generate a resolution refutation.
- ▶ Most use depth-first search [Davis–Logemann–Loveland'62], and conflict-driven clause learning (CDCL) [Marques-Silva–Sakallah'99]
- ▶ Use a suite of other methods to speed search: fast backtracking, restarts, 2-clause methods, self-subsumption, lazy data structures, etc.
- ▶ Algorithms lift to a useful fragments of first-order logic. (SMT “Satisfiability Modulo Theory” solvers.)

## DPLL algorithm

**Input:** set  $\Gamma$  of clauses.

**Algorithm:**

- Initialize  $\rho$  to be the empty partial truth assignment
- Recursively do:
  - If  $\rho$  falsifies any clause, then return.
  - If  $\rho$  satisfies all clauses, then halt.  $\Gamma$  is satisfiable.
  - Choose literal  $x \notin \text{dom}(\rho)$ .
  - Set  $\rho(x) = \text{True}$  and do a recursive call.
  - Set  $\rho(x) = \text{False}$  and do a recursive call.
  - Restore the assignment  $\rho$  and return.
- If return from top level,  $\Gamma$  is unsatisfiable.

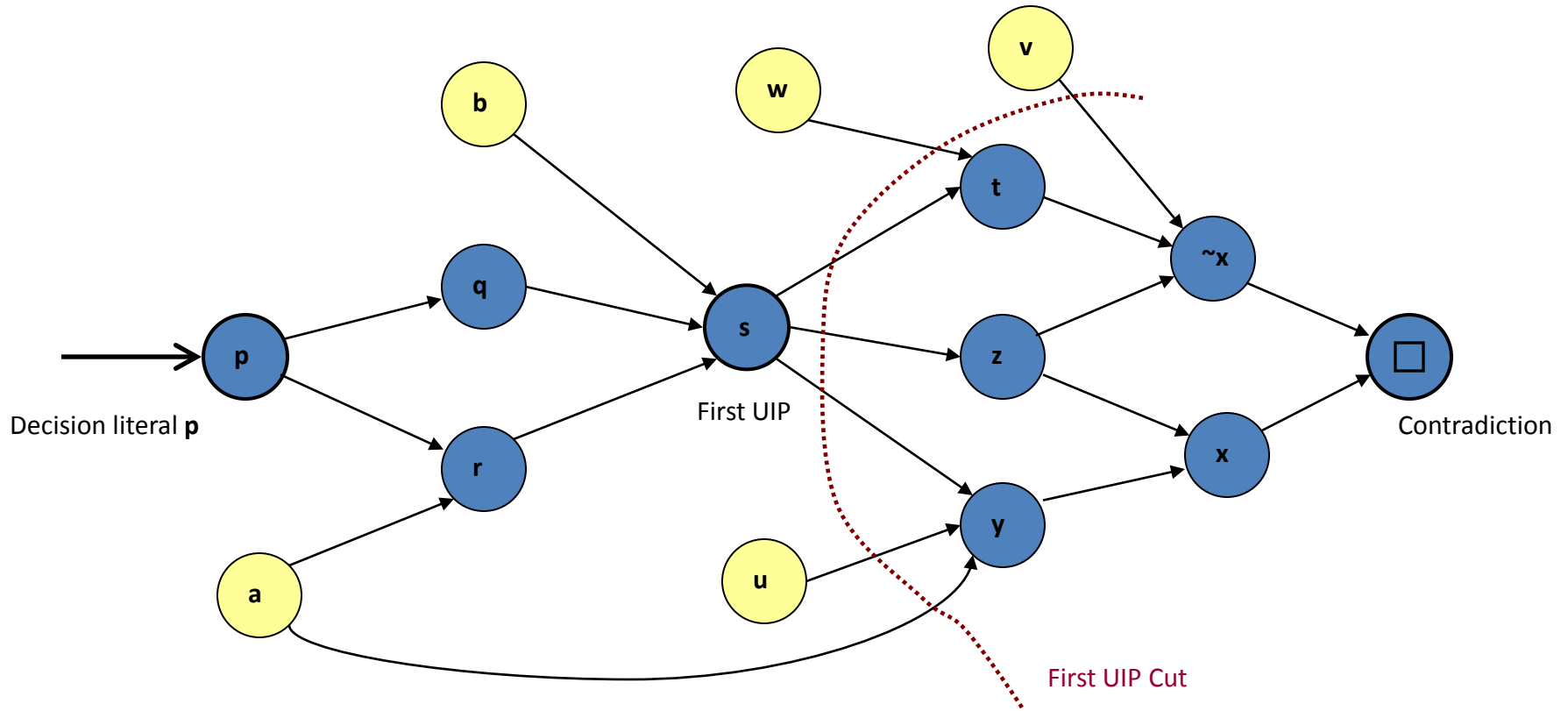
## DPLL algorithm with unit propagation and CDCL

**Input:** set  $\Gamma$  of clauses.

**Algorithm:**

- Initialize  $\rho$  to be the empty partial truth assignment
- Recursively do:
  - **Extend  $\rho$  by unit propagation as long as possible.**
  - If a clause is falsified by  $\rho$ , then  
**analyze the conflict to learn new clauses implied by  $\Gamma$ ,  
add these clauses to  $\Gamma$** , and return.
  - If  $\rho$  satisfies all clauses, then halt.  $\Gamma$  is satisfiable.
  - Choose literal  $x \notin \text{dom}(\rho)$ .
  - Set  $\rho(x) = \text{True}$  and do a recursive call.
  - Set  $\rho(x) = \text{False}$  and do a recursive call.
  - Restore the assignment  $\rho$  and return.
- If return from top level,  $\Gamma$  is unsatisfiable.

# Example of clause learning



Clauses:  $\{\sim y, \sim z, x\}$ ,  $\{\sim p, \sim a, r\}$ , etc. (One per unit propagation.)

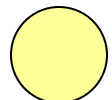
First UIP Learned Clause:  $\{\sim a, \sim u, \sim s, \sim w, \sim v\}$ .

Whole top level learned clause:  $\{\sim p, \sim a, \sim u, \sim b, \sim w, \sim v\}$ .

With First-UIP: Both **p** and **s** can be set false when backtracking.

New level of  $\sim s$  is set to max. level of **u, v, w**.

 Blue for top level

 Yellow for lower level literal

## Experiments with Pigeonhole Principles

Formula	Clause Learning		No Clause Learning	
	Steps	Time (s)	Steps	Time (s)
PHP <sub>3</sub> <sup>4</sup>	5	0.0	5	0.0
PHP <sub>6</sub> <sup>7</sup>	129	0.0	719	0.0
PHP <sub>8</sub> <sup>9</sup>	769	0.0	40319	0.3
PHP <sub>9</sub> <sup>10</sup>	1793	0.5	362879	2.5
PHP <sub>10</sub> <sup>11</sup>	4097	2.7	3628799	32.6
PHP <sub>11</sub> <sup>12</sup>	9217	14.9	39916799	303.8
PHP <sub>12</sub> <sup>13</sup>	20481	99.3	479001599	4038.1

“Steps” are decision literals set. (Equals  $n! - 1$  with no learning).

Variables selected in “clause greedy order”.

Software: SatDiego 1.0 (an older, much slower version).

Improvements strikingly good, even though PHP is not particularly well-suited to clause learning.

**Resolution** is the inference rule

$$\frac{C, x \quad D, \bar{x}}{C \cup D}$$

A **resolution refutation** of  $\Gamma$  is a derivation of the empty clause  $\square$  from  $\Gamma$ .

[Davis-Putnam'60]: Resolution is complete.

**Observation:** All DPLL-style methods for showing  $\Gamma$  unsatisfiable (implicitly) generate resolution refutations. Thus resolution refutations encompass all DPLL-style methods for proving satisfiability with only a polynomial increase in refutation size.



## Relationship to resolution?

**Open problem:** Can a DPLL search procedure, for unsatisfiable  $\Gamma$  polynomially simulate resolution? (This is without restarts.)

**Theorem** [Beame-Kautz-Sabharwal'04] Non-greedy DPLL with clause learning **and restarts** simulates full resolution.

*Proof idea:* Simulate a resolution refutation, using a new restart for each clause in the refutation. Ignore contradictions (hence: non-greedy) until able to learn the desired clause. □

**Theorem** [Pipatsrisawat-Darwiche'10] (Greedy) DPLL with clause learning **and (many) restarts** simulates full resolution.

This built on [Atserias-Fichte-Thurley'11].

## DPLL and clause learning without restarts:

[BKS '04; Baachus-Hertel-Pitassi-van Gelder'08; Buss-Hoffmann-Johannsen'08] It is possible to add new variables and clauses *that syntactically preserve (un)satisfiability*, so that DPLL with clause learning can refute the augmented set of clauses in time polynomially bounded by a resolution refutation of the original set of clauses.

In this way, DPLL with clause learning can “effectively p-simulate” resolution.

These new variables and clauses are *proof trace extensions* or *variable extensions*.

### Drawback:

- ▶ The variable extensions yields contrived sets of clauses, and the resulting DPLL executions are unnatural.

[Van Gelder, 2005] introduced “**pool resolution**” as a system that can simulate DPLL clause learning without restarts. Pool resolution consists of:

- a. A degenerate resolution inference rule, where the resolution literal may be missing from either hypothesis. If so, the conclusion is equal to one of the hypotheses.
- b. A dag-like degenerate resolution refutation with a **regular depth-first traversal**.

The degenerate rule incorporates weakening into resolution.

The *regular* property means no variable can be resolved on twice along a path. This models the fact that DPLL algorithms do not change the value of literals without backtracking.

**Theorem** [VG'05] Pool resolution p-simulates DPLL clause learning without restarts.

[Buss-Hoffmann-Johannsen '08] gave a system that is *equivalent* to non-greedy DPLL clause learning without restarts.

**w-resolution:** 
$$\frac{C \quad D}{(C \setminus \{x\}) \cup (D \setminus \{\bar{x}\})} \quad \text{where } \bar{x} \notin C \text{ and } x \notin D.$$

[BHJ'08] uses **tree-like proofs with lemmas** to simulate dag like proofs. A lemma must be earlier derived in left-to-right order. A lemma is **input** if derived by an input subderivation (allowing lemmas in the subderivation). An **input** derivation is one in which each resolution inference has one hypothesis which is a leaf node.

**Theorem** [Chang'70]. Input derivations are equivalent to unit derivations.

**Theorem** [BHJ'08]. Resolution trees with input lemmas simulates general resolution (i.e., with arbitrary lemmas).

**Defn** A “regWRTI” derivation is a regular tree-like w-resolution with input lemmas.

**Theorem** [BHJ’08] regWRTI p-simulates DPLL clause learning without restarts. Conversely, non-greedy DPLL clause learning (without restarts) p-simulates regWRTI.

(The above theorem allows very general schemes of clause learning; however it does not cover the technique of “on-the-fly self-subsumption” .)

**Consequently:** To separate DPLL with clause learning and no restarts from full resolution, it suffices to separate either regWRTI or pool resolution from full resolution.

**Fact:** DPLL clause learning without restarts (and regWRTI and pool resolution) simulates regular resolution.

**Theorem** [Alekhnovich-Johanssen-Pitassi-Urquhart'02]  
Regular resolution does not p-simulate resolution.

[APJU'02] gave two examples of separations.

- ▶ Graph tautologies (GGT) expressing the existence of a minimal element in a linear order, obfuscated by making the axioms more complicated.
- ▶ A *Stone* principle about pebbling dag's.

[Urquhart'11] gave a third example using obfuscated pebbling tautologies expressing the well-foundedness of (finite) dags.

**Clauses for the Stone tautologies:** Let  $G$  be a dag on nodes  $\{1, \dots, n\}$ , with single sink 1, all non-source nodes have indegree 2. Let  $m > 0$  be the number of “stones” (pebbles).

- ▶  $\bigvee_{j=1}^m p_{i,j}$  for each vertex  $i$  in  $G$ ,
- ▶  $\bar{p}_{i,j} \vee r_j$ , for each  $j = 1, \dots, m$ , and each  $i$  which is a source in  $G$ ,
- ▶  $\bar{p}_{1,j} \vee \bar{r}_j$ , for each  $j = 1, \dots, m$ . Node 1 is the sink node.
- ▶  $\bar{p}_{i',j'} \vee \bar{r}_{j'} \vee \bar{p}_{i'',j''} \vee \bar{r}_{j''} \vee \bar{p}_{i,j} \vee r_j$  with  $i'$  and  $i''$  the predecessors of  $i$ , and  $j \notin \{j', j''\}$ . These are the *induction clauses* for  $i$ .

“ $p_{i,j}$ ” - Stone  $j$  is on node  $i$ .

“ $r_j$ ” - Stone  $j$  is red.

If all nodes have a stone, and source nodes have red stones, and any stone on any node with red stones on both predecessors is red, then any stone on the sink node is red.



**Theorem** [Bonet-Buss'12] There are polynomial size pool refutations and also regRTI refutations for the obfuscated GGT tautologies.

**Theorem** [Bonet-Buss-Johannsen'ip] There are polynomial size pool refutations and regRTI refutations for the obfuscated pebbling tautologies.

**Theorem** [Buss-Kołodziejczyk'su] There are polynomial size pool refutations and regRTI refutations for the Stone tautologies.

**Corollary** [see also BKS, vG] Regular resolution does not simulate regWRTI, or pool resolution, or DPLL clause learning without restarts.

**Consequently:** Although we still conjecture that resolution cannot be simulated by DPLL with clause learning and no restarts, we currently have no conjectures for unsatisfiable formulas that might separate them.

## Questions

1. Give examples of sets of clauses which have polynomial size resolution refutations, but do not have polynomial size pool or regWRTI refutations. Or, prove no such sets of clauses exist.
2. Does regWRTI faithfully capture present day DPLL clause learning without restarts? For example, can “on-the-fly” self-subsumption improve on regWRTI?

## Meta-questions:

1. Can we find more systematic or rigorous methods for evaluating the effectiveness of different proof search heuristics. For example, can we better understand why restarts are so useful in practice?
2. Can logical considerations help us understand the effectiveness of different DPLL strategies? Can regWRTI or other similar formal systems help design better DPLL algorithms?

## Meta-questions:

1. Can we find more systematic or rigorous methods for evaluating the effectiveness of different proof search heuristics. For example, can we better understand why restarts are so useful in practice?
2. Can logical considerations help us understand the effectiveness of different DPLL strategies? Can regWRTI or other similar formal systems help design better DPLL algorithms?
3. Is automated theorem proving part of Logic? Or is it just computation?

## Meta-questions:

1. Can we find more systematic or rigorous methods for evaluating the effectiveness of different proof search heuristics. For example, can we better understand why restarts are so useful in practice?
2. Can logical considerations help us understand the effectiveness of different DPLL strategies? Can regWRTI or other similar formal systems help design better DPLL algorithms?
3. Is automated theorem proving part of Logic? Or is it just computation?

*Answer: Yes!*

Happy Birthday, Sy!