Tutorial on Proof Systems Connected to SAT Solving

Sam Buss
Univ. of California, San Diego

Theory and Practice of SAT Solving
Dagstuhl, Germany

April 21, 2015

Satisfiability (SAT); The basics

Satisfiability: Given a propositional formula, determine
- if it has a satisfying assignment
- find a satisfying assignment or a refutation (optional).

Satisfiability is NP-complete. Indeed, many of the "standard" NP-complete problems are many-one reducible to satisfiability (even on *clauses*) in quasilinear time $n(\log n)^{O(1)}$.
For instance: the question of whether a given Turing machine halts in n steps is reducible to SAT in this way.

The best algorithms we know for general satisfiability (even on clauses) have exponential runtime 2^n, where n is the number of variables. Any substantial improvement, even to just $2^{n/2}$, would be a substantial breakthrough, and give improved algorithms for a broad range of problems.

Thus, it comes as a shock that in practice many instances of SAT that arise from hard problem domains can be solved efficiently. Even for tens or hundreds of thousands of variables.

Most SAT solvers use clauses and (implicitly) generate refutations in *resolution*.

Variables: x, y, ..., x_1, x_2, ...
Literals: x and $\overline{x}$ for x a variable. (Involutive negation.)

Clause: a finite set of literals.
Intended meaning is the disjunction (OR) of the literals.
Generally require that clauses are not tautologies (do not contain both x and $\overline{x}$).

Instance of satisfiability: A set Γ of clauses.
Intended meaning is the conjunction of the clauses.
Goal is to assert satisfiable or unsatisfiable.
Usually the answer can be augmented with a satisfying assignment or a (resolution) refutation, respectively.

Resolution inference:

$$\frac{\Gamma, x \qquad \Delta, \overline{x}}{\Gamma, \Delta}$$

where $x, \overline{x} \notin \Gamma \cup \Delta$.

We write $\Gamma \vdash C$ to mean C can be derived from Γ by resolution inferences.
We write $\Gamma \models C$ for logical implication.

Completeness and soundness:
$\Gamma \vdash \emptyset$ iff Γ is unsatisfiable.

Implicational completeness and soundness:
$\Gamma \models C$ iff for some $C' \subseteq C$, $\Gamma \vdash C'$.

A **partial truth assignment**, also called a *restriction*, is a mapping ρ from variables to $\{\top, \bot\}$ (i.e., {True,False}). Thus, ρ gives truth values to some of the literals.

The restriction $\Gamma \restriction \rho$ is the set of clauses obtained from Γ by

- Remove from Γ every clause containing a literal x set true.
- In remaining clauses, remove any literal set false.

A **unit clause** is a clause with a single literal $\{x\}$. Any satisfying assignment must set x true.

A **pure** literal x of Γ is one such that $\overline{x}$ does not appear in Γ.

DPLL algorithm (without unit propagation)

```
Input:  Set of clauses Γ
Output:  A satisfying assignment or "Not Satisfiable".

DPLL_No_UP( Γ ) {
    DPLL_No_UP( Γ, ∅ ).
    Output "Unsatisfiable" and halt.
}

DPLL_No_UP( Γ, ρ ) {
    If Γ|ρ contains the empty clause ∅, return.
    If Γ|ρ is the empty set,
        Output "ρ is a satisfying assignment" and halt.
    Choose x ∉ domain(ρ).
    DPLL( Γ, ρ[x ↦ ⊤] ).
    DPLL( Γ, ρ[x ↦⊥] ).
}
```

[Davis-Putnam '60], [Davis, Logemann, Loveland '62]

DPLL algorithm (with unit propagation)

```
DPLL( Γ, ρ ) {
    Repeat while possible {
        If Γ↾ρ contains the empty clause ∅, return.
        If Γ↾ρ is the empty set,
            Output "ρ is a satisfying assignment" and halt.
        If Γ↾ρ contains unit clause {x} (or pure literal x)
            ρ := ρ[x ↦ ⊤]
    }
    Choose x ∉ domain(ρ).
    DPLL( Γ, ρ[x ↦ ⊤] ).
    DPLL( Γ, ρ[x ↦⊥] ).
}
```

CDCL: Conflict-Driven Clause Learning

CDCL is DPLL plus clause learning

```
CDCL( Γ, ρ ) {
    Loop {
        If Γ↾ρ contains the empty clause ∅,
            Learn one or more clauses and return.
        If Γ↾ρ is the empty set,
            Output "ρ is a satisfying assignment" and halt.
        If Γ↾ρ contains unit clause {x} or pure literal x
            ρ := ρ[x ↦ ⊤]
    }
    Choose x ∉ domain(ρ).
    DPLL( Γ, ρ[x ↦ ⊤] ).
    DPLL( Γ, ρ[x ↦⊥] ).
}
```

Learning a clause means: add it persistently to Γ.
(That is, Γ is a global variable — unlike ρ.)

Clause learning possibilities:

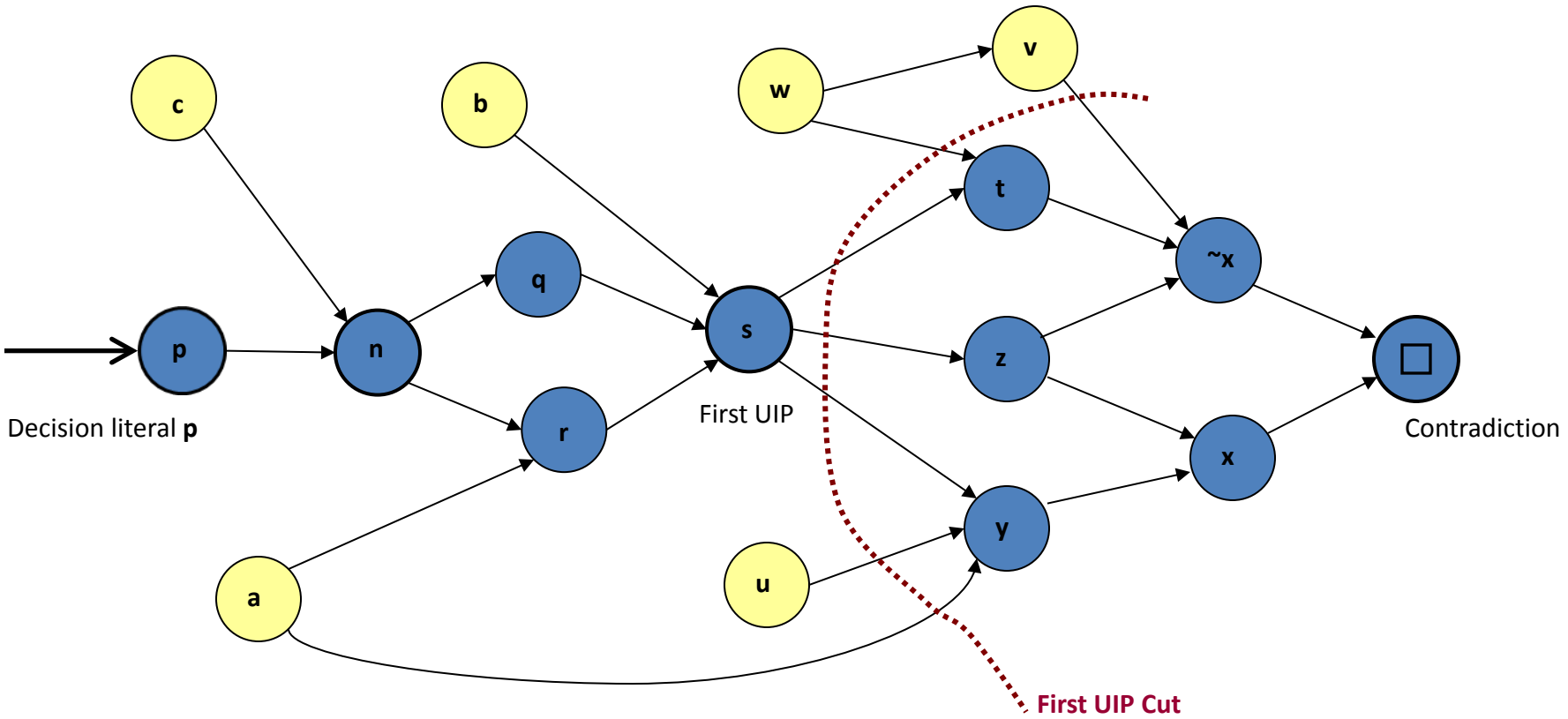
Generally based on the *conflict graph* of variables set at the current decision level with unit propagation.

Methods include:

- First UIP [Marques-Silva, Sakallah '96]
- rel-sat [Bayardo, Schrag '97]
- Second UIP, Third Uip, . . . , [M-S,S'96]

First UIP is by far the most popular.
SatDiego experiments: using all UIP's is somewhat better.

Clause Learning – First UIP

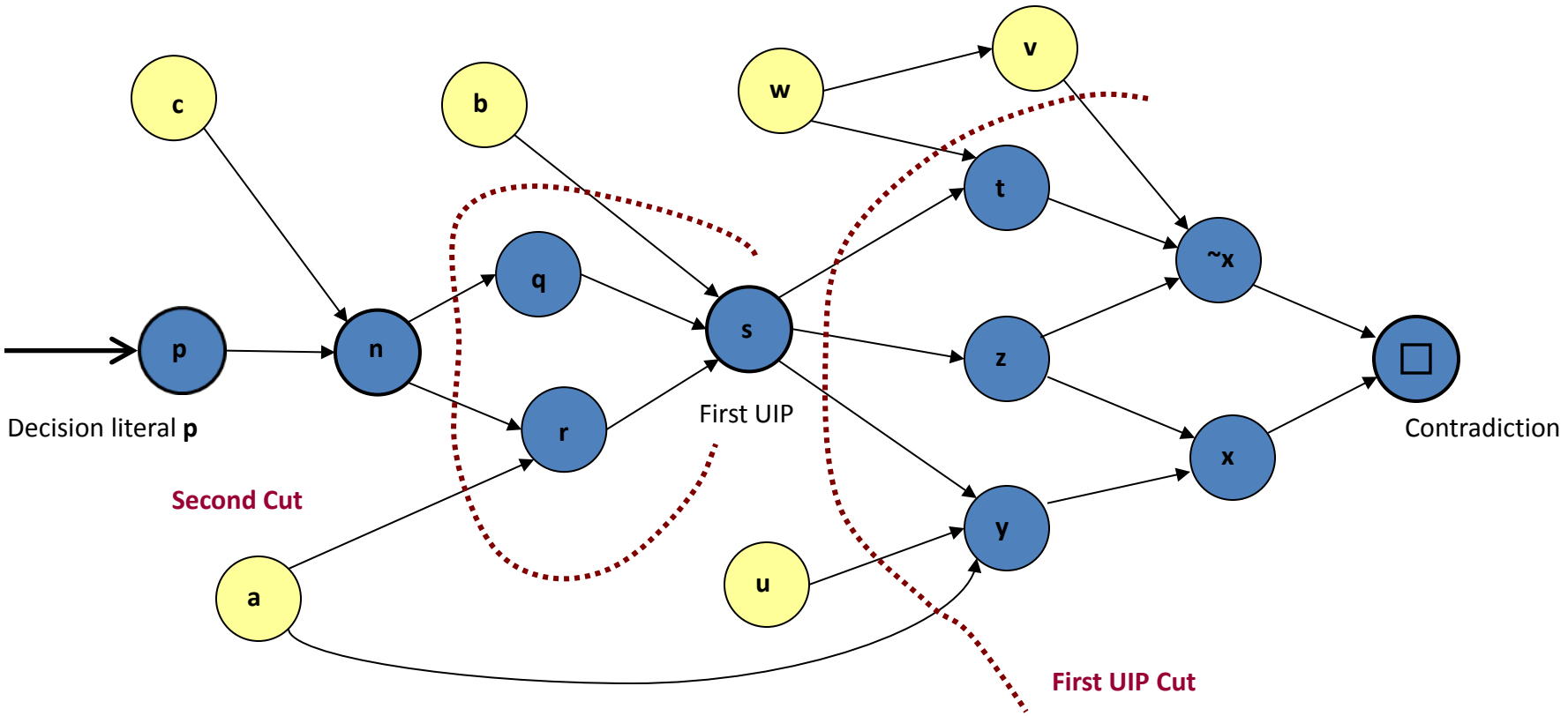


Clauses: {~y,~z, x}, {~p,~a,r}, etc. (One per unit propagation.)
First UIP Learned Clause: {~a,~u,~s,~w,~v}.
Whole top level learned clause: {~p,~a,~u,~b,~w,~v}.
With First-UIP: Both **p** and **s** can be set false when backtracking.
New level of **~s** is set to maximum level of **u,v,w.**

Blue for top level

Yellow for lower level literal

Clause Learning – Example of Two UIP Learning



First UIP Learned Clause: **{~a,~u,~s,~w,~v}.**
Second UIP (Fragment) Clause: **{~n, ~a, ~b, s}**
Both **p** and **s** can be set false when backtracking.
New level of **~s** is set to maximum level of **u,v,w.**
New level of ~p is set to maximum level of **a,b,c,u,v,w.**
With both clauses learned: **Both ~s and ~p have supporting clauses.**

Characterization of clause learning

Def'n: An *input* (aka, *trivial*) resolution refutation is one in which every inference has an initial clause as a hypothesis.

Theorem (Beame-Kautz-Sabharwal'04; Chang'70)

There is an input resolution derivation of C from Γ iff $\Gamma \cup \overline{C}$ has a unit propagation refutation.

$\overline{C}$ means the clauses $\{\overline{x}\}$ for $x \in C$.

[BKS'04]

Clause learning learns only clauses C which can be derived from Γ with an input derivation.
Usually, from clauses used to infer literals at the topmost decision level with unit propagation.

This kind of clause learning thus derives only clauses which are falsified by the current partial assignment ρ.

Learned clause minimization

A learned clause C can often be *minimized* by learning a clause $D \subset C$ by using "recursive minimization".

E.g.: If $C = \{x_1, x_2, \ldots, x_k, y, \vec{z}\}$, and
$\overline{y}$ follows from the k unit clauses $\overline{x}_1, \ldots, \overline{x}_k$ by unit propagation, so $\{x_1, \ldots, x_k, \overline{y}\}$ is derivable by input resolution. Then $D = \{x_1, x_2, \ldots, x_k, \vec{z}\}$ can be inferred instead of C

Simplifications are found by traversing the implication graph used to form C, and looking for literals y in C such that $\overline{y}$ is implied by other literals whose negations appear in C.

This can be done efficiently in time linear in the size of the implication graph. (This can still add quadratic time due to traversaling below the top decision level!)

Extra optimization: Any literal y that is the only literal in the clause at its decision level cannot be optimized away.

Minisat '05 and [Sörensson-Biere '09] and [van Gelder '09]

Experimental results

Minimization can give good improvements in learned clause size

Experiments with SatDiego remove from 5% to 20% of the literals from a learned clause on average.
Initially about 5% when using short restart cycles.

As learned clause length grows (to $\approx$100 literals or more), the percentage of removed literals rises as high as 20%.

Net effect is a noticeable improvement in performance.

Secondary minimization

Secondary minimizations can be carried out by applying minimization to every clause used to derive the learned clause.

This is quadratic time in the worst case, but can be done efficiently in practice, since learned clause simplification is done first, and only those literals which are simplified out of the learned clause are candidates for secondary simplification.

Experimental results with SatDiego, minimization of clauses at the top decision level only: Secondary simplification removes about 1% of the literals.

Net effect is a slight improvement in performance.

[B., SatDiego, unpubl.]

Relationship with resolution

Defn: A refutation of Γ can be either dag-like or tree-like, and has initial clauses from Γ, other clauses inferred by resolution, and final clause $\emptyset$.

Defn: A refutation is *regular* if on each path through the refutation (tree or dag) no variable is resolved on more than once.

Question: How do proofs as implicitly generated by CDCL refutations correspond to resolution refutations? To regular resolution refutations?

Theorem

- *DPLL refutations (without restarts, and with or without unit propagation) can be translated to regular tree-like resolution refutations.*
- *Regular tree-like resolution refutations can be simulated by DPLL refutations.*

The theorem holds even for greedy DPLL algorithms:
'Greedy" means that contradictions cannot be ignored.

Proof idea: The DPLL refutation can be viewed as traversing the refutation in depth-first order. Each clause reached in the traversal is made false by the current assignment ρ.

Question: What about CDCL (that is, with learning)? Can it polynomially simulate unrestricted resolution?

CDCL with restarts and unrestricted resolution

Theorem (BKS'04, AFT'09, PD'09)

CDCL with restarts can polynomially simulate daglike resolution.

Proof idea: Given a resolution proof with clauses $C_1, C_2, \ldots, \emptyset$, successively learn C_1, then C_2, etc., and then finally $\emptyset$.

[BKS'04]'s construction does this exactly, but needs special conventions on learning and is non-greedy. A *non-greedy* algorithm may ignore contradictions and not do all unit propagations.

[AFT'09,PD'09]: Works with general asserting clause learning methods, and works with greedy algorithms. They only "absorb" each C_i and may not learn them: Absorptions allow the same unit propagations to be carried out.

Restarts are very useful in practice too. Why is unclear.

Beame-Kautz-Sabharwal'04; Atserias-Fichte-Thurley'09, Pipatsrisawat-Darwiche'09

CDCL-without-restarts, and unrestricted resolution

Theorem (Effective p-simulation: BKS'04; BHJ'08; HBPvG'08; BS'14)

A set of clauses Γ can be conservatively extended to a set Π so that Γ has a polynomial size dag-like resolution refutation if and only if Π has a polynomial size CDCL refutation.

This theorem is a bit of a cheat however, since it adds extraneous variables which can be branched on merely to learn new clauses.

Corollary (BKS'04)

CDCL can p-simulate resolution iff it is "natural" in the sense that if any Γ has a CDCL refutation of size N, then any restriction $\Gamma\restriction\rho$ has a CDCL refutation of size polynomially bounded by N.

$\overline{\text{B}}$eame-Kautz-Sabharwal'04; B.-Hoffmann-Johannsen'08; Hertel-Bachus-Pitassi-Van Gelder'08; Beame-Sabharwal'14

Pool resolution and regWRTI and CDCL

An attempt to better model CDCL as a proof system:

Degenerate resolution inference: $\dfrac{\Gamma \qquad \Delta}{\Pi}$ resolving on x, where:

- This is a valid resolution inference on x, or
- $x, \overline{x}$ not in Γ, and Π is Γ, or
- $x, \overline{x}$ not in Δ, and Π is Δ.

Definition (van Gelder'05; HBPvG'08)

A *pool resolution refutation* is a dag-like refutation of degenerate resolution inferences, and

> *There is a depth-first traversal of the refutation which is regular (no literal is repeated on any single branch of the traversal).*

The motivation for pool resolution is that a depth-first regular traversal corresponds to a CDCL algorithm which learns clauses once they have been traversed.

The degenerate resolution rule corresponds to a CDCL algorithm branching on a literal, but not using it for learning in both branches.

Theorem (Van Gelder'04)

Pool resolution can p-simulate CDCL-without-restarts (under a range of possibilities for clause learning).

w-resolution inference:

$$\frac{\Gamma \qquad \Delta}{(\Gamma \setminus \{x\}) \cup (\Delta \setminus \{\overline{x}\})}$$

Definition (B-Hoffmann-Johannsen'09)

A *regWRTI* refutation is a dag-like proof using w-resolution so that

- The dag-like proof is recast into tree-like form by a left-to-right depth first (postorder) traversal.
- Each leaf node is an input clause or an earlier clause from the traversal called a *lemma*.
- Each lemma must be a "input lemma", namely derived by an input subderivation in the tree.

w-resolution combines weakening with resolution: corresponds to the fact that CDCL may branch on some literal which is not used for clause learning.

The use of input lemmas mirrors the characterization of clause learning in terms on input derivations (based on unit propagation).

Theorem (B-Hoffmann-Johannsen'08)

regWRTI is polynomially equivalent to non-greedy CDCL-without-restarts under a wide range of possibilities for clause learning (DLL-L-UP).

It is still open whether regWRTI is equivalent to resolution

Regular resolution is weaker than unrestricted resolution:

Theorem (AJPU'07; U'11)

Dag-like regular resolution refutations may need to be exponentially longer than dag-like resolution refutations. This is known for three principles: an obfuscated graph ordering principle, an obfuscated xor-ified pebbling principle and an indirectly encoded pebbling principle (known as the Stone tautologies).

It was conjectured that some of these principles could separate CDCL-without-restarts from resolution.

Alekhnovich-Johannsen-Pitassi-Urquhart'07; Urquhart'11

Theorem (BBJ'12; BK'14)

- *All three principles have polynomial size regWRTI refutations.*
- *All three principles, have polynomial size non-greedy CDCL refutations provided the CDCL algorithm makes the correct choices for decision literals and clause learning.*

For the first principle, greedy CDCL can work, but this is known only if learned clauses are "forgotten" at the right times.

Open question

Does regWRTI polynomially simulate dag-like resolution refutations?

Bonet-B.-Johannsen'13; B-Kołodziejczyk'14

Learning that does not fit into pool/regWRTI resolution

- Second UIP clause learning, n-th UIP clause learning, for $n > 1$.
- Secondary clause minimization.
- "On the fly" minimization.
- 2-clauses inferred from dominator analysis. [SatDiego,...,]

These examples allow learning clauses in which one literal is set **true** by the current partial assignment ρ,

Suggestion: Reformulate pool resolution and regWRTI to allow learning clauses in which all but one literal is set false — even if it requires resolving on a literal which has already been resolved on.

Some open problems

- Do pool resolution or regWRTI simulate unrestricted daglike resolution?
- Does CDCL-without-restarts, augmented to allow learning of clauses in which all but one literal is set false, simulate unrestricted resolution?
- Do greedy regWRTI, or greedy CDCL-without-restarts, polynomially simulate regular resolution?

Another open problem:

- Explain why restarts work so well.

One more open problem: Give some theoretical justifications (and guidelines) for the effectiveness of:

- Variable selection heuristics, e.g. VSIDS
- Clause forgetting heuristics

These seem to be related to each other, and to the problem about whether CDCL is "local" or "global".

Thank you!