An Improved Separation between Regular Resolution and Pool Resolution

Sam Buss
(joint work with Maria Luisa Bonet)

BIRS Workshop, Proof Complexity, Banff, Oct. 4, 2011

SAT algorithms

SAT algorithms - Remarkably successful

- Routinely solve industrial instances with $\geq$100,000's of variables.
- Mostly based on depth-first search. (DPLL)
- Use a suite of methods to speed search: clause learning, fast backtracking, restarts, implementation tuning.
- Find satisfying assignment or generate a resolution refutation.
- Algorithms lift to a useful fragments of first-order logic. (SMT solvers.)

Questions for this talk: What is the logical complexity of SAT algorithms? How does DPLL with clause learning compare to resolution?

Clause learning is the main heuristic that has a logical justification.

Fundamental idea: Set a trial (partial) satisfying assignment. When blocked, use this "counter-example" to learn a new clause. The new clause helps avoid repeatedly searching the same part of the solution space.
GRASP: Marques-Silva & Sakallah [1999].

Several heuristics are used to decide which clauses to learn, e.g., First-UIP. These are all encompassed by "input resolution", aka "trivial resolution". (Beame-Kautz-Sabarwal [2004].)

- Input resolution corresponds to contradictions that can be discovered by unit propagation.
- Easy to decide if a given clause can be derived by input resolution.

Relationship to resolution?

[Folklore?] Resolution simulates all current DPLL-based algorithms, including with clause learning, restarts, and pure literal selection.

DPLL with clause learning and restarts:

Theorem [BKS 2004] 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, 2010] (Greedy) DPLL with clause learning and (many) restarts simulates full resolution.

[Atserias, Fichte, Thurley '11] - related results for bounded width.

DPLL and clause learning without restarts:

[BKS '04; B-H-P-vG '08; B-H-J '08] It is possible to add new variables and clauses *that preserve (un)satisfiability*, so that DPLL with clause learning can refute the augmented set of clauses if and only if resolution can refute 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.

Pool resolution

[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 is needed to learn more clauses. The regular depth-first traversal corresponds to the fact that DPLL algorithms do not change the value of literals without backtracking.

The "pool" is the set of literals assigned true by the DPLL search.

Thm [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: $\dfrac{C \qquad D}{(C \setminus \{x\}) \cup (D \setminus \{\overline{x}\})}$ where $\overline{x} \notin C$ and $x \notin D$.

[BHJ] 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).

Thm [BHJ]. 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.

Thm [BHJ] 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.

The greedy case still open: No exact formal system is known to be p-equivalent.
However, regWRTI is a reasonable conjecture.

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

Thm [AJPU 2002] Regular resolution does not p-simulate resolution.

[APJU] gave two examples of separations.

- Graph tautologies 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.

The (non-obfuscated) graph tautologies were originally introduced by [Krishnamurthy '85]. Regular refutations were given by [Stålmarck, '96] and [Bonet-Galesi '99].

We use the term *guarded* graph tautologies (GGT_n) for [AJPU]'s obfuscated graph tautologies. In these, initial clauses $x_{i,j}, x_{j,k}, x_{k,i}$ are replaced by

$$x_{i,j}, x_{j,k}, x_{k,i}, x_{r,s} \quad \text{and} \quad x_{i,j}, x_{j,k}, x_{k,i}, \overline{x}_{r,s}$$

for some $r = r(i,j,k)$ and $s = s(i,j,k)$. (All i,j,k,r,s distinct.)

The non-regular refutation of GGT_n comes from resolving the two above clauses to derive $x_{i,j}, x_{j,k}, x_{k,i}$, for all i,j,k, and then using the (regular) refutation of GT_n.

Theorem [Bonet-Buss] There are polynomial size pool refutations and also regRTI refutations of the GGT_n clauses. Consequently, DPLL clause learning without restarts can show the unsatisfiability of the GGT_n clauses in polynomial time.

Note that w-resolution is not needed, only resolution.

Proof sketch: Next two slides...

Parts of the following corollary were already shown by [BKS] and Van Gelder using proof trace extensions:

Corollary Regular resolution does not simulate regWRTI, or pool resolution, or DPLL clause learning without restarts.

Proof sketch.

Idea is to have a partially defined "bipartite" ordering π on the vertices of the underlying graph. The clause $(\bigvee\overline{\pi})$ contains the negations of the literals set true in π (i.e., states that π does not hold). Initially π is empty.

The refutation is constructed in stages, in left-to-right order. At each stage, the goal is to give a subderivation of a clause $(\bigvee\overline{\pi})$. This is done by considering a regular refutation of $\mathrm{GGT}_n \restriction \pi$, and then weakening to get $(\bigvee\overline{\pi})$, and then replacing GT_n initial clauses with GGT_n clauses as needed.

In some cases, it is possible to further transform the refutation (be introducing extra side literals) so as to keep the partially completed refutation valid.

But in some cases, there is an initial axiom $x_{i,j}, x_{j,k}, x_{k,i}$ which cannot be derived from its GGT_n initial clauses without violating regularity.

In these cases, we instead add a subproof that learns $x_{i,j}, x_{j,k}, x_{k,i}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{x_{i,j}, x_{j,k}, x_{k,i}, x_{r.s} \qquad x_{i,j}, x_{j,k}, x_{k,i}, \overline{x}_{r.s}}{x_{i,j}, x_{j,k}, x_{k,i}} \quad \begin{matrix}(\bigvee \pi_1)\\ \vdots\end{matrix}}{x_{i,j}, x_{j,k}, C_1} \quad \begin{matrix}(\bigvee \pi_2)\\ \vdots\end{matrix}}{x_{i,j}, C_2} \quad \begin{matrix}(\bigvee \pi_3)\\ \vdots\end{matrix}}{(\bigvee \overline{\pi})}$$

- The regularity condition will hold.
- Side literals C_1, C_2 can be chosen to make the resolution inferences valid.
- Each $(\bigvee \pi_i)$ is a bipartite partial restriction that will be handled in later stages. The omitted parts contain inferences to ensure this.
- This construction is needed only polynomially many time since there are only polynomially many GT_n initial clauses. $\square$

Open Questions.

Can this be extended, for instance, is it possible that pool resolution or even regWRTI p-simulates full resolution? In this case, (non-greedy) DPLL clause learning without restarts will simulate full resolution.

Next cases to consider: The Stone tautologies of [AJPU] or the refined versions by [Urquhart '11] have polynomial size regWRTI or pool resolution proofs?

Meta-questions:

1. Can we find more systematic or rigorous methods for evaluating the effectiveness of different proof search hueristics.

2. For example, there are at least four competing theories for why restarts are so useful:

- High variance in search space size.
- Back door sets.
- Inferring easy clauses before a deep search.
- Clearing the search space of unwanted decisions.

Can we systematically evaluate why restarts work well?

Can logical considerations help?

Thank you!