60 Years of Boolean Satisfiability Solving

From the Foundations of Mathematics to Industrial Applications

21 February 2020

Wolfgang Küchlin

Symbolic Computation Group
Wilhelm-Schickard-Institute of Informatics
Faculty of Mathematics and Sciences

Universität Tübingen

Steinbeis Technology Transfer Centre
Object- und Internet-Technologies (STZ OIT)

Wolfgang.Kuechlin@uni-tuebingen.de
http://www-sr.informatik.uni-tuebingen.de

SR

Foreword

- 1986: My thesis: *Equational Completion by Proof Simplification*
  - Simplifying equational proofs s = …. = t in term algebras, using critical pairs.
  - Simplest proof has V-shape: reduce both s and t to unique normal form
- 2003: Hilbert´s 24th problem [Thiele. *American Math. Monthly* **110**]
  - *„The 24th problem in my Paris lecture was to be: Criteria of simplicity, or proof of the greatest simplicity of certain proofs. … Attempts at judging the simplicity of a proof are in my examination of syzygies, and syzygies between syzygies.“ [Note left by Hilbert, see Thiele]*

Contents

- 1960: The Origins of Boolean Satisfiability-Solving
  - Proving First Order Inconsistency by Boolean Un-Satisfiability
  - Davis-Putnam (1960): Variable Elimination by Resolution
  - Davis-Logemann-Loveland (1962): Search for a model (DPLL)
- 1996: Conflict Driven Clause Learning
  - J. P. Marques-Silva, K. A. Sakallah (1996): CDCL
    - Combine DPLL search with resolution where search for model fails
- 2000+: Large Scale Industrial Applications
  - Microelectronics, Car Configuration, Software Verification
  - SAT ecosystem. Efficient algorithms for: Prime Implicants, Explanations for SAT and UNSAT, Optimization, Bounded Model Checking, SAT modulo Theories (SMT), …

The origins of SAT Solving: First Order Proof

- Martin Davis, Hilary Putnam (1960)

"The hope that mathematical methods employed in the investigation of formal logic would lead to purely computational methods for obtaining mathematical theorems goes back to Leibniz and has been revived by Peano around the turn of the century and by Hilbert's school in the 1920's.

Hilbert, noting that all of classical mathematics could be formalized within quantification theory, declared that the problem of finding an algorithm for determining whether or not a given formula of quantification theory is valid was ***the central problem of mathematical logic****."*

[Davis, Putnam. A Computing Procedure for Quantification Theory. *J.ACM* 7, 1960].

- First Order Proof by Herbrand´s Theorem

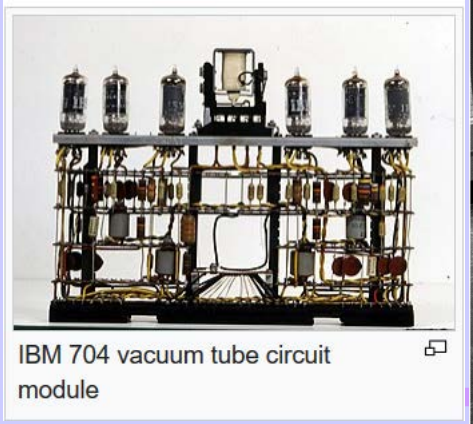
- Method: Enumerate the Herbrand Base of a Predicate Logic formula and check each enumeration level for consistency in Propositional Logic.
  - Herbrand Base: the set of non-variable („ground") instances of the first order formula.
- Example: $\forall x.P(x) \wedge \exists x.\neg P(f(x)) \cong_{(Skolem)} \forall x.P(x) \wedge \neg P(f(a))$
  - 1st level: {P(a), ¬P(f(a)}: consistent (with only a in the Herbrand Universe)
  - 2nd level: {P(a), ¬P(f(a), P(f(a))}: inconsistent (with both a and f(a) in the universe)

Origins of SAT Solving: Methods for First Order Proof

- **Quine**: A proof procedure for quantification theory, 1955.
  - Method: Truth tables
- **Gilmore**: A proof method for quantification theory, 1960.
  - Method: DNF conversion
  - Implemented on IBM 704 (18KB magnetic core memory)
  - „Gilmore formula“: $\exists x,y\ \forall z[F(x,y) \rightarrow (F(y,z)\&F(z,z)) \ \& ((F(x,y)\&G(x,y)) \rightarrow (G(x,z)\&G(z,z)))]$
  - Failed at Herbrand level 7 after 21 minutes on IBM 704 (out of memory)
    - Obviously due to DNF-Explosion
- IBM 704 Computer
  - vacuum tube electronics, 12K floating point additions per sec
  - 18KB magnetic core memory
  - 5 tape units @ 4MB each
  - 123 units sold 1955 – 1960

IBM 704 (1954 – 1960) (source: wikipedia)

- The **IBM 704**, introduced by IBM in 1954, is the first mass-produced computer with floating-point arithmetic hardware. The 704 can execute up to 12,000 floating-point additions per second. Like the 701, the 704 uses vacuum tube logic circuitry and 36-bit binary words. Changes from the 701 include the use of core memory instead of Williams tubes … IBM sold 123 type 704 systems between 1955 and 1960.
- Controls are included in the 704 for: one 711 Punched Card Reader, one 716 Alphabetic Printer, one 721 Punched Card Recorder, five 727 Magnetic Tape Units and one 753 Tape Control Unit, one 733 Magnetic Drum Reader and Recorder, and one 737 Magnetic Core Storage Unit. Weight: about 19,466 pounds (8.8 t).
- The 737 Magnetic Core Storage Unit serves as RAM and provides 4,096 36-bit words, the equivalent of 18,432 bytes. The 727 Magnetic Tape Units store over five million six-bit characters per reel.


IBM 704 vacuum tube circuit module

Origins of SAT Solving: Methods for First Order Proof

- **Davis & Putnam** (1960): Eliminate variables by resolution
  - In clause set F: (1) propagate Unit {x}; (2) eliminate clauses with pure literals.
  - (3) with impure literals p, ¬p, rearrange F into F = (A´ ∨ p) ∧ (B´ ∨ ¬p) ∧ R
  - F is SAT *iff* F´= (A´ ∨ B´) ∧ R is SAT
  - Solved Gilmore Formula by hand in less than 30 minutes
    - Trick: Checked only HB levels 10, 20, 30. Inconsistency first occurs at level 25!
- Example of variable elimination (DP 1960)
  - S_0 = {{x, y, z}, {¬x, y, z}, {¬x}, {z, ¬y}}
  - Rule 3 (resolution on y): S_1 = {{x, z}, {¬x, z}, {¬x}}
  - Rule 1 (unit propagation ¬x): S_2 = {{z}}
  - Rule 2 (Pure Literal z): S_3 = { }, hence consistent.
- DP(1960) solves the Existential QE problem $\exists x_1,\ldots,x_n.F$
  - But EQE is not really SAT-Solving, answer is just *true* or *false*
  - Reason: We may not get a satisfying assignment on impure literals

The origins of SAT Solving: DP vs. D(P)LL

- Davis, Putnam (1960): Eliminate variables by resolution
  - F = **(A´ ∨ p) ∧ (B´ ∨ ¬p) ∧ R** is SAT *iff* F´= **(A´ ∨ B´) ∧ R** is SAT („Rule 3“)
  - **Linear** in #variables! Easy hand computation on small examples.
  - **But** clauses explode: (A´ ∨ B´) equals all n*m resolvents of A and B over p !
- Davis, Logemann, Loveland (1962): Backtrack search for model
  - Try assignment {x, …}, if unsuccessful try {¬x, …}
  - F is SAT *iff* F´= **(A´ ∧ R) ∨ (B´ ∧ R)** is SAT („Rule 3* “)
  - Originally: create both formulas, solve one, put other on stack & solve later
  - Today: Create (A´ ∧ R) as $F_{[p=0]}$, create (B´ ∧ R) as $F_{[p=1]}$ from same F.
  - Formulas are sets of clauses, no clause deletion, creation, CNF conversion.
  - Recursive backtrack search, easy for computers, hard for hand computation
  - Implementation in „SAP“ Assembler *„with many time-saving devices employed“* [DLL 1962] on IBM 704 (32K words memory = 144KB)
  - Gilmore´s example was proved automatically in under 2 minutes!

Lessons Learned from Implementing D(P)LL

The forms of Rule III are interchangeable; although theoretically they are equivalent, in actual applications each has certain desirable features. We used Rule III* because of the fact that Rule III can easily increase the number and the lengths of the clauses in the expression rather quickly after several applications. This is prohibitive in a computer if ones fast access storage is limited. Also, it was observed that after performing Rule III, many duplicated and thus redundant clauses were present. Some success was obtained by causing the machine to systematically eliminate the redundancy; but the problem of total length increasing rapidly still remained when more complicated problems were attempted. Also use of Rule III can seldom yield new one-literal clauses, whereas use of Rule III* often will.

In programming Rule III*, we used auxiliary tape storage. The rest of the testing for consistency is carried out using only fast access storage. When the "Splitting Rule" is used one of the two formulas resulting is placed on tape. Tape memory records are organized in the cafeterial stack-of-plates scheme: the last record written is the first to be read.

- *„we hoped that some mathematically meaningful and, perhaps nontrivial, theorems could be solved. The actual achievements in this direction were somewhat disappointing".*

  [DLL 1962]

Principle of Conflict Driven Clause Learning (CDCL)

- Learning to avoid a bad sequence of decisions
  - A sequence of decisions and propagations may hit a root F=0.
  - But not all of these decisions may be relevant for the root.
- Key insight: start learning process with *conflict clause* K
  - Conflict clause (failure clause) K is the clause which becomes empty in Step 2 of DPLL, i.e. $\beta(K)=0$ under current assignment β
  - The failure is caused by all literals in K becoming 0. This set is already a small subset of β, but may contain propagated literals.
  - Now we can find the subset of decisions, whose conjunction D caused all these literals to become 0.
  - Negating this conjunction gives us a clause L = ¬D which is implied by F, hence can be added to F (learned).
    - D implies ¬F, so $\models(\neg D \vee \neg F)$, i.e. F implies ¬D = L.

Example: Principle of Learning in CDCL

- $S_0 = \{\{x, y\}, \{\neg y, z\}, \{\neg z, x\}\}$. We make the assignments:
  - x=0 (Decision)

Example: Principle of Learning in CDCL

- $S_0 = \{\{x, y\}, \{\neg y, z\}, \{\neg z, x\}\}$. We make the assignments:
  - x=0 (Decision), y=1 (Unit Propagation)

Example: Principle of Learning in CDCL

- $S_0 = \{\{x, y\}, \{\neg y, z\}, \{\neg z, x\}\}$. We make the assignments:
  - x=0 (Decision), y=1 (Unit Propagation), z=1 (Unit Propagation)
  - Conflict clause is K={¬z, x}, Reason for conflict is R = {¬y, z}

Example: Principle of Learning in CDCL

- $S_0 = \{\{x, y\}, \{\neg y, z\}, \{\neg z, x\}\}$. We make the assignments:
  - x=0 (Decision), y=1 (Unit Propagation), z=1 (Unit Propagation)
  - Conflict clause is K={¬z, x}, Reason for conflict is R = {¬y, z}
  - Resolvent on conflict literal z (*first learnt clause*) is $L_1 = \{x, \neg y\}$
  - L_1 is false under current assignment. It contains both a decision variable x and a unit propagation variable y. After backtracking, L_1={x, ¬y} is not unit and not immediately useful.
  - Remove ¬y by resolving with its reason $\{x, y\}, \{x, \neg y\} \vdash \{x\} = L_2$
  - Now backtrack to before the assignment on x. There is no decision left: x=1 now becomes a unit propagation of {x}.
  - In general we continue learning clauses until we hit the first „UIP clause“ (*unique implication point*): It contains a single variable on the highest level of assignment. After backtracking, it is unit.

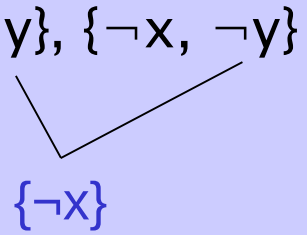
CDCL based proof learning and UNSAT explanation

- Clause set S_0 = { {x, y}, {x, ¬y}, {¬x, y}, {¬x, ¬y} }
  - S_0 [x=1] = { {1, y}, {1, ¬y}, {0, y}, {0, ¬y} } : choose UP y=1
  - S_0 [x=1, y=1] = { {1, 1}, {1, 0}, {0, 1}, {0, 0} } = 0 = conflict!
  - A decision (x=1) forced conflicting propagations y=1 and y=0, obviously by 2 clauses containing {.., y, ..} and {.., ¬y, ..}
  - Hence there is a resolvent on y, in this case {¬x, y}, {¬x, ¬y} ⊢ {¬x}.
  - Add {¬x} to C, because it is a logical consequence of C.
  - Backtrack to just before the decision on x (no matter how far!). Now x=0 is a forced unit propagation by {¬x} (no more decision)
  - S_1 = { {¬x},{x, y}, {x, ¬y}, {¬x, y}, {¬x, ¬y} } : propagate x=0
  - S_1[x=0] = { {1}, {0, y}, {0, ¬y}, {1, y}, {1, ¬y} } : choose UP y=0
  - S_1[x=0, y=1] = { {1}, {0, 0}, {0, 1}, {1, 1}, {1, 0} } = conflict !
  - Hence there is a resolvent on y, in this case {x, y}, {x, ¬y} ⊢ {x}, add {x} to S_1
  - Without any decision on x, we have a final conflict in S_2 = {{¬x}, …, {x}} ⊢ □

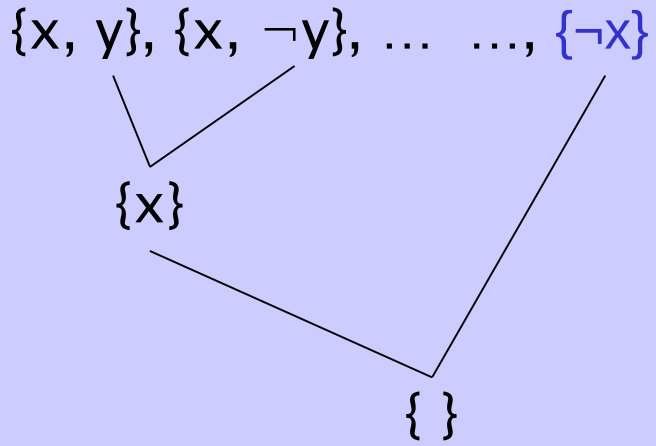
Resolution proof explaining UNSAT(C)

$S_0 = \{ \{x, y\}, \{x, \neg y\}, \{\neg x, y\}, \{\neg x, \neg y\} \}$

Resolution Proof of UNSAT($S_0[x=1]$), respectively of $S_0 \models \{\neg x\}$:



Final proof of UNSAT(S_0):



The answer is easy if you take it logically (Paul Simon)

Some Time-Points in SAT History

- 1960/62: First order proof of Gilmore formula by SAT solving
  - Davis, Putnam (J.ACM 1960); Davis, Logemann, Loveland (C.ACM 1962).
- 1971: SAT is the first NP-complete problem
  - Stephen Cook: *The complexity of theorem-proving procedures*. (STOC'71)
- 1992 / 1994: SAT beats specialized application software
  - H. Kautz, B. Selman. *Planning as satisfiability*. (ECAI'92)
  - H. Zhang: SATO solver. McCune: Quasi-group existence problems (1994)
- 1996: Clause Learning: Combining DPLL with resolution
  - J. P. Marques-Silva, K. A. Sakallah (1996): GRASP solver (CAD 96)
- 2000+: Industrial Applications (Analysis and Verification)
  - N. Eén, N. Sörensson: Minisat Solver (in C). D. LeBerre: SAT4J (in Java)
  - Hardware (Microelectronic Circuits): net-lists are switching algebra
  - Software (Bounded Model Checking: compile software into Boolean circuit)
  - Configuration (variant rich car configuration and parts selection rules)

Literature

1. Paul C. Gilmore. A proof method for quantification theory. *IBM J. Research and Development* **4** (1960), 28—35.
2. M. Davis and H. Putnam. A computing procedure for quantification theory. *J.ACM* 7(3), 1960
3. M. Davis, G. Logemann and D. Loveland. A machine program for theorem proving. *C.ACM* 5, 1962
4. A. Biere, M. Heule, H. van Maaren, T. Walsh (eds.). *Handbok of Satisfiability*. IOS Press 2009. (Comprehensive current account of SAT based methods)
5. J. P. Marques-Silva. *Search Algorithms for Satisfiability Problems in Combinatorial Switching Circuits*. PhD Thesis, U. Michigan, 1995
6. J. P. Marques-Silva, K. A. Sakallah. GRASP: A new search algorithm for satisfiability. In: *Intl. Conf. Computer Aided Design*., Nov 1996.
7. J. P. Marques-Silva, K. A. Sakallah. GRASP: A search algorithm for propositional satisfiability. In: *IEEE Transactions on Computers*., May 1999.
8. D. E. Knuth. Satisfiability. *The Art of Computer Programming* Vol 4 Fasc. 6. Addison Wesley, 2016

Appendix: Some Boolean Decision Procedures

- Bad news: SAT is NP-complete (Cook 1971)
- Good News: SAT(F) is decidable and solves all NP-complete problems!
  - Truth Tables → guaranteed exponential (#variables), toy problems only
  - Disjunctive normal form (DNF) → easily exp., small problems ok.
  - Tableaux → similar to DNF, easily exponential, small problems ok.
  - Boolean Polynomials → „Stone polynomials“, canonical form, little use.
  - Binary Decision Diagrams (ROBDD) → model checking use, 100s variables ok, O(1) SAT-solving, easy model counting, canonical form.
  - Propositional Resolution → too many deductions, theoretical importance
  - Davis-Putnam-Logemann-Loveland (DPLL) → small problems ok.
  - DPLL based CDCL SAT-Solving → practically efficient for science and industry, 100,000+ variables, method of choice, very robust, much research.

Example: SAT-Solving with DPLL

- $S_0 = \{\{x, y, z\}, \{\neg x, y, z\}, \{z, \neg y\}\}$
  - Heuristically choose x as 1st decision variable (level 1):
    - Case 1: let x=1
    - $S_1 = S_0[x=1] = \{\{x, y, z\}, \{\neg x, y, z\}, \{z, \neg y\}\}$
  - Heuristically choose z as 2nd decision variable (level 2):
    - Case 1: let z=0 in S_1
    - $S_2 = S_0[x=1][z=0] = \{\{x, y, z\}, \{\neg x, y, z\}, \{z, \neg y\}\}$
    - Unit propagate y=1:
      $S_3 = S_0[x=1][z=0][y=1] = \{\{x, y, z\}, \{\neg x, y, z\}, \{z, \neg y\}\}$ = false
    - Conflict! Backtrack to last decision!
    - Case 2: let z=1 in S_1
    - $S_4 = S_0[x=1][z=1] = \{\{x, y, z\}, \{\neg x, y, z\}, \{z, \neg y\}\}$
  - Inspection of clauses shows that both y and x are „don‘t care“
    - implicant {x, y, z} can be reduced to prime implicant {z}

The Idea of Conflict Driven Clause Learning (CDCL)

- $S_0 = \{\{x, y, z\}, \{\neg x, y, z\}, \{z, \neg y\}\}$
  - Heuristically choose x as 1st decision variable (level 1):
    - Case 1: let x=1
    - $S_1 = S_0[x=1] = \{\{x, y, z\}, \{\neg x, y, z\}, \{z, \neg y\}\}$
  - Heuristically choose z as 2nd decision variable (level 2):
    - Case 1: let z=0 in S_1
    - $S_2 = S_0[x=1][z=0] = \{\{x, y, z\}, \{\neg x, y, z\}, \{z, \neg y\}\}$
    - Unit propagate y=1:
      $S_3 = S_0[x=1][z=0][y=1] = \{\{x, y, z\}, \{\neg x, y, z\}, \{z, \neg y\}\}$ = false
    - Conflict due to complementary literals y, ¬y!
    - Learning: There is a resolvent $\{\neg x, y, z\}, \{z, \neg y\} \vdash \{\neg x, z\}$ we can learn.
    - Backtrack to level 1: $S_0[x=1] \cup \{\neg x, z\} = \{\{x, y, z\}, \{\neg x, y, z\}, \{z, \neg y\}, \{\neg x, z\}\}$
    - Unit propagate z=1: $\{\{x, y, z\}, \{\neg x, y, z\}, \{z, \neg y\}, \{\neg x, z\}\}$
    - We learned that x=1 implies z=1, no more decision on z.