TOWARDS A DEEPER EMPIRICAL UNDERSTANDING OF CDCL SAT SOLVERS

Vijay Ganesh
Assistant Professor, University of Waterloo, Canada

Constraints Workshop
Automata, Logic and Games
Aug 29, 2016
Institute for Mathematical Sciences, NUS, Singapore
SOFTWARE ENGINEERING & SAT/SMT SOLVERS
AN INDISPENSABLE TACTIC FOR ANY STRATEGY
SAT/SMT SOLVER RESEARCH STORY
A 1000X+ IMPROVEMENT

- Solver-based programming languages
- Compiler optimizations using solvers
- Solver-based debuggers
- Solver-based type systems
- Solver-based concurrency bugfinding
- Solver-based synthesis
- Bio & Optimization

- Concolic Testing
- Program Analysis
- Equivalence Checking
- Auto Configuration

- Bounded MC
- Program Analysis
- AI

1,000,000 Constraints

100,000 Constraints

10,000 Constraints

1,000 Constraints

# PAST CONTRIBUTIONS

<table>
<thead>
<tr>
<th>Name</th>
<th>Key Concept</th>
<th>Impact</th>
<th>Pubs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>STP</strong> Bit-vector &amp; Array Solver(^1,2)</td>
<td>Abstraction-refinement for solving</td>
<td>Concolic Testing</td>
<td>CAV 2007, CCS 2006, TISSEC 2008</td>
</tr>
<tr>
<td><strong>HAMPI</strong> String Solver(^1)</td>
<td>App-driven bounding for solving</td>
<td>Analysis of Web Apps</td>
<td>ISSTA 2009, TOSEM 2012, CAV 2011</td>
</tr>
<tr>
<td>Taint-based Fuzzing</td>
<td>Data flow is cheaper than concolic</td>
<td>Scales better than concolic</td>
<td>ICSE 2009</td>
</tr>
<tr>
<td>Automatic Input Rectification</td>
<td>Automatic security envelope</td>
<td>New security approach</td>
<td>ICSE 2012</td>
</tr>
<tr>
<td>Undecidability results for theories over strings</td>
<td>New reductions</td>
<td>Solved open problems</td>
<td>HVC 2012</td>
</tr>
</tbody>
</table>

1. 100+ research projects use STP, HAMPI, and Z3str2.
2. STP won the SMTCOMP 2006/2010 and second in 2011/2014 competitions for bit-vector solvers.
3. Best paper awards/honors at various conferences including SAT, DATE, SPLC, CAV, and CADE.
MOTIVATION:
THE UNREASONABLE EFFECTIVENESS OF CDCL SAT SOLVERS

• Isn’t SAT supposed to be a hard problem? After all it is NP-complete
  • Yes, SAT is NP-complete. And, yes, in general we believe it to be hard
  • However, for many classes of large industrial instances SAT solvers are very efficient
  • This is a mystery that has stumped theoreticians and practitioners alike for over 2 decades

• Goal
  • Deepen understanding of this phenomenon
  • And as a consequence, build more efficient SAT solvers
  • Apply this understanding to build better SMT solvers, verification, analysis, synthesis tools, …
1. Understand the input structure of SAT instances (e.g., community structure, backdoors,…) [WGS03, BSG09, AL12, NGFAS14,…]

2. Identify the most important techniques in CDCL SAT solvers [KSM11,…]

3. Define a precise model encompassing the most important techniques, e.g., branching [LGZC15]

4. Put all of this together in a complexity-theoretic framework [AFT11, PD11,…]
What is a SAT/SMT Solver?
Automation of Mathematical Logic

Logic Formula
(q v p v r)
(q v p v r)
...

Solver

SAT
UNSAT

- Rich logics (Modular arithmetic, arrays, strings, non-linear arithmetic, theories with quantifiers, ...)
- From proof procedures to validity to satisfiability
- SAT problem is NP-complete, PSPACE-complete,...
- Practical, scalable, usable, automatic
- Enable novel software reliability approaches
A literal $p$ is a Boolean variable $x$ or its negation $\neg x$. A clause $C$ is a disjunction of literals: $x_2 \lor \neg x_4 \lor x_{15}$.

A CNF is a conjunction of clauses: $(x_2 \lor \neg x_1 \lor x_5) \land (x_6 \lor \neg x_2) \land (x_3 \lor \neg x_4 \lor \neg x_6)$.

An assignment is a mapping from variables to Boolean values (True, False). A unit clause $C$ is a clause with a single unbound literal.

The Boolean SAT problem is:

- Find an assignment such that each input clause has a true literal (aka input formula is SAT) OR establish that input formula has no solution (aka input formula is UNSAT).
- SAT solvers are required to output a solution if input is SAT (many solvers also produce a proof if input is UNSAT).
- Boolean formulas are typically represented in DIMACS Format.
DPLL SAT SOLVER ARCHITECTURE
THE BASIC SOLVER

\begin{dpll}
\begin{align*}
&DPLL(\Theta_{cnf}, \text{assign}) \{ \\
&\quad \text{Propagate unit clauses;} \\
&\quad \text{if } "\text{conflict}" : \text{return } \text{FALSE}; \\
&\quad \text{if } "\text{complete assign}" : \text{return } \text{TRUE}; \\
&\quad "\text{pick decision variable } x"; \\
&\quad \text{return } \\
&\quad \quad DPLL(\Theta_{cnf} \mid x=0, \text{assign}[x=0]) || DPLL(\Theta_{cnf} \mid x=1, \text{assign}[x=1]); \\
&\}\end{align*}
\end{dpll}

Key Steps in a DPLL SAT Solver

- Propagate (Boolean Constant Propagation)
  - Propagate inferences due to unit clauses
  - Most of solving “effort” goes into this step

- Detect Conflict
  - Conflict: partial assignment is not satisfying

- Decide (Branch)
  - Choose a variable & assign some value

- Backtracking
  - Implicitly done via recursive calls in DPLL
MODERN CDCL SAT SOLVER ARCHITECTURE
KEY STEPS AND DATA-STRUCTURES

Key steps

- Decide()
- Propagate() (Boolean constant propagation)
- Conflict analysis and learning() (CDCL)
- Backjump()
- Forget()
- Restart()

CDCL: Conflict-Driven Clause-Learning

- Conflict analysis is a key step
- Results in learning a learnt clause
- Prunes the search space

Key data-structures (Solver state)

- Stack or trail of partial assignments (AT)
- Input clause database
- Conflict clause database
- Conflict graph
- Decision level (DL) of a variable
• Motivation
• Background: Internals of SAT solvers
• Problem statement

• Our empirical discoveries
  1. Characterizing industrial instances via community structure [LGRC15, NGFAS14]*
  2. Understanding branching heuristics, a la, VSIDS [LGZC15]
  3. LRB: A new branching heuristic [LGPC16, LGPC+16]**
• Putting it all together, and conclusions

(*Won best paper award at SPLC 2015, and student paper award at SAT 2014. **Won the SAT competition 2016)
Question: How does one go about characterizing the structure of SAT instances?

- Theorists have proposed a variety of metrics like backdoors, treewidth, …

- We considered these measures, but found them to be inadequate in various ways. E.g., weak backdoors do not apply to unsatisfiable instances. Tree width has been shown to not correlate well with SAT solver performance.

Answer: The intuition we had was to focus on metrics that somehow mathematically characterize “important parts” of the SAT formula. We hope to argue that the solver somehow focuses on these “important parts” first, and thus solves very large instances without needing to “process” the entire formula.

- Defining the “important part of a SAT formula” in a generic way is easier said than done

- Community structure to rescue

- View SAT formula as a variable-incidence graph. Partition the graph via the concept of community structure. Focus on highly-connected components.
Community structure [GN03,CNM04,OL13] is used to study all kinds of complex networks.

- Social Networks, e.g. Facebook
- Internet
- Protein networks
- Neural network of the human brain
- Citation graphs
- Business networks
- Populations
- And more recently, the graph of logical formulas
COMMUNITY STRUCTURE IN GRAPHS
VARIABLE-INCIDENCE GRAPH OF NON-RANDOM FORMULA
COMMUNITY STRUCTURE IN GRAPHS

VARIABLE-INCIDENCE GRAPH OF RANDOM FORMULA
Modularity (Q) of graph lies between 0 and 1

- A weighted ratio of the number of edges inside communities to the total number of edges in the graph

- Q measures quality
  - How separable are the communities in the graph
  - Higher Q implies “good community structure”, i.e. highly separable communities
  - Lower Q implies “bad community structure”, i.e. one giant hairy ball
COMMUNITY STRUCTURE AND SAT SOLVER PERFORMANCE
EMPIRICAL RESULTS

- **Result #1**: Hard random instances have low $Q$ ($0.05 \leq Q \leq 0.13$)

- **Result #2**: Number of communities and $Q$ of SAT instances are more predictive of CDCL solver performance than other measures we considered

- **Result #3**: Strong correlation between community structure and LBD (Literal Block Distance) in Glucose solver
COMMUNITY STRUCTURE AND SAT SOLVER PERFORMANCE
CHARACTERIZING TYPICAL INPUTS

- **Take-home Message**
  - Community structure (the quality of which is measured using a metric called Q) of SAT instances strongly affect solver performance

- **Working towards**
  - Small predictive set of features that forms the basis for a more complete explanation
  - Community structure idea needs to be constrained further to eliminate easy theoretical counter-examples
  - More broadly, community structure is a starting point for a line of research that enables us to identify “important sub-parts” of SAT formulas statically
  - Community structure and learning-sensitive with restarts (LSR) backdoors
Impact of the Community Structure Idea

- New techniques to exploit community structure by Giraldez-Cru, Levy, Simon, ...
- A flourishing of approaches being applied to partition SAT instances and identify the “most important” part to focus on first
- Pseudo-industrial instance generators for SAT instances based on community structure
- Theorists have been looking at community structure trying to refine it further
- Comprehensive analyses of other structure metrics to explain SAT solver performance
Motivation

Background: Internals of SAT solvers

Problem statement

Our empirical discoveries

1. Characterizing industrial instances via community structure [LGRC15, NGFAS14]*
2. Understanding branching heuristics, a la VSIDS [LGZC15]
3. LRB: A new branching heuristic [LGPC16, LGPC+16]**

Putting it all together, and conclusions

(* Won best paper award at SPLC 2015, and student paper award at SAT 2014. ** Won the SAT competition 2016)
MODERN CDCL SAT SOLVER ARCHITECTURE
KEY STEPS AND DATA-STRUCTURES

Key steps

- Decide()
- Propagate() (Boolean constant propagation)
- Conflict analysis and learning() (CDCL)
- Backjump()
- Forget()
- Restart()

CDCL: Conflict-Driven Clause-Learning

- Conflict analysis is a key step
- Results in learning a learnt clause
- Prunes the search space

Key data-structures (Solver state)

- Stack or trail of partial assignments (AT)
- Input clause database
- Conflict clause database
- Conflict graph
- Decision level (DL) of a variable
Question: What is a variable selection (branching) heuristic?

Answer: A branching heuristic is a ranking function that takes as input all variables in an input formula, assigns a score to each variable, and outputs a ranking of these variables based on this score (activity).
PROBLEM STATEMENT

What computational problem does a branching heuristic solve?

Use the resultant understanding to improve the effectiveness of branching heuristics in CDCL SAT solvers.

ANSWER

View branching heuristic as technique to solve an optimization problem with the following objective function, namely, learning rate. Use reinforcement learning to solve this problem.

The result is the LRB branching heuristic that outperforms VSIDS on a large, comprehensive set of benchmarks from SAT 2009-2014. MapleSAT solves 100+ more instances than MiniSAT. Also, MapleCMS is competitive against CMS, Glucose, and Lingeling.
Proposed by the authors of the Chaff solver in 2001.

Give more weight to recent conflicts.

Little computational overhead.

Implemented by most competitive CDCL solvers.

**Bumping**

Every variable has a floating-point number called “activity” score.

Activity scores are initialized to zero.

Add one (“bump”) to all the activities of variables occurring in conflict analysis.

**Decaying**

Recent conflict analysis is more pertinent to the current state of the solving.

Multiply the activities of every variable by $0 < d < 1$ after every conflict (“decay”).
VSIDS QUESTIONS

Why does VSIDS perform multiplicative decay?

Which variables does VSIDS bump, and why?

Does VSIDS demonstrate locality?

ANSWERS

Exponential moving average (EMA)

High temporal degree variables, and bridge variables in the community structure

Yes, focuses on few communities (Paper @ HVC 2015)
1. This abstraction (or meta-algorithm) captures the most essential aspects of CDCL. Combines synthesis (induction) with verification (deduction)

2. Common in formal methods and synthesis, e.g., DPLL(T) in SMT solvers, model-checkers, learning-based synthesis algorithms

3. There is similar class of algorithms in reinforcement learning (also, actor-critic model)

4. Enabled us to design a new class of branching heuristics
Maximize:

1. Maximize the “quantity” of learnt clauses per unit time, i.e., maximize learning rate

2. Optimize the “quality” of learnt clauses
   1. We don’t know how to do this yet
   2. Possible candidates for quality include clause length, LBD,…

BRANCHING HEURISTIC SOLVES MULTI-OBJECTIVE OPTIMIZATION PROBLEM
REINFORCEMENT LEARNING AND CDCL

**Reinforcement Learning**
- Agent
- Environment
- Policy
- Action
- Estimated Reward (Q)
- Reward
- Exponential Moving Average

**CDCL**
- Branching Heuristic + BCP
- Clause learning
- Variable Ranking
- Decision
- Activity
- Bump
- Decay
EXPONENTIAL MOVING AVERAGE (EMA)
LEARNING RATE EXAMPLE

A = false, B = true, C = false,…

Learnt Clause: A or C

Student → Teacher

A = false, B = true, D = false,…

Learnt Clause: D or C

Student → Teacher

A = false, C = true, D = false,…

Learnt Clause: A or D

Student → Teacher

B = true, D = false,…

Learnt Clause: D or E

Student → Teacher

sampled_learning_rate(A) = 2/3

sampled_learning_rate(B) = 0/3
LEARNING RATE

learning_rate(\(X\)) = P(\(X\) is in conflict analysis | \(X\) is assigned AND solver is in conflict)

Problem 1: Unfortunately, learning rate is hard to compute.
  • Can we estimate it?

Problem 2: Also, learning rate is constantly changing.
  • Need to adapt our estimate over time in an online fashion

Solution:

Reinforcement learning techniques are ideal for such optimization problems
MULTI-ARMED BANDIT PROBLEM TO MODEL BRANCHING HEURISTICS

Sample average = \[ \frac{1}{3} \times 4 + \frac{1}{3} \times 3 + \frac{1}{3} \times 1 \]

Exponential moving average = \[ (1 - \alpha)^2 \times 4 + (1 - \alpha) \times 3 + (1 - \alpha)^0 \times 1 \]
LEARNING-RATE BRANCHING (LRB) EXAMPLE

sampled_learning_rate(A) = 2/3

exponential moving average = \((1 - \alpha)^1 \times 2/3\) + \((1 - \alpha)^0 \times 1/3\)

A is assigned

B or D

A or D

B or C

B or E

A or B

C or D

A or E

C or E

A is unassigned

A is assigned

A is unassigned

“Rewards”

Activity(A)
**VSIDS**

**The reward is a constant**
Every time a variable appears in a learnt clause, its activity is additively bumped by a constant

**EMA performed for all variables at the same time**
After each conflict, the activities of all variables are decayed

---

**LRB (without extensions)**

**The reward is not constant**
Every time a variable appears in a learnt clause, the numerator of its learning rate reward is incremented. After each conflict, the denominator of each assigned variable's learning rate reward is incremented

**EMA performed only when variable goes from assigned to unassigned**
When a variable is unassigned, the variable receives the learning rate reward, and the estimate $Q$ is updated
EXPERIMENTAL SETUP

• Benchmark consists of all the instances in the application and hard combinatorial categories from four previous SAT competitions (2009, 2011, 2013, 2014)

• Total number of instances: 1975

• Experiments performed on StarExec cluster with 5000 second timeout

• The solvers we used in our experiments are:
  • MiniSat 2.2.0 with VSIDS replaced with CHB and LRB (MapleSAT)
  • CryptoMiniSat 4.5.3 with VSIDS replaced with LRB (MapleCMS)
  • Lingeling bal from 2015 SAT Race
  • Glucose 4.0
APPLE-TO-APPLE RESULTS
(MINISAT WITH VSIDS VS. CHB VS. LRB)
COMPARISON WITH STATE-OF-THE-ART: CRYPTOMINISAT, MAPLECMS, GLUCOSE, AND LINGELING
RESULT: GLOBAL LEARNING-RATE

• Global Learning Rate: \# of conflicts/\# of decisions

• Experimental setup: ran 1200+ application and hand-crafted instances on MapleSAT with VSIDS, CHB, LRB, Berkmin, DLIS, and JW with 5400 sec timeout per instance on StarExec

<table>
<thead>
<tr>
<th>Branching Heuristic</th>
<th>Global Learning Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRB</td>
<td>0.452</td>
</tr>
<tr>
<td>MVSIDS</td>
<td>0.410</td>
</tr>
<tr>
<td>CHB</td>
<td>0.404</td>
</tr>
<tr>
<td>CVSIDS</td>
<td>0.341</td>
</tr>
<tr>
<td>BERKMIN</td>
<td>0.339</td>
</tr>
<tr>
<td>DLIS</td>
<td>0.241</td>
</tr>
<tr>
<td>JW</td>
<td>0.107</td>
</tr>
</tbody>
</table>
LRB makes fewer decisions but each decision has a higher probability of generating a conflict.
RESULT: DECISIONS OVER TIME

LRB makes significantly fewer decisions, but still solves the most instances.
RESULT: BINARY HEAP OVERHEAD

LRB has a higher performance overhead due to more binary heap manipulations.
FUTURE WORK: QUALITY OF LEARNING

• Simultaneously maximize learning rate and maximize the “quality” of learnt clauses.

• Not all learnt clauses are created equal, good learnt clauses prune a lot of remaining search space.

• Should branch on variables that will lead to high quality learnt-clauses.

• Need to consider the trade-off between the two objectives: learning rate and quality.
THREATS TO VALIDITY

- Do we over-tune on the SAT benchmarks?
  - We didn’t do much parameter tuning
  - We designed the solver over SAT competition benchmarks, but tested on a much larger set of benchmarks for crypto and MathCheck examples
  - LRB-based solvers out-perform other solvers including Lingeling

- We implemented the LRB heuristic in a wide variety of solvers including Glucose, CMS, MiniSAT, and COMSPS. The LRB-versions out-perform the non-LRB versions of these SAT solvers

- LRB is better even though its data structures are not tuned relative to VSIDS, MVTF, …
REINFORCEMENT LEARNING FOR POLICY DESIGN

• Solvers can be seen as a collection of heuristics, each of which is solving an optimization problem

• Often adaptive techniques (e.g., reinforcement learning) outperform static ones

• Also, reinforcement learning can be used to choose between different heuristics in an adaptive way

• We have implemented a reinforcement learning hybrid policy for restarts in MapleSAT.
CONCLUSIONS ON BRANCHING HEURISTICS

• View branching heuristics as a technique to solve an optimization problem whose objective function is defined in terms of learning rate

• LRB (and CHB) is shown to be a significant improvement over state-of-the-art VSIDS branching heuristic, for the first time in 15 years

• Reinforcement learning for adaptive dynamic policies, hybrid policies, restarts, clause deletion

• Leverage reinforcement learning to solve such optimization problems, making use of the enormous amount of online data generated by a CDCL SAT solver

• Online policy optimization is everywhere in SAT solvers
TALK OUTLINE

• Motivation

• Background: Internals of SAT solvers

• Problem statement

• Our empirical discoveries
  1. Characterizing industrial instances via community structure [LGRC15, NGFAS14]*
  2. Understanding branching heuristics, a la, VSIDS [LGZC15]
  3. LRB: A new branching heuristic [LGPC16, LGPC+16]**

• Putting it all together, and conclusions
CONCLUSIONS: UNDERSTANDING THE UNREASONABLE EFFECTIVENESS OF CDCL SAT SOLVERS

Result #1: Understand the input structure of industrial SAT instances (contrast with randomly hard ones)

1. **Understanding**: established empirically that community structure is more predictive of solver runtime than other measures
2. **New techniques**: many new proposals use community structure to refine SAT solver heuristics
3. **Impact**: potential deeper theoretical understanding, in conjunction with Result #2

Result #2: Branching heuristics are techniques to solve optimization problem of maximizing learning rate

1. **Understanding**: established much deeper understanding of branching heuristics via a model of SAT solver as a combination of reinforcement learning and a deductive corrective feedback algorithm
2. **New techniques**: invented LRB and CHB, first branching heuristics to beat VSIDS in 15+ years
3. **Impact**: winner of SAT competition 2016

Theoretical work? Ongoing
## CURRENT PROJECTS AND CONTRIBUTIONS

<table>
<thead>
<tr>
<th>Name</th>
<th>Key Concept</th>
<th>Impact</th>
<th>Pubs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Z3-str String and Numeric Solver</strong></td>
<td>Novel techniques for string + integer combination</td>
<td>Analysis of Web Apps</td>
<td>FSE 2013, CAV 2015</td>
</tr>
<tr>
<td><strong>Maple Series of SAT Solvers, and</strong></td>
<td>Branching heuristics, and community structure</td>
<td>Maple series of SAT solvers are among the fastest</td>
<td>AAAI 2016, SAT 2016, HVC 2015, SAT 2014</td>
</tr>
<tr>
<td><strong>understanding SAT</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MathCheck Conjecture Verifier</strong></td>
<td>CAS+SAT combination</td>
<td>Finitely verified math conjectures</td>
<td>IJCAI 2016, CASC 2016, CADE 2015</td>
</tr>
<tr>
<td><strong>Undecidability results for theories</strong></td>
<td>New reductions</td>
<td>Solved open problems</td>
<td>HVC 2012, Under submission</td>
</tr>
<tr>
<td><strong>over strings</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Attack-resistance</strong></td>
<td>A new approach to formally establishing the efficacy of security defenses</td>
<td></td>
<td>PLAS 2015</td>
</tr>
</tbody>
</table>

1. 100+ research projects use STP, HAMPI, and Z3str2.
2. STP won the SMTCOMP 2006/2010 and second in 2011/2014 competitions for bit-vector solvers.
3. Best paper awards/honors at various conferences including SAT, DATE, SPLC, CAV, and CADE.
## PAST CONTRIBUTIONS

<table>
<thead>
<tr>
<th>Name</th>
<th>Key Concept</th>
<th>Impact</th>
<th>Pubs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>STP</strong> Bit-vector &amp; Array Solver&lt;sup&gt;1,2&lt;/sup&gt;</td>
<td>Abstraction-refinement for solving</td>
<td>Concolic Testing</td>
<td>CAV 2007, CCS 2006, TISSEC 2008</td>
</tr>
<tr>
<td><strong>HAMPI</strong> String Solver&lt;sup&gt;1&lt;/sup&gt;</td>
<td>App-driven bounding for solving</td>
<td>Analysis of Web Apps</td>
<td>ISSTA 2009&lt;sup&gt;3&lt;/sup&gt;, TOSEM 2012, CAV 2011</td>
</tr>
<tr>
<td>Taint-based Fuzzing</td>
<td>Data flow is cheaper than concolic</td>
<td>Scales better than concolic</td>
<td>ICSE 2009</td>
</tr>
<tr>
<td>Automatic Input Rectification</td>
<td>Automatic security envelope</td>
<td>New security approach</td>
<td>ICSE 2012</td>
</tr>
<tr>
<td>Undecidability results for theories over strings</td>
<td>New reductions</td>
<td>Solved open problems</td>
<td>HVC 2012</td>
</tr>
</tbody>
</table>

1. 100+ research projects use STP, HAMPI, and Z3str2.
2. STP won the SMTCOMP 2006/2010 and second in 2011/2014 competitions for bit-vector solvers
3. Best paper awards/honors at various conferences including SAT, DATE, SPLC, CAV, and CADE
<table>
<thead>
<tr>
<th>Category</th>
<th>Research Project</th>
<th>Researcher/Institution/Time Period</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Theorem Proving</strong></td>
<td>NuPRL</td>
<td>Robert Constable / Cornell / 1970’s-present</td>
</tr>
<tr>
<td>(very early roots of decision</td>
<td>Boyer-Moore Theorem Prover</td>
<td>Boyer &amp; Moore / UT Austin / 1970’s-present</td>
</tr>
<tr>
<td>procedures)</td>
<td>ACL2</td>
<td>Moore, Kauffman et al. / UT Austin / 1980’s - present</td>
</tr>
<tr>
<td></td>
<td>PVS Proof Checker</td>
<td>Natarajan Shankar / SRI International / 1990’s-present</td>
</tr>
<tr>
<td><strong>SAT Solvers</strong></td>
<td>DPLL</td>
<td>Davis, Putnam, Logemann &amp; Loveland / 1962</td>
</tr>
<tr>
<td></td>
<td>Chaff &amp; zChaff</td>
<td>Zhang, Malik et al. / Princeton / 2001</td>
</tr>
<tr>
<td></td>
<td>MiniSAT</td>
<td>Een &amp; Sorenssson / 2003 - present</td>
</tr>
<tr>
<td><strong>Combinations</strong></td>
<td>Simplify</td>
<td>Nelson &amp; Oppen / DEC and Compaq / late 1980s</td>
</tr>
<tr>
<td></td>
<td>Shostak</td>
<td>Shostak / SRI International / late 1980’s</td>
</tr>
<tr>
<td></td>
<td>ICS</td>
<td>Ruess &amp; Shankar / SRI International / late 1990’s</td>
</tr>
<tr>
<td></td>
<td>SVC, CVC, CVC-Lite, CVC3 ...</td>
<td>Barrett &amp; Dill / Stanford U. / late 1990’s</td>
</tr>
<tr>
<td></td>
<td>Non-disjoint theories</td>
<td>Tinelli, Ghilardi, G.,.. / 2000 - 2008</td>
</tr>
<tr>
<td><strong>DPLL(T)</strong></td>
<td>Barcelogic and Tinelli group</td>
<td>Oliveras, Nieuwenhuis &amp; Tinelli / UPC and Iowa / 2006</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Under/Over Approximations</strong></td>
<td>UCLID</td>
<td>Seshia &amp; Bryant / CMU / 2004 - present</td>
</tr>
<tr>
<td></td>
<td>STP</td>
<td>Ganesh &amp; Dill / Stanford / 2005 - present</td>
</tr>
<tr>
<td><strong>Widely-used SMT Solvers</strong></td>
<td>Z3</td>
<td>DeMoura &amp; Bjorner / Microsoft / 2006 - present</td>
</tr>
<tr>
<td></td>
<td>CVC4</td>
<td>Barrett &amp; Tinelli / NYU and Iowa / early 2000’s - present</td>
</tr>
<tr>
<td></td>
<td>OpenSMT</td>
<td>Bruttomesso / USI Lugano / 2008 - present</td>
</tr>
<tr>
<td></td>
<td>Yices</td>
<td>Deuterre / SRI International / 2005 - present</td>
</tr>
<tr>
<td></td>
<td>MathSAT</td>
<td>Cimatti et al. / Trento / 2005 - present</td>
</tr>
<tr>
<td></td>
<td>STP</td>
<td>Ganesh / Stanford &amp; MIT / 2005 - present</td>
</tr>
<tr>
<td></td>
<td>UCLID</td>
<td>Seshia / CMU &amp; Berkeley / 2004 - present</td>
</tr>
</tbody>
</table>
THANKS!

MAPLE SERIES OF SAT SOLVERS HERE:
HTTPS://SITES.GOOGLE.COM/A/GSD.UWATERLOO.CA/MAPLESAT/