SMT: Where Do We Go From Here?

Clark Barrett, New York University

SMT Workshop, July 17, 2014

Outline

Outline

Automated Reasoning

Philosophers have long dreamed of machines that can reason. The pursuit of this dream has occupied some of the best minds and led both to great achievements and great disappointments.

~1700

Leibniz – mechanized human reasoning

1928

Hilbert

Entscheidungs-problem

1936

Church – lamda calculus

Turing – reduction halting problem

1954

Davis – decision procedure for Presburger arithmetic

Automated Reasoning

Automated Reasoning at the Turn of the Century

- Despite some successes, by the late 90's, automated reasoning was still considered impractical for most real-world applications
- Interesting problems were beyond the reach of automated methods because of decidability and complexity barriers
- The dream of *Hilbert*'s mechanized mathematics or *Leibniz*'s calculating machine appeared to be simply unattainable

Boolean Satisfiability

The SAT problem

One of the simplest problems to understand in automated reasoning is the problem of *Boolean satisfiability (SAT)*

- Given an arbitrary formula with only Boolean variables, does there exist an assignment to the variables that makes the formula true?
- Historically important: first problem shown to be NP-complete

The problem with SAT

- Unless P = NP, there is no algorithm that can efficiently solve the problem in general
- So even the simplest problems in automated reasoning appear to be beyond the reach of efficient algorithms

The Satisfiability Revolution

Princeton, c. 2000

With the new century, something remarkable happened!

- A couple of undergraduates working with Sharad Malik at Princeton built a SAT solver called *Chaff* that could solve all kinds of SAT problems, even very large ones.[a]
- People started using Chaff as a reasoning engine to solve lots of interesting problems
- SAT took off as a research area and as a tool for academic and industrial automated reasoning
- The best modern SAT solvers can routinely solve problems with millions of variables

[a] Moskewicz, Madigan, Zhao, Zhang, Malik. **Chaff: engineering an efficient SAT solver**, DAC '01.

The Satisfiability Revolution

But SAT is NP-Complete: What is Going On?

- Short answer: many problems from real-world applications do not exhibit worst-case performance
- Some observations:
  - Random 3-SAT instances are only hard if the ratio of clauses to variables is in a *narrow region* around 4.25
  - Real-world problems often have *structure*
  - Real-world problems often have *modularity*
  - As a result, real-world problems often have *short proofs*

Satisfiability Modulo Theories

Palo Alto, c. 2000

Resurgence of interest in theory-specific reasoning:

- Stanford: Stanford Validity Checker (SVC)[a, b]
- Stanford: Stanford Temporal Prover (STeP)[c, d]
- SRI: Integrated Canonizer and Solver (ICS)[e, f]

---

[a] Barrett, Dill, Levitt. **Validity Checking for Combinations of Theories with Equality**, FMCAD '96.
[b] Barrett, Dill, Stump. **A Framework for Cooperating Decision Procedures**, CADE '00.
[c] Bjørner, Stickel, Uribe. **A practical integration of first-order reasoning and decision procedures**, CADE '97.
[d] Bjørner. **Integrating Decision Procedures for temporal verification**, PhD Thesis, 1998.
[e] Cyrluk, Lincoln, Shankar. **On Shostak's Decision Procedure for Combinations of Theories**, CADE '96.
[f] Rueß, Shankar. **Deconstructing Shostak**, LICS '01.

UIUC: Revisiting Nelson-Oppen

Tinelli, Harandi. **A New Correctness Proof of the Nelson-Oppen Combination Procedure**, FroCoS'96.

Satisfiability Modulo Theories

Key idea: combine SAT with theory reasoning

- Flurry of research in this direction, with promising results
  - Genoa: TSAT[a]
  - Stanford: Cooperating Validity Checker (CVC)[b]
  - SRI: ICS[c]
  - Trento: MathSAT[d]
  - Compaq/HP SRC: Verifun[e]
- The new area was dubbed Satisfiability Modulo Theories (SMT)[f]

---

[a] Armando, Castellini, Giunchiglia. **SAT-based Procedures for Temporal Reasoning**, ECP '99.

[b] Barrett, Dill, Stump. **Checking Satisfiability of First-Order Formulas by Incremental Translation to SAT**, CAV '02.

[c] de Moura, Reuß. **Lemmas on Demand for Satisfiability Solvers**, SAT '02.

[d] Audemard, Bertoli, Cimatti, Korniłowicz, Sebastiani. **A SAT Based Approach for Solving Formulas over Boolean and Linear Mathematical Propositions**, CADE '02.

[e] Flanagan, Joshi, Ou, Saxe. **Theorem Proving using Lazy Explication**, CAV '03.

[f] Tinelli. **A DPLL-based Calculus for Ground Satisfiability Modulo Theories**, JELIA '02.

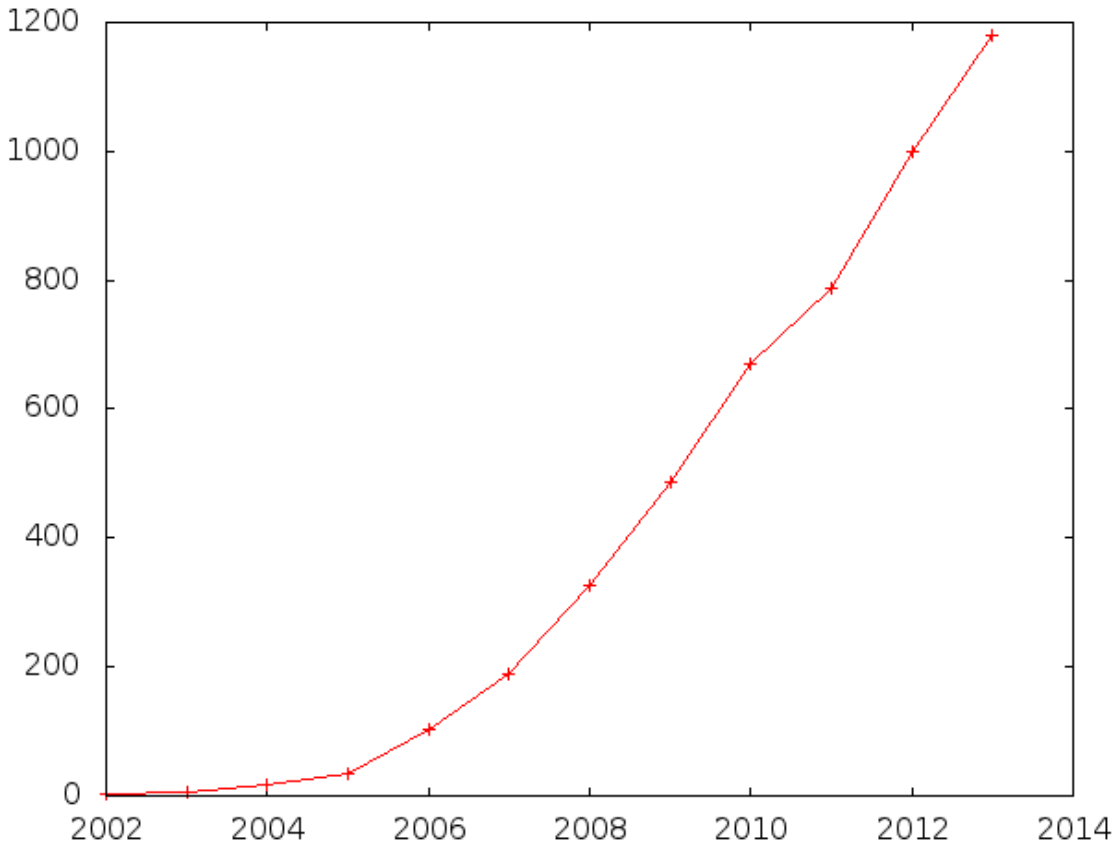
Growth of SMT

Growth of SMT Community

- First PDPAR workshop in 2003, renamed to SMT in 2007
- SMT-LIB standard adopted in 2004, v2 in 2010
- SMT-COMP established in 2005
- SMT-LIB benchmark library established in 2005 (now well over 100,000 benchmarks)
- Emergence of robust, general-purpose solvers implementing the standard: Barcelogic, CVC, MathSAT, Yices, Z3

Growth of SMT

Articles per year referencing "SMT solver" or "Satisfiability Modulo Theories" (from Google Scholar)



Impact of SMT

What people are saying

- *Most promising contribution to fields of software and hardware verification and test in the last five years*
  (from the text of the HVC 2010 award)
- *The biggest advance in formal methods in last 25 years*
  (John Rushby, FMIS 2011)
- *Most successful academic community related to logics and verification ... built in the last decade*
  (editors of FMSD special issue on SMT, 2012)

SMT Today

Growth of SMT

- Initial big breakthroughs in SAT and SMT
- Rapid progress every year since then
- Some current solvers: Boolector, CVC4, MathSAT5, OpenSMT, SMTInterpol, SONOLAR, STP, veriT, Yices2, Z3
  - High-performing, robust tools
  - Widely used for academic and industrial applications

SMT Today

- These solvers are now used as *generic reasoning engines* in a variety of applications
- The theoretical limits still exist, but we have come a long way towards realizing the vision of Leibniz and Hilbert in *practice*

SMT Solving

Guiding Principle

As with SAT, most real-world SMT applications can be solved without running into worst-case performance.

- A guiding principle in SMT
- Also observed empirically
  - Each year, more of the difficult SMT-LIB benchmarks are easily solved by some solver
  - Overall, vast majority of SMT-LIB benchmarks can be solved in reasonable time by some solver

One strength of SMT-based approaches is that they have proved to be very flexible in admitting new "attack surfaces" for tackling hard problems.

Outline

Eager Solving and Preprocessing

Idea

- Transform input problem into simpler equivalent problem
- Some example transformations:
  - Boolean propagation
  - Equality solving and substitution
  - Static learning
  - Rewriting: evaluating/propagating constants
  - Rewriting: normalizing and flattening terms
  - Rewriting: language reduction (e.g. ite removal, writing all inequalities in terms of $\leq$)
  - ...and many more
- In the limit (fully eager), problem is reduced to SAT

Eager Solving and Preprocessing

Advantages of Solving via Preprocessing

- Conceptually simple
- When it works, it's hard to beat
- A good match for real-world applications which often contain redundancy and structure that can be exploited

Challenges

- Make it easier for users to supply their own preprocessing passes
- Make it easier to explore and define different combinations of passes (strategies)
  - Z3 tactic language is a big step forward
- Fully eager approach may not scale well for some applications

Lazy Solving

Lazy Approach

- SAT solver is run on Boolean skeleton of a formula
- Assignment is checked for theory satisfiability by separate theory solver
- Many optimizations, including:
  - Early Pruning
  - Computing small conflict clauses
  - Theory propagation

Lazy Solving

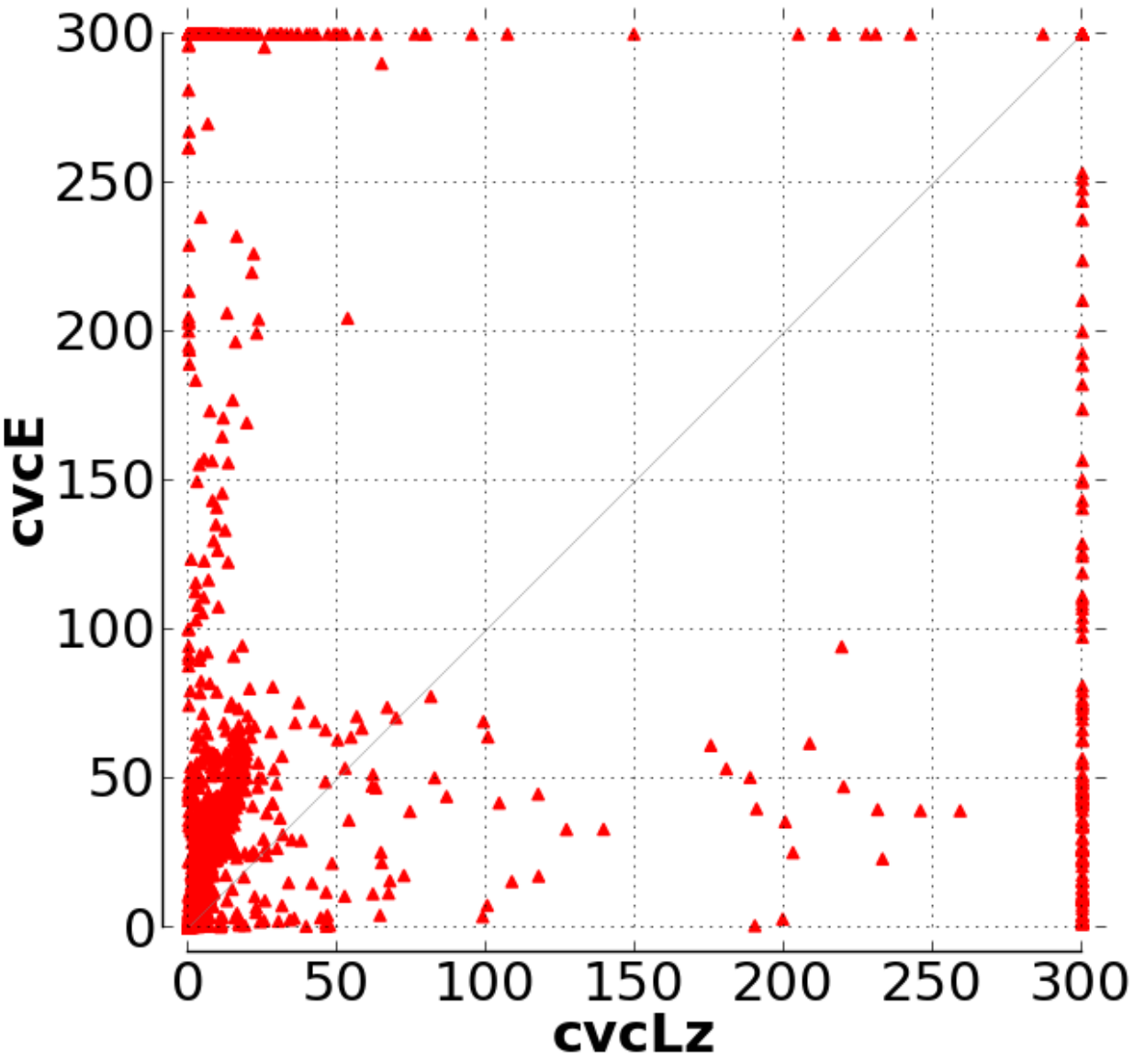
Advantages of Lazy Solving

- Don't pay cost of eager translation
- Only reason about a part of the formula at a time
- Able to handle theories that do not admit a translation to SAT
- Possible to combine theories using modular techniques

Challenges

- Requires significant effort to implement efficiently
- Communication channel between SAT and theory solver can be too narrow
- Incorrect or inefficient theory combination

Lazy vs Eager



Lazy vs Eager: Can we get the best of both?

Best of Both

- In-processing: Use preprocessing/eager techniques on each sub-problem seen by lazy solver
- Use machine learning to detect whether to run lazy or eager
- Use portfolio approach: just do both

Challenges

- More integrated hybrid approach

Outline

Perspective: DP vs DPLL

Davis-Putnam (DP)

- Unit Propagation:
  $(x_1)(x_1 \vee x_2)(\neg x_1 \vee x_3 \vee \neg x_4) \Rightarrow (x_3 \vee \neg x_4)$
- Pure Literal Rule:
  $(x_1 \vee x_2)(\neg x_2 \vee x_3)(x_2 \vee \neg x_3) \Rightarrow (\neg x_2 \vee x_3)(x_2 \vee \neg x_3)$
- Boolean Resolution:
  $(x_1 \vee x_2 \vee \neg x_3)(\neg x_1 \vee x_2 \vee x_4) \Rightarrow (x_2 \vee \neg x_3 \vee x_4)$

Perspective: DP vs DPLL

Davis-Putnam (DP)

- Unit Propagation:
  $(x_1)(x_1 \vee x_2)(\neg x_1 \vee x_3 \vee \neg x_4) \Rightarrow (x_3 \vee \neg x_4)$
- Pure Literal Rule:
  $(x_1 \vee x_2)(\neg x_2 \vee x_3)(x_2 \vee \neg x_3) \Rightarrow (\neg x_2 \vee x_3)(x_2 \vee \neg x_3)$
- Boolean Resolution:
  $(x_1 \vee x_2 \vee \neg x_3)(\neg x_1 \vee x_2 \vee x_4) \Rightarrow (x_2 \vee \neg x_3 \vee x_4)$

DP as Quantifier Elimination

- Notice that each rule eliminates a variable
- DP can be thought of as a decision procedure based on quantifier elimination

Perspective: DP vs DPLL

Boolean Resolution

$$\begin{array}{llcllll} (\neg x_1 \vee x_2) & (x_1 \vee x_5) & & (x_2 \vee x_5) & (x_3 \vee x_5) & (x_4 \vee x_5) \\ (\neg x_1 \vee x_3) & (x_1 \vee x_6) & \Rightarrow & (x_2 \vee x_6) & (x_3 \vee x_6) & (x_4 \vee x_6) \\ (\neg x_1 \vee x_4) & (x_1 \vee x_7) & & (x_2 \vee x_7) & (x_3 \vee x_7) & (x_4 \vee x_7) \end{array}$$

Perspective: DP vs DPLL

Boolean Resolution

$$\begin{array}{ll} (\neg x_1 \vee x_2) & (x_1 \vee x_5) \\ (\neg x_1 \vee x_3) & (x_1 \vee x_6) \\ (\neg x_1 \vee x_4) & (x_1 \vee x_7) \end{array} \Rightarrow \begin{array}{lll} (x_2 \vee x_5) & (x_3 \vee x_5) & (x_4 \vee x_5) \\ (x_2 \vee x_6) & (x_3 \vee x_6) & (x_4 \vee x_6) \\ (x_2 \vee x_7) & (x_3 \vee x_7) & (x_4 \vee x_7) \end{array}$$

DP does not scale: uses exponential space in the worst case

Perspective: DP vs DPLL

Davis-Logemann-Loveland (DPLL)

- Unit Propagation:
  $(x_1)(x_1 \vee x_2)(\neg x_1 \vee x_3 \vee \neg x_4) \Rightarrow (x_3 \vee x_4)$
- Pure Literal Rule:
  $(x_1 \vee x_2)(\neg x_2 \vee x_3)(x_2 \vee \neg x_3) \Rightarrow (\neg x_2 \vee x_3)(x_2 \vee \neg x_3)$
- Branching Rule:
  $(x_1 \vee x_2 \vee \neg x_3)(\neg x_1 \vee x_2 \vee x_4) \Rightarrow$
  (Assume $x1$): $(x_2 \vee x_4)$ OR (Assume $\neg x1$): $(x_2 \vee \neg x_3)$

Perspective: DP vs DPLL

Davis-Logemann-Loveland (DPLL)

- Unit Propagation:
  $(x_1)(x_1 \vee x_2)(\neg x_1 \vee x_3 \vee \neg x_4) \Rightarrow (x_3 \vee x_4)$
- Pure Literal Rule:
  $(x_1 \vee x_2)(\neg x_2 \vee x_3)(x_2 \vee \neg x_3) \Rightarrow (\neg x_2 \vee x_3)(x_2 \vee \neg x_3)$
- Branching Rule:
  $(x_1 \vee x_2 \vee \neg x_3)(\neg x_1 \vee x_2 \vee x_4) \Rightarrow$
  (Assume $x1$): $(x_2 \vee x_4)$ OR (Assume $\neg x1$): $(x_2 \vee \neg x_3)$

DPLL as Model Finding

- Notice that each rule assigns a specific value to a variable
- DPLL can be thought of as a decision procedure based on model finding

Modern SAT solvers: CDCL

Conflict-Driven-Clause-Learning (CDCL)

- *Model finding* as in DPLL
- When there is a conflict, a new conflict clause is learned
- The conflict clause enables backjumping - going back several levels of decisions at once
- The conflict clause is formed by *Boolean resolution* on the clauses contributing to the conflict

Lazy Solving with DPLL(T): Strengths and Limitations

DPLL(T) works well

- Problems requiring non-trivial Boolean reasoning (big search tree)
- Where the theory-check for satisfiability of literals is cheap (minimal work at each node of tree)

DPLL(T) struggles

- Problems with little or no Boolean reasoning (small tree)
- Theory-check for satisfiability of literals is hard (hard work at each node)

Idea: move more work into the search tree (but how?)

Example

Consider a solver for linear real arithmetic that uses Fourier-Motzkin (FM) to check conjunctions of literals

Example

For a fixed n, consider the set of equations $\pm x_i \pm x_j \pm x_k \geq 0$ with $n > i > j > k > 0$

FM generates exponentially many (in n) inequalities for this example

Example

Example

$$-x_1 + x_2 + x_3 \geq 0 \quad x_1 + x_2 + x_3 \geq 0$$
$$-x_1 + x_2 - x_3 \geq 0 \quad x_1 + x_2 - x_3 \geq 0$$
$$-x_1 - x_2 + x_3 \geq 0 \quad x_1 - x_2 + x_3 \geq 0$$
$$-x_1 - x_2 - x_3 \geq 0 \quad x_1 - x_2 - x_3 \geq 0$$

$$\Rightarrow$$

$2x_2 + 2x_3 \geq 0$	$2x_2 \geq 0$	$2x_3 \geq 0$	$0 \geq 0$
$2x_2 \geq 0$	$2x_2 - 2x_3 \geq 0$	$0 \geq 0$	$-2x_3 \geq 0$
$2x_3 \geq 0$	$0 \geq 0$	$-2x_2 + 2x_3 \geq 0$	$-2x_2 \geq 0$
$0 \geq 0$	$-2x_3 \geq 0$	$-2x_2 \geq 0$	$-2x_2 - 2x_3 \geq 0$

Recall DP vs DPLL

Quantifier Elimination and Model Finding

- FM is doing Quantifier Elimination
- Can we use model finding?

Simple Fourier-Motzkin with model finding

- Impose an order on the variables: $x_1 \ldots x_n$
- A constraint is a k-constraint if the maximum variable is x_k
- Pick var assignment v by assigning each variable in order
- The assignment to x_k must satisfy all k-constraints
- If this is not possible, there must be:
  - $x_k + p \geq 0$ and $-x_k + q \geq 0$ with $v(p) + v(q) < 0$
  - If there is such a pair, add $p + q \geq 0$ and start again[a]

[a] Korovin, Tsiskaridze, Voronkov. **Conflict Resolution**, CP '09.

Example

Example

For a fixed n, consider the set of equations $\pm x_i \pm x_j \pm x_k \geq 0$ with $n > i > j > k > 0$

FM with model finding

- No 1- or 2-constraints, so x_1 and x_2 are assigned arbitrarily
- After assigning x_1 and x_2, we will do FM on 3-constraints
- This will continue until we learn that x_1 and x_2 must be 0
- After this, the only possible assignment for each x_k will be 0

The Model Construction Calculus (MCC) [1]

Best of Both?

- Theory solvers can reason algebraically OR guess assignments
- Increases expressive power of communication channel between SAT and theory solvers
- Initial prototypes for arithmetic theories are very promising

Challenges

- Need to develop new theory solvers
- Need to be able to generalize from failed model guesses
- Extended SAT module required
- Theory combination needs to be revisited
- How to combine MCC and DPLL(T)?

[1] Jovanović, Barrett, de Moura. **The Design and Implementation of the Model Constructing Satisfiability Calculus**, FMCAD '13.

The Power of Theories

A great advantage of SMT is that it is modular with respect to theories. This allows improved theory solvers to be swapped in and new theories to be added.

Leveraging External Tools

Examples:

- GLPK for arithmetic
- abc for bit-vector reasoning
- Initial results are encouraging

Challenges

- License and build headaches
- External tools are typically not designed for SMT-like environment
  - Many quick calls
  - Need to be able to produce explanations of unsatisfiability

Developing New Theories

Recent New Theories

- Floating Point Arithmetic
- Strings
- Sets

Challenges

- Developing a new theory solver is a huge effort
- Still a large barrier to entry for non-SMT experts to develop a new theory

Developing New Theories

Questions

- What general new theories could have a big impact?
- What specific new theories could help a lot in a specific domain?

Prediction: development of new theories could be biggest future growth area for SMT

Outline

What is SMT good for?

Generic Reasoning

- Given some conditions X, is it possible for Y to happen, and if so how?
- X and Y must be expressible in logic
- SMT offers a lot of expressive power
- Possibility to define a new theory if all else fails

What SMT is NOT good for

- Reasoning in the presense of uncertainty (e.g. probabilities)
- Heavy use of quantifiers
- Difficult constraints with no Boolean structure (e.g. Linear Programs)

Some Applications of SMT

Program Analysis and Verification

- Software Model Checking[a] (e.g. BLAST, SLAM)
- K-Induction-Based Model Checking[b] (e.g. Kind)
- Concolic or Directed Automated Random Testing[c] (e.g. CUTE, KLEE, PEX)
- Program Verifiers (e.g. VCC,[d] Why3[e])
- Translation Validation for Compilers (e.g. TVOC[f])

---

[a] Jhala and Majumdar, **Software Model Checking**, ACM Computing Surveys 2009.
[b] Hagen and Tinelli, **Scaling Up the Formal Verification of Lustre Programs with SMT-Based Techniques**, FMCAD'08.
[c] Godefroid, Klarlund, and Sen, **DART: Directed Automated Random Testing**, PLDI '05
[d] Dahlweid, Moskal, Santen et al. **VCC: Contract-based modular verification of concurrent C**, ICSE '09.
[e] Bobot, Filliâtre, Marché, and Paskevich, **Why3: Shepherd Your Herd of Provers**, Boogie '11.
[f] Zuck, Pnueli, Goldberg, Barrett et al., **Translation and Run-Time Validation of Loop Transformations**, FMSD '05.

Some Applications of SMT

Non-verification Applications

- AI (e.g. Robot Task Planning[a])
- Biology (e.g. Analysis of Synthetic Biology Models[b])
- Databases (e.g. Checking Preservation of Database Integrity[c])
- Network Analysis (e.g. Checking Security of OpenFlow Rules[d])
- Scheduling (e.g. Rotating Workforce Scheduling[e])
- Security (e.g. Automatic Exploit Generation[f])
- Synthesis (e.g. Symbolic Term Exploration[g])

---

[a] Witsch, Skubch, et al., **Using Incomplete Satisfiability Modulo Theories to Determine Robotic Tasks**, IROS '13.

[b] Yordanov and Wintersteiger, **SMT-based analysis of Biological Computation**, NFM '13.

[c] Baltopoulos, Borgström, and Gordon, **Maintaining Database Integrity with Refinement Types**, ECOOP '11.

[d] Son, Shin, Yegneswaran et al., **Model Checking Invariant Security Properties in OpenFlow**, ICC '13.

[e] Erkinger, **Rotating Workforce Scheduling as Satisfiability Modulo Theories**, Master's Thesis, TU Wien, 2013.

[f] Avgerinos, Cha, Rebert et al. **Automatic Exploit Generation**, CACM '14.

[g] Kneuss, Kuraj, Kuncak, and Suter, **Synthesis Modulo Recursive Functions**, OOPSLA '13.

Rotating Workforce Scheduling

Assign Shifts to Employees under Constraints

- Need to fill each shift
- Need to assign enough hours to each employee
- Ensure no employee works too many days without a break
- Avoid illegal shift sequences (e.g. day shift immediately after night shift)

Rotating Workforce Scheduling

Example shift table for 9 employees and 3 shifts

Employee	Mon	Tue	Wed	Thu	Fri	Sat	Sun
1	-	-	-	A	A	A	A
2	A	N	N	-	-	-	-
3	A	A	A	A	A	A	-
4	-	D	D	D	D	N	N
5	N	-	-	A	A	A	A
6	-	-	-	D	D	D	D
7	N	N	N	-	-	D	D
8	D	A	A	N	N	-	-
9	D	D	D	N	N	N	N

Rotating Workforce Scheduling

SMT Encoding

- Encode shift table entries as Integer or bit-vector variables
- Constraints become assertions about table rows, table columns, and consecutive entries

SMT vs traditional techniques

A recent study showed that an SMT-based approach outperformed existing exact solution approaches, solving 38% more instances

Automatic Exploit Generation

Inputs

- Binary Program, Safety Property

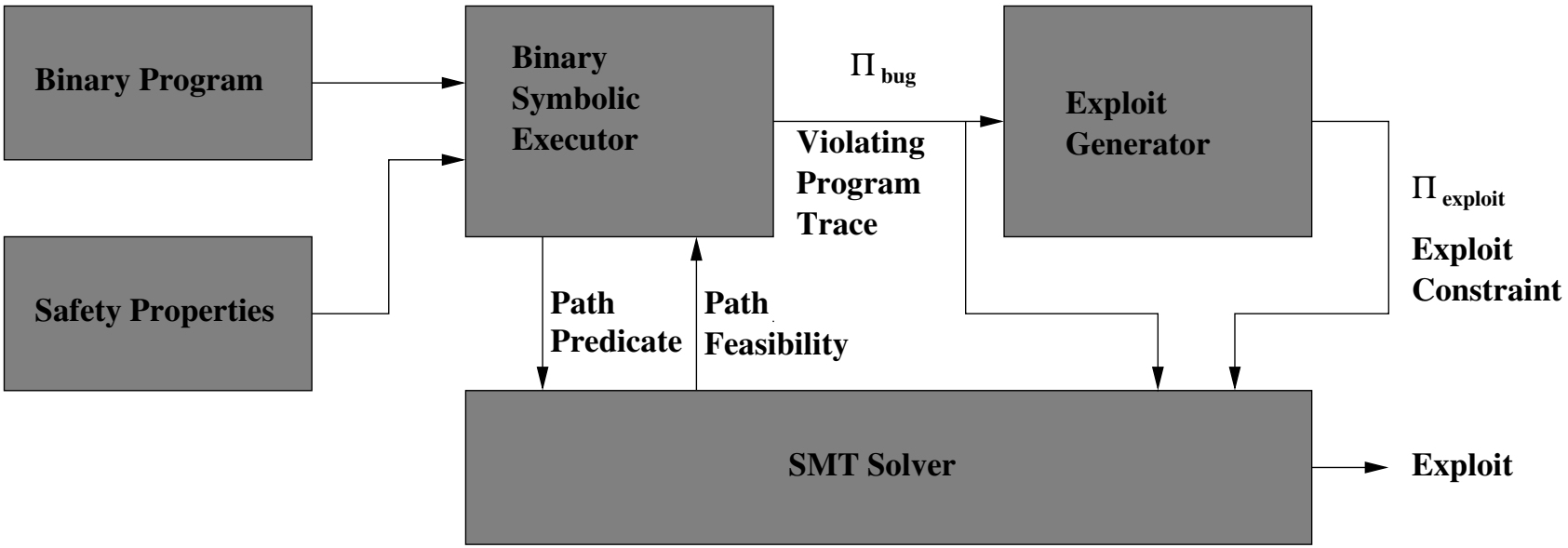
Model

- Memory, registers modeled as arrays of bit-vectors
- Instructions modeled as constraints over bit-vectors and arrays

Symbolic Execution

- Enumerate paths through binary program
- Symbolically simulate each path to generate SMT formula
- SMT solver reports bug if path is feasible but violates safety property

Automatic Exploit Generation



Where is SMT headed?

We have only scratched the surface of what is possible with SMT

Plenty of room for performance improvements

- Each year, solvers solve many problems that were previously too hard (for any solver)
- Parallel computing power still largely untapped

Lots to explore within themes mentioned here

- Exploring lazy and eager approaches
- Exploring algebraic and model-finding approaches
- Leveraging external tools and techniques
- Developing new theories

Where is SMT headed?

More empahsis on additional capabilities

- Not just Sat or Unsat
- Need for fast incremental solving via library interface
- Models, Proofs, Interpolants
- Optimization and MaxSAT
- Interoperability with other tools

New Applications

- More use outside of traditional verification arena
- Domain-specific theories for new domains
- Non-CS applications: biology, finance, etc.
- Your application here?