The Satisfiability Revolution and the Rise of SMT

Clark Barrett, New York University

ExCape Webinar, March 3, 2014
1 Motivation and History

2 The Satisfiability Revolution

3 SMT Solvers: Under the Hood
   - The DPLL(T) Framework
   - Theory Solvers
   - Combining Theories

4 Applications and Future
   - Applications of SMT
   - The Future
Outline

1 Motivation and History

2 The Satisfiability Revolution

3 SMT Solvers: Under the Hood
   - The DPLL(T) Framework
   - Theory Solvers
   - Combining Theories

4 Applications and Future
   - Applications of SMT
   - The Future
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.
Gottfried Wilhelm Leibniz (1646-1716) first proposed the ambitious goal of mechanizing the process of human reasoning:

Once this is done, then when a controversy arises, disputation will no more be needed between two philosophers than between two computers. It will suffice that, pen in hand, they sit down to their abacus and (calling in a friend if they so wish) say to each other: let us calculate.
In 1928, David Hilbert posed the Entscheidungsproblem (decision problem).

The Entscheidungsproblem is solved when we know a procedure that allows for any given logical expression to decide by finitely many operations its validity or satisfiability... The entscheidungsproblem must be considered the main problem of mathematical logic.
Automated Reasoning

In 1936, Alonzo Church and Alan Turing provided a negative answer to the Entscheidungsproblem (Church using lambda calculus and Turing with a reduction to the halting problem), a great disappointment to Hilbert and his program for mechanizing mathematics.
All was not lost, however. By restricting the set of models (e.g. to only those satisfying a specific theory), many interesting decidable fragments can be obtained.

One of these is Presburger (or linear integer) arithmetic, proved decidable by Mojżesz Presburger in 1929.

In 1954, Martin Davis implemented a decision procedure for Presburger arithmetic and used it to produce the first computer-generated mathematical proof. Its great triumph was to prove that the sum of two even numbers is even.
Automated Reasoning

Over the next few decades, automated reasoning became an established research area, with notable milestones including the following:

- **1955-56:** Newell, Shaw, and Simon’s Logic Theorist, designed to mimic human reasoning and capable of proving basic mathematical results.
- **1965:** John Robinson proves that first order resolution is sound and complete, spawning a large research effort in automated pure first-order theorem proving (ATP).
- **1974:** The first Conference on Automated Deduction (CADE) is held at Argonne National Laboratory.
- **1983:** Larry Wos founds the Association for Automated Reasoning (AAR) and the Journal of Automated Reasoning.
Automated Reasoning

Despite many successes, by the early 90’s, automated reasoning was still considered impractical for most real-world applications.

It appeared that most 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.
Outline

1 Motivation and History

2 The Satisfiability Revolution

3 SMT Solvers: Under the Hood
   - The DPLL(T) Framework
   - Theory Solvers
   - Combining Theories

4 Applications and Future
   - Applications of SMT
   - The Future
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
Princeton, 2001

In 2001, 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.
- 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.
But SAT is NP-Complete: What is Going On?

- Short answer: still trying to understand why!
- For some reason, 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
  - Real-world problems often have short proofs
From SAT to SMT

Palo Alto, 2001

About the same time, there was a resurgence of interest in theory-specific automated reasoning techniques:

- Stanford: SVC/CVC (Barrett, Dill, Levitt, Stump)
- Stanford: STeP (Bjørner, Manna)
- Compaq/HP SRC: Verifun (Joshi, Flanagan, Saxe)
- SRI: ICS/Yices (de Moura, Owre, Reuss, Shankar)

Convergence of SAT and theory reasoning

- A natural idea was to combine SAT with theory reasoning
- A flurry of early research produced very promising results
- The new area was dubbed Satisfiability Modulo Theories (SMT)
Growth of SMT

- First PDPAR workshop in 2003, renamed to SMT in 2007
  - SMT is consistently the most-subscribed workshop at every conference it’s been associated with
- SMT-LIB standard adopted in 2004, v2 in 2010
- SMT-COMP established in 2005
- SMT-LIB benchmark library now has over 100,000 benchmarks
- Some current solvers: Boolector, CVC4, MathSAT, Yices, Z3
  - Mature and robust tools
  - Dramatic improvement in performance and capabilities
  - Used in a wide variety of academic and industrial applications
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)
Automated Reasoning Today

A New Age of Automated Reasoning

- Initial big breakthroughs in SAT and SMT
- Rapid progress every year since then
- Advances in processor speeds and memory help

Result

- These solvers are now used as generic reasoning engines in a wide 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
Outline

1 Motivation and History

2 The Satisfiability Revolution

3 SMT Solvers: Under the Hood
   - The DPLL(T) Framework
   - Theory Solvers
   - Combining Theories

4 Applications and Future
   - Applications of SMT
   - The Future
Advantages of SMT over SAT

- Retains speed and automation of SAT solvers
- Supports a higher level of abstraction, can model infinite state
- Much more expressive:
  - atoms can be arbitrary theory formulas instead of just propositional variables
  - modular framework allows new theory solvers to be defined and integrated
  - theories can be used in any combination
Motivation and History
The Satisfiability Revolution
SMT Solvers: Under the Hood
Applications and Future

The DPLL(T) Framework
Theory Solvers
Combining Theories

Arithmetic
Arrays
UF
Bit-Vectors

Core

explanations
conflicts
lemmas
propagations

assertions

SAT Solver

DPLL
The DPLL(T) Framework
Theory Solvers
Combining Theories

Motivation and History
The Satisfiability Revolution
SMT Solvers: Under the Hood
Applications and Future

SAT Solver
- Only sees **Boolean skeleton** of problem
- Builds partial model by assigning truth values to literals
- Informs theories of decisions

- **SAT Solver**
  - *DPLL*

- **Core**
  - **Arithmetic** → **UF** → **Bit-Vectors** → **Arrays**

- **assertions**
  - explanations
  - conflicts
  - lemmas
  - propagation

Clark Barrett
Satisfiability Modulo Theories
Motivation and History
The Satisfiability Revolution
SMT Solvers: Under the Hood
Applications and Future

The DPLL(T) Framework
Theory Solvers
Combining Theories

Arithmetic → Arrays → Bit-Vectors → Core

UF

Core

explanations
conflicts
lemmas
propagations

assertions

SAT Solver

DPLL

Clark Barrett  Satisfiability Modulo Theories
**Core**

- Sends each assertion to the appropriate theory
- Sends deduced literals to other theories/SAT solver
- Handles theory combination using Nelson-Oppen method or variant
Motivation and History
The Satisfiability Revolution
SMT Solvers: Under the Hood
Applications and Future

The DPLL(T) Framework
Theory Solvers
Combining Theories

Clark Barrett
Satisfiability Modulo Theories
Motivation and History
The Satisfiability Revolution
SMT Solvers: Under the Hood
Applications and Future

The DPLL(T) Framework
Theory Solvers
Combining Theories

Theory Solvers
- Decide satisfiability of a conjunction of theory literals
- Incremental
- Backtrackable
- Conflict Generation
- Theory Propagation
Example:

\[ g(a) = c \quad \land \quad f(g(a)) \neq f(c) \quad \lor \quad g(a) = d \quad \land \quad c \neq d \quad \lor \quad g(a) \neq d \]

\[ \emptyset \parallel 1, 2 \lor 3, 4 \lor 3 \]
Example

$$g(a) = c \quad \land \quad f(g(a)) \neq f(c) \quad \lor \quad g(a) = d \quad \land \quad c \neq d \quad \lor \quad g(a) \neq d$$

$$\emptyset \parallel 1, \overline{2} \lor 3, \overline{4} \lor \overline{3} \quad \Rightarrow \quad \text{(UnitProp)}$$

$$1 \parallel 1, \overline{2} \lor 3, \overline{4} \lor \overline{3}$$
Example

\[ g(a) = c \land f(g(a)) \neq f(c) \lor g(a) = d \land c \neq d \lor g(a) \neq d \]

\[ \emptyset \parallel 1, 2 \lor 3, 4 \lor 3 \implies \text{(UnitProp)} \]

\[ 1 \parallel 1, 2 \lor 3, 4 \lor 3 \implies \text{(Theory Propagate)} \]

\[ 1 2 \parallel 1, 2 \lor 3, 4 \lor 3 \]
Example

\[
\begin{align*}
g(a) &= c & \land & & f(g(a)) \neq f(c) \lor & & g(a) = d & \land & & c \neq d \lor & & g(a) \neq d \\
\hline
1 & & 1, \overline{2} \lor 3, \overline{4} \lor \overline{3} & \implies & (\text{UnitProp}) \\
1 & & 1, \overline{2} \lor 3, \overline{4} \lor \overline{3} & \implies & (\text{Theory Propagate}) \\
1 2 & & 1, \overline{2} \lor 3, \overline{4} \lor \overline{3} & \implies & (\text{UnitProp}) \\
1 2 3 & & 1, \overline{2} \lor 3, \overline{4} \lor \overline{3} & & \\
\end{align*}
\]
Example

\[ g(a) = c \quad \land \quad f(g(a)) \neq f(c) \quad \lor \quad g(a) = d \quad \land \quad c \neq d \quad \lor \quad g(a) \neq d \]

\[
\emptyset \quad \parallel \quad 1, \overline{2} \lor 3, \overline{4} \lor \overline{3} \quad \implies \quad \text{(UnitProp)}
\]

\[
1 \quad \parallel \quad 1, \overline{2} \lor 3, \overline{4} \lor \overline{3} \quad \implies \quad \text{(Theory Propagate)}
\]

\[
1 \quad \parallel \quad 1, \overline{2} \lor 3, \overline{4} \lor \overline{3} \quad \implies \quad \text{(UnitProp)}
\]

\[
1 \quad \parallel \quad 1, \overline{2} \lor 3, \overline{4} \lor \overline{3} \quad \implies \quad \text{(Theory Propagate)}
\]
Example

\[
\begin{align*}
g(a) &= c \\ f(g(a)) &\neq f(c) \\ g(a) &= d \\ c &\neq d
\end{align*}
\]

\[
\begin{align*}
\emptyset &\parallel 1, \overline{2} \lor 3, \overline{4} \lor \overline{3} &\implies &\text{(UnitProp)} \\
1 &\parallel 1, \overline{2} \lor 3, \overline{4} \lor \overline{3} &\implies &\text{(Theory Propagate)} \\
1 \ 2 &\parallel 1, \overline{2} \lor 3, \overline{4} \lor \overline{3} &\implies &\text{(UnitProp)} \\
1 \ 2 \ 3 &\parallel 1, \overline{2} \lor 3, \overline{4} \lor \overline{3} &\implies &\text{(Theory Propagate)} \\
1 \ 2 \ 3 \ 4 &\parallel 1, \overline{2} \lor 3, \overline{4} \lor \overline{3} &\implies &\text{(Fail)}
\end{align*}
\]
Given a theory $T$, a Theory Solver for $T$ takes as input a set $\Phi$ of literals and determines whether $\Phi$ is $T$-satisfiable.

$\Phi$ is $T$-satisfiable iff there is some model $M$ of $T$ such that each formula in $\Phi$ holds in $M$.

We consider a simple example: difference logic.
In *difference logic*, we are interested in the satisfiability of a conjunction of arithmetic atoms.

Each atom is of the form $x - y \bowtie c$, where $x$ and $y$ are variables, $c$ is a numeric constant, and $\bowtie \in \{=, <, \leq, >, \geq\}$.

The variables can range over either the *integers* (*QF_IDL*) or the *reals* (*QF_RDL*).
Difference Logic

The first step is to rewrite everything in terms of $\leq$: 
The first step is to rewrite everything in terms of $\leq$:

- $x - y = c \implies x - y \leq c \land x - y \geq c$
Difference Logic

The first step is to rewrite everything in terms of $\leq$:

- $x - y = c \implies x - y \leq c \land x - y \geq c$
- $x - y \geq c \implies y - x \leq -c$
Difference Logic

The first step is to rewrite everything in terms of $\leq$:

- $x - y = c \implies x - y \leq c \land x - y \geq c$
- $x - y \geq c \implies y - x \leq -c$
- $x - y > c \implies y - x < -c$
The first step is to rewrite everything in terms of $\leq$:

- $x - y = c \iff x - y \leq c \land x - y \geq c$
- $x - y \geq c \iff y - x \leq -c$
- $x - y > c \iff y - x < -c$
- $x - y < c \iff x - y \leq c - 1$ (integers)
The first step is to rewrite everything in terms of $\leq$:

- $x - y = c \implies x - y \leq c \land x - y \geq c$
- $x - y \geq c \implies y - x \leq -c$
- $x - y > c \implies y - x < -c$
- $x - y < c \implies x - y \leq c - 1$ (integers)
- $x - y < c \implies x - y \leq c - \delta$ (reals)
Now we have a conjunction of literals, all of the form $x - y \leq c$.

From these literals, we form a weighted directed graph with a vertex for each variable.

For each literal $x - y \leq c$, there is an edge $\xrightarrow{c} y$.

The set of literals is satisfiable iff there is no cycle for which the sum of the weights on the edges is negative.

There are a number of efficient algorithms for detecting negative cycles in graphs.
Difference Logic Example

\[ x - y = 5 \land z - y \geq 2 \land z - x > 2 \land w - x = 2 \land z - w < 0 \]
Difference Logic Example

\[
x - y = 5 \land z - y \geq 2 \land z - x > 2 \land w - x = 2 \land z - w < 0
\]

\[
x - y = 5 \\
z - y \geq 2 \\
z - x > 2 \\
w - x = 2 \\
z - w < 0
\]
Difference Logic Example

\[ x - y = 5 \land z - y \geq 2 \land z - x > 2 \land w - x = 2 \land z - w < 0 \]

\[ x - y = 5 \\
z - y \geq 2 \\
z - x > 2 \implies \\
w - x = 2 \\
z - w < 0 \]
Difference Logic Example

\[ x - y = 5 \land z - y \geq 2 \land z - x > 2 \land w - x = 2 \land z - w < 0 \]

\[ x - y = 5 \quad x - y \leq 5 \land y - x \leq -5 \]
\[ z - y \geq 2 \quad y - z \leq -2 \]
\[ z - x > 2 \quad x - z \leq -3 \]
\[ w - x = 2 \quad w - x \leq 2 \land x - w \leq -2 \]
\[ z - w < 0 \quad z - w \leq -1 \]
Difference Logic Example

\[
\begin{align*}
  x &\rightarrow y \\
  y &\rightarrow z \\
  z &\rightarrow w \\
  w &\rightarrow x
\end{align*}
\]
Combining Theory Solvers

Theory solvers become much more useful if they can be used together.

\[
\begin{align*}
\text{mux\_sel} = 0 & \rightarrow \text{mux\_out} = \text{select}(\text{regfile}, \text{addr}) \\
\text{mux\_sel} = 1 & \rightarrow \text{mux\_out} = \text{ALU}(\text{alu0}, \text{alu1})
\end{align*}
\]

For such formulas, we are interested in satisfiability with respect to a *combination* of theories.

Fortunately, there exist methods for combining theory solvers. The standard technique for this is the Nelson-Oppen method.
The Nelson-Oppen method is applicable when:

1. The theories have *no shared symbols* (other than equality).
2. The theories are *stably-infinite*.
   
   *A theory $T$ is stably-infinite if every $T$-satisfiable quantifier-free formula is satisfiable in an infinite model.*

3. The formulas to be tested for satisfiability are *quantifier-free*

Many theories fit these criteria, and extensions exist in some cases when they do not.
The Nelson-Oppen Method

Suppose that $T_1$ and $T_2$ are theories and that $Sat_1$ is a theory solver for $T_1$-satisfiability and $Sat_2$ for $T_1$-satisfiability.

We wish to determine if $\phi$ is $T_1 \cup T_2$-satisfiable.

1. Convert $\phi$ to its separate form $\phi_1 \land \phi_2$.
2. Let $S$ be the set of variables shared between $\phi_1$ and $\phi_2$.
3. For each arrangement $\Delta$ of $S$:
   1. Run $Sat_1$ on $\phi_1 \cup \Delta$.
   2. Run $Sat_2$ on $\phi_2 \cup \Delta$. 
The Nelson-Oppen Method

If there exists an arrangement such that both $Sat_1$ and $Sat_2$ succeed, then $\phi$ is $T_1 \cup T_2$-satisfiable.

If no such arrangement exists, then $\phi$ is $T_1 \cup T_2$-unsatisfiable.
Consider the following \texttt{QF\_UFLIA} formula:

\[ \phi = 1 \leq x \land x \leq 2 \land f(x) \neq f(1) \land f(x) \neq f(2). \]
Example

Consider the following $QF_{UFLIA}$ formula:

$$\phi = 1 \leq x \land x \leq 2 \land f(x) \neq f(1) \land f(x) \neq f(2).$$

We first convert $\phi$ to a separate form:

$$\phi_{UF} = f(x) \neq f(y) \land f(x) \neq f(z)$$
$$\phi_{LIA} = 1 \leq x \land x \leq 2 \land y = 1 \land z = 2$$

The shared variables are $\{x, y, z\}$. There are 5 possible arrangements based on equivalence classes of $x$, $y$, and $z$. 
Example

\[ \phi_{UF} = f(x) \neq f(y) \land f(x) \neq f(z) \]
\[ \phi_{LIA} = 1 \leq x \land x \leq 2 \land y = 1 \land z = 2 \]

1. \{x = y, x = z, y = z\}
2. \{x = y, x \neq z, y \neq z\}
3. \{x \neq y, x = z, y \neq z\}
4. \{x \neq y, x \neq z, y = z\}
5. \{x \neq y, x \neq z, y \neq z\}
Example

\[
\begin{align*}
\phi_{UF} &= f(x) \neq f(y) \land f(x) \neq f(z) \\
\phi_{LIA} &= 1 \leq x \land x \leq 2 \land y = 1 \land z = 2
\end{align*}
\]

1. \{x = y, x = z, y = z\}: inconsistent with \(\phi_{UF}\).
2. \{x = y, x \neq z, y \neq z\}
3. \{x \neq y, x = z, y \neq z\}
4. \{x \neq y, x \neq z, y = z\}
5. \{x \neq y, x \neq z, y \neq z\}
Motivation and History
The Satisfiability Revolution
SMT Solvers: Under the Hood
Applications and Future

SMT Solvers: Under the Hood

Theory Solvers
Combining Theories

Example

\( \phi_{UF} = f(x) \neq f(y) \land f(x) \neq f(z) \)

\( \phi_{LIA} = 1 \leq x \land x \leq 2 \land y = 1 \land z = 2 \)

1. \{x = y, x = z, y = z\}: inconsistent with \( \phi_{UF} \).
2. \{x = y, x \neq z, y \neq z\}: inconsistent with \( \phi_{UF} \).
3. \{x \neq y, x = z, y \neq z\}
4. \{x \neq y, x \neq z, y = z\}
5. \{x \neq y, x \neq z, y \neq z\}
Example

\[ \phi_{UF} = f(x) \neq f(y) \land f(x) \neq f(z) \]
\[ \phi_{LIA} = 1 \leq x \land x \leq 2 \land y = 1 \land z = 2 \]

1. \{x = y, x = z, y = z\}: inconsistent with \( \phi_{UF} \).
2. \{x = y, x \neq z, y \neq z\}: inconsistent with \( \phi_{UF} \).
3. \{x \neq y, x = z, y \neq z\}: inconsistent with \( \phi_{UF} \).
4. \{x \neq y, x \neq z, y = z\}
5. \{x \neq y, x \neq z, y \neq z\}
Example

\[ \phi_{UF} = f(x) \neq f(y) \land f(x) \neq f(z) \]
\[ \phi_{LIA} = 1 \leq x \land x \leq 2 \land y = 1 \land z = 2 \]

1. \( \{x = y, x = z, y = z\} \): inconsistent with \( \phi_{UF} \).
2. \( \{x = y, x \neq z, y \neq z\} \): inconsistent with \( \phi_{UF} \).
3. \( \{x \neq y, x = z, y \neq z\} \): inconsistent with \( \phi_{UF} \).
4. \( \{x \neq y, x \neq z, y = z\} \): inconsistent with \( \phi_{LIA} \).
5. \( \{x \neq y, x \neq z, y \neq z\} \)
Example

\[ \phi_{UF} = f(x) \neq f(y) \land f(x) \neq f(z) \]
\[ \phi_{LIA} = 1 \leq x \land x \leq 2 \land y = 1 \land z = 2 \]

1. \{x = y, x = z, y = z\}: inconsistent with \( \phi_{UF} \).
2. \{x = y, x \neq z, y \neq z\}: inconsistent with \( \phi_{UF} \).
3. \{x \neq y, x = z, y \neq z\}: inconsistent with \( \phi_{UF} \).
4. \{x \neq y, x \neq z, y = z\}: inconsistent with \( \phi_{LIA} \).
5. \{x \neq y, x \neq z, y \neq z\}: inconsistent with \( \phi_{LIA} \).
Example

\[ \phi_{UF} = f(x) \neq f(y) \land f(x) \neq f(z) \]
\[ \phi_{LIA} = 1 \leq x \land x \leq 2 \land y = 1 \land z = 2 \]

1. \{x = y, x = z, y = z\}: inconsistent with \( \phi_{UF} \).
2. \{x = y, x \neq z, y \neq z\}: inconsistent with \( \phi_{UF} \).
3. \{x \neq y, x = z, y \neq z\}: inconsistent with \( \phi_{UF} \).
4. \{x \neq y, x \neq z, y = z\}: inconsistent with \( \phi_{LIA} \).
5. \{x \neq y, x \neq z, y \neq z\}: inconsistent with \( \phi_{LIA} \).

Therefore, \( \phi \) is unsatisfiable.
Outline

1. Motivation and History
2. The Satisfiability Revolution
3. SMT Solvers: Under the Hood
   - The DPLL(T) Framework
   - Theory Solvers
   - Combining Theories
4. Applications and Future
   - Applications of SMT
   - The Future
Applications of SMT

Program Analysis and Verification
- Predicate Abstraction-Based Software Model Checking (e.g. BLAST, SLAM)
- Concolic or Directed Automated Random Testing (e.g. CUTE, KLEE, PEX)
- Program Verifiers (e.g. VCC, Spec#, Why3)

Other Applications
- Security (e.g. Automatic Exploit Generation)
- Scheduling (e.g. Rotating Workforce Scheduling)
- Synthesis (e.g. Symbolic Term Exploration)
### 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


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**

<table>
<thead>
<tr>
<th>Employee</th>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
<th>Sat</th>
<th>Sun</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>N</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>5</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>7</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>-</td>
<td>-</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>8</td>
<td>D</td>
<td>A</td>
<td>A</td>
<td>N</td>
<td>N</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
Rotating Workforce Scheduling

SMT Encoding
- Encode shift table as a two-dimensional array
- Constraints become assertions about array rows, array 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

Christoph Erkinger, “Rotating Workforce Scheduling as Satisfiability Modulo Theories,” Technical University of Vienna, 2013.
Symbolic Term Exploration

Following the approach of Kneuss et al., consider a synthesis problem defined as a predicate $\phi(\overline{a}, \overline{x})$ where $\overline{a}$ are inputs and $\overline{x}$ are outputs.

Suppose we wish to synthesize a function to delete an element $e$ from a list $a$. We can represent this as the predicate $\alpha(x) = \alpha(a) - \{e\}$, where $\alpha$ is a function that computes the set of all elements of a list.

Symbolic Term Exploration

Using inference rules, we can reduce the problem to two interesting subproblems, with predicates as follows:

\[ \phi_1 \equiv (\alpha(r) = \alpha(t) - \{e\} \land e = h) \rightarrow \alpha(x) = \alpha(Cons(h, t)) - \{e\} \]

\[ \phi_2 \equiv (\alpha(r) = \alpha(t) - \{e\} \land e \neq h) \rightarrow \alpha(x) = \alpha(Cons(h, t)) - \{e\} \]
Symbolic Term Exploration

To synthesize a function for these predicates, *symbolic term exploration* can be used. We use two generator functions:

\[
\text{genList} = \begin{cases} 
\text{Nil} & \text{if } (\ast) \\
\text{if } (\ast) r & \\
\text{if } (\ast) t & \\
\text{else Cons}(\text{genInt}(), \text{genList}())
\end{cases}
\]

\[
\text{genInt} = \begin{cases} 
0 & \text{if } (\ast) \\
\text{if } (\ast) e & \\
\text{if } (\ast) h & \\
1 + \text{genInt}() & \text{else}
\end{cases}
\]
Symbolic Term Exploration

We can encode all possible terms generated by genList and genInt up to some depth:

\[
\begin{align*}
\psi & \equiv
\end{align*}
\]

\[
\begin{align*}
x & = \textit{ite}(b_1, \textit{nil}, \textit{ite}(b_2, r, \textit{ite}(b_3, t, \text{Cons}(c_1, c_2)))) \land \\
c_1 & = \textit{ite}(b_4, 0, \textit{ite}(b_5, e, \textit{ite}(b_6, h, 1 + c_3))) \land \\
c_2 & = \textit{ite}(b_7, \textit{nil}, \textit{ite}(b_8, r, \textit{ite}(b_9, t, \text{Cons}(c_4, c_5)))) \land \\
& b_6 \land b_9
\end{align*}
\]

Now, we can use an SMT solver to solve the formula \( \phi_1 \land \psi \). The solution comes back as \( \neg b_1 \land b_2 \land x = r \).
Symbolic Term Exploration

Similarly, we can solve $\phi_2 \land \psi$ to get:

$$\neg b_1 \land \neg b_2 \land \neg b_3 \land \neg b_4 \land \neg b_5 \land b_6 \land \neg b_7 \land b_8 \land x = Cons(h, r)$$

The final synthesized program looks like this:

```haskell
def rec(a : List) : List = a match {
  case Nil -> Nil
  case Cons(h, t) ->
    val r = rec(t)
    if (e == h) r
  else Cons(h, r)
}
```
Where is SMT headed?

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

Plenty of room for performance improvements

- Example: with recent breakthroughs in arithmetic and bit-vector algorithms, CVC4 can solve many problems that were previously too hard (for any solver)
- Parallel computing power still largely untapped

New Theories Being Developed

- Pointer logic
- IEEE floating point
- Strings
- Non-linear real arithmetic
Where is SMT headed?

More capabilities

- Not just \textit{Sat} or \textit{Unsat}
- 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.
- \textit{Your application here?}
More information:

**SMT resources**
- SMT Survey Articles: available at http://cs.nyu.edu/~barrett/pubs/bytopic.html#Book_Chapters
- SMT-LIB standards and library http://smtlib.org
- SMT Competition http://smtcomp.org
- SMT Workshop http://smt-workshop.org

**CVC4**
- Visit our website: http://cvc4.cs.nyu.edu
- Contact a CVC4 team member
- We welcome questions, feedback, collaboration proposals