Verification Using SAT and SMT Solvers

N. Shankar

Computer Science Laboratory
SRI International
Menlo Park, CA

Aug 15, 2010

1This research was supported NSF Grants CSR-EHCS(CPS)-0834810 and CNS-0917375.
Course Outline

- Boolean satisfiability (SAT)
- Satisfiability Modulo Theories (SMT)
- Verification Techniques
  1. Extended Type Checking
  2. Verification conditions
  3. Test Generation
  4. Model Checking
  5. Induction
  6. Abstraction
Logic studies the *trinity* between *language*, *interpretation*, and *proof*.

*Language* circumscribes the syntax that is used to construct sensible assertions.

*Interpretation* ascribes an intended sense to these assertions by *fixing* the meaning of certain symbols, e.g., the logical connectives, equality, and *delimiting the variation* in the meanings of other symbols, e.g., variables, functions, and predicates.

An assertion is *valid* if it holds in all interpretations.

Checking validity through interpretations is typically *impossible*, so *proofs* built from axioms and inference rules are used to effectively demonstrate the validity of assertions.
Signature $\Sigma[X]$ contains functions and predicate symbols with associated arities, and $X$ is a set of variables.

The signature can be used to construct

- **Terms** $\tau := x | f(\tau_1, \ldots, \tau_n)$
- **Atoms** $\alpha := p(\tau_1, \ldots, \tau_n)$
- **Literals** $\lambda := \alpha | \neg\alpha$
- **Constraints** $\lambda_1 \land \ldots \land \lambda_n$,
- **Clauses** $\lambda_1 \lor \ldots \lor \lambda_n$,
- **Formulas** $\psi := p(\tau_1, \ldots, \tau_n) | \tau_0 = \tau_1 | \neg\psi_0 | \psi_0 \lor \psi_1 | \psi_0 \land \psi_1 | (\exists x : \psi_0) | (\forall x : \psi_0)$
A $\Sigma$-structure $M$ consists of

- A domain $|M|$
- A map $M(f)$ from $|M|^n \rightarrow M$ for each $n$-ary function $f \in \Sigma$
- A map $M(p)$ from $|M|^n \rightarrow \{\top, \bot\}$ for each $n$-ary predicate $p$.

$\Sigma[X]$-structure $M$ also maps variables in $X$ to domain elements in $|M|$.

- The interpretation of terms and formulas in $M$ is standard.
- With this, we have $M \models \psi$ when $M$ satisfies formula $\psi$.
- A theory $\tau$ has a signature $\Sigma_\tau$ and a class of models $M_\tau$. 
Exercises

1. Formalize the statement that a total binary relation over 3 elements must contain cycles.

2. Formalize the 4-pigeonhole principle asserting that if there are 5 pigeons that are each assigned to one of 4 holes, then some hole has two pigeons.

3. Formalize the statement that a transitive graph over 3 elements contains an isolated point.

4. Formalize and prove the statement that given a symmetric and transitive graph over 3 elements, either the graph is complete or contains an isolated point.

5. Formalize *Sudoku* in propositional logic.
More Exercises

1. Show that every \( n \)-ary function from \( \{\top, \bot\}^n \) to \( \{\top, \bot\} \) is expressible using \( \neg \) and \( \lor \).

2. State and prove as many laws as you can find about negation, disjunction, conjunction, and implication.

3. Show that any \( n \)-ary Boolean function can be represented by formulas using \( \neg \) and \( \lor \).

4. State and verify an algorithm to test a Boolean formula for satisfiability and return a satisfying truth assignment when possible.
Two formulas $A$ and $B$ are equivalent, $A \iff B$, if their truth values agree in each interpretation.

Prove that the following are equivalent (TFAE):

1. $\neg\neg A \iff A$
2. $(A \implies B) \iff (\neg A \lor B)$
3. $\neg(A \land B) \iff (\neg A \lor \neg B)$
4. $\neg(A \lor B) \iff (\neg A \land \neg B)$
5. $(\neg A \implies B) \iff (\neg B \implies A)$
A formula where negation is applied only to propositional atoms is said to be in negation normal form (NNF).

A literal is either a propositional atom or its negation.

A formula that is a multiary conjunction of multiary disjunctions of literals is in conjunctive normal form (CNF).

A formula that is a multiary disjunction of multiary conjunctions of literals is in disjunctive normal form (DNF).

Show that every propositional formula is equivalent to one in NNF, CNF, and DNF.
An inference system $\mathcal{I}$ for a $\Sigma$-theory $\mathcal{T}$ is a $\Sigma[X]$-inference structure $\langle \Psi, \Lambda, \vdash \rangle$ that is

1. **Conservative:** Whenever $\varphi \vdash_{\mathcal{I}} \varphi'$, $\Lambda(\varphi)$ and $\Lambda(\varphi')$ are $\mathcal{T}$-equisatisfiable.
2. **Progressive:** The reduction relation $\vdash_{\mathcal{I}}$ should be well-founded, i.e., infinite sequences of the form $\langle \varphi_0 \vdash \varphi_1 \vdash \varphi_2 \vdash \ldots \rangle$ must not exist.
3. **Canonizing:** A state is irreducible only if it is either $\bot$ or is $\mathcal{T}$-satisfiable.

For any class of $\Sigma[X]$-formulas $\Psi$, if there is a mapping $\nu$ from $\Psi$ to $\Phi$ such that $\Lambda(\nu(A)) \iff A$, then a $\mathcal{T}$-inference system is a sound and complete inference procedure for $\mathcal{T}$-satisfiability in $\Psi$.

A computable function $f$ such that $\kappa \vdash f(\kappa)$ whenever there is a $\kappa'$ such that $\kappa \vdash \kappa'$, is a decision procedure for satisfiability.
Ordered Resolution

- Input $K$ is a set of clauses.
- Atoms are ordered by $\succ$ which is lifted to literals so that $\neg p \succ p \succ \neg q \succ q$, if $p \succ q$.
- Literals appear in clauses in decreasing order without duplication.
- Tautologies, clauses containing both $l$ and $\bar{l}$, are deleted from initial input.

<table>
<thead>
<tr>
<th>Res</th>
<th>$K, l \lor \Gamma_1, \bar{l} \lor \Gamma_2$</th>
<th>$\Gamma_1 \lor \Gamma_2 \notin K$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$K, l \lor \Gamma_1, \bar{l} \lor \Gamma_2, \bar{\Gamma}_1 \lor \bar{\Gamma}_2$</td>
<td>$\Gamma_1 \lor \Gamma_2$ is not tautological</td>
</tr>
</tbody>
</table>

| Contrad | $K, l, \bar{l}$ | $\perp$ |
Ordered Resolution: Example

\[
(K_0 =) \quad \neg p \lor \neg q \lor r, \quad \neg p \lor q, \quad p \lor r, \quad \neg r
\]

\[
(K_1 =) \quad \neg q \lor r, \quad K_0
\]

\[
(K_2 =) \quad q \lor r, \quad K_1
\]

\[
(K_3 =) \quad r, \quad K_2
\]

\[
\bot \quad \text{Contrad}
\]
Correctness

- **Progress**: Bounded number of clauses in the given literals. Each application of Res generates a new clause.

- **Conservation**: For any model $M$, if $M \models l \lor \Gamma_1$ and $M \models \overline{l} \lor \Gamma_2$, then $M \models \Gamma_1 \lor \Gamma_2$.

- **Canonicity**: Given an irreducible non-$\bot$ configuration $K$ in the atoms $p_1, \ldots, p_n$ with $p_i \prec p_{i+1}$ for $1 \leq i \leq n$, build a series of partial interpretations $M_i$ as follows:
  1. Let $M_0 = \emptyset$
  2. If $p_{i+1}$ is the maximal literal in a clause $p_{i+1} \lor \Gamma \in K$ and $M_i \not\models \Gamma$, then let $M_{i+1} = M_i \{p_{i+1} \mapsto T\}$.
  3. Otherwise, let $M_{i+1} = M_i \{p_{i+1} \mapsto \bot\}$.

- Each $M_i$ satisfies all the clauses in $K$ in the atoms $p_1, \ldots, p_i$. 
CDCL Informally

- **Goal**: Does a given set of clauses $K$ have a satisfying assignment?
- If $M$ is a total assignment such that $M \models \Gamma$ for each $\Gamma \in K$, then $M \models K$.
- If $M$ is a partial assignment at level $h$, then *propagation* extends $M$ at level $h$ with the *implied literals* $l$ such that $l \lor \Gamma \in K \cup C$ and $M \models \lnot \Gamma$.
- If $M$ detects a conflict, i.e., a clause $\Gamma \in K \cup C$ such that $M \models \lnot \Gamma$, then the conflict is *analyzed* to construct a conflict clause that allows the search to be continued from a prior level.
- If $M$ cannot be extended at level $h$ and no conflict is detected, then an unassigned literal $l$ is *selected* and assigned at level $h + 1$ where the search is continued.
<table>
<thead>
<tr>
<th>Name</th>
<th>Rule</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Propagate</td>
<td>( h, \langle M \rangle, K, C ) [1]</td>
<td>( \Gamma \equiv I \lor \Gamma' \in K \cup C ) ( M \models \neg \Gamma' )</td>
</tr>
<tr>
<td>Select</td>
<td>( h, \langle M \rangle, K, C ) [2]</td>
<td>( M \not\models I ) ( M \not\models \neg I )</td>
</tr>
<tr>
<td>Conflict</td>
<td>( 0, \langle M \rangle, K, C ) [3]</td>
<td>( M \models \neg \Gamma ) for some ( \Gamma \in K \cup C )</td>
</tr>
<tr>
<td>Backjump</td>
<td>( h + 1, \langle M \rangle, K, C ) [4]</td>
<td>( M \models \neg \Gamma ) for some ( \Gamma \in K \cup C ) ( \langle h', \Gamma' \rangle = \text{analyze}(\psi)(\Gamma) ) for ( \psi = h, \langle M \rangle, K, C )</td>
</tr>
</tbody>
</table>
Let $K$ be
\[
\{p \lor q, \neg p \lor q, p \lor \neg q, s \lor \neg p \lor q, \neg s \lor p \lor \neg q, \neg p \lor r, \neg q \lor \neg r\}.
\]

<table>
<thead>
<tr>
<th>step</th>
<th>$h$</th>
<th>$M$</th>
<th>$K$</th>
<th>$C$</th>
<th>$\Gamma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>select $s$</td>
<td>1</td>
<td>$; s$</td>
<td>$K$</td>
<td>$\emptyset$</td>
<td>_</td>
</tr>
<tr>
<td>select $r$</td>
<td>2</td>
<td>$; s; r$</td>
<td>$K$</td>
<td>$\emptyset$</td>
<td>_</td>
</tr>
<tr>
<td>propagate</td>
<td>2</td>
<td>$; s; r, \neg q[\neg q \lor \neg r]$</td>
<td>$K$</td>
<td>$\emptyset$</td>
<td>_</td>
</tr>
<tr>
<td>propagate</td>
<td>2</td>
<td>$; s; r, \neg q, p[p \lor q]$</td>
<td>$K$</td>
<td>$\emptyset$</td>
<td>_</td>
</tr>
<tr>
<td>conflict</td>
<td>2</td>
<td>$; s; r, \neg q, p$</td>
<td>$K$</td>
<td>$\emptyset$</td>
<td>$\neg p \lor q$</td>
</tr>
<tr>
<td>step</td>
<td>h</td>
<td>M</td>
<td>K</td>
<td>C</td>
<td>Γ</td>
</tr>
<tr>
<td>----------</td>
<td>---</td>
<td>----------------</td>
<td>---</td>
<td>---</td>
<td>-----------------</td>
</tr>
<tr>
<td>conflict</td>
<td>2</td>
<td>; s; r, ¬q, p</td>
<td>K</td>
<td>∅</td>
<td>¬p ∨ q</td>
</tr>
<tr>
<td>backjump</td>
<td>0</td>
<td>∅</td>
<td>K</td>
<td>q</td>
<td></td>
</tr>
<tr>
<td>propagate</td>
<td>0</td>
<td>q[q]</td>
<td>K</td>
<td>q</td>
<td></td>
</tr>
<tr>
<td>propagate</td>
<td>0</td>
<td>q, p[p ∨ ¬q]</td>
<td>K</td>
<td>q</td>
<td></td>
</tr>
<tr>
<td>propagate</td>
<td>0</td>
<td>q, p, r[¬p ∨ r]</td>
<td>K</td>
<td>q</td>
<td></td>
</tr>
<tr>
<td>conflict</td>
<td>0</td>
<td>q, p, r</td>
<td>K</td>
<td>q</td>
<td>¬q ∨ ¬r</td>
</tr>
</tbody>
</table>
CDCL Correctness

- **Progress:** Each backjump step adds a new assignment at the level \( h' \) so that \( \sum_{i=0}^{h} |M_i| \times (N + 1)^{(N-h)} \) increases toward the bound \( (N + 1)^{(N+1)} \) for \( N = |\text{vars}(K)| \). In the example, \( N = 4 \), the backjump step goes from a value 1300 in base 5 to the value 100000 which is closer to the bound 40000.

- **Conservation:** In each transition from \( \langle M, K, C \rangle \) to \( \langle M', K', C' \rangle \) (or \( \perp \)), the clause sets \( M_0 \cup K \cup C \) and \( M_0 \cup K' \cup C' \) are equisatisfiable.

- **Canonicity:** In an irreducible non-\( \perp \) state, \( M \) is total assignment and there is no conflict so for each clause \( \Gamma \) in \( K \cup C \), \( M \models \Gamma \).
The input clauses can be preprocessed by resolution, e.g., to eliminate a variable, and subsumption to discard a clause when a subclause is already available.

The selection heuristic can either pick

Propagation uses two-watched literals per clause, so that a clause is visited only when a watched literal is falsified.

Learned clauses can be deleted when they are unused in the partial assignment and not recently active in conflicts.

Frequent restarts are good for learning useful short clauses in order to better direct the search.

All level 0 inferences can be applied permanently.
We can build compact, easily checkable resolution certificates since each literal in $M_0$ and each conflict clause in $C$ has an associated proof.

<table>
<thead>
<tr>
<th>Num.</th>
<th>Clause</th>
<th>Proof</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$p \lor q$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$\neg p \lor q$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$p \lor \neg q$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$\neg p \lor r$</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$\neg q \lor \neg r$</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>$q$</td>
<td>0, 1</td>
</tr>
<tr>
<td>6</td>
<td>$p$</td>
<td>5, 2</td>
</tr>
<tr>
<td>7</td>
<td>$r$</td>
<td>3, 6</td>
</tr>
<tr>
<td>8</td>
<td>$\bot$</td>
<td>4, 5, 7</td>
</tr>
</tbody>
</table>
The input clause set $K$ is partitioned into $K_1$ and $K_2$.

If $K$ is unsatisfiable, there is a formula (interpolant) $I$ such that $K_1 \implies I$ and $K_2 \land I \implies \bot$.

Furthermore, $\text{atoms}(I) \subseteq \text{atoms}(K_1) \cap \text{atoms}(K_2)$.

The interpolant for a proof can be constructed from the interpolant $I_\Gamma$ for each clause $\Gamma$ in the proof.

Each clause $\Gamma$ in the proof is partitioned into $\Gamma_1 \lor \Gamma_2$ with $\text{atoms}(\Gamma_2) \subseteq \text{atoms}(K_2)$ and $\text{atoms}(\Gamma_1) \cap \text{atoms}(K_2) = \emptyset$.

The interpolant $I_\Gamma$ has the property that $K_1 \vdash \neg \Gamma_1 \implies I_\Gamma$ and $K_2 \vdash I_\Gamma \implies \Gamma_2$. 
For an input clauses $\kappa = \kappa_1 \lor \kappa_2$ in $K_1$, the interpolant $I_\kappa = \kappa_2$.

For input clauses $\kappa_2$ in $K_2$, the interpolant is $\top$.

When resolving $\kappa'$, $\kappa''$ to get $\kappa$,

- If resolvent $p$ is in $\kappa'_1$ (i.e., $p \not\in \text{atoms}(K_2)$), then $I_\kappa = I_{\kappa'} \lor I_{\kappa''}$ since $\neg(p \lor \kappa'_1) \implies I_{\kappa'}$ and $I_{\kappa'} \implies \kappa'_2$, and $\neg(\neg p \lor \kappa''_1) \implies I_{\kappa''}$ and $I_{\kappa''} \implies \kappa''_2$.

- If resolvent $p$ is in $\kappa'_2$, then $I_\kappa = I_{\kappa'} \land I_{\kappa''}$ since $\neg(\kappa'_1 \lor \kappa''_1) \implies I_\kappa$ and
  $I_\kappa \implies (p \lor \kappa'_2) \land (\neg p \lor \kappa''_2) \iff \kappa'_2 \lor \kappa''_2$. 
Interpolation Example

- Let $K_1 = \{ a \lor e[e], \neg a \lor b[b], \neg a \lor c[c] \}$, and $K_2 = \{ \neg b \lor \neg c \lor d[\top], \neg d[\top], \neg e[\top] \}$, with shared variables $b, c, \text{and } e$.

- The annotated proof is given by

<table>
<thead>
<tr>
<th>Conc.</th>
<th>Interp.</th>
<th>Clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$[e]$</td>
<td>$a \lor e[e], \neg e[\top]$</td>
</tr>
<tr>
<td>$b$</td>
<td>$[e \lor b]$</td>
<td>$a, \neg a \lor b$</td>
</tr>
<tr>
<td>$c$</td>
<td>$[e \lor c]$</td>
<td>$\neg a \lor c, a$</td>
</tr>
<tr>
<td>$\neg c \lor d$</td>
<td>$[e \lor b]$</td>
<td>$a \lor e, \neg a \lor b$</td>
</tr>
<tr>
<td>$d$</td>
<td>$[(e \lor b) \land (e \lor c)]$</td>
<td>$\neg c \lor d, c$</td>
</tr>
<tr>
<td>$\bot$</td>
<td>$[(e \lor b) \land (e \lor c)]$</td>
<td>$d, \neg d$</td>
</tr>
</tbody>
</table>
To find all satisfying assignments for $K$, add a field $B$ to CDCL to collect the blocking clauses corresponding to the current set of assignments.

For input $\neg a \lor b, c$, the first assignment yields $M = c; a, b$. Add the negation $\neg c \lor \neg a \lor \neg b$ as a blocking clause to $B$ and continue. (This could be reduced to $\neg c \lor \neg b$.)

The next assignment $M' = c; a, \neg b$ generates a conflict, so we add the conflict clause $\neg c \lor \neg a$ to $C$.

Next, $c, \neg a; b$ is a satisfying assignment, so $\neg c \lor a \lor \neg b$ is added to $B$. Finally, $c, \neg a, \neg b$ is also satisfying, and hence $\neg c \lor a \lor b$ is added to $B$.

There is a conflict at level 0, and $\neg \bigwedge B$ is the required DNF form of input $K$.

If $\text{atoms}(K) = X \uplus Y$, then eliminating literals corresponding to $X$ when adding clauses to $B$, computes the DNF of $\exists X.K$. 
With soft constraints, all constraints may not be satisfiable, but the goal is to satisfy as many constraints as possible.

Each constraint $A_i$ can be augmented as $a_i \lor A_i$, for a fresh variable $a_i$.

We can add constraints indicating that at most $k$ of the $a_i$ literals can be assigned $\top$.

By shrinking $k$, we can determine the minimal value of $k$.

Weighted MaxSAT can be solved similarly.

More generally, pseudo-Boolean constraints $\sum_i w_i \ast a_i \leq k$ can be encoded.
Inference systems help structure the correctness arguments.

Several theoretical results are in *Modularity and refinement in inference systems* [Ganzinger, R, S].

Simplifiers are inference systems without canonicity.

Many inference algorithms can be described as inference systems, e.g.,

1. Union-find for equality
2. Propositional resolution
3. Basic superposition for equality/propositional reasoning
4. CDCL
5. Simplex-based linear arithmetic reasoning
6. SMT
In SMT solving, the Boolean atoms represent constraints over individual variables ranging over integers, reals, datatypes, and arrays.

The constraints can involve theory operations, equality, and inequality.

The SAT solver has to interact with a theory constraint solver which propagates truth assignments and adds new clauses.

The theory solver can detect conflicts involving theory reasoning, e.g.,

1. \( f(x) = f(y) \lor x \neq y \)
2. \( f(x - 2) \neq f(y + 3) \lor x - y \leq 5 \lor y - z \leq -2 \lor z - x \leq -3 \)
3. \( x \text{ XOR } y \neq 0b0000000 \lor \text{select}(\text{store}(A, x, v), y) = v \)

The theory solver must produce efficient explanations, incremental assertions, and efficient backtracking.
Core theory: Equalities between variables \( x = y \), offset equalities \( x = y + c \).

Term equality: Congruence closure for uninterpreted function symbols

Difference constraints: Incremental negative cycle detection for inequality constraints of the form \( x - y \leq k \).

Linear arithmetic constraints: Fourier’s method, Simplex.

Bit Vectors: Bit-blasting
The satisfiability procedure uses a theory constraint solver oracle which maintains the theory state $S$ with the interface operations:

1. $\text{assert}(l, S)$ adds literal $l$ to the theory state $S$ returning a new state $S'$ or $\perp[\Delta]$.

2. $\text{check}(S)$ checks if the conjunction of literals asserted to $S$ is satisfiable, and returns either $\top$ or $\perp[\Delta]$.

3. $\text{retract}(S, l)$: Retracts, in reverse chronological order, the assertions up to and including $l$ from state $S$.

4. $\text{model}(S)$: Builds a model for a state known to be satisfiable.
Satisfiability Modulo Theories

- SMT deals with formulas with theory atoms like $x = y$, $x \neq y$, $x - y \leq 3$, and $\text{select}(\text{store}(A, i, v), j) = w$.
- The CDCL search state is augmented with a theory state $S$ in addition to the partial assignment.
- Total assignments are checked for theory satisfiability.
- When a literal is added to $M$ by unit propagation, it is also asserted to $S$.
- When a literal is implied by $S$, it is propagated to $M$.
- When backjumping, the literals deleted from $M$ are also retracted from $S$. 
Input is $y = z$, $x = y \lor x = z$, $x \neq y \lor x \neq z$

<table>
<thead>
<tr>
<th>Step</th>
<th>$M$</th>
<th>$F$</th>
<th>$D$</th>
<th>$C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assert</td>
<td>$y = z$</td>
<td>${y \mapsto z}$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>Select</td>
<td>$y = z; x \neq y$</td>
<td>${y \mapsto z}$</td>
<td>${x \neq y}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>Prop</td>
<td>$\ldots, x \neq z$</td>
<td>${y \mapsto z}$</td>
<td>${x \neq y}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td></td>
<td>$[x \neq z \lor y \neq z \lor x = y]$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conflict</td>
<td>$\ldots$</td>
<td>${y \mapsto z}$</td>
<td>${x \neq y}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>Analyze</td>
<td>$\ldots$</td>
<td>${y \mapsto z}$</td>
<td>${x \neq y}$</td>
<td>${y \neq z \lor x = y}$</td>
</tr>
<tr>
<td>Bkjump</td>
<td>$y = z, x = y$</td>
<td>${y \mapsto z}$</td>
<td></td>
<td>$\ldots$</td>
</tr>
<tr>
<td>Assert</td>
<td>$y = z, x = y$</td>
<td>${x \mapsto y, y \mapsto z}$</td>
<td></td>
<td>$\ldots$</td>
</tr>
<tr>
<td>Scan</td>
<td>$\ldots, x = z$</td>
<td>${x \mapsto y, y \mapsto z}$</td>
<td></td>
<td>$\ldots$</td>
</tr>
<tr>
<td></td>
<td>$[x = z \lor x \neq y \lor y \neq z]$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conflict</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SMT Applications

- **Test generation**: Find assignments to the individual variables satisfying a path constraint in a program.

- **Infinite-state bounded model checking**: BMC for programs with assignments, unbounded arithmetic, arrays, datatypes, and timers.

- **Predicate abstraction and abstract reachability**: For an atom substitution $\gamma$ and formula $\phi$, find Boolean formula $\hat{\phi}$ such that $\phi \implies \gamma(\hat{\phi})$.

- Scheduling, planning, constraint solving, and MaxSAT in unbounded domains.
Given state $\Sigma$, initial state predicate $I(s)$, next-state relation $N(s, s')$, and assertion $P(s)$.

**Bounded Model Checking:**

$satisfiable(I(s_0) \land \bigwedge_{i=0}^{k-1} N(s_i, s_{i+1}) \land \bigvee_{i=0}^{k} \neg P(s_i))$

**$k$-Induction:**

$satisfiable(\bigwedge_{i=0}^{k} N(s_i, s_{i+1}) \land \bigwedge_{i=0}^{k} P(s_i) \land \neg P(s_{k+1}))$

**Image computation:** Compute the formula representing $AX_N P$, or $\forall s' : N(s_0, s') \land P(s')$

**Fixpoints:** Compute the formula representing $AG_N P$.

**Interpolant:** Find interpolant formula $F$ such that $I(s_0) \land N(s_0, s_1) \implies F(s_1)$ and $\neg(F(s_1) \land \bigwedge_{i=1}^{k-1} N(s_i, s_{i+1}) \land \bigvee_{i=0}^{k} P(s_i))$. 
SMT Applications

- **Test generation**: Find assignments to the individual variables satisfying a path constraint in a program.

- **Infinite-state bounded model checking**: BMC for programs with assignments, unbounded arithmetic, arrays, datatypes, and timers.

- **Predicate abstraction and abstract reachability**: For an atom substitution $\gamma$ and formula $\phi$, find Boolean formula $\hat{\phi}$ such that $\phi \implies \gamma(\hat{\phi})$.

- Scheduling, planning, constraint solving, and MaxSAT in unbounded domains.
Quantifiers

- Since first-order logic is undecidable, satisfiability is not solvable for arbitrary quantified formulas.
- The useful $\exists \forall$ (Bernays–Schönfinkel) fragment (with theories) is reducible to SMT.
- Some theories, e.g., datatypes, linear arithmetic over integers, arithmetic over the integers, support quantifier elimination.
- Existential quantifiers can be skolemized, but the problem of instantiating universal quantifiers for detecting unsatisfiability remains.
- **E-graph matching** can be used for example to match $f(g(x))$ with $a$, when the solver is able to show that $a = f(b)$ and $b = g(c)$ in the given context.
Yices is a high-performance SMT solver that supports

1. An *expressive language* with higher-order types, dependent types, and predicate subtypes.

2. A *combination of theories* including uninterpreted functions, linear arithmetic, records, tuples, datatypes, arrays, and bit-vectors.

3. A *command language* with incremental definitions, assertions, context creation and examination, pushing/popping contexts, and MaxSAT.

Yices is integrated with SAL and PVS, and is used in hardware/software verification, bounded model checking, planning, probabilistic consistency using MaxSAT, concolic execution.
bignum [ base : above(1) ] : THEORY
BEGIN
l, m, n: VAR nat
cin : VAR upto(1)
digit : TYPE = below(base)

JUDGEMENT 1 HAS_TYPE digit

i, j, k: VAR digit
bignum : TYPE = list[digit]
X, Y, Z, X1, Y1: VAR bignum

val(X) : RECURSIVE nat =
  CASES X of
  null: 0,
  cons(i, Y): i + base * val(Y)
ENDCASES
MEASURE length(X);
Adding a Digit to a Number

+(X, i): RECURSIVE bignum =
(CASES X of
    null: cons(i, null),
    cons(j, Y):
        (IF i + j < base
            THEN cons(i+j, Y)
            ELSE cons(i + j - base, Y + 1)
        )ENDIF)
ENDCASES
MEASURE length(X);

correct_plus: LEMMA
  val(X + i) = val(X) + i
bigplus(X, Y, (cin : upto(1))): RECURSIVE bignum =
   CASES X of
   null: Y + cin,
   cons(j, X1):
      CASES Y of
      null: X + cin,
      cons(k, Y1):
         (IF cin + j + k < base
          THEN cons((cin + j + k - base),
            bigplus(X1, Y1, 1))
          ELSE cons((cin + j + k), bigplus(X1, Y1, 0))
          ENDF)
      ENDCASES
   ENDCASES
   MEASURE length(X)

bigplus_correct: LEMMA
   val(bigplus(X, Y, cin)) = val(X) + val(Y) + cin
(define c1::(-> int bool))
(define r1::(-> int bool))
(define t::(-> int bool))
(define c2::(-> int bool))
(define r2::(-> int bool))

(define I::bool (and (not (c1 0)) (not (c2 0))(not (r1 0))
   (not (r2 0))(not (t 0))))

(define N1::(-> int bool) ...)
(define N2::(-> int bool) ...)
(define N::(-> int bool) ...)
Yices Example: Property

(define safe::(-> int bool)
    (lambda (i::int)(not (and (c1 i)(c2 i)))))

(define iter_N::(-> int int int bool) (lambda (i::int j::int)
    (if (<= i 0) (N j) (and (N (+ i j)) (iter_N (- i 1) j))))))

(define safeto::(-> int int int bool)(lambda (i::int j::int)
    (if (<= i 0) (safe j)(and (safe (+ i j))
        (safeto (- i 1) j)))))))

(define PBMC::(-> int bool) (lambda (i::int)
    (and I (iter_N i 0) (not (safeto (+ i 1) 0))))))

(define PIND::(-> int bool)(lambda (i::int)(exists (j::int)
    (and (iter_N i j) (safeto i j)(not (safeto (+ i 1) j))))))

(assert (or (PBMC 3) (PIND 3)))
(check)
Hard Sudoku Solved with sal-inf-bmc [Whalen]

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
<td>1</td>
<td>7</td>
<td>3</td>
<td>9</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>8</td>
<td>7</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>8</td>
<td>5</td>
<td>3</td>
<td>7</td>
<td>6</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>4</td>
<td>8</td>
<td>9</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>1</td>
<td>8</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>9</td>
<td>5</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>9</td>
</tr>
</tbody>
</table>
Conclusions

- Powerful, mature, and versatile tools like SAT and SMT solvers can now be exploited in very useful ways.
- They are at the core of powerful, modern verification tools like PVS, SAL, and Yices (see http://fm.csl.sri.com).
- Applications include test generation, model checking, theorem proving, abstraction, scheduling, planning, soft constraint solving.
- These tools can be used either as blackboxes, or through scriptable interaction.
- The construction and application of satisfiability procedures is an active research area with exciting challenges (nonlinear arithmetic, quantifier reasoning, scalability, interfaces, integration).