# Verification Using SAT and SMT Solvers[1]

N. Shankar

Computer Science Laboratory
SRI International
Menlo Park, CA

Aug 15, 2010

[1] This 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 Basics

- 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.

# Language

- 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* $\tau := x \mid f(\tau_1, \ldots, \tau_n)$
  - *Atoms* $\alpha := p(\tau_1, \ldots \tau_n)$,
  - *Literals* $\lambda := \alpha \mid \neg\alpha$
  - *Constraints* $\lambda_1 \wedge \ldots \wedge \lambda_n$,
  - *Clauses* $\lambda_1 \vee \ldots \vee \lambda_n$,
  - *Formulas* $\psi := \begin{array}{l} p(\tau_1, \ldots, \tau_n) \mid \tau_0 = \tau_1 \mid \neg\psi_0 \mid \\ \psi_0 \vee \psi_1 \mid \psi_0 \wedge \psi_1 \mid (\exists x : \psi_0) \mid (\forall x : \psi_0) \end{array}$

# Semantics

- 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 $\mathcal{M}_\tau$.

# 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 $\vee$.
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 $\vee$.
4. State and verify an algorithm to test a Boolean formula for satisfiability and return a satisfying truth assignment when possible.

# More Exercises: Equivalence

- 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. $(A \implies B) \iff (\neg A \vee B)$
3. $\neg(A \wedge B) \iff (\neg A \vee \neg B)$
4. $\neg(A \vee B) \iff (\neg A \wedge \neg B)$
5. $(\neg A \implies B) \iff (\neg B \implies A)$

# More Exercises: Normal Forms

- 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.

# Inference System

- 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 $\perp$ 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.

| | | |
|---|---|---|
| **Res** | $\dfrac{K, l \vee \Gamma_1, \bar{l} \vee \Gamma_2}{K, l \vee \Gamma_1, \bar{l} \vee \Gamma_2, \Gamma_1 \vee \Gamma_2}$ | $\Gamma_1 \vee \Gamma_2 \notin K$<br>$\Gamma_1 \vee \Gamma_2$ is not tautological |
| **Contrad** | $\dfrac{K, l, \bar{l}}{\perp}$ | |

# Ordered Resolution: Example

$$\cfrac{\cfrac{\cfrac{\cfrac{(K_0 =)\ \neg p \vee \neg q \vee r, \quad \neg p \vee q, \quad p \vee r, \quad \neg r}{(K_1 =)\ \neg q \vee r, \quad K_0}\ \textbf{Res}}{(K_2 =)\ q \vee r, \quad K_1}\ \textbf{Res}}{(K_3 =)\ r, \quad K_2}\ \textbf{Res}}{\bot}\ \textbf{Contrad}$$

# Correctness

- **Progress:** Bounded number of clauses in the given literals. Each application of **Res** generates a new clause.
- **Conservation:** For any model $M$, if $M \models l \vee \Gamma_1$ and $M \models \bar{l} \vee \Gamma_2$, then $M \models \Gamma_1 \vee \Gamma_2$.
- **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 $p_{i+1} \vee \Gamma \in K$ and $M_i \not\models \Gamma$, then let $M_{i+1} = M_i\{p_{i+1} \mapsto \top\}$.
    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 $l \vee \Gamma \in K \cup C$ and $M \models \neg\Gamma$.
- If $M$ detects a conflict, i.e., a clause $\Gamma \in K \cup C$ such that $M \models \neg\Gamma$, 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.

# Conflict-Driven Clause Learning (CDCL) SAT

| Name | Rule | Condition |
|---|---|---|
| Propagate | $\dfrac{h, \langle M \rangle, K, C}{h, \langle M, l[\Gamma] \rangle, K, C}$ | $\Gamma \equiv l \vee \Gamma' \in K \cup C$<br>$M \models \neg\Gamma'$ |
| Select | $\dfrac{h, \langle M \rangle, K, C}{h+1, \langle M; l[] \rangle, K, C}$ | $M \not\models l$<br>$M \not\models \neg l$ |
| Conflict | $\dfrac{0, \langle M \rangle, K, C}{\bot}$ | $M \models \neg\Gamma$<br>for some $\Gamma \in K \cup C$ |
| Backjump | $\dfrac{h+1, \langle M \rangle, K, C}{h', \langle M_{\leq h'}, l[\Gamma'] \rangle, K, C \cup \{\Gamma'\}}$ | $M \models \neg\Gamma$<br>for some $\Gamma \in K \cup C$<br>$\langle h', \Gamma' \rangle$<br>$= analyze(\psi)(\Gamma)$<br>for $\psi = h, \langle M \rangle, K, C$ |

# CDCL Example

- Let $K$ be $\{p \vee q, \neg p \vee q, p \vee \neg q, s \vee \neg p \vee q, \neg s \vee p \vee \neg q, \neg p \vee r, \neg q \vee \neg r\}$.
-

| step | $h$ | $M$ | $K$ | $C$ | $\Gamma$ |
|---|---|---|---|---|---|
| select $s$ | 1 | $; s$ | $K$ | $\emptyset$ | _ |
| select $r$ | 2 | $; s; r$ | $K$ | $\emptyset$ | _ |
| propagate | 2 | $; s; r, \neg q[\neg q \vee \neg r]$ | $K$ | $\emptyset$ | _ |
| propagate | 2 | $; s; r, \neg q, p[p \vee q]$ | $K$ | $\emptyset$ | _ |
| conflict | 2 | $; s; r, \neg q, p$ | $K$ | $\emptyset$ | $\neg p \vee q$ |

# CDCL Example (contd.)

| step | $h$ | $M$ | $K$ | $C$ | $\Gamma$ |
|---|---|---|---|---|---|
| conflict | 2 | $; s; r, \neg q, p$ | $K$ | $\emptyset$ | $\neg p \vee q$ |
| backjump | 0 | $\emptyset$ | $K$ | $q$ | _ |
| propagate | 0 | $q[q]$ | $K$ | $q$ | _ |
| propagate | 0 | $q, p[p \vee \neg q]$ | $K$ | $q$ | _ |
| propagate | 0 | $q, p, r[\neg p \vee r]$ | $K$ | $q$ | _ |
| conflict | 0 | $q, p, r$ | $K$ | $q$ | $\neg q \vee \neg r$ |

- **Progress:** Each backjump step adds a new assignment at the level $h'$ so that $\sum_{i=0}^{h} |M_i| * (N+1)^{(N-h)}$ increases toward the bound $(N+1)^{(N+1)}$ for $N = |vars(K)|$. In the example, $N = 4$, the backjump step goes from a value 1300 in base 5 to the value 10000 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 $K \cup C$, $M \models \Gamma$.

# Implementation Notes

- 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.

# CDCL with Proof

- We can build compact, easily checkable resolution certificates since each literal in $M_0$ and each conflict clause in $C$ has an associated proof

| Num. | Clause | Proof |
|---|---|---|
| 0 | $p \vee q$ | |
| 1 | $\neg p \vee q$ | |
| 2 | $p \vee \neg q$ | |
| 3 | $\neg p \vee r$ | |
| 4 | $\neg q \vee \neg r$ | |
| 5 | $q$ | $0, 1$ |
| 6 | $p$ | $5, 2$ |
| 7 | $r$ | $3, 6$ |
| 8 | $\perp$ | $4, 5, 7$ |

# Interpolant

- 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 $K_2 \wedge I \implies \bot$.
- Furthermore, $atoms(I) \subseteq atoms(K_1) \cap atoms(K_2)$.
- 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 $\Gamma_1 \vee \Gamma_2$ with $atoms(\Gamma_2) \subseteq atoms(K_2)$ and $atoms(\Gamma_1) \cap atoms(K_2) = \emptyset$.
- The interpolant $I_\Gamma$ has the property that $K_1 \vdash \neg\Gamma_1 \implies I_\Gamma$ and $K_2 \vdash I_\Gamma \implies \Gamma_2$.

# Interpolants from Resolution

- For an input clauses $\kappa = \kappa_1 \vee \kappa_2$ 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., $p \notin atoms(K_2)$), then $I_\kappa = I_{\kappa'} \vee I_{\kappa''}$ since $\neg(p \vee \kappa'_1) \implies I_{\kappa'}$ and $I_{\kappa'} \implies \kappa'_2$, and $\neg(\neg p \vee \kappa''_1) \implies I_{\kappa''}$ and $I_{\kappa''} \implies \kappa''_2$.
  - 
  - If resolvent $p$ is in $\kappa'_2$, then $I_\kappa = I_{\kappa'} \wedge I_{\kappa''}$ since $\neg(\kappa'_1 \vee \kappa''_1) \implies I_\kappa$ and $I_\kappa \implies (p \vee \kappa'_2) \wedge (\neg p \vee \kappa''_2) \iff \kappa'_2 \vee \kappa''_2$.

# Interpolation Example

- Let $K_1 = \{a \vee e[e], \neg a \vee b[b], \neg a \vee c[c]\}$, and $K_2 = \{\neg b \vee \neg c \vee d[\top], \neg d[\top], \neg e[\top]\}$, with shared variables $b$, $c$, and $e$.
- The annotated proof is given by

| Conc. | Interp. | Clauses |
|---|---|---|
| $a$ | $[e]$ | $a \vee \underline{e}[e], \underline{\neg e}[\top]$ |
| $b$ | $[e \vee b]$ | $\underline{a}, \neg a \vee b$ |
| $c$ | $[e \vee c]$ | $\neg a \vee c, a$ |
| $\neg c \vee d$ | $[e \vee b]$ | $a \vee e, \neg a \vee b$ |
| $d$ | $[(e \vee b) \wedge (e \vee c)]$ | $\neg c \vee d, c$ |
| $\bot$ | $[(e \vee b) \wedge (e \vee c)]$ | $d, \neg d$ |

- 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 $\neg a \vee b, c$, the first assignment yields $M = c; a, b$. Add the negation $\neg c \vee \neg a \vee \neg b$ as a blocking clause to $B$ and continue. (This could be reduced to $\neg c \vee \neg b$.)
- The next assignment $M' = c; a, \neg b$ generates a confict, so we add the conflict clause $\neg c \vee \neg a$ to $C$.
- Next, $c, \neg a; b$ is a satisfying assignment, so $\neg c \vee a \vee \neg b$ is added to $B$. Finally, $c, \neg a, \neg b$ is also satisfying, and hence $\neg c \vee a \vee b$ is added to $B$.
- There is a conflict at level 0, and $\neg \bigwedge B$ is the required DNF form of input $K$.
- If $atoms(K) = X \uplus Y$, then eliminating literals corresponding to $X$ when adding clauses to $B$, computes the DNF of $\exists X.K$.

# MaxSAT

- With soft constraints, all contraints may not be satisfiable, but the goal is to satisfy as many constraints as possible.
- Each constraint $A_i$ can be augmented as $a_i \vee A_i$, 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 $\Sigma_i w_i * a_i \leq k$ can be encoded.

# Example Inference Systems

- 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

# SMT Overview

- 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. $f(x) = f(y) \vee x \neq y$
  2. $f(x-2) \neq f(y+3) \vee x - y \leq 5 \vee y - z \leq -2 \vee z - x \leq -3$
  3. $x\ XOR\ y \neq 0b0000000 \vee select(store(A, x, v), y) = v$
- The theory solver must produce efficient explanations, incremental assertions, and efficient backtracking.

# Example Constraint Solvers

- **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

# Theory Constraint Solver Interface

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

1. $assert(l, S)$ adds literal $l$ to the theory state $S$ returning a new state $S'$ or $\bot[\Delta]$
2. $check(S)$ checks if the conjunction of literals asserted to $S$ is satisfiable, and returns either $\top$ or $\bot[\Delta]$.
3. $retract(S, l)$: Retracts, in reverse chronological order, the assertions up to and including $l$ from state $S$.
4. $model(S)$: 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 $select(store(A, i, v), j) = w$.
- 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 $S$.

# SMT example

Input is $y = z, \quad x = y \vee x = z, \quad x \neq y \vee x \neq z$

| Step | $M$ | $F$ | $D$ | $C$ |
|---|---|---|---|---|
| Assert | $y = z$ | $\{y \mapsto z\}$ | $\emptyset$ | $\emptyset$ |
| Select | $y = z; x \neq y$ | $\{y \mapsto z\}$ | $\{x \neq y\}$ | $\emptyset$ |
| Prop | $\ldots, x \neq z$ $[x \neq z \vee y \neq z \vee x = y]$ | $\{y \mapsto z\}$ | $\{x \neq y\}$ | $\emptyset$ |
| Conflict | $\ldots$ | $\{y \mapsto z\}$ | $\{x \neq y\}$ | $\emptyset$ |
| Analyze | $\ldots$ | $\{y \mapsto z\}$ | $\{x \neq y\}$ | $\{y \neq z \vee x = y\}$ |
| Bkjump | $y = z, x = y$ | $\{y \mapsto z\}$ | | $\ldots$ |
| Assert | $y = z, x = y$ | $\{x \mapsto y, y \mapsto z\}$ | | $\ldots$ |
| Scan | $\ldots, x = z$ $[x = z \vee x \neq y \vee y \neq z]$ | $\{x \mapsto y, y \mapsto z\}$ | | $\ldots$ |
| Conflict | | | | |

# 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.

# Transition Systems

- Given *state* $\Sigma$, *initial state predicate* $I(s)$, *next-state relation* $N(s, s')$, and *assertion* $P(s)$.
- **Bounded Model Checking:** $satisfiable(I(s_0) \wedge \bigwedge_{i=0}^{k-1} N(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{k} \neg P(s_i))$
- ***k*-Induction:** $satisfiable(\bigwedge_{i=0}^{k} N(s_i, s_{i+1}) \wedge \bigwedge_{i=0}^{k} P(s_i) \wedge \neg P(s_{k+1}))$
- **Image computation:** Compute the formula representing $\mathbf{AX}_N P$, or $\forall s' : N(s0, s') \wedge P(s')$
- **Fixpoints:** Compute the formula representing $\mathbf{AG}_N P$.
- **Interpolant:** Find interpolant formula $F$ such that $I(s_0) \wedge N(s_0, s_1) \implies F(s_1)$ and $\neg(F(s_1) \wedge \bigwedge_{i=1}^{k-1} N(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{k} P(s_i))$.

# 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

- 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.

# Introducing PVS: Number Representation

```
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
```

# Adding Two Numbers

```
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))
        ENDIF)
     ENDCASES
   ENDCASES
 MEASURE length(X)

bigplus_correct: LEMMA
 val(bigplus(X, Y, cin)) = val(X) + val(Y) + cin
```

# Yices Examples: Peterson

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

(define safeto::(-> 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 [Wikipedia/Algorithmics_of_Sudoku]

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | |
| | | | | | 3 | | 8 | 5 |
| | | 1 | | 2 | | | | |
| | | | 5 | | 7 | | | |
| | | 4 | | | | 1 | | |
| | 9 | | | | | | | |
| 5 | | | | | | | 7 | 3 |
| | | 2 | | 1 | | | | |
| | | | | 4 | | | | 9 |

# Hard Sudoku Solved with `sal-inf-bmc` [Whalen]

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| 2 | 4 | 6 | 1 | 7 | 3 | 9 | 8 | 5 |
| 3 | 5 | 1 | 9 | 2 | 8 | 7 | 4 | 6 |
| 1 | 2 | 8 | 5 | 3 | 7 | 6 | 9 | 4 |
| 6 | 3 | 4 | 8 | 9 | 2 | 1 | 5 | 7 |
| 7 | 9 | 5 | 4 | 6 | 1 | 8 | 3 | 2 |
| 5 | 1 | 9 | 2 | 8 | 6 | 4 | 7 | 3 |
| 4 | 7 | 2 | 3 | 1 | 9 | 5 | 6 | 8 |
| 8 | 6 | 3 | 7 | 4 | 5 | 2 | 1 | 9 |

# 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).