Developing Efficient SMT Solvers

ESARLT 2007

Leonardo de Moura

leonardo@microsoft.com

Microsoft Research

Introduction

- *Satisfiability Modulo Theories (SMT)*
  - The next generation of verification engines.
  - *SAT solvers + Theories*
    - Arithmetic
    - Arrays
    - Uninterpreted Functions
  - Some problems are more naturally expressed in SMT.
  - More automation.

Applications

- Applications have different requirements.
- Predicate abstraction
  - Fast when unsat.
  - May be incomplete.
  - Examples: *Microsoft SLAM/SDV (device driver verification).*
- Testing
  - Fast when sat.
  - Model generation.
  - May be unsound.
  - Examples: *Microsoft MUTT and Sage.*

Applications (cont.)

- Extended Static Checking.
  - Fast when sat & unsat.
  - Must be sound.
  - “Counterexamples” (execution trace).
  - Incompleteness $\rightsquigarrow$ false alarms.
  - Examples: ESC/Java, *Microsoft Spec# and ESP.*
- Bounded Model Checking (BMC) & k-induction.
- Planning & Scheduling.
- Symbolic Simulation.
- Equivalence Checking.

Roadmap

- Background
- Architecture
- Implementation Techniques
- Applications

Language

- A *signature* Σ is a finite set of: function symbols Σ_F = $\{f, g, \ldots\}$, predicate symbols Σ_P = $\{p, q, \ldots\}$, and an *arity* function $\Sigma \mapsto N$.
- Function symbols with arity 0 are called *constants*.
- A countable set $\mathcal{V}$ of *variables* $\{x, y, \ldots\}$ disjoint of Σ.
- *Terms:*

  $$t \ := \ f(t_1, \ldots, t_n) \mid x$$

- *Formulas:*

  $$\phi := p(t_1, \ldots, t_n) \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \neg\phi_1 \mid \exists x : \phi_1 \mid \forall x : \phi_1$$

- *Free* (occurrences) of *variables* in a formula are those not bound by a quantifier.
- A *sentence* is a first-order formula with no free variables.

Theories

- A *(first-order) theory* $\mathcal{T}$ (over a signature Σ) is a set of (deductively closed) sentences (over Σ and $\mathcal{V}$).
- Let *DC*(Γ) be the deductive closure of a set of sentences Γ.
  - For every theory $\mathcal{T}$, *DC*$(\mathcal{T}) = \mathcal{T}$.
- A theory $\mathcal{T}$ is *consistent* if *false* $\notin \mathcal{T}$.
- We can view a (first-order) theory $\mathcal{T}$ as the class of all *models* of $\mathcal{T}$ (due to completeness of first-order logic).

Models (Semantics)

- A model M is defined as:
  - Domain S: set of elements.
  - Interpretation $f^M : S^n \mapsto S$ for each $f \in \Sigma_F$ with *arity*$(f) = n$.
  - Interpretation $p^M \subseteq S^n$ for each $p \in \Sigma_P$ with *arity*$(p) = n$.
  - Assignment $x^M \in S$ for every variable $x \in \mathcal{V}$.
- A formula ϕ is true in a model M if it evaluates to true under the given interpretations over the domain S.
- M is a *model for the theory* $\mathcal{T}$ if all sentences of $\mathcal{T}$ are true in M.

Satisfiability and Validity

- A formula $\phi(\vec{x})$ is *satisfiable* in a theory $\mathcal{T}$ if there is a model of *DC*$(\mathcal{T} \cup \exists\vec{x}.\phi(\vec{x}))$. That is, there is a model M for $\mathcal{T}$ in which $\phi(\vec{x})$ evaluates to true, denoted by,

$$M \models_{\mathcal{T}} \phi(\vec{x})$$

- This is also called *$\mathcal{T}$-satisfiability*.
- A formula $\phi(\vec{x})$ is *valid* in a theory $\mathcal{T}$ if $\forall\vec{x}.\phi(\vec{x}) \in \mathcal{T}$. That is $\phi(\vec{x})$ evaluates to true in every model M of $\mathcal{T}$.
- *$\mathcal{T}$-validity* is denoted by $\models_{\mathcal{T}} \phi(\vec{x})$.
- The *quantifier free $\mathcal{T}$-satisfiability problem* restricts ϕ to be *quantifier free*.

Combination of Theories

- In practice, we need a combination of theories.
- Examples:
  - $x + 2 = y \Rightarrow f(\mathit{read}(\mathit{write}(a, x, 3), y - 2)) = f(y - x + 1)$
  - $f(f(x) - f(y)) \neq f(z), x + z \leq y \leq x \Rightarrow z < 0$
- Given

$$
\begin{array}{rcl}
\Sigma & = & \Sigma_1 \cup \Sigma_2 \\
\mathcal{T}_1, \mathcal{T}_2 & : & \textit{theories over } \Sigma_1, \Sigma_2 \\
\mathcal{T} & = & \mathit{DC}(\mathcal{T}_1 \cup \mathcal{T}_2)
\end{array}
$$

- Is $\mathcal{T}$ consistent?
- Given satisfiability procedures for conjunction of literals of $\mathcal{T}_1$ and $\mathcal{T}_2$, how to decide the satisfiability of $\mathcal{T}$?

Preamble

- Disjoint signatures: $\Sigma_1 \cap \Sigma_2 = \emptyset$.
- Stably-Infinite Theories.
- Convex Theories.

Stably-Infinite Theories

- A theory is *stably infinite* if every satisfiable QFF is satisfiable in an infinite model.
- Example. Theories with only finite models are not stably infinite. $\mathcal{T}_2 = DC(\forall x, y, z. \, (x = y) \vee (x = z) \vee (y = z))$.
- *The union of two consistent, disjoint, stably infinite theories is consistent.*

Convexity

- A theory $\mathcal{T}$ is *convex* iff

  for all finite sets Γ of literals and

  for all non-empty disjunctions $\bigvee_{i \in I} x_i = y_i$ of variables,

  $\Gamma \models_{\mathcal{T}} \bigvee_{i \in I} x_i = y_i$ iff $\Gamma \models_{\mathcal{T}} x_i = y_i$ for some $i \in I$.
- Every convex theory $\mathcal{T}$ with non trivial models (i.e., $\models_T \exists x, y.\, x \neq y$) is stably infinite.
- All *Horn* theories are convex – this includes all (conditional) equational theories.
- *Linear rational arithmetic is convex.*

Convexity (cont.)

- *Many theories are not convex:*
  - Linear integer arithmetic.

$$y = 1, z = 2, 1 \leq x \leq 2 \models x = y \vee x = z$$

  - Nonlinear arithmetic.

$$x^2 = 1, y = 1, z = -1 \models x = y \vee x = z$$

  - Theory of Bit-vectors.
  - Theory of Arrays.

$$v_1 = \textit{read}(\textit{write}(a, i, v_2), j), v_3 = \textit{read}(a, j) \models$$
$$v_1 = v_2 \vee v_1 = v_3$$

Convexity: Example

- Let $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$, where $\mathcal{T}_1$ is EUF ($O(nlog(n))$) and $\mathcal{T}_2$ is IDL ($O(nm)$).
- *$\mathcal{T}_2$ is not convex*.
- *Satisfiability is NP-Complete for $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$.*
  - Reduce 3CNF satisfiability to $\mathcal{T}$-satisfiability.
  - For each boolean variable p_i add the atomic formulas: $0 \leq x_i, x_i \leq 1$.
  - For a clause $p_1 \vee \neg p_2 \vee p_3$ add the atomic formula: $f(x_1, x_2, x_3) \neq f(0, 1, 0)$

Nelson-Oppen Combination

- Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be consistent, stably infinite theories over disjoint (countable) signatures. Assume satisfiability of conjunction of literals can decided in $O(T_1(n))$ and $O(T_2(n))$ time respectively. Then,
  1. The combined theory $\mathcal{T}$ is consistent and stably infinite.
  2. Satisfiability of quantifier free conjunction of literals in $\mathcal{T}$ can be decided in $O(2^{n^2} \times (T_1(n) + T_2(n))$.
  3. If $\mathcal{T}_1$ and $\mathcal{T}_2$ are convex, then so is $\mathcal{T}$ and satisfiability in $\mathcal{T}$ is in $O(n^4 \times (T_1(n) + T_2(n)))$.

Nelson-Oppen Combination Procedure

- The combination procedure:

  Initial State: ϕ is a conjunction of literals over $\Sigma_1 \cup \Sigma_2$.

  Purification: Preserving satisfiability transform ϕ into $\phi_1 \wedge \phi_2$, such that, $\phi_i \in \Sigma_i$.

  Interaction: Guess a partition of $\mathcal{V}(\phi_1) \cap \mathcal{V}(\phi_2)$ into disjoint subsets. Express it as conjunction of literals ψ.
  Example. The partition $\{x_1\}, \{x_2, x_3\}, \{x_4\}$ is represented as $x_1 \neq x_2, x_1 \neq x_4, x_2 \neq x_4, x_2 = x_3$.

  Component Procedures : Use individual procedures to decide whether $\phi_i \wedge \psi$ is satisfiable.

  Return: If both return yes, return yes. No, otherwise.

Purification

- Purification:

  $$\phi \wedge P(\ldots, s[t], \ldots) \rightsquigarrow \phi \wedge P(\ldots, s[x], \ldots) \wedge x = t,$$

  t is not a variable.

- *Purification is satisfiability preserving and terminating.*

- As most of the SMT developers will tell you, the purification step is not really necessary.

- Given a set of mixed (impure) literal Γ, define a *shared term* to be any term in Γ which is *alien* in some literal or sub-term in Γ.

- In our examples, these were the terms replaced by constants.

- Assume that each satisfiability procedure treats alien terms as constants.

NO procedure: soundness

- Each step is satisfiability preserving.
- Say ϕ is satisfiable (in the combination).
  - Purification: $\phi_1 \wedge \phi_2$ is satisfiable.

NO procedure: soundness

- Each step is satisfiability preserving.
- Say ϕ is satisfiable (in the combination).
  - Purification: $\phi_1 \wedge \phi_2$ is satisfiable.
  - Iteration: for some partition ψ, $\phi_1 \wedge \phi_2 \wedge \psi$ is satisfiable.

NO procedure: soundness

- Each step is satisfiability preserving.
- Say ϕ is satisfiable (in the combination).
  - Purification: $\phi_1 \wedge \phi_2$ is satisfiable.
  - Iteration: for some partition ψ, $\phi_1 \wedge \phi_2 \wedge \psi$ is satisfiable.
  - Component procedures: $\phi_1 \wedge \psi$ and $\phi_2 \wedge \psi$ are both satisfiable in component theories.

NO procedure: soundness

- Each step is satisfiability preserving.
- Say ϕ is satisfiable (in the combination).
  - Purification: $\phi_1 \wedge \phi_2$ is satisfiable.
  - Iteration: for some partition ψ, $\phi_1 \wedge \phi_2 \wedge \psi$ is satisfiable.
  - Component procedures: $\phi_1 \wedge \psi$ and $\phi_2 \wedge \psi$ are both satisfiable in component theories.
  - Therefore, if the procedure return unsatisfiable, then ϕ is unsatisfiable.

NO procedure: correctness

- Suppose the procedure returns satisfiable.
  - Let ψ be the partition and A and B be models of $\mathcal{T}_1 \wedge \phi_1 \wedge \psi$ and $\mathcal{T}_2 \wedge \phi_2 \wedge \psi$.

NO procedure: correctness

- Suppose the procedure returns satisfiable.
  - Let ψ be the partition and A and B be models of $\mathcal{T}_1 \wedge \phi_1 \wedge \psi$ and $\mathcal{T}_2 \wedge \phi_2 \wedge \psi$.
  - The component theories are stably infinite. So, assume the models are infinite (of same cardinality).

NO procedure: correctness

- Suppose the procedure returns satisfiable.
  - Let ψ be the partition and A and B be models of $\mathcal{T}_1 \wedge \phi_1 \wedge \psi$ and $\mathcal{T}_2 \wedge \phi_2 \wedge \psi$.
  - The component theories are stably infinite. So, assume the models are infinite (of same cardinality).
  - Let h be a bijection between S_A and S_B such that $h(x^A) = x^B$ for each shared variable.

NO procedure: correctness

- Suppose the procedure returns satisfiable.
  - Let ψ be the partition and A and B be models of $\mathcal{T}_1 \wedge \phi_1 \wedge \psi$ and $\mathcal{T}_2 \wedge \phi_2 \wedge \psi$.
  - The component theories are stably infinite. So, assume the models are infinite (of same cardinality).
  - Let h be a bijection between S_A and S_B such that $h(x^A) = x^B$ for each shared variable.
  - Extend B to $\bar{B}$ by interpretations of symbols in Σ_1: $f^{\bar{B}}(b_1, \ldots, b_n) = h(f^A(h^{-1}(b_1), \ldots, h^{-1}(b_n)))$

NO procedure: correctness

- Suppose the procedure returns satisfiable.
  - Let ψ be the partition and A and B be models of $\mathcal{T}_1 \wedge \phi_1 \wedge \psi$ and $\mathcal{T}_2 \wedge \phi_2 \wedge \psi$.
  - The component theories are stably infinite. So, assume the models are infinite (of same cardinality).
  - Let h be a bijection between S_A and S_B such that $h(x^A) = x^B$ for each shared variable.
  - Extend B to $\bar{B}$ by interpretations of symbols in Σ_1: $f^{\bar{B}}(b_1, \ldots, b_n) = h(f^A(h^{-1}(b_1), \ldots, h^{-1}(b_n)))$
  - *$\bar{B}$ is a model of:* $\mathcal{T}_1 \wedge \phi_1 \wedge \mathcal{T}_2 \wedge \phi_2 \wedge \psi$

NO deterministic procedure

- Instead of *guessing*, we can *deduce* the equalities to be shared.

  Purification: no changes.

  Interaction: Deduce an equality $x = y$:

  $$\mathcal{T}_1 \vdash (\phi_1 \Rightarrow x = y)$$

  Update $\phi_2 := \phi_2 \wedge x = y$. And vice-versa. Repeat until no further changes.

  Component Procedures : Use individual procedures to decide whether ϕ_i is satisfiable.

- Remark: $\mathcal{T}_i \vdash (\phi_i \Rightarrow x = y)$ iff $\phi_i \wedge x \neq y$ is not satisfiable in $\mathcal{T}_i$.

NO deterministic procedure: correctness

- Assume the theories are convex.
  - Suppose ϕ_i is satisfiable.

NO deterministic procedure: correctness

- Assume the theories are convex.
  - Suppose ϕ_i is satisfiable.
  - Let E be the set of equalities $x_j = x_k$ ($j \neq k$) such that, $\mathcal{T}_i \not\models \phi_i \Rightarrow x_j = x_k$.

NO deterministic procedure: correctness

- Assume the theories are convex.
  - Suppose ϕ_i is satisfiable.
  - Let E be the set of equalities $x_j = x_k$ ($j \neq k$) such that, $\mathcal{T}_i \not\models \phi_i \Rightarrow x_j = x_k$.
  - By convexity, $\mathcal{T}_i \not\models \phi_i \Rightarrow \bigvee_E x_j = x_k$.

NO deterministic procedure: correctness

- Assume the theories are convex.
  - Suppose ϕ_i is satisfiable.
  - Let E be the set of equalities $x_j = x_k$ ($j \neq k$) such that, $\mathcal{T}_i \not\models \phi_i \Rightarrow x_j = x_k$.
  - By convexity, $\mathcal{T}_i \not\models \phi_i \Rightarrow \bigvee_E x_j = x_k$.
  - $\phi_i \wedge \bigwedge_E x_j \neq x_k$ is satisfiable.

NO deterministic procedure: correctness

- Assume the theories are convex.
  - Suppose ϕ_i is satisfiable.
  - Let E be the set of equalities $x_j = x_k$ ($j \neq k$) such that, $\mathcal{T}_i \not\models \phi_i \Rightarrow x_j = x_k$.
  - By convexity, $\mathcal{T}_i \not\models \phi_i \Rightarrow \bigvee_E x_j = x_k$.
  - $\phi_i \wedge \bigwedge_E x_j \neq x_k$ is satisfiable.
  - The proof now is identical to the nondeterministic case.

NO deterministic procedure: correctness

- Assume the theories are convex.
  - Suppose ϕ_i is satisfiable.
  - Let E be the set of equalities $x_j = x_k$ ($j \neq k$) such that, $\mathcal{T}_i \not\models \phi_i \Rightarrow x_j = x_k$.
  - By convexity, $\mathcal{T}_i \not\models \phi_i \Rightarrow \bigvee_E x_j = x_k$.
  - $\phi_i \wedge \bigwedge_E x_j \neq x_k$ is satisfiable.
  - The proof now is identical to the nondeterministic case.
  - Sharing equalities is sufficient, because a theory $\mathcal{T}_1$ can assume that $x^B \neq y^B$ whenever $x = y$ is not implied by $\mathcal{T}_2$ and vice versa.

Roadmap

- Background
- Implementing SMT solvers
- Applications

Architecture

- Preprocessor/Simplifier.
- SAT solver.
- Blackboard: “bus” used to connect the theories.
- Theories:
  - Arithmetic,
  - Bit-vectors,
  - Arrays,
  - etc.
- Heuristic quantifier instantiation.

Preprocessor/Simplifier

- Apply simplification rules:
  - Normalization:
    - Sort arguments of commutative operators.
    - Flat associative operators:

      or$(p_1,$ *or*$(p_2, p_3)) \rightsquigarrow$ *or*(p_1, p_2, p_3)
    - Rewrite arithmetic expressions as sums of monomials.

$$x(y+3) = 5 \rightsquigarrow 3x + xy = 5$$

  - Hash-consing.
  - Lift term if-then-else.
  - $x = t \wedge C[x] \rightsquigarrow C[t]$.
  - etc.

Preprocessor/Simplifier

- CNF translation.
- Rewrite formula to simplify atoms that are asserted during the search.
- Example:

$$x \geq 0 \land (x + y \leq 2 \lor x + 2y \geq 6) \land (x + y = 2 \lor x + 2y > 4)$$

$$\rightsquigarrow$$

$$(s_1 = x + y \land s_2 = x + 2y) \ \land$$
$$(x \geq 0 \land (s_1 \leq 2 \lor s_2 \geq 6) \land (s_1 = 2 \lor s_2 > 4))$$

- Only *bounds* (e.g., $s_1 \leq 2$) are asserted during the search.
- *Unconstrained variables* can be *eliminated* before the beginning of the search.

SMT solvers before SAT breakthrough

- Ad-hoc support for boolean combination of literals.
- Ad-hoc support for (non-convex) theories.
- “Case-splits” should be avoided.
- Few real benchmarks.
- *Breakthrough in SAT solving changed everything.*

Breakthrough in SAT solving

- Breakthrough in SAT solving influenced the way SMT solvers are implemented.
- Modern SAT solvers are based on the DPLL algorithm.
- Modern implementations add several sophisticated *search techniques*.
  - Backjumping
  - Learning
  - Restarts
  - Watched literals

The Original DPLL Procedure

- DPLL tries to *build* incrementally a *satisfying truth assignment* M for a CNF formula F.
- M is grown by
  - *deducing* the truth value of a literal from M and F, or
  - *guessing* a truth value.
- If a wrong guess leads to an inconsistency, the procedure *backtracks* and tries the opposite one.

Lazy approach: SAT solvers + Theories

- This approach was independently developed by several groups: CVC (Stanford), ICS (SRI), MathSAT (Univ. Trento, Italy), and Verifun (HP).
- It was motivated also by the breakthroughs in SAT solving.
- SAT solver "manages" the boolean structure, and assigns truth values to the atoms in a formula.
- Efficient theory solvers are used to validate the (partial) assignment produced by the SAT solver.
- When theory solver detects unsatisfiability $\longrightarrow$ a new clause (*lemma*) is created.

SAT solvers + Theories (cont.)

- Example:
  - Suppose the SAT solver assigns $\{x = y \rightarrow \mathsf{T}, y = z \rightarrow \mathsf{T}, f(x) = f(z) \rightarrow \mathsf{F}\}$.
  - Theory solver detects the conflict, and a *lemma* is created $\neg(x = y) \vee \neg(y = z) \vee f(x) = f(z)$.
- Some theory solvers use the "proof" of the conflict to build the lemma.
- Problems in these tools:
  - The *lemmas are imprecise* (not minimal).
  - The theory solver is "passive": *it just detects conflicts*. There is no propagation step.
  - *Backtracking is expensive*, some tools restart from scratch when a conflict is detected.

Blackboard/Bus

- The *Blackboard/Bus* stores the equalities/disequalities known by the solver.
- The set of known equalities is represented as a set of equivalence classes.
  - Union-Find data structure.
- The bus is used to connect the theories.

Combining theories in practice

- *Propagate all implied equalities.*
  - Deterministic Nelson-Oppen.
  - Complete only for convex theories.
  - It may be expensive for some theories.
- *Delayed Theory Combination.*
  - Nondeterministic Nelson-Oppen.
  - Create set of interface equalities ($x = y$) between shared variables.
  - Use SAT solver to guess the partition.
  - Disadvantage: the number of additional equality literals is quadratic in the number of shared variables.

Combining theories in practice (cont.)

- Common to these methods is that they are *pessimistic* about which equalities are propagated.
- *Model-based Theory Combination*
  - *Optimistic approach.*
  - Use a candidate model M_i for one of the theories $\mathcal{T}_i$ and propagate all equalities implied by the candidate model, hedging that other theories will agree.

    $$\textbf{if } M_i \models \mathcal{T}_i \cup \Gamma_i \cup \{u = v\} \textbf{ then } \text{propagate } u = v\ .$$

  - If not, use backtracking to fix the model.
  - It is cheaper to enumerate equalities that are implied in a particular model than of all models.

Model based theory combination: Example

$$x = f(y - 1), f(x) \neq f(y), 0 \leq x \leq 1, 0 \leq y \leq 1$$

Purifying

Model based theory combination: Example

$$x = f(z), f(x) \neq f(y), 0 \leq x \leq 1, 0 \leq y \leq 1, z = y - 1$$

Model based theory combination: Example

$\mathcal{T}_{\mathcal{E}}$			$\mathcal{T}_{\mathcal{A}}$	
Literals	*Eq. Classes*	*Model*	*Literals*	*Model*
$x = f(z)$	$\{x, f(z)\}$	$x^{\mathcal{E}} = *_1$	$0 \leq x \leq 1$	$x^{\mathcal{A}} = 0$
$f(x) \neq f(y)$	$\{y\}$	$y^{\mathcal{E}} = *_2$	$0 \leq y \leq 1$	$y^{\mathcal{A}} = 0$
	$\{z\}$	$z^{\mathcal{E}} = *_3$	$z = y - 1$	$z^{\mathcal{A}} = -1$
	$\{f(x)\}$	$f^{\mathcal{E}} = \{*_1 \mapsto *_4,$		
	$\{f(y)\}$	$*_2 \mapsto *_5,$		
		$*_3 \mapsto *_1,$		
		else $\mapsto *_6\}$		

Assume x = y

Model based theory combination: Example

$\mathcal{T}_{\mathcal{E}}$			$\mathcal{T}_{\mathcal{A}}$	
Literals	*Eq. Classes*	*Model*	*Literals*	*Model*
$x = f(z)$	$\{x, y, f(z)\}$	$x^{\mathcal{E}} = *_1$	$0 \leq x \leq 1$	$x^{\mathcal{A}} = 0$
$f(x) \neq f(y)$	$\{z\}$	$y^{\mathcal{E}} = *_1$	$0 \leq y \leq 1$	$y^{\mathcal{A}} = 0$
$x = y$	$\{f(x), f(y)\}$	$z^{\mathcal{E}} = *_2$	$z = y - 1$	$z^{\mathcal{A}} = -1$
		$f^{\mathcal{E}} = \{*_1 \mapsto *_3,$	$x = y$	
		$*_2 \mapsto *_1,$		
		else $\mapsto *_4\}$		

Unsatisfiable

Model based theory combination: Example

$\mathcal{T}_{\mathcal{E}}$			$\mathcal{T}_{\mathcal{A}}$	
Literals	*Eq. Classes*	*Model*	*Literals*	*Model*
$x = f(z)$	$\{x, f(z)\}$	$x^{\mathcal{E}} = *_1$	$0 \leq x \leq 1$	$x^{\mathcal{A}} = 0$
$f(x) \neq f(y)$	$\{y\}$	$y^{\mathcal{E}} = *_2$	$0 \leq y \leq 1$	$y^{\mathcal{A}} = 0$
$x \neq y$	$\{z\}$	$z^{\mathcal{E}} = *_3$	$z = y - 1$	$z^{\mathcal{A}} = -1$
	$\{f(x)\}$	$f^{\mathcal{E}} = \{*_1 \mapsto *_4,$	$x \neq y$	
	$\{f(y)\}$	$*_2 \mapsto *_5,$		
		$*_3 \mapsto *_1,$		
		else $\mapsto *_6\}$		

Backtrack, and assert $x \neq y$.

$\mathcal{T}_{\mathcal{A}}$ model need to be fixed.

Model based theory combination: Example

$\mathcal{T}_{\mathcal{E}}$			$\mathcal{T}_{\mathcal{A}}$	
Literals	*Eq. Classes*	*Model*	*Literals*	*Model*
$x = f(z)$	$\{x, f(z)\}$	$x^{\mathcal{E}} = *_1$	$0 \leq x \leq 1$	$x^{\mathcal{A}} = 0$
$f(x) \neq f(y)$	$\{y\}$	$y^{\mathcal{E}} = *_2$	$0 \leq y \leq 1$	$y^{\mathcal{A}} = 1$
$x \neq y$	$\{z\}$	$z^{\mathcal{E}} = *_3$	$z = y - 1$	$z^{\mathcal{A}} = 0$
	$\{f(x)\}$	$f^{\mathcal{E}} = \{*_1 \mapsto *_4,$	$x \neq y$	
	$\{f(y)\}$	$*_2 \mapsto *_5,$		
		$*_3 \mapsto *_1,$		
		else $\mapsto *_6\}$		

Assume $x = z$

Model based theory combination: Example

$\mathcal{T}_\mathcal{E}$			$\mathcal{T}_\mathcal{A}$	
Literals	*Eq. Classes*	*Model*	*Literals*	*Model*
$x = f(z)$	$\{x, z, f(x), f(z)\}$	$x^\mathcal{E} = *_1$	$0 \leq x \leq 1$	$x^\mathcal{A} = 0$
$f(x) \neq f(y)$	$\{y\}$	$y^\mathcal{E} = *_2$	$0 \leq y \leq 1$	$y^\mathcal{A} = 1$
$x \neq y$	$\{f(y)\}$	$z^\mathcal{E} = *_1$	$z = y - 1$	$z^\mathcal{A} = 0$
$x = z$		$f^\mathcal{E} = \{*_1 \mapsto *_1,$	$x \neq y$	
		$*_2 \mapsto *_3,$	$x = z$	
		else $\mapsto *_4\}$		

Satisfiable

Model based theory combination: Example

$\mathcal{T}_\mathcal{E}$			$\mathcal{T}_\mathcal{A}$	
Literals	*Eq. Classes*	*Model*	*Literals*	*Model*
$x = f(z)$	$\{x, z, f(x), f(z)\}$	$x^\mathcal{E} = *_1$	$0 \leq x \leq 1$	$x^\mathcal{A} = 0$
$f(x) \neq f(y)$	$\{y\}$	$y^\mathcal{E} = *_2$	$0 \leq y \leq 1$	$y^\mathcal{A} = 1$
$x \neq y$	$\{f(y)\}$	$z^\mathcal{E} = *_1$	$z = y - 1$	$z^\mathcal{A} = 0$
$x = z$		$f^\mathcal{E} = \{*_1 \mapsto *_1,$	$x \neq y$	
		$*_2 \mapsto *_3,$	$x = z$	
		else $\mapsto *_4\}$		

Let h be the bijection between $S_\mathcal{E}$ and $S_\mathcal{A}$.

$$h = \{*_1 \mapsto 0, *_2 \mapsto 1, *_3 \mapsto -1, *_4 \mapsto 2, \ldots\}$$

Model based theory combination: Example

$\mathcal{T}_{\mathcal{E}}$		$\mathcal{T}_{\mathcal{A}}$	
Literals	*Model*	*Literals*	*Model*
$x = f(z)$	$x^{\mathcal{E}} = *_1$	$0 \leq x \leq 1$	$x^{\mathcal{A}} = 0$
$f(x) \neq f(y)$	$y^{\mathcal{E}} = *_2$	$0 \leq y \leq 1$	$y^{\mathcal{A}} = 1$
$x \neq y$	$z^{\mathcal{E}} = *_1$	$z = y - 1$	$z^{\mathcal{A}} = 0$
$x = z$	$f^{\mathcal{E}} = \{*_1 \mapsto *_1,$	$x \neq y$	$f^{\mathcal{A}} = \{0 \mapsto 0$
	$*_2 \mapsto *_3,$	$x = z$	$1 \mapsto -1$
	else $\mapsto *_4\}$		*else* $\mapsto 2\}$

Extending $\mathcal{A}$ using h.

$$h = \{*_1 \mapsto 0, *_2 \mapsto 1, *_3 \mapsto -1, *_4 \mapsto 2, \ldots\}$$

Simplex: a model base theory solver

- Tableau: $\mathcal{B}$ and $\mathcal{N}$ denote the set of basic and nonbasic variables.

$$x_i = \sum_{x_j \in \mathcal{N}} a_{ij} x_j \quad x_i \in \mathcal{B},$$

- Solver stores upper and lower bounds l_i and u_i, and a mapping β that assigns a value $\beta(x_i)$ to every variable.
- The bounds on nonbasic variables are always satisfied by β, that is, the following invariant is maintained

$$\forall x_j \in \mathcal{N}, \;\; l_j \leq \beta(x_j) \leq u_j.$$

- Bounds constraints for basic variables are not necessarily satisfied by β, but pivoting steps can be used to fix bounds violations.

Simplex: a model based theory solver

- The current model for the simplex solver is given by β.
- *Bound propagation*
  - *Equations + Bounds* can be used to derive *new bounds*.
  - Example: $x = y - z,\ y \leq 2,\ z \geq 3 \rightsquigarrow x \leq -1$.

Opportunistic equality propagation

- Efficient (and incomplete) methods for propagating equalities.
- Notation
  - A variable x_i is *fixed* iff $l_i = u_i$.
  - A linear polynomial $\sum_{x_j \in \mathcal{V}} a_{ij} x_j$ is fixed iff x_j is fixed or $a_{ij} = 0$.
  - Given a linear polynomial $P = \sum_{x_j \in \mathcal{V}} a_{ij} x_j$, $\beta(P)$ denotes $\sum_{x_j \in \mathcal{V}} a_{ij} \beta(x_j)$.

Opportunistic equality propagation

- Equality propagation in arithmetic:

FixedEq

$$l_i \leq x_i \leq u_i,\ \ l_j \leq x_j \leq u_j \Longrightarrow \quad x_i = x_j \ \ \textbf{if} \ \ l_i = u_i = l_j = u_j$$

EqRow

$$x_i = x_j + P \qquad \Longrightarrow \quad x_i = x_j \ \ \textbf{if} \ \ P \text{ is fixed, and } \beta(P) = 0$$

EqOffsetRows

$$\begin{array}{l} x_i = x_k + P_1 \\ x_j = x_k + P_2 \end{array} \qquad \Longrightarrow \quad x_i = x_j \ \ \textbf{if} \ \ \left\{ \begin{array}{l} P_1 \text{ and } P_2 \text{ are fixed, and} \\ \beta(P_1) = \beta(P_2) \end{array} \right.$$

EqRows

$$\begin{array}{l} x_i = P + P_1 \\ x_j = P + P_2 \end{array} \qquad \Longrightarrow \quad x_i = x_j \ \ \textbf{if} \ \ \left\{ \begin{array}{l} P_1 \text{ and } P_2 \text{ are fixed, and} \\ \beta(P_1) = \beta(P_2) \end{array} \right.$$

Opportunistic theory/equality propagation

- These rules can miss some implied equalities.
- Example: $z = w$ is detected, but $x = y$ is not because w is not a fixed variable.

$$\begin{array}{rcl} x & = & y + w + s \\ z & = & w + s \\ 0 & \leq & z \\ & & w \leq 0 \\ 0 & \leq & s \leq 0 \end{array}$$

- Remark: bound propagation can be used imply the bound $0 \leq w$, making w a fixed variable.

Non Stably-Infinite Theories in practice

- Bit-vector theory is not stably-infinite.
- How can we support it?
- *Solution:* add a predicate *is-bv*(x) to the bit-vector theory (intuition: *is-bv*(x) is true iff x is a bitvector).
- The result of the bit-vector operation *op*(x, y) is not specified if $\neg$*is-bv*(x) or $\neg$*is-bv*(y).
- *The new bit-vector theory is stably-infinite.*

Precise Lemmas

- Lemma:

  $\{a_1 = \mathit{T}, a_1 = \mathit{F}, a_3 = \mathit{F}\}$*is inconsistent* $\leadsto \neg a_1 \vee a_2 \vee a_3$

- An inconsistent A set is *redundant* if $A' \subset A$ is also inconsistent.
- Redundant inconsistent sets $\leadsto$ Imprecise Lemmas $\leadsto$ Ineffective pruning of the search space.
- *Noise* of a redundant set: $A \setminus A_{min}$.
- The imprecise lemma is *useless* in any context (partial assignment) where an atom in the noise has a different assignment.
- Example: suppose a_1 is in the noise, then $\neg a_1 \vee a_2 \vee a_3$ is useless when $a_1 = F$.

Precise Lemmas

- Simple approach: track dependencies.
- Record the antecedents $\psi_1, \ldots, \psi_n$ of a consequent ϕ.
- It is the same approach used in SAT solvers:

  Record the clause $C \vee l$ used to imply a literal l.

- It may be imprecise.

Precise Lemmas: simple approach

- Example: assume equations (1), (2) and (3) were asserted into the logical context.

$$
\begin{aligned}
x + w + 3 &= 0 && (1)\\
x + z + 1 &= 0 && (2)\\
x + y + 1 &= 0 && (3)
\end{aligned}
$$

Precise Lemmas: simple approach

- Example: assume equations (1), (2) and (3) were asserted into the logical context.

$$
\begin{array}{rcll}
x + w + 3 & = & 0 & (1) \\
x + z + 1 & = & 0 & (2) \\
x + y + 1 & = & 0 & (3) \\
\hline
-w + z - 2 & = & 0 & (4) = (2) - (1)
\end{array}
$$

Precise Lemmas: simple approach

- Example: assume equations (1), (2) and (3) were asserted into the logical context.

$$
\begin{array}{rcll}
{\color{red}x} + w + 3 & = & 0 & (1) \\
x + z + 1 & = & 0 & (2) \\
{\color{red}x} + y + 1 & = & 0 & (3) \\
\hline
-w + z - 2 & = & 0 & (4) = (2) - (1) \\
-w + y - 2 & = & 0 & (5) = (3) - (1)
\end{array}
$$

Precise Lemmas: simple approach

- Example: assume equations (1), (2) and (3) were asserted into the logical context.

$$
\begin{array}{rcll}
x + w + 3 & = & 0 & (1) \\
x + z + 1 & = & 0 & (2) \\
x + y + 1 & = & 0 & (3) \\
\hline
-w + z - 2 & = & 0 & (4) = (2) - (1) \\
-w + y - 2 & = & 0 & (5) = (3) - (1) \\
y - z & = & 0 & (6) = (5) - (4)
\end{array}
$$

Precise Lemmas: simple approach

- Example: assume equations (1), (2) and (3) were asserted into the logical context.

$$
\begin{array}{rclr}
x + w + 3 & = & 0 & (1) \\
x + z + 1 & = & 0 & (2) \\
x + y + 1 & = & 0 & (3) \\
\hline
-w + z - 2 & = & 0 & (4) = (2) - (1) \\
-w + y - 2 & = & 0 & (5) = (3) - (1) \\
y - z & = & 0 & (6) = (5) - (4)
\end{array}
$$

- Equation (6) implies that $y = z$. *It depends on (1), (2), and (3)*.

Precise Lemmas: simple approach

- Example: assume equations (1), (2) and (3) were asserted into the logical context.

$$
\begin{array}{rcll}
x + w + 3 & = & 0 & (1) \\
x + z + 1 & = & 0 & (2) \\
x + y + 1 & = & 0 & (3) \\
\hline
-w + z - 2 & = & 0 & (4) = (2) - (1) \\
-w + y - 2 & = & 0 & (5) = (3) - (1) \\
y - z & = & 0 & (6) = (5) - (4)
\end{array}
$$

- Equation (6) implies that $y = z$. *It depends on (1), (2), and (3)*.
- *Equation (1) is not necessary to derive $y = z$.*

Precise Lemmas: auxiliary variables

- Use *auxiliary/zero variables* to "name" linear polynomials.

$$
\begin{array}{rcl}
x + w + 3 & = & s_1 \\
x + z + 1 & = & s_2 \\
x + y + 1 & = & s_3 \\
\hline
\end{array}
$$

Precise Lemmas: auxiliary variables

- Use *auxiliary/zero variables* to “name” linear polynomials.

$$
\begin{array}{rcl}
x + w + 3 & = & s_1 \\
x + z + 1 & = & s_2 \\
x + y + 1 & = & s_3 \\
\hline
-w + z - 2 & = & s_2 - s_1
\end{array}
$$

Precise Lemmas: auxiliary variables

- Use *auxiliary/zero variables* to "name" linear polynomials.

$$
\begin{array}{rcl}
x + w + 3 & = & s_1 \\
x + z + 1 & = & s_2 \\
x + y + 1 & = & s_3 \\
\hline
-w + z - 2 & = & s_2 - s_1 \\
-w + y - 2 & = & s_3 - s_1
\end{array}
$$

Precise Lemmas: auxiliary variables

- Use *auxiliary/zero variables* to “name” linear polynomials.

$$
\begin{array}{rcl}
x + w + 3 & = & s_1 \\
x + z + 1 & = & s_2 \\
x + y + 1 & = & s_3 \\
\hline
-w + z - 2 & = & s_2 - s_1 \\
-w + y - 2 & = & s_3 - s_1 \\
y - z & = & s_3 - s_1 - s_2 + s_1
\end{array}
$$

Precise Lemmas: auxiliary variables

- Use *auxiliary/zero variables* to “name” linear polynomials.

$$
\begin{array}{rcl}
x + w + 3 & = & s_1 \\
x + z + 1 & = & s_2 \\
x + y + 1 & = & s_3 \\
\hline
-w + z - 2 & = & s_2 - s_1 \\
-w + y - 2 & = & s_3 - s_1 \\
y - z & = & s_3 - s_2
\end{array}
$$

- The last equation implies $y = z$ when s_2 and s_3 are equal to 0.

Precise Lemmas: auxiliary variables

- Use *auxiliary/zero variables* to “name” linear polynomials.

$$
\begin{array}{rcl}
x + w + 3 & = & s_1 \\
x + z + 1 & = & s_2 \\
x + y + 1 & = & s_3 \\
\hline
-w + z - 2 & = & s_2 - s_1 \\
-w + y - 2 & = & s_3 - s_1 \\
y - z & = & s_3 - s_2
\end{array}
$$

- The last equation implies $y = z$ when s_2 and s_3 are equal to 0.
- This is the approach used in the Simplex based solver.
- A similar approach is used to implement incremental SAT solvers.

Precise "Explanations"

- What is the "explanation" for the implied equality below?

Precise "Explanations"

- What is the "explanation" for the implied equality below?

EqOffsetRows

$$\begin{aligned} x_i &= x_k + P_1 \\ x_j &= x_k + P_2 \end{aligned} \quad \Longrightarrow \quad x_i = x_j \;\; \textbf{if} \;\; \begin{cases} P_1 \text{ and } P_2 \text{ are fixed, and} \\ \beta(P_1) = \beta(P_2) \end{cases}$$

Precise "Explanations"

- What is the "explanation" for the implied equality below?

EqOffsetRows

$$\begin{array}{l} x_i = x_k + P_1 \\ x_j = x_k + P_2 \end{array} \quad \Longrightarrow \quad x_i = x_j \ \textbf{if} \ \begin{cases} P_1 \text{ and } P_2 \text{ are fixed, and} \\ \beta(P_1) = \beta(P_2) \end{cases}$$

- *Explanation: P_1 and P_2 are fixed and $\beta(P_1) = \beta(P_2)$.*

Precise "Explanations"

- What is the "explanation" for the implied equality below?

EqOffsetRows

$$\begin{aligned} x_i &= x_k + P_1 \\ x_j &= x_k + P_2 \end{aligned} \quad \Longrightarrow \quad x_i = x_j \ \textbf{if} \ \begin{cases} P_1 \text{ and } P_2 \text{ are fixed, and} \\ \beta(P_1) = \beta(P_2) \end{cases}$$

- *Explanation: P_1 and P_2 are fixed and $\beta(P_1) = \beta(P_2)$.*
- The union of the explanations for the lower and upper bounds of $x \in \mathit{vars}(P_1) \cup \mathit{vars}(P_2)$.

Precise "Explanations"

- What is the "explanation" for the implied equality below?

EqOffsetRows

$$\begin{array}{l} x_i = x_k + P_1 \\ x_j = x_k + P_2 \end{array} \implies x_i = x_j \ \textbf{if} \ \left\{ \begin{array}{l} P_1 \text{ and } P_2 \text{ are fixed, and} \\ \beta(P_1) = \beta(P_2) \end{array} \right.$$

- *Explanation: P_1 and P_2 are fixed and $\beta(P_1) = \beta(P_2)$.*
- The union of the explanations for the lower and upper bounds of $x \in$ *vars*$(P_1) \cup$ *vars*(P_2).
- *Valley proof problem*. Example: arithmetic propagated $x_1 = x_2$ and $x_1 = x_3$ using the rule above.

Precise "Explanations"

- What is the "explanation" for the implied equality below?

EqOffsetRows

$$\begin{array}{l} x_i = x_k + P_1 \\ x_j = x_k + P_2 \end{array} \quad \Longrightarrow \quad x_i = x_j \ \textbf{if} \ \begin{cases} P_1 \text{ and } P_2 \text{ are fixed, and} \\ \beta(P_1) = \beta(P_2) \end{cases}$$

- *Explanation: P_1 and P_2 are fixed and $\beta(P_1) = \beta(P_2)$.*
- The union of the explanations for the lower and upper bounds of $x \in \mathit{vars}(P_1) \cup \mathit{vars}(P_2)$.
- *Valley proof problem*. Example: arithmetic propagated $x_1 = x_2$ and $x_1 = x_3$ using the rule above.
- *What is the "explanation" for $x_2 = x_3$?*

Efficient Backtracking

- One of the most important improvements in SAT was efficient backtracking.
- Until recently, backtracking was ignored in the design of theory solvers.
- Extreme (inefficient) approach: restart from scratch on every conflict.
- Other approaches:
  - Functional data-structures.
  - Backtrackable data-structures
  - Trail-stack.
- *Restore to a logically equivalent state.*

Reduction Functions

- A *reduction function* reduces the satisfiability problem for a theory $\mathcal{T}_1$ to the satisfiability problem of a simpler theory $\mathcal{T}_2$.
- Reduction functions simplify the implementation.
- Potential disadvantages:
  - "Information loss".
  - Eager addition of irrelevant information.
- Theory of commutative functions.
  - Deductive closure of: $\forall x, y. f(x, y) = f(y, x)$
  - Reduction to $\mathcal{T}_{\mathcal{E}}$.
  - For every $f(a, b)$ in ϕ, add the equality $f(a, b) = f(b, a)$.

Reduction Functions: Ackermann's reduction

- *Ackermann's reduction* is used to remove uninterpreted functions.
  - For each application $f(\vec{a})$ in ϕ create a fresh variable $f_{\vec{a}}$.
  - For each pair of applications $f(\vec{a})$, $f(\vec{c})$ in ϕ add the clause $\vec{a} \neq \vec{c} \vee f_{\vec{a}} = f_{\vec{c}}$.
  - Replace $f(\vec{a})$ with $f_{\vec{a}}$ in ϕ.
- It is used in some SMT solvers to reduce $\mathcal{T}_{\mathcal{LA}} \cup \mathcal{T}_{\mathcal{E}}$ to $\mathcal{T}_{\mathcal{LA}}$.
- *Main problem: quadratic number of new clauses.*
- It is also problematic to use this approach in the context of *several theories* and when combining SMT solvers with *quantifier instantiation*.

Reduction Functions: Ackermann's reduction

- Congruence closure based algorithms miss the following inference rule

$$f(\overline{n}) \neq f(\overline{m}) \implies \bigvee n_i \neq m_i$$

- Following simple formula takes $\mathcal{O}(2^N)$ time to be solved using SAT + Congruence closure.

$$\bigwedge_{i=1}^{N} (p_i \vee x_i = v_0),\ (\neg p_i \vee x_i = v_1),\ (p_i \vee y_i = v_0),\ (\neg p_i \vee y_i = v_1),$$
$$f(x_N, \ldots, f(x_2, x_1) \ldots) \neq f(y_N, \ldots, f(y_2, y_1) \ldots)$$

- It can be solved in polynomial time with Ackermann's reduction.
- A similar behavior is also observed in several pipeline verification problems.

Dynamic Ackermann's reduction

- This performance problem reflects a limitation in the current congruence closure algorithms used in SMT solvers.
- It is not related with the theory combination problem.
- *Dynamic Ackermannization*: clauses corresponding to Ackermann's reduction are added when a congruence rule participates in a conflict.

	CC		Ack		Dyn Ack	
	conflicts	time (s)	conflicts	time (s)	conflicts	time (s)
c10bi	217232	143.87	6880	6.09	5885	1.75
f10id	$>$ 8752181	$>$ 1800	22038	16.20	21220	7.20

Modularity issues

- Modular implementations are attractive.
- *Potential problem:* theories fail to share relevant information.
  - *Arithmetic:* $i = s + 1, j = s + 2$
  - *Array theory:*

    $v_1 = \textit{read}(\textit{write}(a_0, i, v_0), j), v_2 = \textit{read}(a_0, j)$.
  - Arithmetic implies $i \neq j$. If this disequality is shared with array theory, then $v_1 = v_2$.
- It is infeasible to propagate all implied disequalities.
- *Blackboard solution:*
  - Theories post on the blackboard the equations they are "interested".

Delaying inference rules

- A commonly used approach: delay the application of “expensive” inference rules.
- Examples:
  - Inference rules that produce new case-splits.
  - Non-linear arithmetic.
- Potential problem: solver may waste time searching an infeasible part of the search space.

Quantifiers

- Since first-order logic is undecidable, satisfiability is not solvable for arbitrary quantified formulas.
- Some theories, e.g., datatypes, linear arithmetic over integers, arithmetic over reals, support quantifier elimination.
- Existential quantifiers can be skolemized, but the problem of instantiating universal quantifiers for detecting unsatisfiability remains.

Heuristic Quantifier Instantiation

- Semantically, $\forall x_1, \ldots, x_n.F$ is equivalent to the infinite conjunction $\bigwedge_\beta \beta(F)$.
- Solvers use heuristics to select from this infinite conjunction those instances that are "relevant".
- The key idea is to treat an instance $\beta(F)$ as relevant whenever it contains enough terms that are represented in the solver state.
- Non ground terms p from F are selected as *patterns*.
- *E-matching* (matching modulo equalities) is used to find instances of the patterns.
- Example: $f(a, b)$ matches the pattern $f(g(x), x)$ if $a = g(b)$.

E-matching

- E-matching is NP-hard.
- The number of matches can be exponential.
- It is not refutationally complete.
- In practice:
  - Indexing techniques for fast retrieval.
  - Incremental E-matching.

E-matching: example

- $\forall x. f(g(x)) = x$
- Pattern: $f(g(x))$
- Atoms: $a = g(b), b = c, f(a) \neq c$
- $\longrightarrow$ *instantiate* $f(g(b)) = b$

Quantifiers in Z3

- Z3 uses a E-matching abstract machine.
  - Patterns $\rightsquigarrow$ code sequence.
  - Abstract machine executes the code.
- *Z3 uses new algorithms that identify matches on E-graphs incrementally and efficiently.*
  - E-matching code trees.
  - Inverted path index.
- Z3 garbage collects clauses, together with their atoms and terms, that were useless in closing branches.

E-matching code trees

- In practice, there are several similar patterns.
- *Idea: combine several code sequences in a code tree.*
- Factor out redundant work.
- Match several patterns simultaneously.
- Saturation based theorem provers use a different kind of code tree to implement:
  - Forward subsumption.
  - Forward demodulation.

Incremental E-matching

- Z3 uses a backtracking search.
- *New terms are created during the search.*
  - A code tree for each function symbol f.

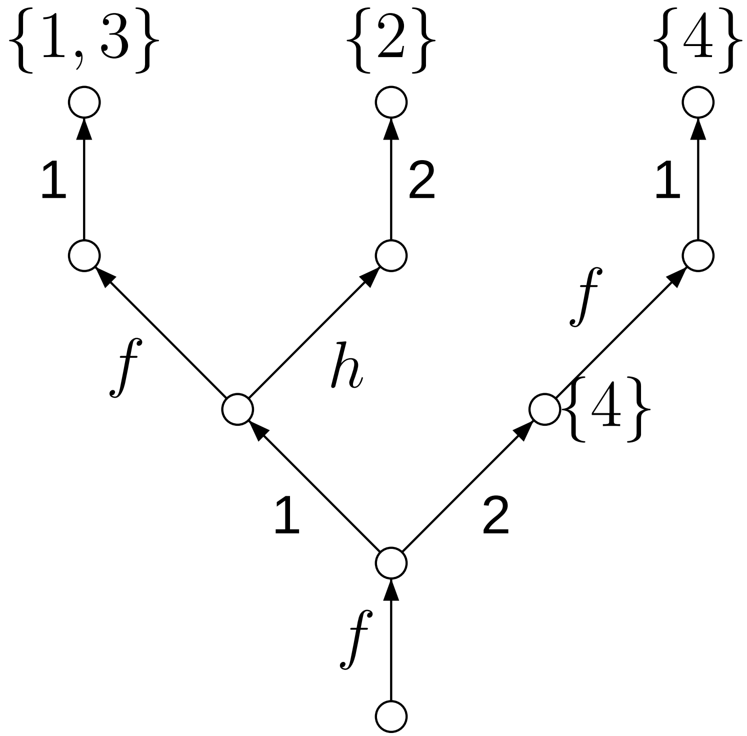
    Patterns that start with a f-application.
  - Execute code-tree for each new term.
- *New equalities are asserted during the search.*
  - New equalities $\rightsquigarrow$ new E-matching instances.
  - Example:

    $f(a, b)$ matches $f(g(x), x)$ after

    $a = g(b)$ is asserted.

Inverted path index

- It is used to find which patterns may have new instances after an equality is asserted.
- Inverted path index for pc-pair (f, g) and patterns $f(f(g(x), a), x), h(c, f(g(y), x)), f(f(g(x), b), y),$ $f(f(a, g(x)), g(y))$.



E-matching limitations

- E-matching needs ground (seed) terms.
  - It fails to prove simple properties when ground (seed) terms are not available.
  - Example:

$$(\forall x. f(x) \leq 0) \land (\forall x. f(x) > 0)$$

- Matching loops

$$(\forall x. f(x) = g(f(x))) \land (\forall x. g(x) = f(g(x)))$$

  - Inefficiency and/or non-termination.
  - Some solvers have support for detecting matching loops based on instantiation chain length.

Quantifiers: future work

- Model checking.
- Superposition calculus + SMT.
- Decidable fragments.

Roadmap

- Background
- Architecture
- Applications

Spec#: Extended Static Checking

- `http://research.microsoft.com/specsharp/`
- Superset of C#
  - non-null types
  - pre- and postconditions
  - object invariants
- Static program verification
- Example:

```
public StringBuilder Append(char[] value, int startIndex,
                            int charCount);
 requires value == null ==> startIndex == 0 && charCount == 0;
 requires 0 <= startIndex;
 requires 0 <= charCount;
 requires value == null ||
          startIndex + charCount <= value.Length;
```

Spec#: Architecture

- Verification condition generation:

  Spec# compiler: Spec# $\rightsquigarrow$ MSIL (bytecode).

  Bytecode translator: MSIL $\rightsquigarrow$ Boogie PL.

  V.C. generator: Boogie PL $\rightsquigarrow$ SMT formula.

- SMT solver is used to prove the verification conditions.
- Counterexamples are traced back to the source code.
- *The formulas produces by Spec# are not quantifier free.*

SLAM: device driver verification

- `http://research.microsoft.com/slam/`
- *SLAM/SDV* is a software model checker.
- Application domain: *device drivers*.
- Architecture

  c2bp C program $\rightsquigarrow$ boolean program (*predicate abstraction*).

  bebop Model checker for boolean programs.

  newton Model refinement (*check for path feasibility*)
- SMT solvers are used to perform predicate abstraction and to check path feasibility.
- c2bp makes several calls to the SMT solver. The formulas are relatively small.

MUTT: MSIL Unit Testing Tools

- `http://research.microsoft.com/projects/mutt`
- *Unit tests are popular*, but it is far from trivial to write them.
- It is quite laborious to write enough of them to have confidence in the correctness of an implementation.
- Approach: *symbolic execution*.
- Symbolic execution builds a path condition over the input symbols.
- A *path condition* is a mathematical formula that encodes data constraints that result from executing a given code path.

MUTT: MSIL Unit Testing Tools

- When symbolic execution reaches a if-statement, it will explore two execution paths:
  1. The if-condition is conjoined to the path condition for the then-path.
  2. The negated condition to the path condition of the else-path.
- SMT solver must be able to produce models.
- SMT solver is also used to test path *feasibility*.

Conclusion

- SMT is the next generation of verification engines.
- More automation: it is push-button technology.
- SMT solvers are used in different applications.
- The breakthrough in SAT solving influenced the new generation of SMT solvers:
  - Precise lemmas.
  - Theory Propagation.
  - Incrementality.
  - Efficient Backtracking.
- Z3 website:
  `http://research.microsoft.com/projects/z3`