SMT Solvers: Theory and Implementation

Summer School on Logic and Theorem Proving

Leonardo de Moura

leonardo@microsoft.com

Microsoft Research

Overview

- Satisfiability is the problem of determining if a formula has a model.
- In the purely Boolean case, a model is a truth assignment to the Boolean variables.
- In the first-order case, a model assigns values from a domain to variables and interpretations over the domain to the function and predicate symbols.
- For theories such arithmetic, a model admits a specific (range of) interpretation to the arithmetic symbols.
- Efficient SAT and SMT solvers have many applications.

Applications

- Extended Static Checking.
  - Spec#, VCC, HAVOC
  - ESC/Java
- Predicate Abstraction.
  - SLAM/SDV (device driver verification).
- Test-case generation.
  - Pex, Sage
- Bounded Model Checking (BMC) & k-induction.
- Symbolic Simulation.
- Planning & Scheduling.
- Equivalence checking.

SMT-Solvers & SMT-Lib & SMT-Comp

- SMT-Solvers:

  Alt-Ergo, Ario, Barcelogic, Beaver, Boolector, CVC, CVC Lite, CVC3, DPT (Intel), ExtSAT, Harvey, HTP, ICS (SRI), Jat, MathSAT, OpenSMT, Sateen, Simplify, Spear, STeP, STP, SVC, Sword, TSAT, UCLID, Yices (SRI), Zap, Zapato, Z3 (Microsoft)

- SMT-Lib: library of benchmarks

  `http://www.smtlib.org`

- SMT-Comp: annual SMT-Solver competition

  `http://www.smtcomp.org`

Goals

- This tutorial covers pragmatic issues in the theory, implementation, and use of SMT solvers.
- It is not a comprehensive survey, but a basic and rigorous introduction to some of the key ideas.
- It is not directed at experts but at potential users and developers of SMT solvers.

Roadmap

- Background
- SAT & SMT
- Equality
- Arithmetic
- Combining theories
- Quantifiers
- Applications

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, 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 not feasible, so proofs in the form axioms and inference rules are used to demonstrate the validity of assertions.

Language: Signatures

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

Language: Terms

- The set $T(\Sigma, \mathcal{V})$ of terms is the smallest set such that:
  - $\mathcal{V} \subset T(\Sigma, \mathcal{V})$
  - $f(t_1, \ldots, t_n) \in T(\Sigma, \mathcal{V})$ whenever $f \in \Sigma_F, t_1, \ldots, t_n \in T(\Sigma, \mathcal{V})$ and *arity*$(f) = n$.
- The set of ground terms is defined as $T(\Sigma, \emptyset)$.

Language: Atomic Formulas

- $p(t_1, \ldots, t_n)$ is an atomic formula whenever $p \in \Sigma_P$, *arity*$(p) = n$, and $t_1, \ldots, t_n \in T(\Sigma, \mathcal{V})$.
- true and false are atomic formulas.
- If $t_1, \ldots, t_n$ are ground terms, then $p(t_1, \ldots, t_n)$ is called a ground (atomic) formula.
- We assume that the binary predicate $=$ is present in Σ_P.
- A literal is an atomic formula or its negation.

Language: Quantifier Free Formulas

- The set $QFF(\Sigma, \mathcal{V})$ of quantifier free formulas is the smallest set such that:
  - Every atomic formulas is in $QFF(\Sigma, \mathcal{V})$.
  - If $\phi \in QFF(\Sigma, \mathcal{V})$, then $\neg\phi \in QFF(\Sigma, \mathcal{V})$.
  - If $\phi_1, \phi_2 \in QFF(\Sigma, \mathcal{V})$, then

$$\begin{aligned} \phi_1 \wedge \phi_2 &\in QFF(\Sigma, \mathcal{V}) \\ \phi_1 \vee \phi_2 &\in QFF(\Sigma, \mathcal{V}) \\ \phi_1 \Rightarrow \phi_2 &\in QFF(\Sigma, \mathcal{V}) \\ \phi_1 \Leftrightarrow \phi_2 &\in QFF(\Sigma, \mathcal{V}) \end{aligned}$$

Language: Formulas

- The set of first-order formulas is the closure of $QFF(\Sigma, \mathcal{V})$ under existential ($\exists$) and universal ($\forall$) quantification.
- 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.

Models (Semantics)

- A model M is defined as:
  - Domain $|M|$: set of elements.
  - Interpretation $M(f) : |M|^n \mapsto |M|$ for each $f \in \Sigma_F$ with *arity*$(f) = n$.
  - Interpretation $M(p) \subseteq |M|^n$ for each $p \in \Sigma_P$ with *arity*$(p) = n$.
  - Assignment $M(x) \in |M|$ 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 $|M|$.

Interpreting Terms

$$
\begin{aligned}
M[\![x]\!] &= M(x) \\
M[\![f(a_1, \ldots, a_n)]\!] &= M(f)(M[\![a_1]\!], \ldots, M[\![a_n]\!])
\end{aligned}
$$

Interpreting Formulas

The interpretation of a formula F in M, $M[\![F]\!]$, is defined as

$$
\begin{aligned}
M \models a = b &\iff M[\![a]\!] = M[\![b]\!] \\
M \models p(a_1, \ldots, a_n) &\iff \langle M[\![a_1]\!], \ldots, M[\![a_n]\!] \rangle \in M(p) \\
M \models \neg\psi &\iff M \not\models \psi \\
M \models \psi_1 \vee \psi_2 &\iff M \models \psi_1 \text{ or } M \models \psi_2 \\
M \models \psi_1 \wedge \psi_2 &\iff M \models \psi_1 \text{ and } M \models \psi_2 \\
M \models (\forall x : \psi) &\iff M\{x \mapsto a\} \models \psi, \text{ for all } a \in |M| \\
M \models (\exists x : \psi) &\iff M\{x \mapsto a\} \models \psi, \text{ for some } a \in |M|
\end{aligned}
$$

Interpretation Example

$$
\begin{aligned}
\Sigma &= \{0, +, <\}, \text{ and } M \text{ such that } |M| = \{a, b, c\} \\
M(0) &= a, \\
M(+) &= \{\langle a, a \mapsto a\rangle, \langle a, b \mapsto b\rangle, \langle a, c \mapsto c\rangle, \langle b, a \mapsto b\rangle, \langle b, b \mapsto c\rangle, \\
&\quad \langle b, c \mapsto a\rangle, \langle c, a \mapsto c\rangle, \langle c, b \mapsto a\rangle, \langle c, c \mapsto b\rangle\} \\
M(<) &= \{\langle a, b\rangle, \langle a, c\rangle, \langle b, c\rangle\}
\end{aligned}
$$

If $M(x) = a, M(y) = b, M(z) = c$, then

$$M[\![+(+(x, y), z)]\!] =$$

$$M(+)(M(+)(M(x), M(y)), M(z)) = M(+)(M(+)(a, b), c) =$$

$$M(+)(b, c) = a$$

Interpretation Example

$$
\begin{aligned}
\Sigma &= \{0, +, <\}, \text{ and } M \text{ such that } |M| = \{a, b, c\} \\
M(0) &= a, \\
M(+) &= \{\langle a, a \mapsto a\rangle, \langle a, b \mapsto b\rangle, \langle a, c \mapsto c\rangle, \langle b, a \mapsto b\rangle, \langle b, b \mapsto c\rangle, \\
&\quad \langle b, c \mapsto a\rangle, \langle c, a \mapsto c\rangle, \langle c, b \mapsto a\rangle, \langle c, c \mapsto b\rangle\} \\
M(<) &= \{\langle a, b\rangle, \langle a, c\rangle, \langle b, c\rangle\}
\end{aligned}
$$

$$M \models (\forall x : (\exists y : +(x, y) = 0))$$

$$M \not\models (\forall x : (\exists y : x < y))$$

$$M \models (\forall x : (\exists y : +(x, y) = x))$$

Validity

- A formula F is satisfiable if there is an interpretation M such that $M \models F$.
- Otherwise, the formula F is unsatisfiable.
- If a formula is satisfiable, so is its existential closure $\exists \vec{x} : F$, where $\vec{x}$ is *vars*(F), the set of free variables in F.
- If a formula F is unsatisfiable, then the negation of its existential closure $\neg \exists \vec{x} : F$ is valid.

Theories

- A (first-order) theory $\mathcal{T}$ (over a signature Σ) is a set of (deductively closed) sentences (over Σ and $\mathcal{V}$).
- Let $DC(\Gamma)$ 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).

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.

Roadmap

- Background
- SAT & SMT
- Combining theories
- Equality
- Arithmetic
- Quantifiers
- Applications

Clausal (CNF) Form

In clausal form, the formula is a set (conjunction) of clauses $\bigwedge_i C_i$, and each clause C_i is a disjunction of literals. A literal is an atom or the negation of an atom.

$$p_1 \vee \neg p_2, \quad \neg p_1 \vee p_2 \vee p_3, \quad p_3$$

Most SAT solvers assume the formula is in CNF.

Naïve translation to CNF is too expensive.

Conversion to Clausal (CNF) Form

$$
\begin{aligned}
CNF(p, \Delta) &= \langle p, \Delta \rangle \\
CNF(\neg\phi, \Delta) &= \langle \neg l, \Delta' \rangle, \text{ where } \langle l, \Delta' \rangle = CNF(\phi, \Delta) \\
CNF(\phi_1 \wedge \phi_2, \Delta) &= \langle p, \Delta' \rangle, \text{ where} \\
&\quad \langle l_1, \Delta_1 \rangle = CNF(\phi_1, \Delta) \\
&\quad \langle l_2, \Delta_2 \rangle = CNF(\phi_2, \Delta_1) \\
&\quad p \text{ is fresh} \\
&\quad \Delta' = \Delta_2 \cup \{\neg p \vee l_1, \neg p \vee l_2, \neg l_1 \vee \neg l_2 \vee p\} \\
CNF(\phi_1 \vee \phi_2, \Delta) &= \langle p, \Delta' \rangle, \text{ where } \ldots \\
&\quad \Delta' = \Delta_2 \cup \{\neg p \vee l_1 \vee l_2, \neg l_1 \vee p, \neg l_2 \vee p\}
\end{aligned}
$$

Theorem: ϕ and $l \wedge \Delta$ are equisatisfiable, where $CNF(\phi, \emptyset) = \langle l, \Delta \rangle$.

Conversion to CNF: Example

$$CNF(\neg(\underbrace{q_1 \wedge \overbrace{(q_2 \vee \neg q_3)}^{p_1}}_{p_2}), \emptyset) =$$

$$\begin{aligned}
\langle \neg p_2, \{ \quad & \neg p_1 \vee q_2 \vee \neg q_3, \\
& \neg q_2 \vee p_1, \\
& q_3 \vee p_1, \\
& \neg p_2 \vee q_1, \\
& \neg p_2 \vee p_1, \\
& \neg q_1 \vee \neg p_1 \vee p_2 \} \rangle
\end{aligned}$$

Conversion to CNF

- Improvements:
  - Maximize sharing & canonicity in the input formula F.
  - Cache $\phi \mapsto l$, when $CNF(\phi, \Delta) = \langle l, \Delta' \rangle$.
  - Support for multiary $\vee$ and $\wedge$.
  - ...

Resolution

- Input: a set of clauses.
- No duplicate literals in clauses.
- Tautologies, clauses containing l and $\bar{l}$, are deleted.
- Rules:

$$F,\ C \vee l,\ D \vee \bar{l} \implies F,\ C \vee l,\ D \vee \bar{l},\ C \vee D$$

$$F,\ l,\ \bar{l} \implies \textbf{unsat}$$

- Improvement: ordered resolution.

Resolution: Example

$$\neg p \vee \neg q \vee r,\ \neg p \vee q,\ p \vee r,\ \neg r$$

Resolution: Example

$\neg p \vee \neg q \vee r,\ \neg p \vee q,\ p \vee r,\ \neg r \quad \Rightarrow$

$\neg p \vee \neg q \vee r,\ \neg p \vee q,\ p \vee r,\ \neg r,\ \neg q \vee r$

Resolution: Example

$$\neg p \vee \neg q \vee r,\ \neg p \vee q,\ p \vee r,\ \neg r \quad \Rightarrow$$

$$\neg p \vee \neg q \vee r,\ \neg p \vee q,\ p \vee r,\ \neg r,\ \neg q \vee r \quad \Rightarrow$$

$$\neg p \vee \neg q \vee r,\ \neg p \vee q,\ p \vee r,\ \neg r,\ \neg q \vee r,\ q \vee r$$

Resolution: Example

$$\neg p \vee \neg q \vee r,\ \neg p \vee q,\ p \vee r,\ \neg r \quad \Rightarrow$$

$$\neg p \vee \neg q \vee r,\ \neg p \vee q,\ p \vee r,\ \neg r,\ \neg q \vee r \quad \Rightarrow$$

$$\neg p \vee \neg q \vee r,\ \neg p \vee q,\ p \vee r,\ \neg r,\ \neg q \vee r,\ q \vee r \quad \Rightarrow$$

$$\neg p \vee \neg q \vee r,\ \neg p \vee q,\ p \vee r,\ \neg r,\ \neg q \vee r,\ q \vee r,\ r$$

Resolution: Example

$$\neg p \vee \neg q \vee r,\ \neg p \vee q,\ p \vee r,\ \neg r \quad \Rightarrow$$

$$\neg p \vee \neg q \vee r,\ \neg p \vee q,\ p \vee r,\ \neg r,\ \neg q \vee r \quad \Rightarrow$$

$$\neg p \vee \neg q \vee r,\ \neg p \vee q,\ p \vee r,\ \neg r,\ \neg q \vee r,\ q \vee r \quad \Rightarrow$$

$$\neg p \vee \neg q \vee r,\ \neg p \vee q,\ p \vee r,\ \neg r,\ \neg q \vee r,\ q \vee r,\ r \quad \Rightarrow$$

unsat

Resolution: Correctness

Progress: Bounded number of clauses. Each application of resolution generates a new clause.

Conservation: For any model M, if $M \models C \vee l$ and $M \models D \vee \bar{l}$, then $M \models C \vee D$.

Canonicity: Given an irreducible non-**unsat** state in the atoms $p_1, \ldots, p_n$ with $p_i \prec p_{i+1}$, build a series of partial interpretations M_i as follows:

1. Let $M_0 = \emptyset$
2. If p_{i+1} is not the maximal atom in some clause that is not already satisfied in M_i, then $M_{i+1} = M_i[p_{i+1} := \mathit{false}]$.
3. If some $p_{i+1} \vee C$ is not already satisfied in M_i, then $M_{i+1} = M_i[p_{i+1} := \mathit{true}]$.

The (original) DPLL Procedure

- Resolution is not practical (exponential amount of memory).
- DPLL tries to build incrementally a model 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.

Breakthrough in SAT solving

- Modern SAT solvers are based on the DPLL algorithm.
- Modern implementations add several sophisticated search techniques.
  - Backjumping
  - Learning
  - Restarts
  - Indexing

Abstract DPLL

$$M \parallel F \implies M\ l \parallel F \quad \textbf{if} \quad \begin{cases} l \text{ or } \bar{l} \text{ occurs in } F, \\ l \text{ is undefined in } M \end{cases} \quad \textbf{(Decide)}$$

$$M \parallel F, C \vee l \implies M\ l_{C \vee l} \parallel F, C \vee l \quad \textbf{if} \quad \begin{cases} M \models \neg C, \\ l \text{ is undefined in } M \end{cases} \quad \textbf{(UnitPropagate)}$$

$$M \parallel F, C \implies M \parallel F, C \parallel C \quad \textbf{if} \quad M \models \neg C \quad \textbf{(Conflict)}$$

$$M \parallel F \parallel C \vee \bar{l} \implies M \parallel F \parallel D \vee C \quad \textbf{if} \quad l_{D \vee l} \in M, \quad \textbf{(Resolve)}$$

$$M \parallel F \parallel C \implies M \parallel F, C \parallel C \quad \textbf{if} \quad C \notin F \quad \textbf{(Learn)}$$

$$M\ l'\ M' \parallel F \parallel C \vee l \implies M\ l_{C \vee l} \parallel F \quad \textbf{if} \quad \begin{cases} M \models \neg C, \\ l \text{ is undefined in } M \end{cases} \quad \textbf{(Backjump)}$$

$$M \parallel F \parallel \square \implies \textit{unsat} \quad \textbf{(Unsat)}$$

Abstract DPLL: Example

$$\| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$$

Abstract DPLL: Example

$$\| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(Decide)}$$

$$1 \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$$

Abstract DPLL: Example

$$\| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(Decide)}$$

$$1 \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(UnitProp)}$$

$$1\ 2_{\overline{1} \vee 2} \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$$

Abstract DPLL: Example

$$\| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(Decide)}$$

$$1 \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(UnitProp)}$$

$$1\ 2_{\overline{1} \vee 2} \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(Decide)}$$

$$1\ 2_{\overline{1} \vee 2}\ 3 \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$$

Abstract DPLL: Example

$$\| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(Decide)}$$

$$1 \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(UnitProp)}$$

$$1\ 2_{\overline{1} \vee 2} \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(Decide)}$$

$$1\ 2_{\overline{1} \vee 2}\ 3 \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(UnitProp)}$$

$$1\ 2_{\overline{1} \vee 2}\ 3\ 4_{\overline{3} \vee 4} \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$$

Abstract DPLL: Example

$$\| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(Decide)}$$

$$1 \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(UnitProp)}$$

$$1\ 2_{\overline{1} \vee 2} \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(Decide)}$$

$$1\ 2_{\overline{1} \vee 2}\ 3 \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(UnitProp)}$$

$$1\ 2_{\overline{1} \vee 2}\ 3\ 4_{\overline{3} \vee 4} \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(Decide)}$$

$$1\ 2_{\overline{1} \vee 2}\ 3\ 4_{\overline{3} \vee 4}\ 5 \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$$

Abstract DPLL: Example

$$\| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(Decide)}$$

$$1 \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(UnitProp)}$$

$$1\ 2_{\overline{1} \vee 2} \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(Decide)}$$

$$1\ 2_{\overline{1} \vee 2}\ 3 \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(UnitProp)}$$

$$1\ 2_{\overline{1} \vee 2}\ 3\ 4_{\overline{3} \vee 4} \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(Decide)}$$

$$1\ 2_{\overline{1} \vee 2}\ 3\ 4_{\overline{3} \vee 4}\ 5 \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(UnitProp)}$$

$$1\ 2_{\overline{1} \vee 2}\ 3\ 4_{\overline{3} \vee 4}\ 5\ \overline{6}_{\overline{5} \vee \overline{6}} \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$$

Abstract DPLL: Example

$$\| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(Decide)}$$

$$1 \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(UnitProp)}$$

$$1\ 2_{\overline{1} \vee 2} \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(Decide)}$$

$$1\ 2_{\overline{1} \vee 2}\ 3 \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(UnitProp)}$$

$$1\ 2_{\overline{1} \vee 2}\ 3\ 4_{\overline{3} \vee 4} \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(Decide)}$$

$$1\ 2_{\overline{1} \vee 2}\ 3\ 4_{\overline{3} \vee 4}\ 5 \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(UnitProp)}$$

$$1\ 2_{\overline{1} \vee 2}\ 3\ 4_{\overline{3} \vee 4}\ 5\ \overline{6}_{\overline{5} \vee \overline{6}} \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(Conflict)}$$

$$1\ 2_{\overline{1} \vee 2}\ 3\ 4_{\overline{3} \vee 4}\ 5\ \overline{6}_{\overline{5} \vee \overline{6}} \ \| \quad \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} \quad \| \quad 6 \vee \overline{5} \vee \overline{2}$$

Abstract DPLL: Example

$$
\begin{array}{rcll}
 & \| & \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} & \Rightarrow \quad \text{(Decide)} \\
1 & \| & \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} & \Rightarrow \quad \text{(UnitProp)} \\
1\ 2_{\overline{1} \vee 2} & \| & \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} & \Rightarrow \quad \text{(Decide)} \\
1\ 2_{\overline{1} \vee 2}\ 3 & \| & \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} & \Rightarrow \quad \text{(UnitProp)} \\
1\ 2_{\overline{1} \vee 2}\ 3\ 4_{\overline{3} \vee 4} & \| & \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} & \Rightarrow \quad \text{(Decide)} \\
1\ 2_{\overline{1} \vee 2}\ 3\ 4_{\overline{3} \vee 4}\ 5 & \| & \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} & \Rightarrow \quad \text{(UnitProp)} \\
1\ 2_{\overline{1} \vee 2}\ 3\ 4_{\overline{3} \vee 4}\ 5\ \overline{6}_{\overline{5} \vee \overline{6}} & \| & \overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2} & \Rightarrow \quad \text{(Conflict)} \\
1\ 2_{\overline{1} \vee 2}\ 3\ 4_{\overline{3} \vee 4}\ 5\ \overline{6}_{\overline{5} \vee \overline{6}} & \| & \underbrace{\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}}_{F} & \| \quad 6 \vee \overline{5} \vee \overline{2}
\end{array}
$$

Abstract DPLL: Example (cont.)

$$1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}} \quad \| \quad F \quad \| \quad 6 \vee \overline{5} \vee \overline{2}$$

Abstract DPLL: Example (cont.)

$$1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}} \parallel \quad F \quad \parallel \quad 6 \vee \overline{5} \vee \overline{2} \quad \Rightarrow \quad \text{(Resolve)}$$

$$1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}} \parallel \quad F \quad \parallel \quad \overline{5} \vee \overline{2}$$

Abstract DPLL: Example (cont.)

$$1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}} \parallel F \parallel 6\vee\overline{5}\vee\overline{2} \Rightarrow \text{(Resolve)}$$

$$1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}} \parallel F \parallel \overline{5}\vee\overline{2} \Rightarrow \text{(Learn)}$$

$$1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}} \parallel F, \overline{5}\vee\overline{2} \parallel \overline{5}\vee\overline{2}$$

Abstract DPLL: Example (cont.)

$$1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}} \parallel F \parallel 6\vee\overline{5}\vee\overline{2} \Rightarrow \text{(Resolve)}$$

$$1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}} \parallel F \parallel \overline{5}\vee\overline{2} \Rightarrow \text{(Learn)}$$

$$1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}} \parallel F, \overline{5}\vee\overline{2} \parallel \overline{5}\vee\overline{2} \Rightarrow \text{(Resolve)}$$

$$1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}} \parallel F, \overline{5}\vee\overline{2} \parallel \overline{5}\vee\overline{1}$$

Abstract DPLL: Example (cont.)

$$
\begin{array}{rlll}
1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}} & \| \quad F & \| \quad 6\vee\overline{5}\vee\overline{2} & \Rightarrow \quad \text{(Resolve)} \\
1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}} & \| \quad F & \| \quad \overline{5}\vee\overline{2} & \Rightarrow \quad \text{(Learn)} \\
1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}} & \| \quad F,\ \overline{5}\vee\overline{2} & \| \quad \overline{5}\vee\overline{2} & \Rightarrow \quad \text{(Resolve)} \\
1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}} & \| \quad F,\ \overline{5}\vee\overline{2} & \| \quad \overline{5}\vee\overline{1} & \Rightarrow \quad \text{(Backjump)} \\
1\ 2_{\overline{1}\vee 2}\ \overline{5}_{\overline{5}\vee\overline{1}} & \| \quad F,\ \overline{5}\vee\overline{2} & &
\end{array}
$$

Abstract DPLL (cont.)

- Support different strategies.
  - Example: learn 0 or several clauses per conflict.
- Does it terminate?
  - Each decision defines a new scope level.
  - Metric: number of assigned literals per level.

$$1\ 2_{\overline{1}\vee 2}\ 3\ 4_{\overline{3}\vee 4}\ 5\ \overline{6}_{\overline{5}\vee\overline{6}} \quad \mapsto \quad (2,2,2)$$

$$1\ 2_{\overline{1}\vee 2}\ \overline{5}_{\overline{5}\vee\overline{1}} \quad \mapsto \quad (3)$$

  - **Decide**, **UnitPropagate**, and **Backjump** increase the metric.
  - It can not increase forever (finite number of variables).
  - Conflict resolution rules (**Conflict**, **Resolve**, **Learn**) are also terminating.

Abstract DPLL: Strategy

- Abstract DPLL is very flexible.
- Basic Strategy:
  - Only apply **Decide** if **UnitPropagate** and **Conflict** cannot be applied.
- Conflict Resolution:
  - Learn only one clause per conflict (the clause used in **Backjump**).
  - Use **Backjump** as soon as possible (FUIP).
  - Use the rightmost (applicable) literal in M when applying **Resolve**.

$$M \parallel F \parallel C \vee \bar{l} \quad \Longrightarrow \quad M \parallel F \parallel D \vee C \qquad \textbf{if} \;\; l_{D \vee l} \in M, \qquad \textbf{(Resolve)}$$

Abstract DPLL: Decision Strategy

- Decision heuristic:
  - Associate a score with each boolean variable.
  - Select the variable with highest score when **Decide** is used.
  - Increase by δ the score of *var*(l) when **Resolve** is used:

    $M \parallel F \parallel C \vee \bar{l} \quad \Longrightarrow \quad M \parallel F \parallel D \vee C \qquad$ **if** $\; l_{D \vee l} \in M, \qquad$ (**Resolve**)
  - Increase the score of every variable in the clause $C \vee l$ when **Backjump** is used:

    $$M\, l'\, M' \parallel F \parallel C \vee l \quad \Longrightarrow \quad M\, l_{C \vee l} \parallel F' \qquad \text{if} \begin{cases} M \models \neg C, \\ l \text{ is undefined in } M \end{cases} \qquad \text{(Backjump)}$$
  - After each conflict: slightly increase the value of δ.
  - From time to time renormalize the scores and δ to avoid overflows.

Abstract DPLL: Phase Selection

- Assume p was selected by a decision strategy.

  Should we assign p or $\neg p$ in **Decide**?

  Always False Guess $\neg p$ (works well in practice).

  Always True Guess p.

  Score Associate a score with each literal instead of each variable.

  Pick the phase with highest score.

  Caching Caches the last phase of variables during conflict resolution. Improvement: except for variables in the last decision level.

  Greedy Select the phase that satisfies most clauses.

Abstract DPLL: Extra Rules

- Extra rules:

$M \parallel F, C \quad \Longrightarrow \quad M \parallel F$ **if** C is a learned clause (**Forget**)

$M \parallel F \quad \Longrightarrow \quad \parallel F$ (**Restart**)

- **Forget** in practice:
  - Associate a score with each learned clause C.
  - Increase by δ_c the score of $D \vee l$ when **Resolve** is used.

    $M \parallel F \parallel C \vee \bar{l} \quad \Longrightarrow \quad M \parallel F \parallel D \vee C$ **if** $l_{D \vee l} \in M,$ (**Resolve**)
  - From time to time use **Forget** to delete learned clauses with low score.

Abstract DPLL: Restart Strategies

No restarts

Linear Restart after every k conflicts, update $k := k + \delta$.

Geometric Restart after every k conflicts, update $k := k \times \delta$.

Inner-Out Geometric "Two dimensional pattern" that increases in both dimensions.

- Initially $k := x$, the inner loop multiplies k by δ at each restart.
- When $k > y$, $k := x$ and $y := y \times \delta$.

Luby Restarts are performed according to the following series: $1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, \ldots$, multiplied by a constant c (e.g., 100, 256, 512).

$$\mathit{luby}(i) = \begin{cases} 2^{k-1}, & \text{if } \exists k.\ i = 2^k - 1 \\ \mathit{luby}(i - 2^{k-1} + 1), & \text{if } \exists k.\ 2^{k-1} \leq i < 2^k - 1 \end{cases}$$

Indexing

- Indexing techniques are very important.
- How to implement **UnitPropagate** and **Conflict**?
- Scanning the set of clauses will not scale.
- Simple index: mapping from literals to clauses (occurrences).
  - *watch*$(l) = \{C_1, \ldots, C_n\}$, where $\bar{l} \in C_i$
  - If l is assigned, check each clause $C \in$ *watch*(l) for **UnitPropagate** and **Conflict**.
  - Most of the time C has more than one unassigned literal.
  - Improvement: associate a counter u with each clause (number of unassigned literals).
  - Problem: counters must be decremented when literals are assigned, and restored during **Backjump**.

Indexing: Two Watch Literal

- Insight:
  - No need to include clause C in every set *watch*(l) where $\bar{l} \in C$.
  - It suffices to include C in at most 2 such sets.
- Invariant:

  If some literal l in C is not assigned to false, then $C \in$ *watch*(l') of some l' that is not assigned to false.

Indexing: Two watch Literal

- Maintain 2-watch invariant:
  - Whenever l is assigned.
  - For each clause $C \in \mathit{watch}(l)$
    - If the other watch literal l' ($C \in \mathit{watch}(l')$) is assigned to true, then do nothing.
    - Else if some other literal l' is true or unassigned

$$\begin{aligned} \mathit{watch}(l') &:= \mathit{watch}(l') \cup \{C\} \\ \mathit{watch}(l) &:= \mathit{watch}(l) \setminus \{C\} \end{aligned}$$

    - Else if all literals in C are assigned to false, then **Backjump**.
    - Else (all but one literal in C is assigned to false) **Propagate**.

Preprocessor

- Preprocessing step is very important for industrial benchmarks.
- Formula $\leadsto$ CNF (already covered).
- Subsumption: C subsumes D if $C \subseteq D$.
- Resolution: eliminate cheap variables.
  - $\mathit{occs}(l) = \{\text{clauses that contain } l\}$
  - $|\mathit{occs}(p)| * |\mathit{occs}(\neg p)| < k$
  - $|\mathit{occs}(p)| = 1$ or $|\mathit{occs}(\neg p)| = 1$

Satisfiability Modulo Theories (SMT)

- In SMT solving, the Boolean atoms represent constraints over individual variables ranging over integer, reals, bit-vectors, datatypes, and arrays.
- The constraints can involve theory operations, equality, and inequality.
- Now, the SAT solver has to interact with theory solvers.
- The constraint solver can detect conflicts involving theory reasoning, e.g., $f(x) \neq f(y), x = y$, or $x - y \leq 2, y - z \leq -1, z - x \leq -3$.

Pure Theory of Equality (EUF)

- The theory $\mathcal{T}_E$ of equality is the theory $DC(\emptyset)$.
- The exact set of sentences of $\mathcal{T}_E$ depends on the signature in question.
- The theory does not restrict the possibles values of the symbols in its signature in any way. For this reason, it is sometimes called the theory of equality and uninterpreted functions.
- The satisfiability problem for $\mathcal{T}_E$ is the satisfiability problem for first-order logic, which is undecidable.
- The satisfiability problem for conjunction of literals in $\mathcal{T}_E$ is decidable in polynomial time using congruence closure.

Linear Integer Arithmetic

- $\Sigma_P = \{\leq\}, \Sigma_F = \{0, 1, +, -\}$.
- Let M_{LIA} be the standard model of integers.
- Then $\mathcal{T}_{LIA}$ is defined to be the set of all Σ sentences true in the model M_{LIA}.
- As showed by Presburger, the general satisfiability problem for $\mathcal{T}_{LIA}$ is decidable, but its complexity is triply-exponential.
- The quantifier free satisfiability problem is NP-complete.
- Remark: non-linear integer arithmetic is undecidable even for the quantifier free case.

Linear Real Arithmetic

- The general satisfiability problem for $\mathcal{T}_{LRA}$ is decidable, but its complexity is doubly-exponential.
- The quantifier free satisfiability problem is solvable in polynomial time, though exponential methods (Simplex) tend to perform best in practice.

Difference Logic

- Difference logic is a fragment of linear arithmetic.
- Atoms have the form: $x - y \leq c$.
- Most linear arithmetic atoms found in hardware and software verification are in this fragment.
- The quantifier free satisfiability problem is solvable in $O(nm)$.

Theory of Arrays

- $\Sigma_P = \emptyset, \Sigma_F = \{\mathit{read}, \mathit{write}\}$.
- Non-extensional arrays
  - Let $\Lambda_{\mathcal{A}}$ be the following axioms:

$$\forall a, i, v.\ read(write(a, i, v), i) = v$$

$$\forall a, i, j, v.\ i \neq j \Rightarrow read(write(a, i, v), j) = read(a, j)$$

  - $\mathcal{T}_{\mathcal{A}} = DC(\Lambda_{\mathcal{A}})$
- For extensional arrays, we need the following extra axiom:

$$\forall a, b.\ (\forall i. read(a, i) = read(b, i)) \Rightarrow a = b$$

- The satisfiability problem for $\mathcal{T}_{\mathcal{A}}$ is undecidable, the quantifier free case is NP-complete.

Other theories

- Bit-vectors
- Partial orders
- Tuples & Records
- Algebraic data types
- . . .

Theory Solver: Rules

We use $F \models_T G$ to denote the fact that F *entails* G in theory T.

T-Propagate

$$M \parallel F \implies M\ l_{(\neg l_1 \vee \ldots \vee \neg l_n \vee l)} \parallel F \qquad \textbf{if} \begin{cases} l \text{ occurs in } F, \\ l \text{ is undefined in } M, \\ l_1 \wedge \ldots \wedge l_n \models_T l, \\ l_1, \ldots, l_n \in \mathit{lits}(M) \end{cases}$$

T-Conflict

$$M \parallel F \implies M \parallel F \parallel \neg l_1 \vee \ldots \vee \neg l_n \quad \textbf{if} \begin{cases} l_1 \wedge \ldots \wedge l_n \models_T \mathit{false}, \\ l_1, \ldots, l_n \in \mathit{lits}(M) \end{cases}$$

DPLL + Theory Solver

$$
\begin{aligned}
p &\equiv 3 < x \\
q &\equiv x < 0 \\
r &\equiv x < y \\
s &\equiv y < 0
\end{aligned}
$$

$$\| \quad p,\ q \vee r,\ s \vee \neg r$$

DPLL + Theory Solver

$$
\begin{aligned}
p &\equiv 3 < x \\
q &\equiv x < 0 \\
r &\equiv x < y \\
s &\equiv y < 0
\end{aligned}
$$

$$
\begin{array}{rcl}
 & \| \quad p,\ q \vee r,\ s \vee \neg r & \Rightarrow \quad \text{(UnitPropagate)} \\
p_p & \| \quad p,\ q \vee r,\ s \vee \neg r &
\end{array}
$$

DPLL + Theory Solver

$$
\begin{aligned}
p &\equiv 3 < x \\
q &\equiv x < 0 \\
r &\equiv x < y \\
s &\equiv y < 0
\end{aligned}
$$

$$
\begin{aligned}
&\| \quad p,\ q \vee r,\ s \vee \neg r \quad \Rightarrow \quad \text{(UnitPropagate)} \\
p_p \ &\| \quad p,\ q \vee r,\ s \vee \neg r \quad \Rightarrow \quad \text{(T-Propagate)} \\
p_p\ \neg q_{\neg p \vee \neg q} \ &\| \quad p,\ q \vee r,\ s \vee \neg r
\end{aligned}
$$

$\underbrace{3 < x}_{p}$ implies $\neg\, \underbrace{x < 0}_{q}$

DPLL + Theory Solver

$$
\begin{aligned}
p &\equiv 3 < x \\
q &\equiv x < 0 \\
r &\equiv x < y \\
s &\equiv y < 0
\end{aligned}
$$

$$
\begin{array}{rcll}
 & \| & p,\ q \vee r,\ s \vee \neg r & \Rightarrow \quad \text{(UnitPropagate)} \\
p_p & \| & p,\ q \vee r,\ s \vee \neg r & \Rightarrow \quad \text{(T-Propagate)} \\
p_p\ \neg q_{\neg p \vee \neg q} & \| & p,\ q \vee r,\ s \vee \neg r & \Rightarrow \quad \text{(UnitPropagate)} \\
p_p\ \neg q_{\neg p \vee \neg q}\ r_{q \vee r} & \| & p,\ q \vee r,\ s \vee \neg r &
\end{array}
$$

DPLL + Theory Solver

$$
\begin{aligned}
p &\equiv 3 < x \\
q &\equiv x < 0 \\
r &\equiv x < y \\
s &\equiv y < 0
\end{aligned}
$$

$$
\begin{array}{rcll}
 & \| & p,\ q \vee r,\ s \vee \neg r & \Rightarrow \quad \text{(UnitPropagate)} \\
p_p & \| & p,\ q \vee r,\ s \vee \neg r & \Rightarrow \quad \text{(T-Propagate)} \\
p_p\ \neg q_{\neg p \vee \neg q} & \| & p,\ q \vee r,\ s \vee \neg r & \Rightarrow \quad \text{(UnitPropagate)} \\
p_p\ \neg q_{\neg p \vee \neg q}\ r_{q \vee r} & \| & p,\ q \vee r,\ s \vee \neg r & \Rightarrow \quad \text{(UnitPropagate)} \\
p_p\ \neg q_{\neg p \vee \neg q}\ r_{q \vee r}\ s_{s \vee \neg r} & \| & p,\ q \vee r,\ s \vee \neg r &
\end{array}
$$

DPLL + Theory Solver

$$
\begin{aligned}
p &\equiv 3 < x \\
q &\equiv x < 0 \\
r &\equiv x < y \\
s &\equiv y < 0
\end{aligned}
$$

$$
\begin{aligned}
&\| \quad p,\ q \vee r,\ s \vee \neg r \quad \Rightarrow && \text{(UnitPropagate)} \\
p_p \ &\| \quad p,\ q \vee r,\ s \vee \neg r \quad \Rightarrow && \text{(T-Propagate)} \\
p_p \ \neg q_{\neg p \vee \neg q} \ &\| \quad p,\ q \vee r,\ s \vee \neg r \quad \Rightarrow && \text{(UnitPropagate)} \\
p_p \ \neg q_{\neg p \vee \neg q} \ r_{q \vee r} \ &\| \quad p,\ q \vee r,\ s \vee \neg r \quad \Rightarrow && \text{(UnitPropagate)} \\
p_p \ \neg q_{\neg p \vee \neg q} \ r_{q \vee r} \ s_{s \vee \neg r} \ &\| \quad p,\ q \vee r,\ s \vee \neg r \quad \Rightarrow && \text{(T-Conflict)} \\
p_p \ \neg q_{\neg p \vee \neg q} \ r_{q \vee r} \ s_{s \vee \neg r} \ &\| \quad p,\ q \vee r,\ s \vee \neg r \quad \| \quad \neg p \vee \neg r \vee \neg s
\end{aligned}
$$

$\underbrace{3 < x}_{p}, \underbrace{x < y}_{r}, \underbrace{y < 0}_{s}$ implies false

DPLL + Theory Solver

- Do we need T-Propagate?

DPLL + Theory Solver

- Do we need T-Propagate?
  - No
  - Trade-off between precision and performance.

DPLL + Theory Solver

- Do we need T-Propagate?
  - No
  - Trade-off between precision and performance.
- What is the minimal functionality of a theory solver?

DPLL + Theory Solver

- Do we need T-Propagate?
  - No
  - Trade-off between precision and performance.
- What is the minimal functionality of a theory solver?
  - Check the unsatisfiability of conjunction of literals.

DPLL + Theory Solver

- Do we need T-Propagate?
  - No
  - Trade-off between precision and performance.
- What is the minimal functionality of a theory solver?
  - Check the unsatisfiability of conjunction of literals.
- Efficiently mining T-justifications

T-Propagate

$$M \parallel F \implies M\ l_{(\neg l_1 \vee \ldots \vee \neg l_n \vee l)} \parallel F \quad \textbf{if} \quad \begin{cases} l \text{ occurs in } F, \\ l \text{ is undefined in } M, \\ l_1 \wedge \ldots \wedge l_n \models_T l, \\ l_1, \ldots, l_n \in \mathit{lits}(M) \end{cases}$$

T-Conflict

$$M \parallel F \implies M \parallel F \parallel \neg l_1 \vee \ldots \vee \neg l_n \quad \textbf{if} \quad \begin{cases} l_1 \wedge \ldots \wedge l_n \models_T \mathit{false}, \\ l_1, \ldots, l_n \in \mathit{lits}(M) \end{cases}$$

The Ideal Theory Solver

- Incremental
- Efficient Backtracking
- Efficient T-Propagate
- Precise T-Justifications

Roadmap

- Background
- SAT & SMT
- Combining theories
- Equality
- Arithmetic
- Quantifiers
- Applications

Combination of Theories

- In practice, we need a combination of theories.
- Example:
  - $x + 2 = y \Rightarrow f(\mathit{read}(\mathit{write}(a, x, 3), y - 2)) = f(y - x + 1)$
- 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: $\Sigma_1 \cap \Sigma_2 = \emptyset$.
- Purification
- Stably-Infinite Theories.
- Convex Theories.

Purification

- Purification:

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

  t is not a variable.

- Purification is satisfiability preserving and terminating.

Purification

- Purification:

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

  t is not a variable.

- Purification is satisfiability preserving and terminating.

- Example:

$$f(x-1) - 1 = x, f(y) + 1 = y \rightsquigarrow$$

Purification

- Purification:

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

  t is not a variable.

- Purification is satisfiability preserving and terminating.

- Example:

$$f(x-1) - 1 = x, f(y) + 1 = y \rightsquigarrow$$

$$f(u_1) - 1 = x, f(y) + 1 = y, u_1 = x - 1 \rightsquigarrow$$

Purification

- Purification:

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

  t is not a variable.

- Purification is satisfiability preserving and terminating.

- Example:

$$f(x-1) - 1 = x, f(y) + 1 = y \rightsquigarrow$$

$$f(u_1) - 1 = x, f(y) + 1 = y, u_1 = x - 1 \rightsquigarrow$$

$$u_2 - 1 = x, f(y) + 1 = y, u_1 = x - 1, u_2 = f(u_1) \rightsquigarrow$$

Purification

- Purification:

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

  t is not a variable.

- Purification is satisfiability preserving and terminating.

- Example:

$$f(x-1) - 1 = x, f(y) + 1 = y \rightsquigarrow$$

$$f(u_1) - 1 = x, f(y) + 1 = y, u_1 = x - 1 \rightsquigarrow$$

$$u_2 - 1 = x, f(y) + 1 = y, u_1 = x - 1, u_2 = f(u_1) \rightsquigarrow$$

$$u_2 - 1 = x, u_3 + 1 = y, u_1 = x - 1, u_2 = f(u_1), u_3 = f(y)$$

Purification

- Purification:

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

  t is not a variable.

- Purification is satisfiability preserving and terminating.

- Example:

$$f(x-1) - 1 = x, f(y) + 1 = y \rightsquigarrow$$

$$f(u_1) - 1 = x, f(y) + 1 = y, u_1 = x - 1 \rightsquigarrow$$

$$u_2 - 1 = x, f(y) + 1 = y, u_1 = x - 1, u_2 = f(u_1) \rightsquigarrow$$

$$u_2 - 1 = x, u_3 + 1 = y, u_1 = x - 1, u_2 = f(u_1), u_3 = f(y)$$

After Purification

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

After Purification

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

Red Model	Blue Model
$\lvert R\rvert = \{*_1, \ldots, *_6\}$	$\lvert B\rvert = \{\ldots, -1, 0, 1, \ldots\}$
$R(x) = *_1$	$B(x) = 0$
$R(y) = *_2$	$B(y) = 0$
$R(z) = *_3$	$B(z) = -1$
$R(f) = \{*_1 \mapsto *_4,$	
$*_2 \mapsto *_5,$	
$*_3 \mapsto *_1,$	
$\mathit{else} \mapsto *_6\}$	

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.

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

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 $O(n^3 \times (T_1(n) + T_2(n)))$.

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.

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 $\left|A\right|$ and $\left|B\right|$ such that $h(A(x)) = B(x)$ 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 $|A|$ and $|B|$ such that $h(A(x)) = B(x)$ for each shared variable.
  - Extend B to $\bar{B}$ by interpretations of symbols in Σ_1: $\bar{B}(f)(b_1, \ldots, b_n) = h(A(f)(h^{-1}(b_1), \ldots, h^{-1}(b_n)))$

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 $|A|$ and $|B|$ such that $h(A(x)) = B(x)$ for each shared variable.
  - Extend B to $\bar{B}$ by interpretations of symbols in Σ_1: $\bar{B}(f)(b_1, \ldots, b_n) = h(A(f)(h^{-1}(b_1), \ldots, h^{-1}(b_n)))$
  - $\bar{B}$ 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, $\mathcal{T}_i \not\vdash \phi_i \Rightarrow x_j = x_k$.

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, $\mathcal{T}_i \nvdash \phi_i \Rightarrow x_j = x_k$.
  - By convexity, $\mathcal{T}_i \nvdash \phi_i \Rightarrow \bigvee_E x_j = x_k$.

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, $\mathcal{T}_i \nvdash \phi_i \Rightarrow x_j = x_k$.
  - By convexity, $\mathcal{T}_i \nvdash \phi_i \Rightarrow \bigvee_E x_j = x_k$.
  - $\phi_i \wedge \bigwedge_E x_j \neq x_k$ 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, $\mathcal{T}_i \nvdash \phi_i \Rightarrow x_j = x_k$.
  - By convexity, $\mathcal{T}_i \nvdash \phi_i \Rightarrow \bigvee_E x_j = x_k$.
  - $\phi_i \wedge \bigwedge_E x_j \neq x_k$ 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, $\mathcal{T}_i \nvdash \phi_i \Rightarrow x_j = x_k$.
  - By convexity, $\mathcal{T}_i \nvdash \phi_i \Rightarrow \bigvee_E x_j = x_k$.
  - $\phi_i \wedge \bigwedge_E x_j \neq x_k$ 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.

NO procedure: example

$x + 2 = y \wedge f(\mathit{read}(\mathit{write}(a, x, 3), y - 2)) \neq f(y - x + 1)$

$\mathcal{T}_E$	$\mathcal{T}_A$	$\mathcal{T}_{Ar}$

Purifying

NO procedure: example

$f(\mathit{read}(\mathit{write}(a, x, 3), y - 2)) \neq f(y - x + 1)$

$\mathcal{T}_E$	$\mathcal{T}_A$	$\mathcal{T}_{Ar}$
	$x + 2 = y$	

Purifying

NO procedure: example

$f(\mathit{read}(\mathit{write}(a, x, u_1), y - 2)) \neq f(y - x + 1)$

$\mathcal{T}_E$	$\mathcal{T}_A$	$\mathcal{T}_{Ar}$
	$x + 2 = y$ $u_1 = 3$	

Purifying

NO procedure: example

$$f(\mathit{read}(\mathit{write}(a, x, u_1), u_2)) \neq f(y - x + 1)$$

$\mathcal{T}_E$	$\mathcal{T}_A$	$\mathcal{T}_{Ar}$
	$x + 2 = y$ $u_1 = 3$ $u_2 = y - 2$	

Purifying

NO procedure: example

$f(u_3) \neq f(y - x + 1)$

$\mathcal{T}_E$	$\mathcal{T}_A$	$\mathcal{T}_{Ar}$
	$x + 2 = y$ $u_1 = 3$ $u_2 = y - 2$	$u_3 =$ $\mathit{read}(\mathit{write}(a, x, u_1), u_2)$

Purifying

NO procedure: example

$f(u_3) \neq f(u_4)$

$\mathcal{T}_E$	$\mathcal{T}_A$	$\mathcal{T}_{Ar}$
	$x + 2 = y$ $u_1 = 3$ $u_2 = y - 2$ $u_4 = y - x + 1$	$u_3 =$ $\mathit{read}(\mathit{write}(a, x, u_1), u_2)$

Purifying

NO procedure: example

$\mathcal{T}_E$	$\mathcal{T}_A$	$\mathcal{T}_{Ar}$
$f(u_3) \neq f(u_4)$	$x + 2 = y$ $u_1 = 3$ $u_2 = y - 2$ $u_4 = y - x + 1$	$u_3 =$ $\mathit{read}(\mathit{write}(a, x, u_1), u_2)$

Solving $\mathcal{T}_A$

NO procedure: example

$\mathcal{T}_E$	$\mathcal{T}_A$	$\mathcal{T}_{Ar}$
$f(u_3) \neq f(u_4)$	$y = x + 2$ $u_1 = 3$ $u_2 = x$ $u_4 = 3$	$u_3 =$ $\mathit{read}(\mathit{write}(a, x, u_1), u_2)$

Propagating $u_2 = x$

NO procedure: example

$\mathcal{T}_E$	$\mathcal{T}_A$	$\mathcal{T}_{Ar}$
$f(u_3) \neq f(u_4)$	$y = x + 2$	$u_3 =$
$u_2 = x$	$u_1 = 3$	$\mathit{read}(\mathit{write}(a, x, u_1), u_2)$
	$u_2 = x$	$u_2 = x$
	$u_4 = 3$	

Solving $\mathcal{T}_{Ar}$

NO procedure: example

$\mathcal{T}_E$	$\mathcal{T}_A$	$\mathcal{T}_{Ar}$
$f(u_3) \neq f(u_4)$	$y = x + 2$	$u_3 = u_1$
$u_2 = x$	$u_1 = 3$	$u_2 = x$
	$u_2 = x$	
	$u_4 = 3$	

Propagating $u_3 = u_1$

NO procedure: example

$\mathcal{T}_E$	$\mathcal{T}_A$	$\mathcal{T}_{Ar}$
$f(u_3) \neq f(u_4)$	$y = x + 2$	$u_3 = u_1$
$u_2 = x$	$u_1 = 3$	$u_2 = x$
$u_3 = u_1$	$u_2 = x$	
	$u_4 = 3$	
	$u_3 = u_1$	

Propagating $u_1 = u_4$

NO procedure: example

$\mathcal{T}_E$	$\mathcal{T}_A$	$\mathcal{T}_{Ar}$
$f(u_3) \neq f(u_4)$	$y = x + 2$	$u_3 = u_1$
$u_2 = x$	$u_1 = 3$	$u_2 = x$
$u_3 = u_1$	$u_2 = x$	
$u_4 = u_1$	$u_4 = 3$	
	$u_3 = u_1$	

Congruence $u_3 = u_1 \wedge u_4 = u_1 \Rightarrow f(u_3) = f(u_4)$

NO procedure: example

$\mathcal{T}_E$	$\mathcal{T}_A$	$\mathcal{T}_{Ar}$
$f(u_3) \neq f(u_4)$	$y = x + 2$	$u_3 = u_1$
$u_2 = x$	$u_1 = 3$	$u_2 = x$
$u_3 = u_1$	$u_2 = x$	
$u_4 = u_1$	$u_4 = 3$	
$f(u_3) = f(u_4)$	$u_3 = u_1$	

Unsatisfiable!

NO deterministic procedure

- Deterministic procedure does not work for non convex theories.
- Example (integer arithmetic):

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

- (Expensive) solution: deduce disjunctions of equalities.

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.

    if $M_i \models \mathcal{T}_i \cup \Gamma_i \cup \{u = v\}$ **then** 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

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

Assume x = y

Model based theory combination: Example

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

Unsatisfiable

Model based theory combination: Example

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

Backtrack, and assert $x \neq y$.

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

Model based theory combination: Example

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

Assume x = z

Model based theory combination: Example

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

Satisfiable

Model based theory combination: Example

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

Let h be the bijection between $|E|$ and $|A|$.

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

Model based theory combination: Example

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

Extending A using h.

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

Model mutation

- Sometimes $M(x) = M(y)$ by accident.

$$\bigwedge_{i=1}^{N} f(x_i) \geq 0 \ \wedge \ x_i \geq 0$$

- Model mutation: diversify the current model.

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.

Reduction Functions

- A reduction function reduces the satisfiability of a complex theory to the satisfiability problem of a simpler theory.
- Ackerman 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 formula $\vec{a} = \vec{c} \Rightarrow f_{\vec{a}} = f_{\vec{c}}$.
  - It is used in some SMT solvers to reduce $\mathcal{T}_A \cup \mathcal{T}_E$ to $\mathcal{T}_A$.

Ackerman Reduction: Example

$$f(x-1)-1=f(z),$$

$$f(y)+1=y$$

$$\rightsquigarrow$$

$$x-1=z \Rightarrow f_{x-1}=f_z,$$

$$x-1=y \Rightarrow f_{x-1}=f_y,$$

$$z=y \Rightarrow f_z=f_y,$$

$$f_{x-1}-1=f_z,$$

$$f_y+1=y$$

$f_{x-1}, f_z,$ and f_y are new variables.

Reduction Functions

- Theory of commutative functions.
  - Deductive closure of: $\forall x, y. f(x, y) = f(y, x)$
  - Reduction to $\mathcal{T}_E$.
  - For every $f(a, b)$ in ϕ, do $\phi := \phi \wedge f(a, b) = f(b, a)$.
- Theory of "lists".
  - Deductive closure of:

$$\forall x, y.\ car(cons(x, y)) = x$$

$$\forall x, y.\ cdr(cons(x, y)) = y$$

  - Reduction to $\mathcal{T}_E$
  - For each term $cons(a, b)$ in ϕ, do
    $\phi := \phi \wedge car(cons(a, b)) = a \wedge cdr(cons(a, b)) = b$.

Roadmap

- Background
- SAT & SMT
- Combining theories
- Equality
- Arithmetic
- Quantifiers
- Applications

Theory of Equality: Axioms

Reflexivity $x = x$

Symmetry $x = y \Rightarrow y = x$

Transitivity $x = y, y = z \Rightarrow x = z$

Congruence

$$x_1 = y_1, \ldots, x_n = y_n \Rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$$

Example

$$f(f(a)) = a,\ b = f(a),\ \neg f(f(f(a))) = b$$

Example

$f(f(a)) = a,\ b = f(a),\ \neg f(f(f(a))) = b$

congruence $\rightsquigarrow\ f(f(f(a))) = f(a)$

Example

$f(f(a)) = a,\ b = f(a),\ \neg f(f(f(a))) = b,$
$f(f(f(a))) = f(a)$

symmetry $\rightsquigarrow f(a) = b$

Example

$f(f(a)) = a,\ b = f(a),\ \neg f(f(f(a))) = b,$

$f(f(f(a))) = f(a),\ f(a) = b$

transitivity $\rightsquigarrow$ $f(f(f(a))) = b$

Example

$$f(f(a)) = a,\ b = f(a),\ \neg f(f(f(a))) = b,$$
$$f(f(f(a))) = f(a),\ f(a) = b,\ f(f(f(a))) = b$$

unsatisfiable

Example

- A conjunction of equalities is trivially satisfiable.
- Example: $f(x) = y,\ x = y,\ g(x) = z,\ f(y) = f(z)$

Example

- A conjunction of equalities is trivially satisfiable.
- Example: $f(x) = y,\ x = y,\ g(x) = z,\ f(y) = f(z)$
- Model:
  - $|M| = \{*_1\}$
  - $M(x) = M(y) = M(z) = *_1$
  - $M(f)(*_1) = *_1$
  - $M(g)(*_1) = *_1$

Variable equality

- Assume the problem has not function symbols.
- Use union-find data structure to represent equalities.
- The state consists of a find structure F that maintains equivalence classes and a set of disequalities D.
- Initially, $F(x) = x$ for each variable x.
- $F^*(x)$ is the root of the equivalence class containing x:

$$F^*(x) = \begin{cases} x, & \text{if } F(x) = x \\ F^*(F(x)) & \text{otherwise} \end{cases}$$

- Let $sz(F, x)$ denote the size of the equivalence class containing x.

Variable equality: union

- An equality $x = y$ is processed by merging distinct equivalence classes using the *union* operation:

$$\mathit{union}(F, x, y) = \begin{cases} F[x' := y'], & \mathit{sz}(F, x) < \mathit{sz}(F, y) \\ F[y' := x'], & \text{otherwise} \end{cases}$$

where $x' \equiv F^*(x) \not\equiv F^*(y) \equiv y'$

- Optimization: path compression, update F when executing $F^*(x)$. $F[x := F^*(x)]$

Processing equalities

- The entire inference system consists of operations for adding equalities, disequalities, and dectecting unsatisfiability.

$$\mathit{addeq}(x = y, F, D) \quad := \quad \langle F, D \rangle, \text{ if } F^*(x) \equiv F^*(y)$$

$$\mathit{addeq}(x = y, F, D) \quad := \quad \begin{cases} \textbf{unsat}, & \text{if } F'^*(u) \equiv F'^*(v) \text{ for some} \\ & u \neq v \in D \\ \langle F', D \rangle, & \text{otherwise} \end{cases}$$

where $F^*(x) \not\equiv F^*(y)$

$F' = \mathit{union}(F, x, y)$

Processing disequalities

$$
\begin{aligned}
\textit{addneq}(x \neq y, F, D) &:= \textbf{unsat}, \text{ if } F^*(x) \equiv F^*(y) \\
\textit{addneq}(x \neq y, F, D) &:= \langle F, D \rangle, \text{ if} \\
&\quad F^*(x) = F^*(u), F^*(y) = F^*(v), \\
&\quad \text{for } u \neq v \in D \text{ or } v \neq u \in D \\
\textit{addneq}(x \neq y, F, D) &:= \langle F, D \cup \{x \neq y\} \rangle, \text{ otherwise}
\end{aligned}
$$

Example

$$x_1 = x_2, x_1 = x_3, x_2 = x_3, x_2 \neq x_4, x_4 = x_5$$

$$\begin{aligned} F &= \{x_1 \mapsto x_1, x_2 \mapsto x_2, x_3 \mapsto x_3, x_4 \mapsto x_4, x_5 \mapsto x_5\} \\ D &= \{\} \end{aligned}$$

Example

$$x_1 = x_2, x_1 = x_3, x_2 = x_3, x_2 \neq x_4, x_4 = x_5$$

$$\begin{aligned} F &= \{x_1 \mapsto x_1, x_2 \mapsto x_2, x_3 \mapsto x_3, x_4 \mapsto x_4, x_5 \mapsto x_5\} \\ D &= \{\} \end{aligned}$$

Merge equivalence classes of x_1 and x_2.

Example

$$x_1 = x_2, x_1 = x_3, x_2 = x_3, x_2 \neq x_4, x_4 = x_5$$

$$\begin{aligned} F &= \{x_1 \mapsto x_1, x_2 \mapsto x_1, x_3 \mapsto x_3, x_4 \mapsto x_4, x_5 \mapsto x_5\} \\ D &= \{\} \end{aligned}$$

Example

$$x_1 = x_2, x_1 = x_3, x_2 = x_3, x_2 \neq x_4, x_4 = x_5$$

$$\begin{aligned} F &= \{x_1 \mapsto x_1, x_2 \mapsto x_1, x_3 \mapsto x_3, x_4 \mapsto x_4, x_5 \mapsto x_5\} \\ D &= \{\} \end{aligned}$$

Merge equivalence classes of x_1 and x_3.

Example

$$x_1 = x_2, x_1 = x_3, x_2 = x_3, x_2 \neq x_4, x_4 = x_5$$

$$\begin{aligned} F &= \{x_1 \mapsto x_1, x_2 \mapsto x_1, x_3 \mapsto x_1, x_4 \mapsto x_4, x_5 \mapsto x_5\} \\ D &= \{\} \end{aligned}$$

Example

$$x_1 = x_2, x_1 = x_3, x_2 = x_3, x_2 \neq x_4, x_4 = x_5$$

$$\begin{aligned} F &= \{x_1 \mapsto x_1, x_2 \mapsto x_1, x_3 \mapsto x_1, x_4 \mapsto x_4, x_5 \mapsto x_5\} \\ D &= \{\} \end{aligned}$$

Skip equality

Example

$$x_1 = x_2, x_1 = x_3, x_2 = x_3, x_2 \neq x_4, x_4 = x_5$$

$$\begin{aligned} F &= \{x_1 \mapsto x_1, x_2 \mapsto x_1, x_3 \mapsto x_1, x_4 \mapsto x_4, x_5 \mapsto x_5\} \\ D &= \{\} \end{aligned}$$

Add disequality

Example

$$x_1 = x_2, x_1 = x_3, x_2 = x_3, x_2 \neq x_4, x_4 = x_5$$

$$\begin{aligned} F &= \{x_1 \mapsto x_1, x_2 \mapsto x_1, x_3 \mapsto x_1, x_4 \mapsto x_4, x_5 \mapsto x_5\} \\ D &= \{x_2 \neq x_4\} \end{aligned}$$

Example

$$x_1 = x_2, x_1 = x_3, x_2 = x_3, x_2 \neq x_4, x_4 = x_5$$

$$\begin{aligned} F &= \{x_1 \mapsto x_1, x_2 \mapsto x_1, x_3 \mapsto x_1, x_4 \mapsto x_4, x_5 \mapsto x_5\} \\ D &= \{x_2 \neq x_4\} \end{aligned}$$

Merge equivalence classes of x_4 and x_5.

Example

$$x_1 = x_2, x_1 = x_3, x_2 = x_3, x_2 \neq x_4, x_4 = x_5$$

$$\begin{aligned} F &= \{x_1 \mapsto x_1, x_2 \mapsto x_1, x_3 \mapsto x_1, x_4 \mapsto x_4, x_5 \mapsto x_4\} \\ D &= \{x_2 \neq x_4\} \end{aligned}$$

Example

$$x_1 = x_2, x_1 = x_3, x_2 = x_3, x_2 \neq x_4, x_4 = x_5$$

$$\begin{aligned} F &= \{x_1 \mapsto x_1, x_2 \mapsto x_1, x_3 \mapsto x_1, x_4 \mapsto x_4, x_5 \mapsto x_4\} \\ D &= \{x_2 \neq x_4\} \end{aligned}$$

Model M:

$|M| = \{*_1, *_2\}$

$M(x_1), M(x_2), M(x_3) = *_1$

$M(x_4), M(x_5) = *_2$

Equality with offsets

- Many terms are equal modulo a numeric offset (e.g., $x = y + 1$).
- If these are placed in separate equivalence classes, then the equality reasoning on these terms must invoke the arithmetic module.
- We can modify the *find* data structure so that $F(x)$ returns $y + c$, and similarly $F^*(x)$.
- Example: $x_1 \neq x_2 + c$ if $F^*(x_1) = y + c_1$ and $F^*(x_2) = y + c_2$, where $c \neq c_1 - c_2$.

Retracting assertions

- Checkpointing the find data structure can be expensive.
- A disequality can be retracted by just deleting it from D.
- Retracting equality assertions is more difficult, the history of the merge operations have to be maintained.
- On retraction, the find values have to be restored.

Congruence Closure

- Equivalence is extended to *congruence* with the rule that for each n-ary function f, $f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)$ if $s_i = t_i$ for each $1 \leq 1 \leq n$.
- New index: $\pi(t)$ is the set of parents of the equivalence class rooted by t (aka use-list).
- Example:

$\{f(f(a)),\ g(a),\ a,\ g(b)\}$ $F = \{b \mapsto a, g(a) \mapsto g(b), \ldots\}$

$$\begin{aligned} \pi(a) &= \{f(a),\ g(a),\ g(b)\} \\ \pi(f(a)) &= \{f(f(a))\} \\ \pi(g(a)) &= \emptyset \\ \pi(f(f(a))) &= \emptyset \end{aligned}$$

Congruence Closure (cont.)

- As with equivalence, the *find* roots $s' = F^*(s)$ and $t' = F^*(t)$ are merged. The use lists $\pi(s')$ and $\pi(t')$ are also merged.
- How to merge use-lists?
  1. Use-lists are circular lists:
     - Constant time merge and unmerge.
  2. Use-lists are vectors:
     - Linear time merge: copy $\pi(s')$ to $\pi(t')$.
     - Constant time unmerge: shrink the vector.
  3. Do not merge: to traverse the set of parents, traverse the equivalence class.
- Any pair p_1 in $\pi(s')$ and p_2 in $\pi(t')$ that are congruent in F is added to a queue of equalities to be merged.

Congruence Closure (cont.)

- Any pair p_1 in $\pi(s')$ and p_2 in $\pi(t')$ that are congruent in F is added to a queue of equalities to be merged.
  - Naïve solution: for each p_i of $\pi(s')$ traverse $\pi(t')$ looking for a congruence p_j.
  - Efficient solution: congruence table.
    - Hashtable of ground terms.
    - Hash of $f(t_1, \ldots, t_n)$ is based on $f, F^*(t_1), \ldots, F^*(t_n)$
    - $f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)$ if $F^*(s_1) = F^*(t_1), \ldots, F^*(s_n) = F^*(t_n)$
    - The operation $F[x' := y']$ affects the hashcode of $\pi(x')$, before executing it remove terms in $\pi(x')$ from the table, and reinsert them back after.
    - Detect new congruences during reinsertion.

Example

$$f(g(a)) = c, c \neq f(g(b)), a = b$$

$$\begin{aligned}
F &= \{a \mapsto a,\ b \mapsto b,\ c \mapsto c,\ g(a) \mapsto g(a),\ g(b) \mapsto g(b) \\
&\quad f(g(a)) \mapsto f(g(a)),\ f(g(b)) \mapsto f(g(b))\} \\
D &= \{\} \\
\pi(a) &= \{g(a)\} \\
\pi(b) &= \{g(b)\} \\
\pi(g(a)) &= \{f(g(a))\} \\
\pi(g(b)) &= \{f(g(b))\}
\end{aligned}$$

Example

$$f(g(a)) = c, c \neq f(g(b)), a = b$$

$$\begin{aligned}
F &= \{a \mapsto a,\ b \mapsto b,\ c \mapsto c,\ g(a) \mapsto g(a),\ g(b) \mapsto g(b) \\
&\quad f(g(a)) \mapsto f(g(a)),\ f(g(b)) \mapsto f(g(b))\} \\
D &= \{\} \\
\pi(a) &= \{g(a)\} \\
\pi(b) &= \{g(b)\} \\
\pi(g(a)) &= \{f(g(a))\} \\
\pi(g(b)) &= \{f(g(b))\}
\end{aligned}$$

Merge equivalence classes of $f(g(a))$ and c.

Example

$$f(g(a)) = c, c \neq f(g(b)), a = b$$

$$\begin{aligned}
F &= \{a \mapsto a,\ b \mapsto b,\ c \mapsto c,\ g(a) \mapsto g(a),\ g(b) \mapsto g(b) \\
&\quad f(g(a)) \mapsto c,\ f(g(b)) \mapsto f(g(b))\} \\
D &= \{\} \\
\pi(a) &= \{g(a)\} \\
\pi(b) &= \{g(b)\} \\
\pi(g(a)) &= \{f(g(a))\} \\
\pi(g(b)) &= \{f(g(b))\}
\end{aligned}$$

Example

$$f(g(a)) = c, c \neq f(g(b)), a = b$$

$$\begin{aligned}
F &= \{a \mapsto a,\ b \mapsto b,\ c \mapsto c,\ g(a) \mapsto g(a),\ g(b) \mapsto g(b) \\
&\quad f(g(a)) \mapsto c,\ f(g(b)) \mapsto f(g(b))\} \\
D &= \{\} \\
\pi(a) &= \{g(a)\} \\
\pi(b) &= \{g(b)\} \\
\pi(g(a)) &= \{f(g(a))\} \\
\pi(g(b)) &= \{f(g(b))\}
\end{aligned}$$

Add disequality

Example

$$f(g(a)) = c, c \neq f(g(b)), a = b$$

$$\begin{aligned}
F &= \{a \mapsto a,\ b \mapsto b,\ c \mapsto c,\ g(a) \mapsto g(a),\ g(b) \mapsto g(b) \\
&\quad f(g(a)) \mapsto c,\ f(g(b)) \mapsto f(g(b))\} \\
D &= \{c \neq f(g(b))\} \\
\pi(a) &= \{g(a)\} \\
\pi(b) &= \{g(b)\} \\
\pi(g(a)) &= \{f(g(a))\} \\
\pi(g(b)) &= \{f(g(b))\}
\end{aligned}$$

Example

$$f(g(a)) = c, c \neq f(g(b)), a = b$$

$$\begin{aligned}
F &= \{a \mapsto a,\ b \mapsto b,\ c \mapsto c,\ g(a) \mapsto g(a),\ g(b) \mapsto g(b) \\
&\quad f(g(a)) \mapsto c,\ f(g(b)) \mapsto f(g(b))\} \\
D &= \{c \neq f(g(b))\} \\
\pi(a) &= \{g(a)\} \\
\pi(b) &= \{g(b)\} \\
\pi(g(a)) &= \{f(g(a))\} \\
\pi(g(b)) &= \{f(g(b))\}
\end{aligned}$$

Merge equivalence classes of a and b.

Example

$$f(g(a)) = c, c \neq f(g(b)), a = b, g(a) = g(b)$$

$$\begin{aligned}
F &= \{a \mapsto a,\ b \mapsto a,\ c \mapsto c,\ g(a) \mapsto g(a),\ g(b) \mapsto g(b) \\
&\quad f(g(a)) \mapsto c,\ f(g(b)) \mapsto f(g(b))\} \\
D &= \{c \neq f(g(b))\} \\
\pi(a) &= \{g(a), g(b)\} \\
\pi(b) &= \{g(b)\} \\
\pi(g(a)) &= \{f(g(a))\} \\
\pi(g(b)) &= \{f(g(b))\}
\end{aligned}$$

Example

$$f(g(a)) = c, c \neq f(g(b)), a = b, g(a) = g(b)$$

$$\begin{aligned}
F &= \{a \mapsto a,\ b \mapsto a,\ c \mapsto c,\ g(a) \mapsto g(a),\ g(b) \mapsto g(b) \\
&\quad f(g(a)) \mapsto c,\ f(g(b)) \mapsto f(g(b))\} \\
D &= \{c \neq f(g(b))\} \\
\pi(a) &= \{g(a), g(b)\} \\
\pi(b) &= \{g(b)\} \\
\pi(g(a)) &= \{f(g(a))\} \\
\pi(g(b)) &= \{f(g(b))\}
\end{aligned}$$

Merge equivalence classes of $g(a)$ and $g(b)$.

Example

$$f(g(a)) = c, c \neq f(g(b)), a = b, g(a) = g(b), f(g(a)) = f(g(b))$$

$$\begin{aligned}
F &= \{a \mapsto a,\ b \mapsto a,\ c \mapsto c,\ g(a) \mapsto g(b),\ g(b) \mapsto g(b) \\
&\quad f(g(a)) \mapsto c,\ f(g(b)) \mapsto f(g(b))\} \\
D &= \{c \neq f(g(b))\} \\
\pi(a) &= \{g(a), g(b)\} \\
\pi(b) &= \{g(b)\} \\
\pi(g(a)) &= \{f(g(a))\} \\
\pi(g(b)) &= \{f(g(b)), f(g(a))\}
\end{aligned}$$

Example

$$f(g(a)) = c, c \neq f(g(b)), a = b, g(a) = g(b), f(g(a)) = f(g(b))$$

$$\begin{aligned}
F &= \{a \mapsto a,\ b \mapsto a,\ c \mapsto c,\ g(a) \mapsto g(b),\ g(b) \mapsto g(b) \\
&\quad\ f(g(a)) \mapsto c,\ f(g(b)) \mapsto f(g(b))\} \\
D &= \{c \neq f(g(b))\} \\
\pi(a) &= \{g(a), g(b)\} \\
\pi(b) &= \{g(b)\} \\
\pi(g(a)) &= \{f(g(a))\} \\
\pi(g(b)) &= \{f(g(b)), f(g(a))\}
\end{aligned}$$

Merge equivalence classes of $f(g(a))$ and $f(g(b))$ $\rightsquigarrow$ **unsat**.

Example: Satisfiable Version

$$f(g(a)) = c, a \neq f(g(b)), a = b, g(a) = g(b), f(g(a)) = f(g(b))$$

$$\begin{aligned} F &= \{a \mapsto a,\ b \mapsto a,\ c \mapsto c,\ g(a) \mapsto g(b),\ g(b) \mapsto g(b) \\ &\quad\ f(g(a)) \mapsto c,\ f(g(b)) \mapsto c\} \\ D &= \{a \neq f(g(b))\} \end{aligned}$$

Example: Satisfiable Version

$$f(g(a)) = c, a \neq f(g(b)), a = b, g(a) = g(b), f(g(a)) = f(g(b))$$

$$\begin{aligned} F &= \{a \mapsto a,\ b \mapsto a,\ c \mapsto c,\ g(a) \mapsto g(b),\ g(b) \mapsto g(b) \\ &\quad\ f(g(a)) \mapsto c,\ f(g(b)) \mapsto c\} \\ D &= \{a \neq f(g(b))\} \end{aligned}$$

Model: $|M| = \{*_1, *_2, *_3\}$ One value for each eq. class root.

$M(a) = M(b) = *_1$

$M(c) = *_2$

$M(g) = \{*_1 \mapsto *_3, \mathit{else} \mapsto *_?\}$ $*_?$ can be any value.

$M(f) = \{*_3 \mapsto *_2, \mathit{else} \mapsto *_?\}$

Example: Satisfiable Version

$$f(g(a)) = c, a \neq f(g(b)), a = b, g(a) = g(b), f(g(a)) = f(g(b))$$

$$\begin{aligned} F &= \{a \mapsto a,\ b \mapsto a,\ c \mapsto c,\ g(a) \mapsto g(b),\ g(b) \mapsto g(b) \\ &\quad\ f(g(a)) \mapsto c,\ f(g(b)) \mapsto c\} \\ D &= \{a \neq f(g(b))\} \end{aligned}$$

Model: $|M| = \{*_1, *_2, *_3\}$ One value for each eq. class root.

$M(a) = M(b) = *_1$

$M(c) = *_2$

$M(g) = \{*_1 \mapsto *_3, \mathit{else} \mapsto *_?\}$ $*_?$ can be any value.

$M(f) = \{*_3 \mapsto *_2, \mathit{else} \mapsto *_?\}$

Equality: T-Justifications

- A T-Justification for F is a set of literals S such that $S \models_T F$.
- S is a non-redudant if there is no $S' \subset S$ such that $S' \models_T F$.
- Non-redundant T-Justifications for variable equalities is easy: shortest-path between two variables.
- With uninterpreted functions the problem is more difficult:
- Example:

$$
\begin{aligned}
f_1(x_1) &= x_1 = x_2 = f_1(x_{n+1}), \\
&\ldots, \\
f_n(x_1) &= x_n = x_{n+1} = f_n(x_{n+1}), \\
g(f_1(x_1), \ldots, f_n(x_1)) &\neq g(f_1(x_{n+1}), \ldots, f_n(x_{n+1}))
\end{aligned}
$$

Roadmap

- Background
- SAT & SMT
- Combining theories
- Equality
- Arithmetic
- Quantifiers
- Applications

Linear Arithmetic

- Algorithms:
  - Graph based for difference logic ($x \leq y - k$).
  - Fourier-Motzkin elimination.

    $t_1 \leq ax, \;\; bx \leq t_2 \;\; \Rightarrow \;\; bt_1 \leq at_2$
  - Standard Simplex.
- Standard Simplex based solvers:
  - Standard Form: $Ax = b$ and $x \geq 0$.
  - Incremental: add/remove equations (i.e., rows).
  - Slow backtracking.
  - No theory propagation.

Fast Linear Arithmetic

- Simplex General Form.
- Algorithm based on the Dual Simplex.
- Non-redundant T-Justifications.
- Efficient Backtracking.
- Efficient T-Propagate.
- Support for strict inequalities ($t > 0$).
- Presimplification step.
- Integer problems: Gomory cuts, Branch & Bound, GCD test.

General Form

- General Form: $Ax = 0$ and $l_j \leq x_j \leq u_j$
- Example:

$$x \geq 0, (x + y \leq 2 \vee x + 2y \geq 6), (x + y = 2 \vee x + 2y > 4)$$

$$\rightsquigarrow$$

$$s_1 = x + y, s_2 = x + 2y,$$

$$x \geq 0, (s_1 \leq 2 \vee s_2 \geq 6), (s_1 = 2 \vee s_2 > 4)$$

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

Model + Equations + Bounds

- An assignment (model) is a mapping from variables to values.
- We maintain an assignment that satisfies all equations and bounds.
- The assignment of non dependent variables implies the assignment of dependent variables.
- Equations + Bounds can be used to derive new bounds.
- Example: $x = y - z,\ y \leq 2,\ z \geq 3 \rightsquigarrow x \leq -1$.
- The new bound may be inconsistent with the already known bounds.
- Example: $x \leq -1,\ x \geq 0$.

Strict Inequalities

- The method described only handles non-strict inequalities (e.g., $x \leq 2$).
- For integer problems, strict inequalities can be converted into non-strict inequalities. $x < 1 \rightsquigarrow x \leq 0$.
- For rational/real problems, strict inequalities can be converted into non-strict inequalities using a small δ. $x < 1 \rightsquigarrow x \leq 1 - \delta$.
- We do not compute a δ, we treat it symbolically.
- δ is an infinitesimal parameter: $(c, k) = c + k\delta$

Example

- Initial state

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$s = x + y$	
$M(y) = 0$	$u = x + 2y$	
$M(s) = 0$	$v = x - y$	
$M(u) = 0$		
$M(v) = 0$		

Example

- Asserting $s \geq 1$

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$s = x + y$	
$M(y) = 0$	$u = x + 2y$	
$M(s) = 0$	$v = x - y$	
$M(u) = 0$		
$M(v) = 0$		

Example

- Asserting $s \geq 1$ assignment does not satisfy new bound.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	0	s	$=$	$x + y$	s	$\geq$	1
$M(y)$	$=$	0	u	$=$	$x + 2y$			
$M(s)$	$=$	0	v	$=$	$x - y$			
$M(u)$	$=$	0						
$M(v)$	$=$	0						

Example

- Asserting $s \geq 1$ pivot s and x (s is a dependent variable).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$s = x + y$	$s \geq 1$
$M(y) = 0$	$u = x + 2y$	
$M(s) = 0$	$v = x - y$	
$M(u) = 0$		
$M(v) = 0$		

Example

- Asserting $s \geq 1$ pivot s and x (s is a dependent variable).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$x = s - y$	$s \geq 1$
$M(y) = 0$	$u = x + 2y$	
$M(s) = 0$	$v = x - y$	
$M(u) = 0$		
$M(v) = 0$		

Example

- Asserting $s \geq 1$ pivot s and x (s is a dependent variable).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$x = s - y$	$s \geq 1$
$M(y) = 0$	$u = s + y$	
$M(s) = 0$	$v = s - 2y$	
$M(u) = 0$		
$M(v) = 0$		

Example

- Asserting $s \geq 1$ update assignment.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	0	x	$=$	$s - y$	s	$\geq$	1
$M(y)$	$=$	0	u	$=$	$s + y$			
$M(s)$	$=$	1	v	$=$	$s - 2y$			
$M(u)$	$=$	0						
$M(v)$	$=$	0						

Example

- Asserting $s \geq 1$ update dependent variables assignment.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 1$	$x = s - y$	$s \geq 1$
$M(y) = 0$	$u = s + y$	
$M(s) = 1$	$v = s - 2y$	
$M(u) = 1$		
$M(v) = 1$		

Example

- Asserting $x \geq 0$

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 1$	$x = s - y$	$s \geq 1$
$M(y) = 0$	$u = s + y$	
$M(s) = 1$	$v = s - 2y$	
$M(u) = 1$		
$M(v) = 1$		

Example

- Asserting $x \geq 0$ assignment satisfies new bound.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 1$	$x = s - y$	$s \geq 1$
$M(y) = 0$	$u = s + y$	$x \geq 0$
$M(s) = 1$	$v = s - 2y$	
$M(u) = 1$		
$M(v) = 1$		

Example

- Case split $\neg y \leq 1$

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 1$	$x = s - y$	$s \geq 1$
$M(y) = 0$	$u = s + y$	$x \geq 0$
$M(s) = 1$	$v = s - 2y$	
$M(u) = 1$		
$M(v) = 1$		

Example

- Case split $\neg y \leq 1$ assignment does not satisfies new bound.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 1$	$x = s - y$	$s \geq 1$
$M(y) = 0$	$u = s + y$	$x \geq 0$
$M(s) = 1$	$v = s - 2y$	$y > 1$
$M(u) = 1$		
$M(v) = 1$		

Example

- Case split $\neg y \leq 1$ update assignment.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	1	x	$=$	$s - y$	s	$\geq$	1
$M(y)$	$=$	$1 + \delta$	u	$=$	$s + y$	x	$\geq$	0
$M(s)$	$=$	1	v	$=$	$s - 2y$	y	$>$	1
$M(u)$	$=$	1						
$M(v)$	$=$	1						

Example

- Case split $\neg y \leq 1$ update dependent variables assignment.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	$-\delta$	x	$=$	$s - y$	s	$\geq$	1
$M(y)$	$=$	$1 + \delta$	u	$=$	$s + y$	x	$\geq$	0
$M(s)$	$=$	1	v	$=$	$s - 2y$	y	$>$	1
$M(u)$	$=$	$2 + \delta$						
$M(v)$	$=$	$-1 - 2\delta$						

Example

- Bound violation

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	$-\delta$	x	$=$	$s - y$	s	$\geq$	1
$M(y)$	$=$	$1 + \delta$	u	$=$	$s + y$	x	$\geq$	0
$M(s)$	$=$	1	v	$=$	$s - 2y$	y	$>$	1
$M(u)$	$=$	$2 + \delta$						
$M(v)$	$=$	$-1 - 2\delta$						

Example

- Bound violation pivot x and s (x is a dependent variables).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	$-\delta$	x	$=$	$s - y$	s	$\geq$	1
$M(y)$	$=$	$1 + \delta$	u	$=$	$s + y$	x	$\geq$	0
$M(s)$	$=$	1	v	$=$	$s - 2y$	y	$>$	1
$M(u)$	$=$	$2 + \delta$						
$M(v)$	$=$	$-1 - 2\delta$						

Example

- Bound violation pivot x and s (x is a dependent variables).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	$-\delta$	s	$=$	$x + y$	s	$\geq$	1
$M(y)$	$=$	$1 + \delta$	u	$=$	$s + y$	x	$\geq$	0
$M(s)$	$=$	1	v	$=$	$s - 2y$	y	$>$	1
$M(u)$	$=$	$2 + \delta$						
$M(v)$	$=$	$-1 - 2\delta$						

Example

- Bound violation pivot x and s (x is a dependent variables).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	$-\delta$	s	$=$	$x + y$	s	$\geq$	1
$M(y)$	$=$	$1 + \delta$	u	$=$	$x + 2y$	x	$\geq$	0
$M(s)$	$=$	1	v	$=$	$x - y$	y	$>$	1
$M(u)$	$=$	$2 + \delta$						
$M(v)$	$=$	$-1 - 2\delta$						

Example

- Bound violation update assignment.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	0	s	$=$	$x + y$	s	$\geq$	1
$M(y)$	$=$	$1 + \delta$	u	$=$	$x + 2y$	x	$\geq$	0
$M(s)$	$=$	1	v	$=$	$x - y$	y	$>$	1
$M(u)$	$=$	$2 + \delta$						
$M(v)$	$=$	$-1 - 2\delta$						

Example

- Bound violation update dependent variables assignment.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	0	s	$=$	$x + y$	s	$\geq$	1
$M(y)$	$=$	$1 + \delta$	u	$=$	$x + 2y$	x	$\geq$	0
$M(s)$	$=$	$1 + \delta$	v	$=$	$x - y$	y	$>$	1
$M(u)$	$=$	$2 + 2\delta$						
$M(v)$	$=$	$-1 - \delta$						

Example

- Theory propagation $x \geq 0, y > 1 \rightsquigarrow u > 2$

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	0	s	$=$	$x + y$	s	$\geq$	1
$M(y)$	$=$	$1 + \delta$	u	$=$	$x + 2y$	x	$\geq$	0
$M(s)$	$=$	$1 + \delta$	v	$=$	$x - y$	y	$>$	1
$M(u)$	$=$	$2 + 2\delta$						
$M(v)$	$=$	$-1 - \delta$						

Example

- Theory propagation $u > 2 \rightsquigarrow \neg u \leq -1$

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	0	s	$=$	$x + y$	s	$\geq$	1
$M(y)$	$=$	$1 + \delta$	u	$=$	$x + 2y$	x	$\geq$	0
$M(s)$	$=$	$1 + \delta$	v	$=$	$x - y$	y	$>$	1
$M(u)$	$=$	$2 + 2\delta$				u	$>$	2
$M(v)$	$=$	$-1 - \delta$						

Example

- Boolean propagation $\neg y \leq 1 \rightsquigarrow v \geq 2$

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	0	s	$=$	$x + y$	s	$\geq$	1
$M(y)$	$=$	$1 + \delta$	u	$=$	$x + 2y$	x	$\geq$	0
$M(s)$	$=$	$1 + \delta$	v	$=$	$x - y$	y	$>$	1
$M(u)$	$=$	$2 + 2\delta$				u	$>$	2
$M(v)$	$=$	$-1 - \delta$						

Example

- Theory propagation $v \geq 2 \rightsquigarrow \neg v \leq -2$

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	0	s	$=$	$x + y$	s	$\geq$	1
$M(y)$	$=$	$1 + \delta$	u	$=$	$x + 2y$	x	$\geq$	0
$M(s)$	$=$	$1 + \delta$	v	$=$	$x - y$	y	$>$	1
$M(u)$	$=$	$2 + 2\delta$				u	$>$	2
$M(v)$	$=$	$-1 - \delta$						

Example

- Conflict empty clause

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	0	s	$=$	$x + y$	s	$\geq$	1
$M(y)$	$=$	$1 + \delta$	u	$=$	$x + 2y$	x	$\geq$	0
$M(s)$	$=$	$1 + \delta$	v	$=$	$x - y$	y	$>$	1
$M(u)$	$=$	$2 + 2\delta$				u	$>$	2
$M(v)$	$=$	$-1 - \delta$						

Example

- Backtracking

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	0	s	$=$	$x+y$	s	$\geq$	1
$M(y)$	$=$	$1+\delta$	u	$=$	$x+2y$	x	$\geq$	0
$M(s)$	$=$	$1+\delta$	v	$=$	$x-y$			
$M(u)$	$=$	$2+2\delta$						
$M(v)$	$=$	$-1-\delta$						

Example

- Asserting $y \leq 1$

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	0	s	$=$	$x + y$	s	$\geq$	1
$M(y)$	$=$	$1 + \delta$	u	$=$	$x + 2y$	x	$\geq$	0
$M(s)$	$=$	$1 + \delta$	v	$=$	$x - y$			
$M(u)$	$=$	$2 + 2\delta$						
$M(v)$	$=$	$-1 - \delta$						

Example

- Asserting $y \leq 1$ assignment does not satisfy new bound.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	0	s	$=$	$x + y$	s	$\geq$	1
$M(y)$	$=$	$1 + \delta$	u	$=$	$x + 2y$	x	$\geq$	0
$M(s)$	$=$	$1 + \delta$	v	$=$	$x - y$	y	$\leq$	1
$M(u)$	$=$	$2 + 2\delta$						
$M(v)$	$=$	$-1 - \delta$						

Example

- Asserting $y \leq 1$ update assignment.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	0	s	$=$	$x + y$	s	$\geq$	1
$M(y)$	$=$	1	u	$=$	$x + 2y$	x	$\geq$	0
$M(s)$	$=$	$1 + \delta$	v	$=$	$x - y$	y	$\leq$	1
$M(u)$	$=$	$2 + 2\delta$						
$M(v)$	$=$	$-1 - \delta$						

Example

- Asserting $y \leq 1$ update dependent variables assignment.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$s = x + y$	$s \geq 1$
$M(y) = 1$	$u = x + 2y$	$x \geq 0$
$M(s) = 1$	$v = x - y$	$y \leq 1$
$M(u) = 2$		
$M(v) = -1$		

Example

- Theory propagation $s \geq 1, y \leq 1 \rightsquigarrow v \geq -1$

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$x = s - y$	$s \geq 1$
$M(y) = 1$	$u = s + y$	$x \geq 0$
$M(s) = 1$	$v = s - 2y$	$y \leq 1$
$M(u) = 2$		
$M(v) = -1$		

Example

- Theory propagation $v \geq -1 \rightsquigarrow \neg v \leq -2$

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$x = s - y$	$s \geq 1$
$M(y) = 1$	$u = s + y$	$x \geq 0$
$M(s) = 1$	$v = s - 2y$	$y \leq 1$
$M(u) = 2$		$v \geq -1$
$M(v) = -1$		

Example

- Boolean propagation $\neg v \leq -2 \rightsquigarrow v \geq 0$

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$x = s - y$	$s \geq 1$
$M(y) = 1$	$u = s + y$	$x \geq 0$
$M(s) = 1$	$v = s - 2y$	$y \leq 1$
$M(u) = 2$		$v \geq -1$
$M(v) = -1$		

Example

- Bound violation assignment does not satisfy new bound.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$x = s - y$	$s \geq 1$
$M(y) = 1$	$u = s + y$	$x \geq 0$
$M(s) = 1$	$v = s - 2y$	$y \leq 1$
$M(u) = 2$		$v \geq 0$
$M(v) = -1$		

Example

- Bound violation pivot u and s (u is a dependent variable).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	0	x	$=$	$s - y$	s	$\geq$	1
$M(y)$	$=$	1	u	$=$	$s + y$	x	$\geq$	0
$M(s)$	$=$	1	v	$=$	$s - 2y$	y	$\leq$	1
$M(u)$	$=$	2				v	$\geq$	0
$M(v)$	$=$	-1						

Example

- Bound violation pivot u and s (u is a dependent variable).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	0	x	$=$	$s - y$	s	$\geq$	1
$M(y)$	$=$	1	u	$=$	$s + y$	x	$\geq$	0
$M(s)$	$=$	1	s	$=$	$v + 2y$	y	$\leq$	1
$M(u)$	$=$	2				v	$\geq$	0
$M(v)$	$=$	-1						

Example

- Bound violation pivot u and s (u is a dependent variable).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	0	x	$=$	$v + y$	s	$\geq$	1
$M(y)$	$=$	1	u	$=$	$v + 3y$	x	$\geq$	0
$M(s)$	$=$	1	s	$=$	$v + 2y$	y	$\leq$	1
$M(u)$	$=$	2				v	$\geq$	0
$M(v)$	$=$	-1						

Example

- Bound violation update assignment.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 0$	$x = v + y$	$s \geq 1$
$M(y) = 1$	$u = v + 3y$	$x \geq 0$
$M(s) = 1$	$s = v + 2y$	$y \leq 1$
$M(u) = 2$		$v \geq 0$
$M(v) = 0$		

Example

- Bound violation update dependent variables assignment.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 1$	$x = v + y$	$s \geq 1$
$M(y) = 1$	$u = v + 3y$	$x \geq 0$
$M(s) = 2$	$s = v + 2y$	$y \leq 1$
$M(u) = 3$		$v \geq 0$
$M(v) = 0$		

Example

- Boolean propagation $\neg v \leq -2 \rightsquigarrow u \leq -1$

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 1$	$x = v + y$	$s \geq 1$
$M(y) = 1$	$u = v + 3y$	$x \geq 0$
$M(s) = 2$	$s = v + 2y$	$y \leq 1$
$M(u) = 3$		$v \geq 0$
$M(v) = 0$		

Example

- Bound violation assignment does not satisfy new bound.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	1	x	$=$	$v + y$	s	$\geq$	1
$M(y)$	$=$	1	u	$=$	$v + 3y$	x	$\geq$	0
$M(s)$	$=$	2	s	$=$	$v + 2y$	y	$\leq$	1
$M(u)$	$=$	3				v	$\geq$	0
$M(v)$	$=$	0				u	$\leq$	-1

Example

- Bound violation pivot u and y (u is a dependent variable).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	1	x	$=$	$v + y$	s	$\geq$	1
$M(y)$	$=$	1	u	$=$	$v + 3y$	x	$\geq$	0
$M(s)$	$=$	2	s	$=$	$v + 2y$	y	$\leq$	1
$M(u)$	$=$	3				v	$\geq$	0
$M(v)$	$=$	0				u	$\leq$	-1

Example

- Bound violation pivot u and y (u is a dependent variable).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	1	x	$=$	$v + y$	s	$\geq$	1
$M(y)$	$=$	1	y	$=$	$\frac{1}{3}u - \frac{1}{3}v$	x	$\geq$	0
$M(s)$	$=$	2	s	$=$	$v + 2y$	y	$\leq$	1
$M(u)$	$=$	3				v	$\geq$	0
$M(v)$	$=$	0				u	$\leq$	-1

Example

- Bound violation pivot u and y (u is a dependent variable).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	1	x	$=$	$\frac{1}{3}u + \frac{2}{3}v$	s	$\geq$	1
$M(y)$	$=$	1	y	$=$	$\frac{1}{3}u - \frac{1}{3}v$	x	$\geq$	0
$M(s)$	$=$	2	s	$=$	$\frac{2}{3}u + \frac{1}{3}v$	y	$\leq$	1
$M(u)$	$=$	3				v	$\geq$	0
$M(v)$	$=$	0				u	$\leq$	-1

Example

- Bound violation update assignment.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	1	x	$=$	$\frac{1}{3}u + \frac{2}{3}v$	s	$\geq$	1
$M(y)$	$=$	1	y	$=$	$\frac{1}{3}u - \frac{1}{3}v$	x	$\geq$	0
$M(s)$	$=$	2	s	$=$	$\frac{2}{3}u + \frac{1}{3}v$	y	$\leq$	1
$M(u)$	$=$	-1				v	$\geq$	0
$M(v)$	$=$	0				u	$\leq$	-1

Example

- Bound violation update dependent variables assignment.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	$-\frac{1}{3}$	x	$=$	$\frac{1}{3}u + \frac{2}{3}v$	s	$\geq$	1
$M(y)$	$=$	$-\frac{1}{3}$	y	$=$	$\frac{1}{3}u - \frac{1}{3}v$	x	$\geq$	0
$M(s)$	$=$	$-\frac{2}{3}$	s	$=$	$\frac{2}{3}u + \frac{1}{3}v$	y	$\leq$	1
$M(u)$	$=$	-1				v	$\geq$	0
$M(v)$	$=$	0				u	$\leq$	-1

Example

- Bound violations

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	$-\frac{1}{3}$	x	$=$	$\frac{1}{3}u + \frac{2}{3}v$	s	$\geq$	1
$M(y)$	$=$	$-\frac{1}{3}$	y	$=$	$\frac{1}{3}u - \frac{1}{3}v$	x	$\geq$	0
$M(s)$	$=$	$-\frac{2}{3}$	s	$=$	$\frac{2}{3}u + \frac{1}{3}v$	y	$\leq$	1
$M(u)$	$=$	-1				v	$\geq$	0
$M(v)$	$=$	0				u	$\leq$	-1

Example

- Bound violations pivot s and v (s is a dependent variable).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model		Equations		Bounds		
$M(x) = -\frac{1}{3}$		$x = \frac{1}{3}u + \frac{2}{3}v$		$s \geq 1$		
$M(y) = -\frac{1}{3}$		$y = \frac{1}{3}u - \frac{1}{3}v$		$x \geq 0$		
$M(s) = -\frac{2}{3}$		$s = \frac{2}{3}u + \frac{1}{3}v$		$y \leq 1$		
$M(u) = -1$				$v \geq 0$		
$M(v) = 0$				$u \leq -1$		

Example

- Bound violations pivot s and v (s is a dependent variable).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	$-\frac{1}{3}$	x	$=$	$\frac{1}{3}u + \frac{2}{3}v$	s	$\geq$	1
$M(y)$	$=$	$-\frac{1}{3}$	y	$=$	$\frac{1}{3}u - \frac{1}{3}v$	x	$\geq$	0
$M(s)$	$=$	$-\frac{2}{3}$	v	$=$	$3s - 2u$	y	$\leq$	1
$M(u)$	$=$	-1				v	$\geq$	0
$M(v)$	$=$	0				u	$\leq$	-1

Example

- Bound violations pivot s and v (s is a dependent variable).

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	$-\frac{1}{3}$	x	$=$	$2s - u$	s	$\geq$	1
$M(y)$	$=$	$-\frac{1}{3}$	y	$=$	$-s + u$	x	$\geq$	0
$M(s)$	$=$	$-\frac{2}{3}$	v	$=$	$3s - 2u$	y	$\leq$	1
$M(u)$	$=$	-1				v	$\geq$	0
$M(v)$	$=$	0				u	$\leq$	-1

Example

- Bound violations update assignment.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	$-\frac{1}{3}$	x	$=$	$2s - u$	s	$\geq$	1
$M(y)$	$=$	$-\frac{1}{3}$	y	$=$	$-s + u$	x	$\geq$	0
$M(s)$	$=$	1	v	$=$	$3s - 2u$	y	$\leq$	1
$M(u)$	$=$	-1				v	$\geq$	0
$M(v)$	$=$	0				u	$\leq$	-1

Example

- Bound violations update dependent variables assignment.

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model	Equations	Bounds
$M(x) = 3$	$x = 2s - u$	$s \geq 1$
$M(y) = -2$	$y = -s + u$	$x \geq 0$
$M(s) = 1$	$v = 3s - 2u$	$y \leq 1$
$M(u) = -1$		$v \geq 0$
$M(v) = 5$		$u \leq -1$

Example

- Found satisfying assignment

$$s \geq 1, x \geq 0$$

$$(y \leq 1 \vee v \geq 2), (v \leq -2 \vee v \geq 0), (v \leq -2 \vee u \leq -1)$$

Model			Equations			Bounds		
$M(x)$	$=$	3	x	$=$	$2s - u$	s	$\geq$	1
$M(y)$	$=$	-2	y	$=$	$-s + u$	x	$\geq$	0
$M(s)$	$=$	1	v	$=$	$3s - 2u$	y	$\leq$	1
$M(u)$	$=$	-1				v	$\geq$	0
$M(v)$	$=$	5				u	$\leq$	-1

Questions

- Indexing: pivoting?

Questions

- Indexing: pivoting?
- Does it terminate?

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 $P = \sum_{x_j \in \mathcal{V}} a_{ij} x_j$, and a model M: $M(P)$ denotes $\sum_{x_j \in \mathcal{V}} a_{ij} M(x_j)$.

Opportunistic equality propagation

- Equality propagation in arithmetic:

FixedEq

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

EqRow

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

EqOffsetRows

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

EqRows

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

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.

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

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

Linear Integer Arithmetic

- GCD test
- Gomory Cuts
- Branch and Bound

Beyond Linear Arithmetic

- Gröbner Basis
- Cylindric Algebraic Decomposition

Roadmap

- Background
- SAT & SMT
- Combining theories
- Equality
- Arithmetic
- Quantifiers
- Applications

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/handling universal quantifiers for detecting unsatisfiability remains.
- Approaches:
  - Heuristic instantiation (E-matching).
  - Decidable fragments.
  - SMT + Superposition Calculus.

Negation Normal Form (NNF)

$$
\begin{aligned}
\mathit{NNF}(p) &= p \\
\mathit{NNF}(\neg p) &= \neg p \\
\mathit{NNF}(\neg\neg\phi) &= \mathit{NNF}(\phi) \\
\mathit{NNF}(\phi_0 \vee \phi_1) &= \mathit{NNF}(\phi_0) \vee \mathit{NNF}(\phi_1) \\
\mathit{NNF}(\neg(\phi_0 \vee \phi_1)) &= \mathit{NNF}(\neg\phi_0) \wedge \mathit{NNF}(\neg\phi_1) \\
\mathit{NNF}(\phi_0 \wedge \phi_1) &= \mathit{NNF}(\phi_0) \wedge \mathit{NNF}(\phi_1) \\
\mathit{NNF}(\neg(\phi_0 \wedge \phi_1)) &= \mathit{NNF}(\neg\phi_0) \vee \mathit{NNF}(\neg\phi_1) \\
\mathit{NNF}(\forall x : \phi) &= \forall x : \mathit{NNF}(\phi) \\
\mathit{NNF}(\neg(\forall x : \phi)) &= \exists x : \mathit{NNF}(\neg\phi) \\
\mathit{NNF}(\exists x : \phi) &= \exists x : \mathit{NNF}(\phi) \\
\mathit{NNF}(\neg(\exists x : \phi)) &= \forall x : \mathit{NNF}(\neg\phi)
\end{aligned}
$$

Theorem: $F \Leftrightarrow \mathit{NNF}(F)$

Ex.: $\mathit{NNF}(\neg(p \wedge (\neg r \vee \forall x : q(x)))) = \neg p \vee (r \wedge \exists x : \neg q(x))$.

Skolemization

- After NNF, Skolemization can be used to eliminate existential quantifiers.
- $\exists y : F[x, y] \quad \rightsquigarrow \quad F[x, f(x)]$
- The resultant formula is equisatisfiable.
- Example:

$$\forall x : p(x) \Rightarrow \exists y : q(x, y)$$

$$\forall x : p(x) \Rightarrow q(x, f(x))$$

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 problem

Input: A set of ground equations E, a ground term t, and a pattern p, where p possibly contains variables.

Output: The set of substitutions β over the variables in p, such that:

$$E \models t = \beta(p)$$

E-matching problem

Input: A set of ground equations E, a ground term t, and a pattern p, where p possibly contains variables.

Output: The set of substitutions β over the variables in p, such that:

$$E \models t = \beta(p)$$

Example:

$$\begin{aligned} E &\equiv \{a = f(b),\ a = f(c)\} \\ t &\equiv g(a) \\ p &\equiv g(f(x)) \end{aligned}$$

E-matching problem

Input: A set of ground equations E, a ground term t, and a pattern p, where p possibly contains variables.

Output: The set of substitutions β over the variables in p, such that:

$$E \models t = \beta(p)$$

Example:

$$\begin{aligned} E &\equiv \{a = f(b),\ a = f(c)\} \\ t &\equiv g(a) \\ p &\equiv g(f(x)) \\ R &\equiv \{\underbrace{\{x \mapsto b\}}_{\beta_1}, \underbrace{\{x \mapsto c\}}_{\beta_2}\} \end{aligned}$$

E-matching problem

Input: A set of ground equations E, a ground term t, and a pattern p, where p possibly contains variables.

Output: The set of substitutions β over the variables in p, such that:

$$E \models t = \beta(p)$$

Example:

$$\begin{aligned} E &\equiv \{a = f(b),\ a = f(c)\} \\ t &\equiv g(a) \\ p &\equiv g(f(x)) \\ R &\equiv \{\underbrace{\{x \mapsto b\}}_{\beta_1}, \underbrace{\{x \mapsto c\}}_{\beta_2}\} \end{aligned}$$

Applying β_1: $a = f(b), a = f(c) \models g(a) = g(f(b))$

E-matching problem

Input: A set of ground equations E, a ground term t, and a pattern p, where p possibly contains variables.

Output: The set of substitutions β over the variables in p, such that:

$$E \models t = \beta(p)$$

Example:

$$\begin{aligned} E &\equiv \{a = f(b),\ a = f(c)\} \\ t &\equiv g(a) \\ p &\equiv g(f(x)) \\ R &\equiv \{\underbrace{\{x \mapsto b\}}_{\beta_1}, \underbrace{\{x \mapsto c\}}_{\beta_2}\} \end{aligned}$$

Applying β_2: $a = f(b), a = f(c) \models g(a) = g(f(c))$

The E-matching challenge

- E-matching is NP-hard.
- The number of matches can be exponential.
- It is not refutationally complete.
- The real challenge is finding new matches:
  - Incrementally during backtracking search.
  - In a large database of patterns, many share substantial structure.

E-matching

$$
\begin{aligned}
\mathit{match}(x, t, S) &= \{\beta \cup \{x \mapsto t\} \mid \beta \in S, x \notin \mathit{dom}(\beta)\} \cup \\
&\quad \{\beta \mid \beta \in S, F^*(\beta(x)) = F^*(t)\} \\
\mathit{match}(c, t, S) &= S \ \mathit{if} \ F^*(c) = F^*(t) \\
\mathit{match}(c, t, S) &= \emptyset \ \mathit{if} \ F^*(c) \neq F^*(t) \\
\mathit{match}(f(p_1, \ldots, p_n), t, S) &= \\
&\bigcup_{F^*(f(t_1,\ldots,t_n))=F^*(t)} \mathit{match}(p_n, t_n, \ldots, \mathit{match}(p_1, t_1, S) \ldots)
\end{aligned}
$$

$\mathit{match}(p, t, \{\emptyset\})$ returns the desired set of substitutions.

E-matching: Example

$$
\begin{aligned}
F \quad = \quad & \{a \mapsto c,\ b \mapsto b,\ c \mapsto c,\ d \mapsto d, \\
& f(c,b) \mapsto f(c,b),\ \ f(g(a),b) \mapsto f(c,b), \\
& g(a) \mapsto c,\ \ g(b) \mapsto g(b),\ \ g(c) \mapsto c,\ \ g(d) \mapsto c, \\
& h(a,d) \mapsto b,\ \ h(c,a) \mapsto b\}
\end{aligned}
$$

E-match t and p:

$$
\begin{aligned}
t \quad &= \quad f(c,b) \\
p \quad &= \quad f(g(x),h(x,a))
\end{aligned}
$$

E-matching: Example

$$
\begin{aligned}
F \quad = \quad & \{a \mapsto c,\ b \mapsto b,\ c \mapsto c,\ d \mapsto d, \\
& f(c,b) \mapsto f(c,b),\ \ f(g(a),b) \mapsto f(c,b), \\
& g(a) \mapsto c,\ \ g(b) \mapsto g(b),\ \ g(c) \mapsto c,\ \ g(d) \mapsto c, \\
& h(a,d) \mapsto b,\ \ h(c,a) \mapsto b\}
\end{aligned}
$$

$\mathsf{match}(f(g(x),h(x,a)), f(c,b), \{\emptyset\}) =$

E-matching: Example

$$F = \{a \mapsto c,\ b \mapsto b,\ c \mapsto c,\ d \mapsto d,$$
$$f(c,b) \mapsto f(c,b),\ \ f(g(a),b) \mapsto f(c,b),$$
$$g(a) \mapsto c,\ \ g(b) \mapsto g(b),\ \ g(c) \mapsto c,\ \ g(d) \mapsto c,$$
$$h(a,d) \mapsto b,\ \ h(c,a) \mapsto b\}$$

$$\mathit{match}(f(g(x),h(x,a)), f(c,b), \{\emptyset\}) =$$

$\mathit{match}(g(x), c, \mathit{match}(h(x,a), b, \{\emptyset\}))$ for $f(c,b)$

$\cup$

$\mathit{match}(g(x), g(a), \mathit{match}(h(x,a), b, \{\emptyset\}))$ for $f(g(a),b)$

E-matching: Example

$$
\begin{aligned}
F \;=\; & \{a \mapsto c,\ b \mapsto b,\ c \mapsto c,\ d \mapsto d, \\
& f(c,b) \mapsto f(c,b),\ \ f(g(a),b) \mapsto f(c,b), \\
& g(a) \mapsto c,\ \ g(b) \mapsto g(b),\ \ g(c) \mapsto c,\ \ g(d) \mapsto c, \\
& h(a,d) \mapsto b,\ \ h(c,a) \mapsto b\}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{match}(f(g(x),h(x,a)), f(c,b), \{\emptyset\}) = \\
&\quad \mathit{match}(g(x), c, \mathit{match}(x, a, \mathit{match}(a, d, \{\emptyset\}))) \qquad \text{for } h(a,d) \\
&\qquad\qquad \cup \\
&\qquad\qquad \mathit{match}(x, c, \mathit{match}(a, a, \{\emptyset\}))) \qquad \text{for } h(c,a) \\
&\quad \cup \\
&\quad \mathit{match}(g(x), g(a), \mathit{match}(h(x,a), b, \{\emptyset\}))
\end{aligned}
$$

E-matching: Example

$$
\begin{aligned}
F \;=\; & \{a \mapsto c,\ b \mapsto b,\ c \mapsto c,\ d \mapsto d, \\
& f(c,b) \mapsto f(c,b),\ \ f(g(a),b) \mapsto f(c,b), \\
& g(a) \mapsto c,\ \ g(b) \mapsto g(b),\ \ g(c) \mapsto c,\ \ g(d) \mapsto c, \\
& h(a,d) \mapsto b,\ \ h(c,a) \mapsto b\}
\end{aligned}
$$

$\mathit{match}(f(g(x),h(x,a)), f(c,b), \{\emptyset\}) =$

$\mathit{match}(g(x), c, \mathit{match}(x, a, \mathit{match}(a, d, \{\emptyset\}))$ for $h(a,d)$

$\cup$

$\mathit{match}(x, c, \mathit{match}(a, a, \{\emptyset\})))$ for $h(c,a)$

$\cup$

$\mathit{match}(g(x), g(a), \mathit{match}(h(x,a), b, \{\emptyset\}))$

a and d are not in the same equivalence class.

E-matching: Example

$$
\begin{aligned}
F \;=\; & \{a \mapsto c,\ b \mapsto b,\ c \mapsto c,\ d \mapsto d, \\
& f(c,b) \mapsto f(c,b),\ \ f(g(a),b) \mapsto f(c,b), \\
& g(a) \mapsto c,\ \ g(b) \mapsto g(b),\ \ g(c) \mapsto c,\ \ g(d) \mapsto c, \\
& h(a,d) \mapsto b,\ \ h(c,a) \mapsto b\}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{match}(f(g(x),h(x,a)), f(c,b), \{\emptyset\}) = \\
&\quad \mathit{match}(g(x), c, \mathit{match}(x, a, \emptyset) \\
&\qquad\qquad \cup \\
&\qquad\qquad \mathit{match}(x, c, \mathit{match}(a, a, \{\emptyset\}))) \\
&\quad \cup \\
&\quad \mathit{match}(g(x), g(a), \mathit{match}(h(x,a), b, \{\emptyset\}))
\end{aligned}
$$

E-matching: Example

$$
\begin{aligned}
F \quad = \quad & \{a \mapsto c,\ b \mapsto b,\ c \mapsto c,\ d \mapsto d, \\
& f(c,b) \mapsto f(c,b),\ \ f(g(a),b) \mapsto f(c,b), \\
& g(a) \mapsto c,\ \ g(b) \mapsto g(b),\ \ g(c) \mapsto c,\ \ g(d) \mapsto c, \\
& h(a,d) \mapsto b,\ \ h(c,a) \mapsto b\}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{match}(f(g(x),h(x,a)), f(c,b), \{\emptyset\}) = \\
&\quad \mathit{match}(g(x), c, \emptyset \\
&\qquad\qquad \cup \\
&\qquad\qquad \mathit{match}(x, c, \mathit{match}(a, a, \{\emptyset\}))) \\
&\quad \cup \\
&\quad \mathit{match}(g(x), g(a), \mathit{match}(h(x,a), b, \{\emptyset\}))
\end{aligned}
$$

E-matching: Example

$$
\begin{aligned}
F \;=\; & \{a \mapsto c,\ b \mapsto b,\ c \mapsto c,\ d \mapsto d, \\
& f(c,b) \mapsto f(c,b),\ \ f(g(a),b) \mapsto f(c,b), \\
& g(a) \mapsto c,\ \ g(b) \mapsto g(b),\ \ g(c) \mapsto c,\ \ g(d) \mapsto c, \\
& h(a,d) \mapsto b,\ \ h(c,a) \mapsto b\}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{match}(f(g(x),h(x,a)), f(c,b), \{\emptyset\}) = \\
&\quad \mathit{match}(g(x), c, \emptyset \\
&\qquad\qquad \cup \\
&\qquad\qquad \mathit{match}(x, c, \mathit{match}(a, a, \{\emptyset\}))) \\
&\quad \cup \\
&\quad \mathit{match}(g(x), g(a), \mathit{match}(h(x,a), b, \{\emptyset\}))
\end{aligned}
$$

$$F^*(a) = F^*(a)$$

E-matching: Example

$$F = \{a \mapsto c,\ b \mapsto b,\ c \mapsto c,\ d \mapsto d,$$
$$f(c,b) \mapsto f(c,b),\ \ f(g(a),b) \mapsto f(c,b),$$
$$g(a) \mapsto c,\ \ g(b) \mapsto g(b),\ \ g(c) \mapsto c,\ \ g(d) \mapsto c,$$
$$h(a,d) \mapsto b,\ \ h(c,a) \mapsto b\}$$

$$\mathit{match}(f(g(x),h(x,a)), f(c,b), \{\emptyset\}) =$$

$$\mathit{match}(g(x), c, \emptyset$$
$$\cup$$
$$\mathit{match}(x, c, \{\emptyset\}))$$

$$\cup$$

$$\mathit{match}(g(x), g(a), \mathit{match}(h(x,a), b, \{\emptyset\}))$$

E-matching: Example

$$
\begin{aligned}
F \quad = \quad & \{a \mapsto c,\ b \mapsto b,\ c \mapsto c,\ d \mapsto d, \\
& f(c,b) \mapsto f(c,b),\ \ f(g(a),b) \mapsto f(c,b), \\
& g(a) \mapsto c,\ \ g(b) \mapsto g(b),\ \ g(c) \mapsto c,\ \ g(d) \mapsto c, \\
& h(a,d) \mapsto b,\ \ h(c,a) \mapsto b\}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{match}(f(g(x),h(x,a)), f(c,b), \{\emptyset\}) = \\
&\quad \mathit{match}(g(x), c, \emptyset \\
&\qquad\qquad \cup \\
&\qquad\qquad \{\{x \mapsto c\}\}) \\
&\quad \cup \\
&\quad \mathit{match}(g(x), g(a), \mathit{match}(h(x,a), b, \{\emptyset\}))
\end{aligned}
$$

E-matching: Example

$$
\begin{aligned}
F \quad = \quad & \{a \mapsto c,\ b \mapsto b,\ c \mapsto c,\ d \mapsto d, \\
& f(c,b) \mapsto f(c,b),\ \ f(g(a),b) \mapsto f(c,b), \\
& g(a) \mapsto c,\ \ g(b) \mapsto g(b),\ \ g(c) \mapsto c,\ \ g(d) \mapsto c, \\
& h(a,d) \mapsto b,\ \ h(c,a) \mapsto b\}
\end{aligned}
$$

$$
\mathit{match}(f(g(x),h(x,a)), f(c,b), \{\emptyset\}) =
$$

$$
\mathit{match}(g(x), c, \{\{x \mapsto c\}\})
$$

$$
\cup
$$

$$
\mathit{match}(g(x), g(a), \mathit{match}(h(x,a), b, \{\emptyset\}))
$$

E-matching: Example

$$
\begin{aligned}
F \quad = \quad & \{a \mapsto c,\ b \mapsto b,\ c \mapsto c,\ d \mapsto d, \\
& f(c,b) \mapsto f(c,b),\ \ f(g(a),b) \mapsto f(c,b), \\
& g(a) \mapsto c,\ \ g(b) \mapsto g(b),\ \ g(c) \mapsto c,\ \ g(d) \mapsto c, \\
& h(a,d) \mapsto b,\ \ h(c,a) \mapsto b\}
\end{aligned}
$$

$\mathit{match}(f(g(x),h(x,a)), f(c,b), \{\emptyset\}) =$

$\mathit{match}(x, a, \{\{x \mapsto c\}\}) \cup$ for $g(a)$

$\mathit{match}(x, c, \{\{x \mapsto c\}\}) \cup$ for $g(c)$

$\mathit{match}(x, d, \{\{x \mapsto c\}\}) \cup$ for $g(d)$

$\mathit{match}(g(x), g(a), \mathit{match}(h(x,a), b, \{\emptyset\}))$

E-matching: Example

$$
\begin{aligned}
F \quad = \quad & \{a \mapsto c,\ b \mapsto b,\ c \mapsto c,\ d \mapsto d, \\
& f(c,b) \mapsto f(c,b),\ \ f(g(a),b) \mapsto f(c,b), \\
& g(a) \mapsto c,\ \ g(b) \mapsto g(b),\ \ g(c) \mapsto c,\ \ g(d) \mapsto c, \\
& h(a,d) \mapsto b,\ \ h(c,a) \mapsto b\}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{match}(f(g(x),h(x,a)), f(c,b), \{\emptyset\}) = \\
&\quad \{\{x \mapsto c\}\} \cup \\
&\quad \{\{x \mapsto c\}\} \cup \\
&\quad \emptyset \cup \\
&\quad \mathit{match}(g(x), g(a), \mathit{match}(h(x,a), b, \{\emptyset\}))
\end{aligned}
$$

E-matching: Example

$$
\begin{aligned}
F \quad = \quad & \{a \mapsto c,\ b \mapsto b,\ c \mapsto c,\ d \mapsto d, \\
& f(c,b) \mapsto f(c,b),\ \ f(g(a),b) \mapsto f(c,b), \\
& g(a) \mapsto c,\ \ g(b) \mapsto g(b),\ \ g(c) \mapsto c,\ \ g(d) \mapsto c, \\
& h(a,d) \mapsto b,\ \ h(c,a) \mapsto b\}
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{match}(f(g(x),h(x,a)), f(c,b), \{\emptyset\}) = \\
&\quad \{\{x \mapsto c\}\} \cup \\
&\quad \mathsf{match}(g(x), g(a), \mathsf{match}(h(x,a), b, \{\emptyset\}))
\end{aligned}
$$

E-matching: Example

$$
\begin{aligned}
F \quad = \quad & \{a \mapsto c,\ b \mapsto b,\ c \mapsto c,\ d \mapsto d, \\
& f(c,b) \mapsto f(c,b),\ \ f(g(a),b) \mapsto f(c,b), \\
& g(a) \mapsto c,\ \ g(b) \mapsto g(b),\ \ g(c) \mapsto c,\ \ g(d) \mapsto c, \\
& h(a,d) \mapsto b,\ \ h(c,a) \mapsto b\}
\end{aligned}
$$

$$
\mathsf{match}(f(g(x),h(x,a)), f(c,b), \{\emptyset\}) =
$$

$$
\{\{x \mapsto c\}\} \cup
$$

$$
\{\{x \mapsto c\}\}
$$

E-matching: Example

$$
\begin{aligned}
F \quad = \quad & \{a \mapsto c,\ b \mapsto b,\ c \mapsto c,\ d \mapsto d, \\
& f(c,b) \mapsto f(c,b),\ \ f(g(a),b) \mapsto f(c,b), \\
& g(a) \mapsto c,\ \ g(b) \mapsto g(b),\ \ g(c) \mapsto c,\ \ g(d) \mapsto c, \\
& h(a,d) \mapsto b,\ \ h(c,a) \mapsto b\}
\end{aligned}
$$

$$
\mathsf{match}(f(g(x),h(x,a)), f(c,b), \{\emptyset\}) =
$$

$$
\{\{x \mapsto c\}\}
$$

E-matching: example

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

E-matching 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.

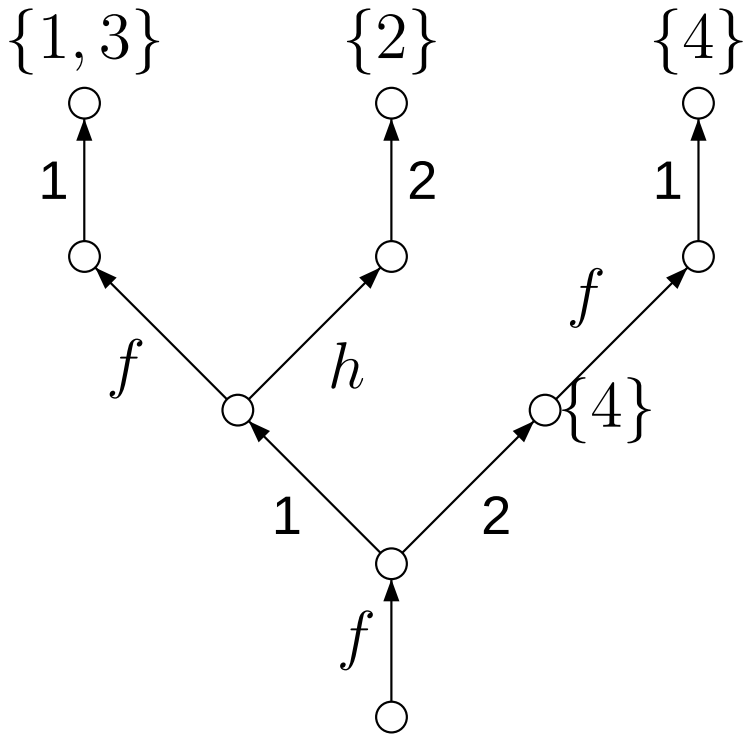
    Patterns that start with a f-application.
  - Execute code-tree for each new term.
- New equalities are assigned during the search.
  - New equalities $\leadsto$ new E-matching instances.
  - Example:

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

    $a = g(b)$ is assigned.

Inverted path index

- It is used to find which patterns may have new instances after an equality is assigned.
- 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:

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

- Matching loops

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

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

E-matching: Conclusion

- E-matching is a heuristic and (blatantly) incomplete.
- Saturation calculi (e.g., Superposition Calculus) offer a strong (and in principle complete) alternative.
- Plug: Engineering DPLL(T) + Saturation. [de Moura & Bjørner IJCAR 2008]

Decidable fragments

- Some fragments of first-order logic are decidable.
- Fragments supported by Z3:
  - Bernays-Schönfinkel class (aka EPR): $\forall^*$, predicates, variables and constants (no function symbols).
    - NEXPTIME-complete
    - QBF
    - Encode useful theories (e.g., partial orders).
    - Finite model finding of arbitrary first-order formulas.
  - Array property fragment.
  - More fragments coming soon.

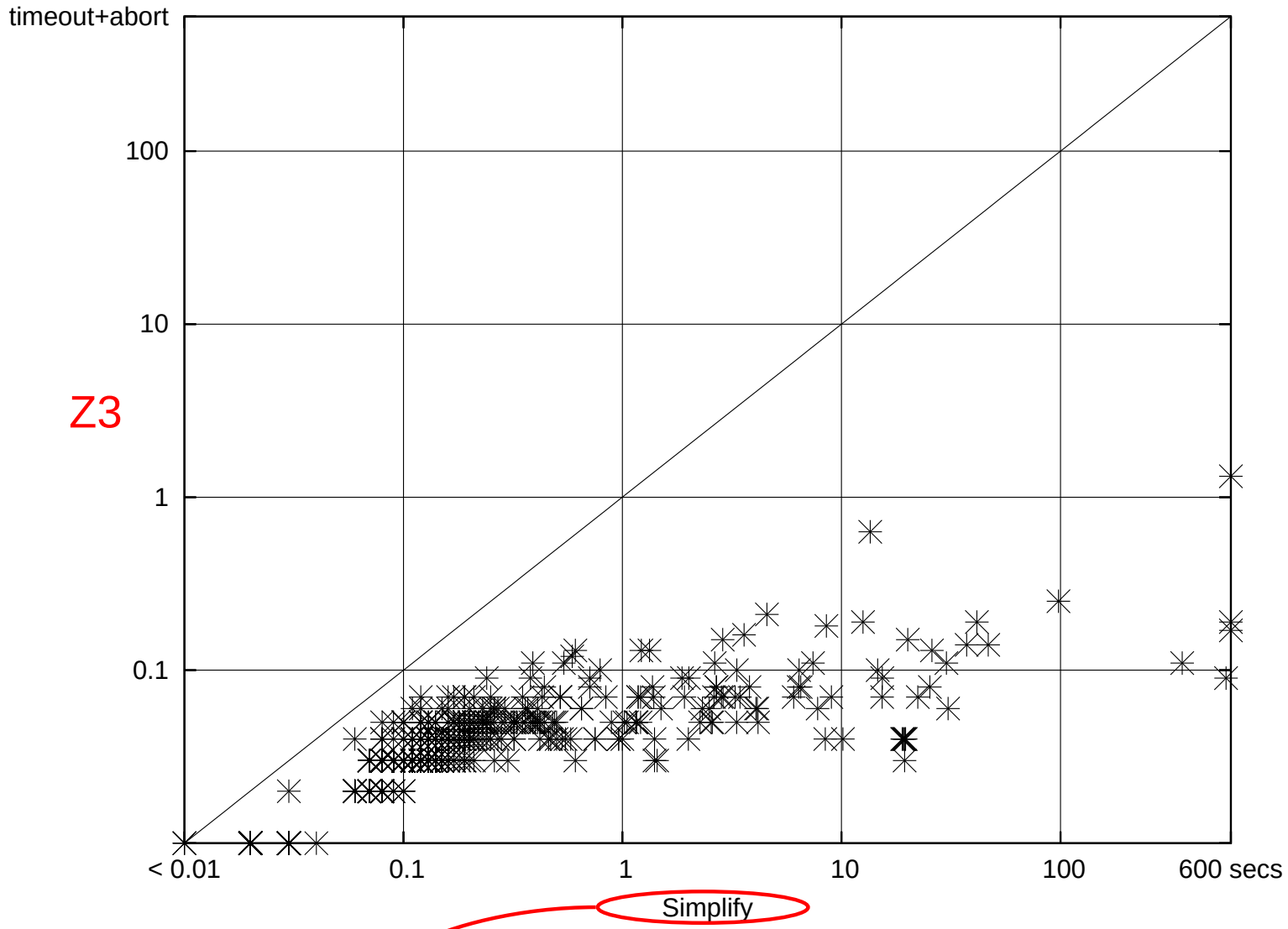
Roadmap

- Background
- SAT & SMT
- Combining theories
- Equality
- Arithmetic
- Quantifiers
- Applications

SMT@Microsoft: Solver

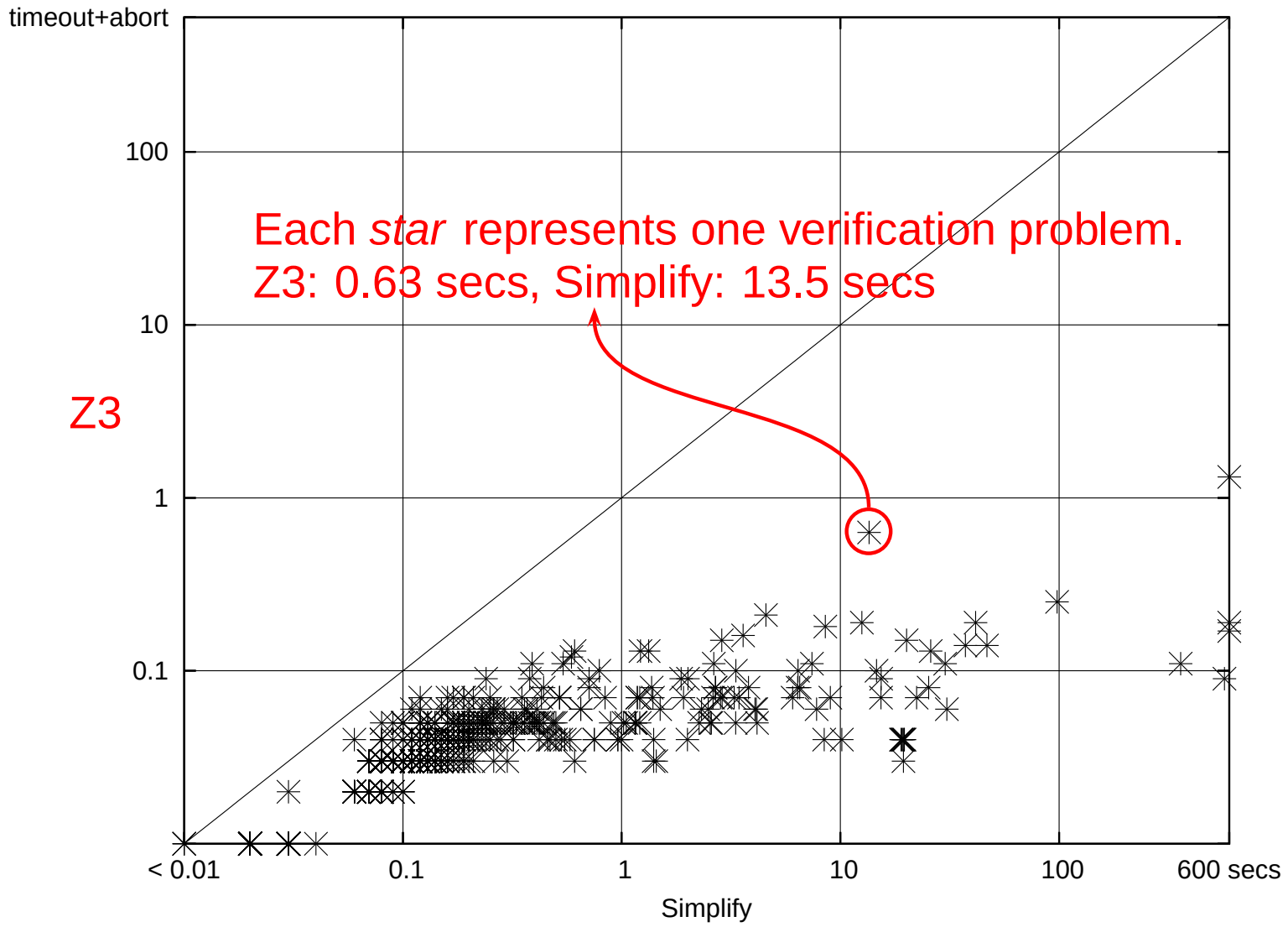
- Z3 is a new SMT solver developed at Microsoft Research.
- Development/Research driven by internal customers.
- Textual input & APIs (C/C++, .NET, OCaml).
- Free for non-commercial use.
- Very efficient: SMT-COMP'08 (15 divisions)
  - 9 first places
  - 6 second places
- `http://research.microsoft.com/projects/z3`

Performance (Spec#/Boogie): Z3 × Simplify

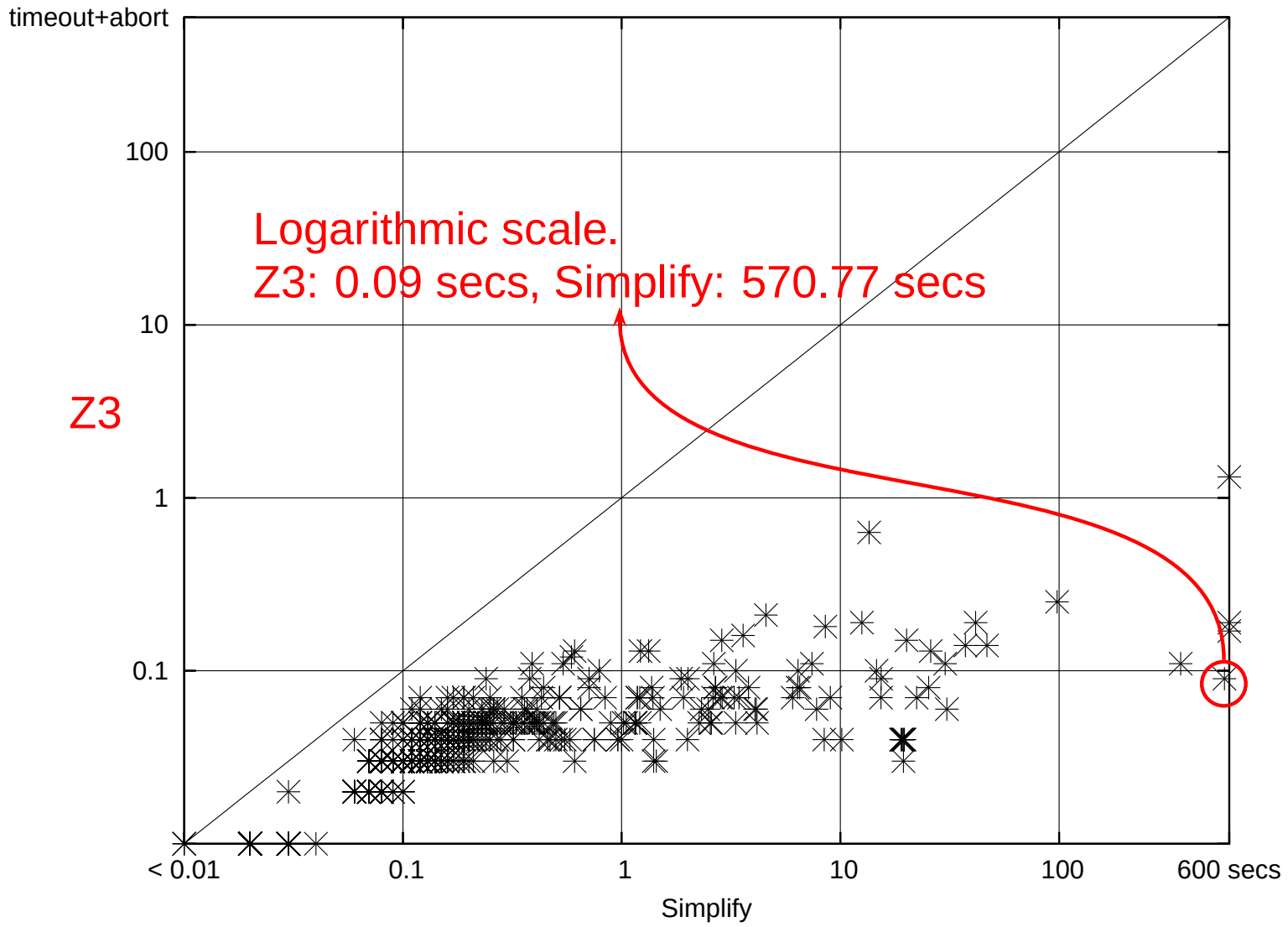


Spec#/Boogie was using *Simplify* from HP Labs.

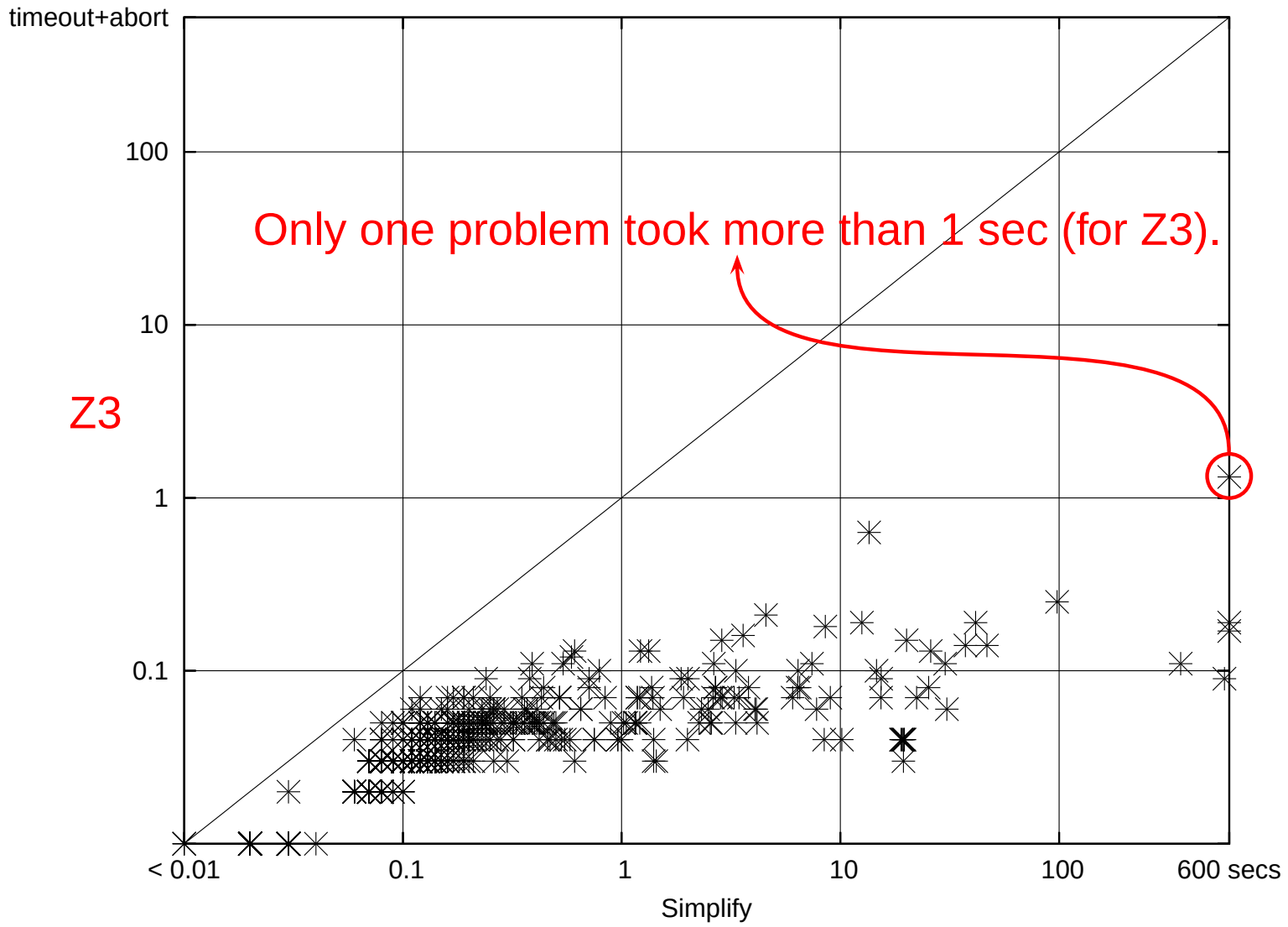
Performance (Spec#/Boogie): Z3 × Simplify



Performance (Spec#/Boogie): Z3 × Simplify



Performance (Spec#/Boogie): Z3 × Simplify



SMT@Microsoft: Applications

- Test-case generation:

  Pex, SAGE, and Vigilante.

- Verifying Compiler:

  Spec#/Boogie, HAVOC, and VCC.

- Model Checking & Predicate Abstraction:

  SLAM/SDV and Yogi.

- Bounded Model Checking (BMC):

  AsmL model checker.

- Other: invariant generation, crypto, etc.

Test-case generation

- Test (correctness + usability) is 95% of the deal:
  - Dev/Test is 1-1 in products.
  - Developers are responsible for unit tests.
- Tools:
  - Annotations and static analysis (SAL, ESP)
  - File Fuzzing
  - Unit test case generation

Security is Critical

- Security bugs can be very expensive:
  - Cost of each MS Security Bulletin: $600K to $Millions.
  - Cost due to worms (Slammer, CodeRed, Blaster, etc.): $Billions.
  - The real victim is the customer.
- Most security exploits are initiated via files or packets:
  - Ex: Internet Explorer parses dozens of files formats.
- Security testing: hunting for million-dollar bugs
  - Write A/V (always exploitable),
  - Read A/V (sometimes exploitable),
  - NULL-pointer dereference,
  - Division-by-zero (harder to exploit but still DOS attack), ...

Hunting for Security Bugs

- Two main techniques used by “black hats”:
  - Code inspection (of binaries).
  - Black box fuzz testing.
- **Black box** fuzz testing:
  - A form of black box random testing.
  - Randomly fuzz (=modify) a well formed input.
  - Grammar-based fuzzing: rules to encode how to fuzz.
- **Heavily** used in security testing
  - At MS: several internal tools.
  - Conceptually simple yet effective in practice

    Has been instrumental in weeding out 1000 of bugs during development and test.

Automatic Code-Driven Test Generation

Input x, y

requires$(y > 0)$

while (true) $\{$

$m = x \,\%\, y$

if $(m == 0)$ **return** y

$x = y$

$y = m$

$\}$

We want a trace where the loop is executed twice.

Automatic Code-Driven Test Generation

Input x, y

requires$(y > 0)$

$m = x \,\%\, y$

if $(m == 0)$ **return** y

$x = y$

$y = m$

$m = x \,\%\, y$

if $(m == 0)$ **return** y

$x = y$

$y = m$

Unfolded the loop twice.

Automatic Code-Driven Test Generation

Input x_0, y_0

requires$(y_0 > 0)$

$m_0 = x_0 \,\%\, y_0$

if $(m_0 == 0)$ **return** y_0

$x_1 = y_0$

$y_1 = m_0$

$m_1 = x_1 \,\%\, y_1$

if $(m_1 == 0)$ **return** y_1

$x_2 = y_1$

$y_2 = m_1$

Conveted to static single assignment form.

Automatic Code-Driven Test Generation

Input x_0, y_0

requires$(y_0 > 0)$

$m_0 = x_0 \,\%\, y_0$

if $(m_0 == 0)$ **return** y_0

$x_1 = y_0$

$y_1 = m_0$

$m_1 = x_1 \,\%\, y_1$

if $(m_1 == 0)$ **return** y_1

$x_2 = y_1$

$y_2 = m_1$

The first condition should be false, the second true.

Automatic Code-Driven Test Generation

$$
\begin{aligned}
& y_0 > 0 \wedge \\
& m_0 = x_0 \,\%\, y_0 \wedge \\
& \neg m_0 = 0 \wedge \\
& x_1 = y_0 \wedge \\
& y_1 = m_0 \wedge \\
& m_1 = x_1 \,\%\, y_1 \wedge \\
& m_1 = 0
\end{aligned}
$$

Converted to formula. Use bit-vector decision procedure.

Automatic Code-Driven Test Generation

$$
\begin{aligned}
& y_0 > 0 \wedge \\
& m_0 = x_0 \,\%\, y_0 \wedge \\
& \neg m_0 = 0 \wedge \\
& x_1 = y_0 \wedge \\
& y_1 = m_0 \wedge \\
& m_1 = x_1 \,\%\, y_1 \wedge \\
& m_1 = 0
\end{aligned}
$$

$$
\begin{aligned}
M(x_0) &= 2 \\
M(y_0) &= 4 \\
M(m_0) &= 2 \\
M(x_1) &= 4 \\
M(y_1) &= 2 \\
M(m_1) &= 0
\end{aligned}
$$

Executed SMT Solver (Z3).

Method: Dynamic Test Generation

- Run program with random inputs.
- Collect constraints on inputs.
- Use SMT solver to generate new inputs.
- Combination with randomization: DART (Godefroid-Klarlund-Sen-05)

Method: Dynamic Test Generation

- Run program with random inputs.
- Collect constraints on inputs.
- Use SMT solver to generate new inputs.
- Combination with randomization: DART (Godefroid-Klarlund-Sen-05)

Repeat while finding new execution paths.

DARTish projects at Microsoft

- SAGE (CSE) implements DART for x86 binaries and merges it with "fuzz" testing for finding security bugs.
- PEX (MSR-Redmond FSE Group) implements DART for .NET binaries in conjunction with "parameterized-unit tests" for unit testing of .NET programs.
- YOGI (MSR-India) implements DART to check the feasibility of program paths generated statically using a SLAM-like tool.
- Vigilante (MSR Cambridge) partially implements DART to dynamically generate worm filters.

Inital Experiences with SAGE

25+ security bugs and counting. (most missed by blackbox fuzzers)

- OS component X

  4 new bugs: “This was an area that we heavily fuzz tested in Vista”.

- OS component Y

  Arithmetic/stack overflow in y.dll

- Media format A

  Arithmetic overflow; DOS crash in previously patched component

- Media format B & C

  Hard-to-reproduce uninitialized-variable bug

Pex

- Pex monitors the execution of .NET application using the CLR profiling API.
- Pex dynamically checks for violations of programming rules, e.g. resource leaks.
- Pex suggests code snippets to the user, which will prevent the same failure from happening again.
- Very instrumental in exposing bugs in .NET libraries.
- Free for non-commercial use.
- `http://research.microsoft.com/Pex/`

Test-case generation & SMT

- Formulas are usually a big conjunction.
- Incremental: solve several similar formulas.
- “Small models”.
- Arithmetic $\times$ Machine Arithmetic.

Test-case generation & SMT

- Formulas are usually a big conjunction.
  - Pre-processing step.
  - Eliminate variables and simplify input formula.
  - Significant performance impact.
- Incremental: solve several similar formulas.
- “Small models”.
- Arithmetic $\times$ Machine Arithmetic.

Test-case generation & SMT

- Formulas are usually a big conjunction.
- Incremental: solve several similar formulas.
  - New constraints can be asserted.
  - **push** and **pop**: (user) backtracking.
  - Reuse (some) lemmas.
- “Small models”.
- Arithmetic $\times$ Machine Arithmetic.

Test-case generation & SMT

- Formulas are usually a big conjunction.
- Incremental: solve several similar formulas.
- “Small models”.
  - **Given** a set of constraints C, find a model M that minimizes the value of the variables $x_0, \ldots, x_n$.
- Arithmetic $\times$ Machine Arithmetic.

Test-case generation & SMT

- Formulas are usually a big conjunction.
- Incremental: solve several similar formulas.
- “Small models”.
  - **Given** a set of constraints C, find a model M that minimizes the value of the variables $x_0, \ldots, x_n$.
  - **Eager (cheap) Solution:**

    Assert C.

    While satisfiable

        Peek x_i such that $M[x_i]$ is big

        Assert $x_i < c$, where c is a small constant

    Return last found model
- Arithmetic $\times$ Machine Arithmetic.

Test-case generation & SMT

- Formulas are usually a big conjunction.
- Incremental: solve several similar formulas.
- “Small models”.
  - **Given** a set of constraints C, find a model M that minimizes the value of the variables $x_0, \ldots, x_n$.
  - **Refinement:**
    - Eager solution stops as soon as the context becomes unsatisfiable.
    - A “bad” choice (peek x_i) may prevent us from finding a good solution.
    - Use **push** and **pop** to retract “bad” choices.
- Arithmetic $\times$ Machine Arithmetic.

Test-case generation & SMT

- Formulas are usually a big conjunction.
- Incremental: solve several similar formulas.
- “Small models”.
- Arithmetic $\times$ Machine Arithmetic.
  - Precision $\times$ Performance.
  - SAGE has flags to abstract expensive operations.

The Verifying Compiler

A verifying compiler uses automated reasoning to check the correctness of a program that is compiles.

Correctness is specified by types, assertions, . . . and other redundant annotations that accompany the program.

Hoare 2004

Spec# Approach for a Verifying Compiler

Presented by Rustan Leino on Monday and Tuesday.

Microsoft Hypervisor

- **Meta OS:** small layer of software between hardware and OS.
- **Mini:** 60K lines of non-trivial concurrent systems C code.
- **Critical:** simulates a number of virtual x64 machines.
- **Trusted:** a grand verification challenge.

Tool: A Verified C Compiler

- VCC translates an annotated C program into a Boogie PL program.
- Boogie generates verification conditions.
- A C-ish memory model
  - Abstract heaps
  - Bit-level precision
- The verification project has very recently started.
- It is a multi-man multi-year effort.
- More news coming soon.

Tool: HAVOC

- HAVOC also translates annotated C into Boogie PL.
- It allows the expression of richer properties about the program heap and data structures such as linked lists and arrays.
- Used to check NTFS-specific properties.
- Found 50 bugs, most confirmed.
  - 250 lines required to specify properties.
  - 600 lines of manual annotations.
  - 3000 lines of inferred annotations.

Verifying Compilers & SMT

- Quantifiers, Quantifiers, . . .
  - Modeling the runtime.
  - Frame axioms (“what didn’t change”).
  - User provided assertions (e.g., the array is sorted).
  - Prototyping decision procedures (e.g., reachability, partial orders, . . .).
- Solver must be fast in satisfiable instances.
- First-order logic is undecidable.
- Z3:
  - Heuristic Quantifier Instantiation: E-matching.
  - Decidable fragments.
  - Superposition Calculus engine.

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.

Predicate Abstraction: c2bp

- **Given** a C program P and $F = \{p_1, \ldots, p_n\}$.
- **Produce** a boolean program $B(P, F)$
  - Same control flow structure as P.
  - Boolean variables $\{b_1, \ldots, b_n\}$ to match $\{p_1, \ldots, p_n\}$.
  - Properties true of $B(P, F)$ are true of P.
- Example $F = \{x > 0, x = y\}$.

Abstracting Expressions via F

- *Implies*$_F(e)$
  - Best boolean function over F that implies e
- *ImpliedBy*$_F(e)$
  - Best boolean function over F that is implied by e
  - *ImpliedBy*$_F(e) = \neg$*Implies*$_F(\neg e)$

Computing $Implies_F(e)$

- minterm $m = l_1 \wedge \ldots \wedge l_n$, where $l_i = p_i$, or $l_i = \neg p_i$.
- $Implies_F(e)$ is the disjunction of all minterms that imply e.
- Naive approach
  - Generate all 2^n possible minterms.
  - For each minterm m, use SMT solver to check validity of $m \implies e$.
- Many possible optimizations.

Computing $Implies_F(e)$: $Example$

- $F = \{x < y, x = 2\}$
- $e : y > 1$
- Minterms over P
  - $x \geq y, x \neq 2$
  - $x < y, x \neq 2$
  - $x \geq y, x = 2$
  - $x < y, x = 2$
- $Implies_F(e) = \{x < y, x = 2\}$

Newton

- Given an error path π in the boolean program B.
- Is π a feasible path of the corresponding C program?
  - Yes: found a bug.
  - No: find predicates that explain the infeasibility.
- Execute path symbolically.
- Check conditions for inconsistency using SMT solver.

Model Checking & SMT

- All-SAT
  - Fast Predicate Abstraction.
- Unsatisfiable Cores
  - Why the abstract path is not feasible?
  - Fast Predicate Abstraction.

Other Microsoft Clients

- Termination
- Security protocols
- Business application modeling
- Cryptography
- Verifying garbage collectors
- Model based testing (SQL)
- Semantic type checking for D models
- More coming soon

Other applications

Bounded Model Checking

- To check whether a program with initial state I and next-state relation T violates the invariant P in the first k steps, one checks:

$$I(s_0) \wedge T(s_0, s_1) \wedge \ldots \wedge T(s_{k-1}, s_k) \wedge (\neg P(s_0) \vee \ldots \vee \neg P(s_k))$$

- This formula is satisfiable if and only if there exists a path of length at most k from the initial state s_0 which violates the invariant k.
- Formulas produced in BMC are usually quite big.
- k-Induction

$$T(s_0, s_1) \wedge \ldots \wedge T(s_{k-1}, s_k) \wedge P(s_0) \wedge \ldots \wedge P(s_{k-1}) \wedge \neg P(s_k)$$

Scheduling

- Given j jobs and m machines, each job consists of a sequence of tasks $t_{i_1}, \ldots, t_{i_n}$, where each task t_{i_k} is a pair $\langle M, \delta \rangle$ for machine M and duration δ.
- Find a schedule with a minimum duration, e.g.,

Jobs	Tasks
a	$\langle 1, 2 \rangle, \langle 2, 6 \rangle$
b	$\langle 2, 5 \rangle, \langle 1, 3 \rangle, \langle 2, 3 \rangle$
c	$\langle 2, 4 \rangle$
d	$\langle 1, 5 \rangle, \langle 2, 2 \rangle$

Planning

- Given c cities, t trucks each located at a specific city, and p packages each with a source city and a destination city.
- In each step, packages can be loaded and unloaded, or the trucks can be driven from one city to another.
- Find a plan with a minimum number of steps for delivering the packages from source to destination.
- For each step i, we have Booleans: *location*(t, c, i), *at*(p, c, i), and *on*(p, t, i).
- Domain constraints assert that a package can be either on one truck or at a city, a package can be loaded or unloaded from a truck to a city only if the truck is at the city, etc.

MaxSAT

- 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 $a_i \vee A_i$, for a fresh variable a_i.
- We can add constraints indicating that at most k of the a_i literals can be assigned **true**.
- By shrinking k, we can determine the minimal value of k.
- Weighted MaxSAT can be solved similarly.
- More generally, pseudo-Boolean constraints $\Sigma_i w_i \times a_i \leq k$ can be encoded.

Several satisfying assignments

- Related to All-SAT.
- Many applications (e.g., Spec# uses it).
- How to:
  - Add a clause blocking the current solution
  - Reinvoke the SMT solver.

Conclusion

- Powerful, mature, and versatile tools like SMT solvers can now be exploited in very useful ways.
- The construction and application of satisfiability procedures is an active research area with exciting challenges.
- SMT is hot at Microsoft.
- Z3 is a new SMT solver.
- Main applications:
  - Test-case generation.
  - Verifying compiler.
  - Model Checking & Predicate Abstraction.

Reading Material

Books

- Bradley & Manna: **The Calculus of Computation**
- Kroening & Strichman: **Decision Procedures** An Algorithmic Point of View

Web Links

- Z3:
  `http://research.microsoft.com/projects/z3`
- `http://research.microsoft.com/~leonardo`
  - Slides & Papers
- `http://www.smtlib.org`
- `http://www.smtcomp.org`

Lab Exercises

References

[Ack54] W. Ackermann. Solvable cases of the decision problem. *Studies in Logic and the Foundation of Mathematics*, 1954

[ABC$^+$02] G. Audemard, P. Bertoli, A. Cimatti, A. Kornilowicz, and R. Sebastiani. A SAT based approach for solving formulas over boolean and linear mathematical propositions. In *Proc. of CADE'02*, 2002

[BDS00] C. Barrett, D. Dill, and A. Stump. A framework for cooperating decision procedures. In *17th International Conference on Computer-Aided Deduction*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 79–97. Springer-Verlag, 2000

[BdMS05] C. Barrett, L. de Moura, and A. Stump. SMT-COMP: Satisfiability Modulo Theories Competition. In *Int. Conference on Computer Aided Verification (CAV'05)*, pages 20–23. Springer, 2005

[BDS02] C. Barrett, D. Dill, and A. Stump. Checking satisfiability of first-order formulas by incremental translation to SAT. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Proceedings of the* 14^{th} *International Conference on Computer Aided Verification (CAV '02)*, volume 2404 of *Lecture Notes in Computer Science*, pages 236–249. Springer-Verlag, July 2002. Copenhagen, Denmark

[BBC$^+$05] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Ranise, and R. Sebastiani. Efficient satisfiability modulo theories via delayed theory combination. In *Int. Conf. on Computer-Aided Verification (CAV)*, volume 3576 of *LNCS*. Springer, 2005

[Chv83] V. Chvatal. *Linear Programming*. W. H. Freeman, 1983

References

[CG96] B. Cherkassky and A. Goldberg. Negative-cycle detection algorithms. In *European Symposium on Algorithms*, pages 349–363, 1996

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962

[DNS03] D. Detlefs, G. Nelson, and J. B. Saxe. Simplify: A theorem prover for program checking. Technical Report HPL-2003-148, HP Labs, 2003

[DST80] P. J. Downey, R. Sethi, and R. E. Tarjan. Variations on the Common Subexpression Problem. *Journal of the Association for Computing Machinery*, 27(4):758–771, 1980

[dMR02] L. de Moura and H. Rueß. Lemmas on demand for satisfiability solvers. In *Proceedings of the Fifth International Symposium on the Theory and Applications of Satisfiability Testing (SAT 2002)*. Cincinnati, Ohio, 2002

[DdM06] B. Dutertre and L. de Moura. Integrating simplex with DPLL(T). Technical report, CSL, SRI International, 2006

[dMB07b] L. de Moura and N. Bjørner. Efficient E-Matching for SMT solvers. In *CADE-21*, pages 183–198, 2007

References

[dMB07c] L. de Moura and N. Bjørner. Model Based Theory Combination. In *SMT'07*, 2007

[dMB07a] L. de Moura and N. Bjørner. Relevancy Propagation . Technical Report MSR-TR-2007-140, Microsoft Research, 2007

[dMB08a] L. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *TACAS 08*, 2008

[dMB08c] L. de Moura and N. Bjørner. Engineering DPLL(T) + Saturation. In *IJCAR'08*, 2008

[dMB08b] L. de Moura and N. Bjørner. Deciding Effectively Propositional Logic using DPLL and substitution sets. In *IJCAR'08*, 2008

[GHN+04] H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast decision procedures. In R. Alur and D. Peled, editors, *Int. Conference on Computer Aided Verification (CAV 04)*, volume 3114 of *LNCS*, pages 175–188. Springer, 2004

[MSS96] J. Marques-Silva and K. A. Sakallah. GRASP - A New Search Algorithm for Satisfiability. In *Proc. of ICCAD'96*, 1996

[NO79] G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, 1979

[NO05] R. Nieuwenhuis and A. Oliveras. DPLL(T) with exhaustive theory propagation and its application to difference logic. In *Int. Conference on Computer Aided Verification (CAV'05)*, pages 321–334. Springer, 2005

References

[Opp80] D. Oppen. Reasoning about recursively defined data structures. *J. ACM*, 27(3):403–411, 1980

[PRSS99] A. Pnueli, Y. Rodeh, O. Shtrichman, and M. Siegel. Deciding equality formulas by small domains instantiations. *Lecture Notes in Computer Science*, 1633:455–469, 1999

[Pug92] William Pugh. The Omega test: a fast and practical integer programming algorithm for dependence analysis. In *Communications of the ACM*, volume 8, pages 102–114, August 1992

[RT03] S. Ranise and C. Tinelli. The smt-lib format: An initial proposal. In *Proceedings of the 1st International Workshop on Pragmatics of Decision Procedures in Automated Reasoning (PDPAR'03), Miami, Florida*, pages 94–111, 2003

[RS01] H. Ruess and N. Shankar. Deconstructing shostak. In *16th Annual IEEE Symposium on Logic in Computer Science*, pages 19–28, June 2001

[SLB03] S. Seshia, S. Lahiri, and R. Bryant. A hybrid SAT-based decision procedure for separation logic with uninterpreted functions. In *Proc. 40th Design Automation Conference*, pages 425–430. ACM Press, 2003

[Sho81] R. Shostak. Deciding linear inequalities by computing loop residues. *Journal of the ACM*, 28(4):769–779, October 1981