Combining Abstract Interpreters

Ashish Tiwari
Tiwari@csl.sri.com

Computer Science Laboratory
SRI International
Menlo Park CA 94025
http://www.csl.sri.com/~tiwari

Joint work with Sumit Gulwani

Outline of this Talk

• Abstract Interpretation
• Logical Lattices
• Combining Logical Lattices
• Combination can be hard
• Logical Product: The Correct Combination Lattice
• Combination Abstract Interpreter

Abstract Interpretation

\( X \) : state space
\( \rightarrow \) : binary transition relation on \( X \)
\( X_{\text{init}} \) : set of initial states, subset of \( X \)
\( \langle X, \rightarrow, X_{\text{init}} \rangle \) : Program

\( 2^X \rightarrow, X_{\text{init}} \) : Dynamical system
\( \bigcup_i \rightarrow^i (X_{\text{init}}) = \text{reachable states} \)

\( \langle A, \rightarrow, a_{\text{init}} \rangle \) : Approximate system over a lattice \( A \)
\( \bigcup_i \rightarrow^i (a_{\text{init}}) = \text{approx reachable states} \)
\( \) : fixpoint computation

Abstract Interpretation: Lattice

To build an abstract interpreter, we require

\( A \) : lattice
\( \rightarrow \) : transfer function
\( \) : ability to compute \( \rightarrow \) given \( \langle X, \rightarrow, X_{\text{init}} \rangle \) and \( A \)
\( \) : ability to compute the join in \( A \)
\( \) : ability to decide the lattice pre-order

For imperative programming languages, computing \( \rightarrow (a) \) often requires computing \( \cap \) and more.
**Abstract Interpretation: Example**

\[
\begin{align*}
x &:= 0; \quad \text{while (1) } \{ \ x := x+2; \}
\end{align*}
\]

The **concrete** state transition system:

\[
\begin{align*}
X &\quad : \ Z \\
\rightarrow &\quad : i \rightarrow i + 2 \\
X_{\text{init}} &\quad : \{0\} \\
\langle X, \rightarrow, X_{\text{init}} \rangle &\quad : \text{Program}
\end{align*}
\]

**Lattice:**

\[
\begin{align*}
\text{Even} &\quad : \{\ldots, -2, 0, 2, 4, \ldots\} \\
\text{Odd} &\quad : \{\ldots, -3, -1, 1, 3, \ldots\} \\
A &\quad : \{\emptyset, \text{Even}, \text{Odd}, Z\} \\
\sqsubseteq &\quad : \emptyset \sqsubseteq \text{Even}, \text{Odd} \sqsubseteq Z
\end{align*}
\]

**Logical Theory**

Components of a logical theory \( T_h \):

\[
\begin{align*}
\Sigma &\quad : \text{Signature containing function symbols, predicates} \\
T(\Sigma, \mathcal{V}) &\quad : \text{terms, } t := c \ | \ x \ | \ f(t, \ldots, t) \\
AF(\Sigma, \mathcal{V}) &\quad : \text{atomic formulas, } \phi := t = t \ | \ p(t, \ldots, t) \\
\text{Formulas} &\quad : \text{atomic formulas combined with boolean connectives} \\
T_h &\quad : \text{Set of sentences (valid in the theory)} \\
T_h \models \phi &\quad : \phi \text{ is valid in the theory } T_h
\end{align*}
\]

**Example: Contd**

In the abstract lattice,

\[
\begin{align*}
A &\quad : \{\emptyset, \text{Even}, \text{Odd}, Z\} \\
\rightarrow &\quad : a \rightarrow a \text{ for all } a \in A \\
\text{a}_{\text{init}} &\quad : \text{Even}
\end{align*}
\]

Reachable states = \( \bigcup_i \rightarrow^i (\text{a}_{\text{init}}) \)

= Even \sqcup Even \sqcup Even \sqcup \cdots

= Even

Thus, we have generated the invariant “\( x \) is even.”

**Logical Theory: Examples**

\[
\begin{align*}
\Sigma_{\text{LAE}} &\quad : \{0, 1, +, -\} \\
T_{\text{LAE}} &\quad : \text{Equality Axioms of } +, - \text{ (linear arithmetic with equality)} \\
\Sigma_{\text{LA}} &\quad : \{0, 1, +, -, <\} \\
T_{\text{LA}} &\quad : \text{Equality and inequality axioms of } +, - \text{ (LA with inequalities)} \\
\Sigma_{\text{Pol}} &\quad : \{0, 1, +, -, \ast\} \\
T_{\text{Pol}} &\quad : \text{Polynomial ring axioms} \\
\Sigma_{\text{UF}} &\quad : \{e_1, e_2, \ldots, f, g, \ldots\} \\
T_{\text{UF}} &\quad : \text{No axioms (Theory of uninterpreted functions/pure equality)}
\end{align*}
\]
Logical Lattices

Semi-lattice defined by
- elements: conjunction \( \phi \) of atomic formulas in \( Th \)
- preorder: \( \phi \subseteq \phi' \) if \( Th \models \phi \Rightarrow \phi' \)

We have
- meet \( \cap \) \( \Rightarrow \) logical and \( \land \)
- join \( \sqcup \) \( \Rightarrow \phi_1 \sqcup \phi_2 \) is the strongest \( \phi \) s.t. \( Th \models (\phi_1 \lor \phi_2) \Rightarrow \phi \)

Question: Is this semi-lattice a lattice?

Logical Lattices

Answer depends on the theory. Theories that define a logical lattice:

- Linear arithmetic with equality (Karr 1976)
  \( \text{Eg. } \{x = 0, y = 1\} \sqcup \{x = 1, y = 0\} = (x + y = 1) \)

- Linear arithmetic with inequalities (Cousot and Halbwachs 1978)
  \( \text{Eg. } \{x = 0\} \sqcup \{x = 1\} = \{0 \leq x, x \leq 1\} \)

  \( \text{Eg. } \{x = 0\} \sqcup \{x = 1\} = \{x(x - 1) = 0\} \)

- UFS + injectivity/acyclicity (Gulwani, T. and Necula 2004)
  \( \text{Eg. } \{x = a, y = f(a)\} \sqcup \{x = b, y = f(b)\} = \{y = f(x)\} \)

When this semilattice is a lattice, we call it a logical lattice.

UFS does not define a logical lattice

The join of two finite sets of facts need not be finitely presented. [Gulwani, T. and Necula 2004]

\[
\begin{align*}
\phi_1 & \equiv a = b \\
\phi_2 & \equiv fa = a \land fb = b \land ga = gb \\
\phi_1 \sqcup \phi_2 & \equiv \bigwedge_i g f^i a = g f^i b 
\end{align*}
\]

The formula \( \bigwedge_i g f^i a = g f^i b \) can not be represented by finite set of ground equations.

Proof. It induces infinitely many congruence classes with more than one signature.

Example: Abstract Intprtn over acyclic UFS lattice

With additional acyclicity restriction, UFS can be used to define a logical lattice.

\[
\begin{align*}
u & := c; v := c; \\
[ & u = c \land v = c ]
\end{align*}
\]

while (*) {
  \[
  \begin{align*}
  u & := F(u); \\
  v & := F(v);
  \end{align*}
  \]
}

[ \{u = F(c) \land v = F(c)\} \sqcup \{u = c \land v = c\}] 
\[u = v\]

We generate the invariant \( u = v \) this way.
Abstract Interpreter for Logical Lattices

<table>
<thead>
<tr>
<th>Lattice Op</th>
<th>Computing</th>
<th>When required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meet ⊓</td>
<td>∧</td>
<td>computing transfer functions</td>
</tr>
<tr>
<td>Join ⊔</td>
<td>??</td>
<td>control-flow merge (loop, if-then-else)</td>
</tr>
<tr>
<td>Preorder ⊆</td>
<td>⇒_{Th}</td>
<td>fixpoint detection</td>
</tr>
<tr>
<td>??</td>
<td>Quant Elim</td>
<td>transfer function for assignments</td>
</tr>
</tbody>
</table>

Join computation for logical lattices is not well-studied.

Examples: Logical Lattices

Most of the standard lattices considered for AI can be described as logical lattices over an appropriate theory $Th$

- Parity : $Σ = \{0, 1, +, -, even, odd\}$, $Th =$ axioms of even,odd (no =)
- Sign : $Σ = \{0, 1, +, -, pos, neg\}$, $Th =$ axioms of pos,neg (no =)
- Intervals : $Σ = \{0, 1, +, -, <e, >e\}$

In the above cases, atomic formulas of only special form (predicate applied on variables) are considered as lattice elements.

Recap

- Overview of abstract interpretation
  - Abstract interpretation can be used to generate invariants
- Overview of logical theories
  - Logical theories are described over a signature (a set of symbols) by axioms for those symbols
- Interesting lattices for AI obtained by considering conjunctions of atomic formulas in a given theory
- These semilattices may not be a lattice for arbitrary theories $Th$.
  As they are missing $∨ (⊔)$

Join Algorithms for Logical Lattices: Examples

- $Th_{LAE}$ : $\{x = z - 1, y = 1\} ⊔ \{z = y + 2, x = 2\} = \{x + y = z\}$
  - Karr’s 1976 algorithm
- $Th_{UP}$ : $\{x = a, y = fa\} ⊔ \{x = fa, y = ffa\} = \{y = fx\}$
  - Gulwani, T., Necula 2004
- $Th_{LA}$ : $\{x < 1, y < 0\} ⊔ \{x < 0, y < 1\} = \{x < 1, y < 1, x + y < 1\}$
  - Convex Hull
- $Th_{Pol}$ : $\{x = 0\} ⊔ \{y = 0\} = \{xy = 0\}$
  - Ideal Intersection

Many interesting unexplored problems here.
Combining Abstract Interpreters: Motivation

<table>
<thead>
<tr>
<th>Kind</th>
<th>Lattice elements</th>
<th>Lattice Preorder</th>
<th>Can verify</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical+</td>
<td>Inf conj of atm facts in $T_1 \cup T_2$</td>
<td>$\Rightarrow T_1 \cup T_2$</td>
<td>1, 2, 3, 4</td>
</tr>
<tr>
<td>Logical</td>
<td>conj of atm facts in $T_1 \cup T_2$</td>
<td>$\Rightarrow T_1 \cup T_2$</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>Reduced</td>
<td>$L_1 \times L_2$</td>
<td>$\Rightarrow T_1 \cup T_2$</td>
<td>1</td>
</tr>
<tr>
<td>Direct</td>
<td>$L_1 \times L_2$</td>
<td>$\Rightarrow T_1 \times T_2$</td>
<td>1</td>
</tr>
</tbody>
</table>

if (*)

\[
x := 1; \ y := F(1); \ z := G(2);
\]

else

\[
x := 4; \ y := F(8-x); \ z := G(5);
\]

Assertions:

\[
x \geq 1, \ y = F(x), \ z = G(x+1),
\]

\[
H(x) + H(5-x) = H(1) + H(4)
\]

Combining Logical Lattices

Combining abstract interpreters is not easy \cite{Cousot76}

Given logical lattices $L_1$ and $L_2$:

- Direct product: $\langle L_1 \times L_2, t \Rightarrow T_{h_1} \times T_{h_2} \rangle$
- Reduced product: $\langle L_1 \times L_2, T_{h_1 \cup T_{h_2}} \rangle$
- Logical+ product: (Infinite conjunctions of $AF(\Sigma_1 \cup \Sigma_2, \nu), T_{h_{1 \cup T_{h_2}}}$)
- Logical product: (Conjunctions of $AF(\Sigma_1 \cup \Sigma_2, \nu), T_{h_{1 \cup T_{h_2}}}$ with some restriction)

Issues in Combining Logical Lattices

Why not use the logical+ product?

The logical+ product is undesirable for two reasons:

1. $T_{h_1 \cup T_{h_2}}$ need not define a lattice on finite conjunctions even if $T_{h_1}$ and $T_{h_2}$ define logical lattices

   $T_{h_{UF1}}$ : theory of uninterpreted functions with injectivity

   $T_{h_{LAE}}$ : theory of linear arithmetic with only equality

   Now,

   \[
   (x = 0 \land y = 1) \cup (x = 1 \land y = 0)
   = x + y = 1 \land C[x] + C[y] = C[0] + C[1]
   \]

2. Combination can be hard

   Let us consider the decision version of the abstract interpretation problem
Assertion Checking Problem

Given:
P : Program
φ : Assertion over program variables at point π in P

Problem: Is φ an invariant at π?

In contrast, assertion generation problem seeks to synthesize all invariants at point π.

Assertion Checking over Logical Lattices

Undecidable in general for most theories
So we consider non-deterministic conditionals in the program model

• Acyclic UFS theory: Polynomial time [Gulwani and Necula 2004]
• Linear arithmetic with equality. Polynomial time [Karr 1976]

Question. What about the combination?

Logical+ product:

elements : inf conjunction φ of atomic formulas in Th₁ ∪ Th₂
preorder : ⇒Th₁∪Th₂

Program Model

A program is given as a flowchart with three kinds of nodes:

(a) Join Node
(b) Assignment Node
(c) Conditional Node

Fixing a theory Th:
e : term (expression) in the theory
p : atomic formula in the theory
E : elements of the logical lattice induced by Th

Example

x :=0; y := 0; x := c; y := c; x :=0; y := 0;
u := 0; v := 0; u := c; v := c; u := 0; v := 0;

x :=0; y := 0;
u := 0; v := 0;
while (*) {
    x := u + 1;
y := 1 + v;
u := F(x);
v := F(y);
}
assert( x = y )

Σ = ΣLA ∪ ΣUFS
Th = ThLA + ThUFS

Σ = ΣUFS
Σ = ΣLA
Th = ThUFS
Th = ThLA
coNP-hardness of Assertion Checking

for Combination

Key Idea: Disjunctive assertion can be encoded in the combination.

\[ x = a \lor x = b \iff F(a) + F(b) = F(x) + F(a + b - x) \]

Using this recursively, we can write an assertion (atomic formula) which holds iff \( x = 0 \lor x = 1 \lor \cdots \lor x = m - 1 \) holds.

For e.g., encoding for \( x = 0 \lor x = 1 \lor x = 2 \) is obtained by encoding

\[ Fx = F2 \lor Fx = F0 + F1 - F(1 - x); \]
\[ F(F0 + F1 - F(1 - x)) + FF2 = FFx + F(F0 + F1 + F2 - F(1 - x) - Fx) \]

\[ \psi: \text{boolean 3-SAT instance with } m \text{ clauses and } k \text{ variables} \]

\[ x_i := 0, \text{for } i = 1, 2, \ldots, m \]

for \( i = 1 \) to \( k \) do

if (*) then

\[ x_j := 1, \forall j: \text{variable } i \text{ occurs positively in clause } j \]

else

\[ x_j := 1, \forall j: \text{variable } i \text{ occurs negatively in clause } j \]

\[ \text{sum} := x_1 + \cdots + x_m \]

assert(sum = 0 \lor \cdots \lor sum = m - 1)

Assertion is valid IFF \( \psi \) is unsatisfiable

Recap

- Logical theories used to define logical lattices
- There are different ways of combining these logical lattices
- The ideal way would have been the logical++ product
- Logical++ product has two problems:
  - In general, it is a lattice only if we consider infinite conjunctions
  - Assertion checking for nondeterministic programs is hard on logical++ products even when it is in PTime for individual lattices

Is assertion checking for UFS+LA language even decidable?
**Assertion Checking Algorithm**

Backward analysis:
- Starting with the assertion, use weakest precondition computation
- At each step, replace the formula $\psi$ computed at any program point by $Unif(\psi)$

This method is both sound and complete due to
- correctness of WP computation
- connection between unification and assertion checking

**Question.** What is $Unif(\psi)$?
**Question.** Does it terminate (reach fixpoint across loops)?

---

**Why is Unification Sound in Backward Analysis?**

First, if $Th \models Unif(e_1 = e_2)$ then $Th \models e_1 = e_2$.

Conversely, let $\theta: substitution$ that maps $x$ to a symbolic value of $x$ at point $\pi$ (along some execution path)

(Symbolic value is in terms of input variables)

If assertion $e_1 = e_2$ holds at $\pi$, then,

$Th \models \theta \Rightarrow e_1 = e_2$, i.e., $Th \models e_1\theta = e_2\theta$

Since $\{\sigma_1, \ldots, \sigma_k\}$ is a complete set of $Th$-unifiers, $\vdash \theta = Th \sigma_j\theta'$ for some $j$

We will show

$Th \models \theta \Rightarrow x = x\sigma_j$, i.e., $Th \models x\theta = x\sigma_j\theta$

But

$Th \models (x\theta = x\sigma_j\theta' = x\sigma_j\sigma_j' = x\sigma_j\theta)$

---

**Unification in Assertion Checking**

Assume that all assignments in program $P$ are of the form

$x := e$

An assertion $e_1 = e_2$ holds at point $\pi$ in $P$ iff the assertion $Unif(e_1 = e_2)$ hold at $\pi$ in $P$.

This also extends to arbitrary assertion $\phi$.

If $\{\sigma_1, \ldots, \sigma_k\}$ is a complete set of $Th$-unifiers for $e_1 = e_2$, then

$$Unif(e_1 = e_2) = \bigvee_{i=1}^{k} (\bigwedge x = x\sigma_i)$$

---

**Why backward analysis need not terminate?**

Forward analysis will not terminate since the lattice has infinite height:

$x := 0;$
while (*) do
$x := x + 1;$
Assert($x = 0 \lor x = 1 \lor \cdots \lor x = m)$;

But due to the unifier computations, backward analysis terminates
Termination of Algorithm

At each program point, the proof obligation formula is of the form

$$\bigvee_{l=1}^{m} \bigwedge_{x} (x = x \sigma_l)$$

In backward analysis across a loop, in each successive iteration, this formula will become stronger

But this can not happen indefinitely:
Assign the following measure to the above formula

$$\{n - || \bigwedge_{x} (x = x \sigma)||\}$$

This measure decreases in the well-founded ordering $>^m$.

Assertion Checking and Unification

<table>
<thead>
<tr>
<th>UFS</th>
<th>unitary</th>
<th>PTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>LA</td>
<td>unitary</td>
<td>PTime</td>
</tr>
<tr>
<td>UFS+LA</td>
<td>finitary*</td>
<td>coNP-hard for loop-free, decidable in general</td>
</tr>
</tbody>
</table>

*Skipped detail:
Unification in Abelian Groups + free function symbols follows from general combination result

- Schmidt-Schuass 1989
- Baader-Schulz 1992

Recap

- Logical+ product is not a good choice for defining combinations
- Digression:
  - UFS+LA assertion checking problem is coNP-hard—even for loop-free programs
  - UFS+LA assertion checking problem is decidable
Both these results depend on a novel connection between unification and assertion checking

We wish to get PTime overhead for the operations $\bigvee, \bigwedge, \subseteq$, fixpoint, and $SP$ in the combination

Logical Product

Given two logical lattices, we define the logical product as:

- elements: conjunction $\phi$ of atomic formulas in $Th_1 \cup Th_2$
- $E \subseteq E'$: $E \Rightarrow_{Th_1 \cup Th_2} E'$ and $\text{AlienTerms}(E') \subseteq \text{Terms}(E)$

| Alien Terms($E$) | subterms in $E$ that belong to different theory |
| Terms($E$) | all subterms in $E$, plus all terms equivalent to these subterms (in $Th_1 \cup Th_2 \cup E$) |

Eg. $\{x = F(a + 1), y = a\} \cup \{x = F(b + 1), y = b\} = \{x = F(y + 1)\}$:

- $x = F(a + 1) \land y = a \Rightarrow x = F(y + 1)$
- $x = F(b + 1) \land y = b \Rightarrow x = F(y + 1)$
- $x = F(a + 1) \land y = a \Rightarrow y + 1 = a + 1$
- $x = F(b + 1) \land y = b \Rightarrow y + 1 = b + 1$
Satisfiability in $Th_1 \cup Th_2$

Nelson-Oppen presented a general method for combining satisfiability decision procedures of $Th_1$ and $Th_2$ to get one for $Th_1 \cup Th_2$

$E$: conjunction of atomic formulas in $Th_1 \cup Th_2$

1. First purify $E$ into $E_1$ and $E_2$
   Eg. $4y_3 \leq f(2y_2 - y_1) \implies \{4y_3 \leq a_2, a_1 = 2y_2 - y_1 \}$ and $\{a_2 = f(a_1)\}$
2. Each $Th_i$ generates variable equalities implied by $E_i$ and passes it on the other theory
   This step can be done by a $Th_i$-satisfiability procedure
3. Repeat Step (2) until no more variable equalities can be exchanged
4. Declare satisfiable if $Th_1$ and $Th_2$ both declare satisfiable

Works for convex, stably-infinite, disjoint theories

Combining the Preorder Test

Required for testing convergence of fixpoint

$E \subseteq E'$ iff
1. $Th_1 \cup Th_2 \models E \Rightarrow E'$
2. $\text{AlienTerms}(E') \subseteq \text{Terms}(E)$

So, the crucial problem is (1)

(1) is solved by combining satisfiability testing decision procedures for $Th_1$ and $Th_2$

$Th_1 \cup Th_2 \models E \Rightarrow E'$

IFF $E \land \neg E'$ is unsatisfiable

IFF $E \land \neg e$ is unsatisfiable for all $e \in E'$

Combining preorder test: NO Procedure

We can modify NO procedure to also deduce facts

$y_1 \leq 4y_3 \leq f(2y_2 - y_1), y_1 = f(y_1), y_2 = f(f(y_1)) \Rightarrow y_1 = 4y_3$

$a_1 = 2y_2 - y_1\quad a_2 = f(a_1)$

$y_1 \leq 4y_3 \leq a_2\quad y_1 = f(y_1), y_2 = f(f(y_1))$

$y_1 = y_2\quad \rightarrow y_1 = a_1$

$y_1 = a_2\quad \leftarrow$

Now, the linear arithmetic procedure can deduce $y_1 = 4y_3$

Works for convex, stably-infinite, disjoint theories
Combining Join Operator

Given procedures:

\( \text{Join}_{L_1}(E_1, E_r) \): Computes \( E_1 \sqcup E_r \) in lattice \( L_1 \)

\( \text{Join}_{L_2}(E_1, E_r) \): Computes \( E_1 \sqcup E_r \) in lattice \( L_2 \)

We wish to compute \( E_1 \sqcup E_r \) in the logical product \( L_1 \ast L_2 \)

Example.

\[ \{ z = a + 1, y = f(a) \} \sqcup \{ z = b - 1, y = f(b) \} = \{ y = f(1 + z) \} \]

Existential Quantification Operator

Required to compute transfer function for assignments

\( E = QE_L(E', V) \) if \( E \) is the least element in lattice \( L \) s.t.

- \( E' \subseteq L \)
- \( \text{Vars}(E) \cap V = \emptyset \)

Examples:

- \( QE_L\{x < a, a < y\}, \{a\} = \{x \leq y\} \)
- \( QE_L\{x = f(a), y = f(f(a))\}, \{a\} = \{y = f(x)\} \)
- \( QE_L\{a < b < y, z = c + 1, a = fb, c = fb\}, \{a, b, c\} = \{f(z - 1) \leq y\} \)

How to construct \( QE_L\ast UF \) using \( QE_L\) and \( QE_U \)?

Combining QE Operators

Problem

\[ a < b \leq y, z = c + 1, a = ffb, c = fb \quad \{a, b, c\} \]

Purify+NOSat

\[ a < b \leq y \quad a = ffb, c = fb \]

QSat

\[ a \mapsto fc, c \mapsto z - 1 \]

QSat

\[ a \mapsto fc \]

Base QEs

\[ QE_L, QE_U \]

Substitute

\[ c \mapsto z - 1, a \mapsto fc \]

Return

\[ f(z - 1) \leq y \]
**Fixpoint Computation**

Termination of analysis across loops required bounding height of lattice

\[ H_L(E) = \text{no. of elements in any chain above } E \text{ in lattice } L \]

\[ H_{L_1 \times L_2}(E) \leq H_{L_1}(E_1) + H_{L_2}(E_2) + |\text{AlienTerms}(E)| \]

where \( E_1, E_2 \) are purified and NO-saturated components of \( E \)

---

**Correctness and Complexity**

- Algorithms \( QE_{L_1 \times L_2} \) and \( \text{Join}_{L_1 \times L_2} \) are sound
- They are complete when the underlying theories \( T_1 \) and \( T_2 \) are convex, stably infinite and disjoint
- Proof of correctness is technical
  
  Heavily based on the correctness of NO procedure
- Complexity of \( QE \) and \( \text{Join} \) is worst-case quadratic in the complexity of these operations for individual lattices

---

**Example of Incompleteness**

\[ QE_P(\{\text{odd}(x'), x = x' - 1\}, \{x'\}) = \{\text{even}(x)\} \]

\[ QE_S(\{\text{pos}(x'), x = x' - 1\}, \{x'\}) = \{\} \]

\[ QE_{P \times S}(\{\text{pos}(x'), \text{odd}(x'), x = x' - 1\}, \{x'\}) = \{\text{pos}(x), \text{even}(x)\} \]

But our algorithm only outputs \( \text{even}(x) \). Why?

The theories of parity and sign are not disjoint.

---

**Conclusions**

- Logical lattices are good candidates for thinking about and building abstract interpreters
- Logical lattices can be combined in a new and important way

**Logical Products:**

- Logical product is more powerful than direct or reduced product
- Operations on logical lattices can be modularly combined to yield operations for logical products
- Using ideas from the classical Nelson-Oppen combination method
Conclusions

- The assertion checking problem:
  - Equations in an assertion can be replaced by its complete set of $T_h$-unifiers for purposes of assertion checking
  - Assertion checking over “lattices” defined by combination of two logical lattices can be hard, even when it is in PTime for the lattices defined by individual theories
  - Finitary $T_h$-unification algorithm implies decidability of assertion checking for the logical lattices defined by $T_h$

References