Unification in Assertion Checking

Over Logical Lattices

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
Assertion Checking Problem

Given:

\[ P \quad : \quad \text{Program} \]
\[ \phi \quad : \quad \text{An assertion over program variables at point } \pi \text{ in } P \]

Problem: Is \( \phi \) an invariant at \( \pi \)?

In contrast, assertion generation problem seeks to synthesize all invariants at point \( \pi \).
Language and Theory Restrictions

Assume the symbols used for specifying the program $P$ and the assertion $\phi$ come from some

$\Sigma$: signature

$Th$: theory

General programs are abstracted to the chosen language by abstracting each assignment and conditional in the program (preserving its control flow)

Skipped Detail: How do we go from general program to such an abstraction.
Example

\begin{align*}
x & := 0; \ y := 0; \\
u & := 0; \ v := 0; \\
\text{while } (*) \{ \\
& \quad x := u + 1; \ \\
& \quad y := 1 + v; \\
& \quad u := F(x); \\
& \quad v := F(y); \\
\}& \}
\end{align*}

\begin{align*}
x & := c; \ y := c; \\
u & := c; \ v := c; \\
\text{while } (*) \{ \\
& \quad x := G(u, 1); \ \\
& \quad y := G(1, v); \\
& \quad u := F(x); \\
& \quad v := F(y); \\
\}& \}
\end{align*}

\begin{align*}
x & := 0; \ y := 0; \\
u & := 0; \ v := 0; \\
\text{while } (*) \{ \\
& \quad x := u + 1; \ \\
& \quad y := 1 + v; \\
& \quad u := *; \\
& \quad v := *; \\
\}& \}
\end{align*}

assert( x = y )

\begin{align*}
\Sigma & = \Sigma_{LA} \cup \Sigma_{UFS} \\
Th & = Th_{LA} + Th_{UFS}
\end{align*}

\begin{align*}
\Sigma & = \Sigma_{UFS} \\
Th & = Th_{UFS}
\end{align*}

\begin{align*}
\Sigma & = \Sigma_{LA} \\
Th & = Th_{LA}
\end{align*}
Outline of this Talk

• Abstract interpretation for assertion generation+checking over logical lattices

• Link between unification and assertion checking

• Two consequences:
  ○ NP-hardness of assertion checking (for loop-free programs) over UFS+LA language
  ○ decidability of assertion checking for UFS+LA language
Abstract Interpretation

- Fix a lattice
- Map sets of state $\phi$ of the program onto lattice elements $\alpha(\phi)$
- Compute transfer functions:

$$\{\phi_1\} x := e \{\phi_2\} \iff \alpha(\phi_1) \rightarrow \alpha(\phi_2)$$
$$\{\phi_1\} \text{ if } (c) \text{ then } \{\phi_2\} \text{ else } \{\phi_3\} \iff \alpha(\phi_1) \rightarrow \alpha(\phi_1) \land \alpha(c);$$
$$\alpha(\phi_1) \rightarrow \alpha(\phi_1) \land \alpha(\neg c);$$

conditionals $\iff$ meet in the lattice
merges $\iff$ join in the lattice
loop $\iff$ fixpoint in the lattice
Logical Lattices

Lattice defined over conjunction $\phi$ of atomic formulas in $Th$ by

- **meet** in the lattice $\Rightarrow$ logical **and**
- **join** in the lattice $\Rightarrow$ \{ $\phi : Th \models (\phi_1 \lor \phi_2) \Rightarrow \phi$ \}

**Question 1.** Is this a well-defined lattice?

**Answer.** Depends on the theory.

- Linear arithmetic with equality (Karr 1976)
- Linear arithmetic with inequalities (Cousot and Halbwachs 1978)
- Nonlinear (polynomial) equations (Rodriguez-Carbonell and Kapur 2004)
- UFS + injectivity/acyclicity (Gulwani, T. and Necula 2004)
  
  ...
The join of two finite sets of facts need not be finitely presented. [Gulwani, T. and Necula 2004]

\[ \phi_1 \equiv a = b \]
\[ \phi_2 \equiv f a = a \land f b = b \land g a = g b \]
\[ \phi_1 \sqcup \phi_2 \equiv \bigwedge_i g f^i a = g f^i b \]

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. Ex: Complete the proof.
Example: Abstract Intprtn over acyclic UFS lattice

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

\[
\begin{align*}
u := c; \quad v := c; \\
\left[ u = c \land v = c \right] \\
\text{while (*) } \{ \\
\quad u := F(u); \\
\quad v := F(v); \\
\quad \left[ (u = F(c) \land v = F(c)) \sqcup (u = c \land v = c) \right] \\
\} \\
\left[ u = v \right]
\end{align*}
\]

We generate the invariant \( u = v \) this way.
Known Results

Assertion checking over lattices defined by:

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

Question. What about the combination?
Outline of this Talk

- Abstract interpretation for assertion generation+checking over logical lattices
- Link between unification and assertion checking
- Two consequences for UFS+LA combination:
  - NP-hardness of assertion checking (for loop-free programs) over above language
  - decidability of assertion checking for above language
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 $\text{Unif}(e_1 = e_2)$ 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

$$\text{Unif}(e_1 = e_2) = \bigvee_{i=1}^{k} (\bigwedge x = x\sigma_i)$$
Proof of Main Result

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,

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

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

We will show

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

But

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

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

\[ F(F0 + F1 - F(1 - x)) + FF2 = FFx + F(F0 + F1 + F2 - F(1 - x) - Fx) \]
**coNP-hardness of Assertion Checking**

\( \psi: \) boolean 3-SAT instance with \( m \) clauses

\[ x_i := 0, \text{for } i = 1, 2, \ldots, m \]
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\( (\text{sum} = 0 \lor \cdots \lor \text{sum} = m - 1) \)

Assertion is valid IFF \( \psi \) is unsatisfiable
This procedure checks whether $x \in \{0, \ldots, m - 1\}$. $h_0 := F(x)$;

for $j = 0$ to $m - 1$ do

\[ h_{0,j} := F(j); \]

for $i = 1$ to $m - 1$ do

\[ s_{i-1} := h_{i-1,0} + h_{i-1,i}; \]
\[ h_{i} := F(h_{i-1}) + F(s_{i-1} - h_{i-1}); \]

for $j = 0$ to $m - 1$ do

\[ h_{i,j} := F(h_{i-1,j}) + F(s_{i-1} - h_{i-1,j}); \]

Assert($h_{m-1} = h_{m-1,0}$);

The assertion holds iff $x \in \{0, \ldots, m - 1\}$.

Assertion checking on combination lattice is coNP-hard.
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
- main result of this talk

**Question.** Does it terminate (reach fixpoint across loops)?
Forward analysis will **not** terminate since the **lattice** has **infinite** height:

```plaintext
x := 0;
while (*) do
    x := x + 1;
Assert(x = 0 ∨ x = 1 ∨ ⋯ ∨ 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 cannot 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>System</th>
<th>Complexity</th>
<th>Complexity Additional Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>UFS</td>
<td>unitary PTime</td>
<td></td>
</tr>
<tr>
<td>LA</td>
<td>unitary PTime</td>
<td></td>
</tr>
<tr>
<td>UFS+LA</td>
<td>finitary* coNP-hard</td>
<td>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
Conclusion

- **Equations** in an assertion can be replaced by its **complete set of $Th$-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 $Th$-unification algorithm implies **decidability** of assertion checking for the logical lattices defined by $Th$