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 : Program

ϕ : An 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.

Language and Theory Restrictions

Assume the symbols used for specifying the program P and the assertion ϕ come from some

Σ: 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

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

$\Sigma = \Sigma_{LA} \cup \Sigma_{UFS}$

$Th = Th_{LA} + Th_{UFS}$

```
x := c; y := c;
u := c; v := c;
while (*) {
    x := G(u, 1);
    y := G(1, v);
    u := F(x);
    v := F(y);
}
assert( x = y )
```

$\Sigma = \Sigma_{UFS}$

$Th = Th_{UFS}$

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

$\Sigma = \Sigma_{LA}$

$Th = Th_{LA}$

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 ϕ of the program onto lattice elements $\alpha(\phi)$
- Compute transfer functions:

$$
\begin{array}{rcl}
\{\phi_1\}x := e\{\phi_2\} & \mapsto & \alpha(\phi_1) \rightarrow \alpha(\phi_2) \\
\{\phi_1\}\ \text{if}\ (c)\ \text{then}\ \{\phi_2\}\ \text{else}\ \{\phi_3\} & \mapsto & \alpha(\phi_1) \rightarrow \alpha(\phi_1) \wedge \alpha(c); \\
 & & \alpha(\phi_1) \rightarrow \alpha(\phi_1) \wedge \alpha(\neg c); \\
\text{conditionals} & \mapsto & \text{meet in the lattice} \\
\text{merges} & \mapsto & \text{join in the lattice} \\
\text{loop} & \mapsto & \text{fixpoint in the lattice}
\end{array}
$$

Logical Lattices

Lattice defined over conjunction ϕ of atomic formulas in Th by

meet in the lattice $\mapsto$ logical and

join in the lattice $\mapsto$ $\{\phi : Th \models (\phi_1 \vee \phi_2) \Rightarrow \phi\}$

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)

$\vdots$

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]

$$\begin{aligned}
\phi_1 &\equiv a = b \\
\phi_2 &\equiv fa = a \wedge fb = b \wedge ga = gb \\
\phi_1 \sqcup \phi_2 &\equiv \bigwedge_i gf^i a = gf^i b
\end{aligned}$$

The formula $\bigwedge_i gf^i a = gf^i b$ 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.

u := c; v := c;

[$u = c \wedge v = c$]

while (*) {

u := F(u);

v := F(v);

[$(u = F(c) \wedge v = F(c)) \sqcup (u = c \wedge v = c)$]

}

[$u = v$]

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

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* π *in* P *iff*
the assertion $Unif(e_1 = e_2)$ *hold at* π *in* P.
This also extends to arbitrary assertion ϕ.

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

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

Proof of Main Result

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

Conversely, let θ: substitution that maps x to a symbolic value of x at point π (along some exectution path)

(Symbolic value is in terms of input variables)

If assertion $e_1 = e_2$ holds at π, 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.

$$x = a \vee x = b \quad \Leftrightarrow \quad F(a) + F(b) = F(x) + F(a + b - x)$$

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

For e.g., encoding for $x = 0 \vee x = 1 \vee x = 2$ is obtained by encoding $Fx = F2 \vee Fx = F0 + F1 - F(1 - x)$:

$$F(F0 + F1 - F(1 - x)) + FF2 = FFx + F(F0 + F1 + F2 - F(1 - x) - Fx)$$

coNP-hardness of Assertion Checking

ψ: boolean 3-SAT instance with m clauses

$x_i := 0$, for $i = 1, 2, \ldots, m$
for $i = 1$ to k do
    if (*) then
        $x_j := 1, \forall j$: variable i occurs positively in clause j
    else
        $x_j := 1, \forall j$: variable i occurs negatively in clause j
$sum := x_1 + \cdots + x_m$
assert($sum = 0 \vee \cdots \vee sum = m - 1$)

Assertion is valid IFF ψ is unsatisfiable

coNP-hardness of Assertion Checking

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 ψ 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)?

Why it need not terminate?

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

$x := 0;$

while (*) do

$\quad x := x + 1;$

Assert($x = 0 \vee x = 1 \vee \cdots \vee x = m$);

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 abovw formula

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

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

Assertion Checking and Unification

UFS	unitary	PTime
LA	unitary	PTime
UFS+LA	finitary*	coNP-hard for loop-free, decidable in general

*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