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_{init}	:	set of initial states, subset of X
$\langle X, \rightarrow, X_{init} \rangle$	:	Program
$\langle 2^X, \rightarrow, X_{init} \rangle$	:	Dynamical system
	:	$\bigcup_i \rightarrow^i (X_{init})$ = reachable states
$\langle A, \rightarrow, a_{init} \rangle$	:	Approximate system over a lattice A
	:	$\bigsqcup_i \rightarrow^i (a_{init})$ = 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 $\langle X, \rightarrow, X_{init} \rangle$ and A
$\sqcup$	:	ability to compute the join in A
$\sqsubseteq$	:	ability to decide the lattice pre-order

For imperative programming languages, computing $\rightarrow (a)$ often requires computing $\sqcap$ and more.

Abstract Interpretation: Example

```
x := 0;   while (1) { x := x+2; }
```

The concrete state transition system:

X	:	$\mathbb{Z}$
$\rightarrow$	:	$i \rightarrow i + 2$
X_{init}	:	$\{0\}$
$\langle X, \rightarrow, X_{init} \rangle$	:	Program

Lattice:

$Even$	:	$\{\ldots, -2, 0, 2, 4, \ldots\}$
Odd	:	$\{\ldots, -3, -1, 1, 3, \ldots\}$
A	:	$\{\emptyset, Even, Odd, \mathbb{Z}\}$
$\sqsubseteq$	:	$\emptyset \sqsubseteq Even, Odd \sqsubseteq \mathbb{Z}$

Example: Contd

In the abstract lattice,

$$\begin{array}{lcl} A & : & \{\emptyset, Even, Odd, \mathbb{Z}\} \\ \rightarrow & : & a \rightarrow a \text{ for all } a \in A \\ a_{init} & : & Even \end{array}$$

$$\begin{array}{rcl} \text{Reachable states} & = & \bigsqcup_i \rightarrow^i (a_{init}) \\ & = & Even \sqcup Even \sqcup Even \sqcup \cdots \\ & = & Even \end{array}$$

Thus, we have generated the invariant "x is even."

Logical Theory

Components of a logical theory Th:

Σ	:	Signature containing function symbols, predicates
$T(\Sigma, \mathcal{V})$	:	terms, $t := c \mid x \mid f(t, \ldots, t)$
$AF(\Sigma, \mathcal{V})$	:	atomic formulas, $\phi := t = t \mid p(t, \ldots, t)$
Formulas	:	atomic formulas combined with boolean connectives
Th	:	Set of sentences (valid in the theory)
$Th \models \phi$	:	ϕ is valid in the theory Th

Logical Theory: Examples

Σ_{LAE} : $\{0, 1, +, -\}$

Th_{LAE} : Equality Axioms of $+, -$ (linear arithmetic with equality)

Σ_{LA} : $\{0, 1, +, -, <\}$

Th_{LA} : Equality and inequality axioms of $+, -$ (LA with inequalities)

Σ_{Pol} : $\{0, 1, +, -, *\}$

Th_{Pol} : Polynomial ring axioms

Σ_{UF} : $\{c_1, c_2, \ldots, f, g, \ldots\}$

Th_{UF} : No axioms (Theory of uninterpreted functions/pure equality)

Logical Lattices

Semi-lattice defined by

elements	:	conjunction ϕ of atomic formulas in Th
preorder	:	$\phi \sqsubseteq \phi'$ if $Th \models \phi \Rightarrow \phi'$

We have

meet $\sqcap$	$\mapsto$	logical and $\wedge$
join $\sqcup$	$\mapsto$	$\phi_1 \sqcup \phi_2$ is the strongest ϕ s.t. $Th \models (\phi_1 \vee \phi_2) \Rightarrow \phi$

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)
  Eg. $\{x = 0, y = 1\} \sqcup \{x = 1, y = 0\} = (x + y = 1)$
- Linear arithmetic with inequalities (Cousot and Halbwachs 1978)
  Eg. $\{x = 0\} \sqcup \{x = 1\} = \{0 \leq x, x \leq 1\}$
- Nonlinear equations (polynomials) (Rodriguez-Carbonell and Kapur 2004)
  Eg. $\{x = 0\} \sqcup \{x = 1\} = \{x(x - 1) = 0\}$
- UFS + injectivity/acyclicity (Gulwani, T. and Necula 2004)
  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]

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

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.

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.

Examples: Logical Lattices

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

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

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 $\vee$ ($\sqcup$)

Abstract Interpreter for Logical Lattices

Lattice Op		Computing		When required
Meet $\sqcap$	:	$\wedge$	:	computing transfer functions
Join $\sqcup$	:	??	:	control-flow merge (loop, if-then-else)
Preorder $\sqsubseteq$	:	$\Rightarrow_{Th}$	:	fixpoint detection
??	:	Quant Elim	:	transfer function for assignments

Join computation for logical lattices is not well-studied.

Join Algorithms for Logical Lattices: Examples

Th_{LAE} : $\{x = z - 1, y = 1\} \sqcup \{z = y + 2, x = 2\} = \{x + y = z\}$

Karr's 1976 algorithm

Th_{UF} : $\{x = a, y = fa\} \sqcup \{x = fa, y = ffa\} = \{y = fx\}$

Gulwani, T., Necula 2004

Th_{LA} : $\{x < 1, y < 0\} \sqcup \{x < 0, y < 1\} = \{x < 1, y < 1, x + y < 1\}$

Convex Hull

Th_{Pol} : $\{x = 0\} \sqcup \{y = 0\} = \{xy = 0\}$

Ideal Intersection

Many interesting unexplored problems here.

Combining Abstract Interpreters: Motivation

```
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 )
```

$\Sigma = \Sigma_{LA}$

$Th = Th_{LA}$

Combining Logical Lattices

Combining abstract interpreters is not easy [Cousot76]

Given logical lattices L_1 and L_2:

- Direct product: $\langle L_1 \times L_2, \ \Rightarrow_{Th_1} \times \Rightarrow_{Th_2} \rangle$
- Reduced product: $\langle L_1 \times L_2, \ \Rightarrow_{Th_1 \cup Th_2} \rangle$
- Logical+ product: $\langle$Infinite* conjunctions of $AF(\Sigma_1 \cup \Sigma_2, \mathcal{V}), \Rightarrow_{Th_1 \cup Th_2} \rangle$
- Logical product:
  $\langle$Conjunctions of $AF(\Sigma_1 \cup \Sigma_2, \mathcal{V}), \Rightarrow_{Th_1 \cup Th_2}$ with some restriction$\rangle$

Different Kinds of Combinations

Kind	Lattice elements	Lattice Preorder	Can verify
Logical+	Inf conj of atm facts in $T_1 \cup T_2$	$\Rightarrow_{T_1 \cup T_2}$	1,2, 3 , 4
Logical	conj of atm facts in $T_1 \cup T_2$	$\Rightarrow^{\preceq}_{T_1 \cup T_2}$	1,2, 3
Reduced	$L_1 \times L_2$	$\Rightarrow_{T_1 \cup T_2}$	1,2
Direct	$L_1 \times L_2$	$\Rightarrow_{T_1} \times \Rightarrow_{T_2}$	1

```
if (*)
   x := 1; y := F(1); z := G(2);
else
   x := 4; y := F(8-x); z := G(5);
```

Assertions: $x \geq 1$, $y = F(x)$, $z = G(x+1)$,
$H(x) + H(5-x) = H(1) + H(4)$

Issues in Combining Logical Lattices

Why not use the logical+ product?

The logical+ product is undesirable for two reasons:

1. $Th_1 \cup Th_2$ need not define a lattice on finite conjunctions even if Th_1 and Th_2 define logical lattices

   Th_{UFI} : theory of uninterpreted functions with injectivity

   Th_{LAE} : theory of linear arithmetic with only equality

   Now,

$$\begin{aligned}
&(x = 0 \wedge y = 1) \sqcup (x = 1 \wedge y = 0) \\
= \quad & x + y = 1 \wedge C[x] + C[y] = C[0] + C[1]
\end{aligned}$$

2. Combination can be hard

Let us consider the decision version of the abstract interpretation problem

Assertion Checking Problem

Given:

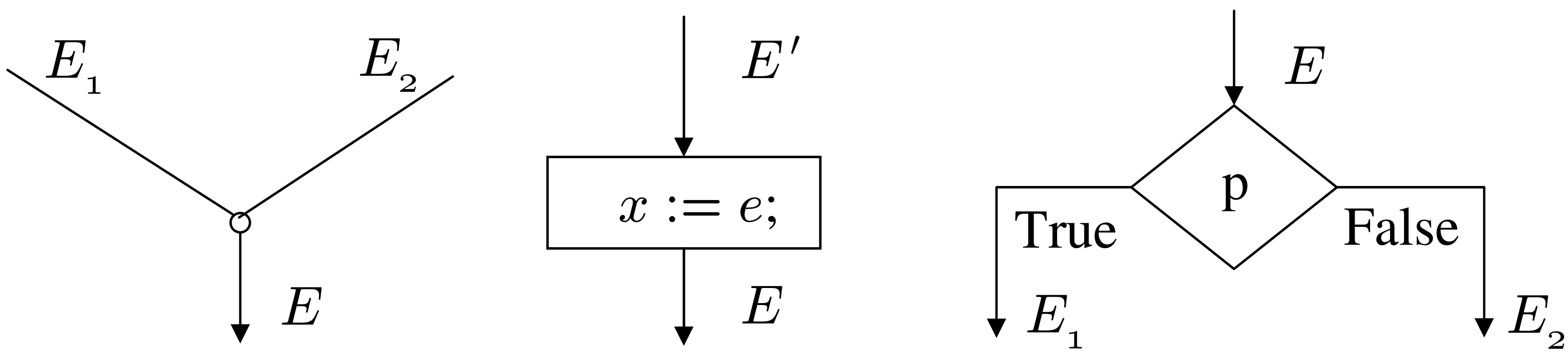
P	:	Program
ϕ	:	Assertion over program variables at point π in P

Problem: Is ϕ an invariant at π ?

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

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

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 $Th_1 \cup Th_2$

preorder : $\Rightarrow_{Th_1 \cup Th_2}$

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;
while (*) {	while (*) {	while (*) {
x := u + 1;	x := G(u, 1);	x := u + 1;
y := 1 + v;	y := G(1, v);	y := 1 + v;
u := F(x);	u := F(x);	u := *;
v := F(y);	v := F(y);	v := *;
}	}	}
assert(x = y)	assert(x = y)	assert(x = y)
$\Sigma = \Sigma_{LA} \cup \Sigma_{UFS}$	$\Sigma = \Sigma_{UFS}$	$\Sigma = \Sigma_{LA}$
$Th = Th_{LA} + Th_{UFS}$	$Th = Th_{UFS}$	$Th = Th_{LA}$

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 and k variables

$x_i := 0$, 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.

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 ψ 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)?

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 $\mathit{Unif}(e_1 = e_2)$ *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

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

Why is Unification Sound in Backward Analysis?

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

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

Why backward analysis 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 above 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

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 $\sqcup$, $\sqcap$, $\sqsubseteq$, fixpoint, and SP in the combination

Logical Product

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

elements	:	conjunction ϕ of atomic formulas in $Th_1 \cup Th_2$
$E \sqsubseteq E'$	:	$E \Rightarrow_{Th_1 \cup Th_2} E'$ and $\mathit{AlienTerms}(E') \subseteq \mathit{Terms}(E)$

$\mathit{AlienTerms}(E)$	=	subterms in E that belong to different theory
$\mathit{Terms}(E)$	=	all subterms in E, plus all terms equivalent to these subterms (in $Th_1 \cup Th_2 \cup E$)

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

$$
\begin{array}{rcl}
x = F(a+1) \wedge y = a & \Rightarrow & x = F(y+1) \\
x = F(b+1) \wedge y = b & \Rightarrow & x = F(y+1) \\
x = F(\underline{a+1}) \wedge y = a & \Rightarrow & y+1 = \underline{a+1} \\
x = F(\underline{b+1}) \wedge y = b & \Rightarrow & y+1 = \underline{b+1}
\end{array}
$$

Logical Product

- Includes only those atomic facts in the least upper bound of E and E' whose alien terms occur semantically in both elements E and E'
- Is more powerful than direct product and reduced product
- Allows us to combine the abstract interpreters modularly in some cases

We will discuss how to combine the abstract interpretation operations

Combining the Preorder Test

Required for testing convergence of fixpoint

$E \sqsubseteq E'$ iff

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

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 $E \wedge \neg E'$ is unsatisfiable

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

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. $4y_3 \leq f(2y_2 - y_1) \mapsto \{4y_3 \leq a_2, a_1 = 2y_2 - y_1\}$ 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 preorder test: NO Procedure

We can modify NO procedure to also deduce facts

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

$$\begin{array}{lll} a_1 = 2y_2 - y_1 & & a_2 = f(a_1) \\ y_1 \le 4y_3 \le a_2 & & y_1 = f(y_1), y_2 = f(f(y_1)) \\ y_1 = y_2 & \leftarrow & \\ & \rightarrow & y_1 = a_1 \\ y_1 = a_2 & \leftarrow & \end{array}$$

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

Works for convex, stably-infinite, disjoint theories

Combining Join Operator

Given procedures:

$$
\begin{array}{lcl}
Join_{L_1}(E_l, E_r) & : & \text{Computes } E_l \sqcup E_r \text{ in lattice } L_1 \\
Join_{L_2}(E_l, E_r) & : & \text{Computes } E_l \sqcup E_r \text{ in lattice } L_2
\end{array}
$$

We wish to compute $E_l \sqcup E_r$ in the logical product $L_1 * L_2$

Example.

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

Combining Join Operators

	$\boxed{z = a - 1, y = f(a)}$		$\boxed{z = b - 1, y = f(b)}$	
Purify+NOSat	$z = a - 1$	$y = f(a)$	$z = b - 1$	$y = f(b)$
LR-Exchange	$a = \langle a, b\rangle$	$a = \langle a, b\rangle$	$b = \langle a, b\rangle$	$b = \langle a, b\rangle$
Base Joins	$\boxed{Join_{LA}}$		$\boxed{Join_{UF}}$	
	$\langle a, b\rangle = 1 + z$		$y = f(\langle a, b\rangle)$	
Quant Elim	$\boxed{QE_{UF*LA}}$			
Return	$\boxed{y = f(1 + z)}$			

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.

- $E' \sqsubseteq_L E$
- $Vars(E) \cap V = \emptyset$

Examples:

- $QE_{LA}(\{x < a, a < y\}, \{a\}) = \{x \leq y\}$
- $QE_{UF}(\{x = f(a), y = f(f(a))\}, \{a\}) = \{y = f(x)\}$
- $QE_{LA*UF}(\{a < b < y, z = c + 1, a = ffb, c = fb\}, \{a, b, c\}) = \{f(z - 1) \leq y\}$

How to construct QE_{LA*UF} using QE_{LA} and QE_{UF}?

Combining QE Operators

Problem	$\boxed{a < b < y, z = c+1, a = ffb, c = fb}$		$\boxed{\{a, b, c\}}$
Purify+NOSat	$\boxed{a < b < y, z = c+1}$		$\boxed{a = ffb, c = fb}$
QSat		$\longrightarrow$	$c \mapsto z - 1$
QSat	$a \mapsto fc$	$\longleftarrow$	
Base QEs	$\boxed{QE_{LA}}$		$\boxed{QE_{UF}}$
	$\boxed{a \leq y, z = c+1}$		$\boxed{a = fc}$
Substitute		$c \mapsto z - 1, a \mapsto fc$	
Return		$\boxed{f(z-1) \leq y}$	

Fixpoint Computation

Termination of analysis across loops required bounding height of lattice

$H_L(E) \doteq$ no. of elements in any chain above E in lattice L

$$H_{L_1 * L_2}(E) \quad \leq \quad H_{L_1}(E_1) + H_{L_2}(E_2) + |AlienTerms(E)|$$

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

Correctness and Complexity

- Algorithms $QE_{L_1 * L_2}$ and $Join_{L_1 * L_2}$ 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 $Join$ is worst-case quadratic in the complexity of these operations for individual lattices

Example of Incompleteness

$QE_P(\{odd(x'), x = x' - 1\}, \{x'\}) = \{even(x)\}$

$QE_S(\{pos(x'), x = x' - 1\}, \{x'\}) = \{\}$

$QE_{P*S}(\{pos(x'), odd(x'), x = x' - 1\}, \{x'\}) = \{pos(x), even(x)\}$

But our algorithm only outputs $even(x)$. 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 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

References

1. S. Gulwani and A. Tiwari, "*Combining abstract interpreters*". PLDI 2006.
2. S. Gulwani and A. Tiwari, "*Assertion checking in the combined abstraction of linear arithmetic and uninterpreted functions*". ESOP 2006.