Abstract Interpretation with Alien Expressions and Heap Structures

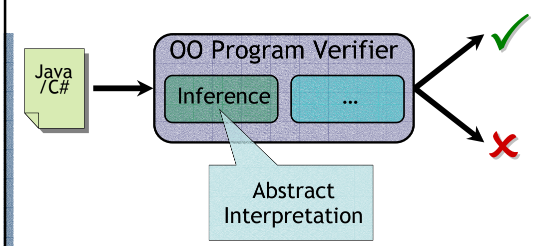
Bor-Yuh Evan Chang
University of California, Berkeley

K. Rustan M. Leino
Microsoft Research

January 18, 2005

VMCAI 2005
Paris, France

Verifying Object-Oriented Programs



Problem and Motivation

- Standard abstract interpretation infer properties following a domain specific-schema of relations among (program) variables

$0 \leq x \leq y$

z := 2 · y – 2 · x;

$0 \leq z$

  – e.g., can infer this with Polyhedra [CH78]

Problem and Motivation

- But …

alien expression to Polyhedra

$0 \leq$ **this.x** $\leq y$
z := 2 · y – 2 · **this.x**;
$0 \leq z$?

alien expression to Polyhedra

$0 \leq$ **length(x)** $\leq y$
z := 2 · y – 2 · **length(x)**;
$0 \leq z$?

$0 \leq$ **this.x** $\leq y$ $\wedge$ **o $\neq$ this**
o.x := 2 · y
z := 2 · y – 2 · **this.x**;
$0 \leq z$?

Goal

Given a *base abstract domain* that can represent certain kind of constraints on variables, use it to represent constraints on arbitrary *alien expressions* (e.g., fields of objects)

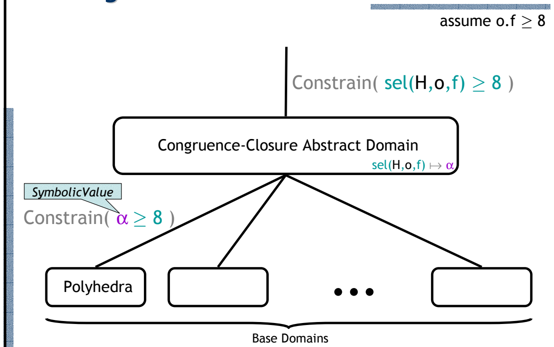
Outline

- Overview
- Handling Alien Expressions
- Handling Heap Updates
- Concluding Remarks

Overview of Contributions

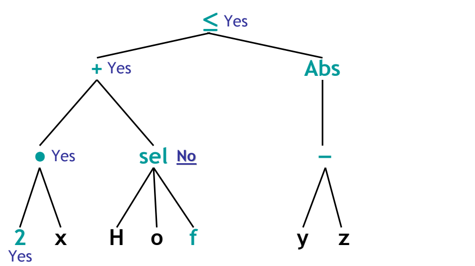
- To extend base domains to work with alien expressions
  - use a general abstract domain parameterized by base domains that hide alien expressions as fresh variables (*cf.* Nelson-Oppen)
  - congruence-closure abstract domain
- To deal with heap updates
  - track successive heaps as a separate base domain
  - heap succession abstract domain

Fooling the Base Domains



Understandable to the Base Domain

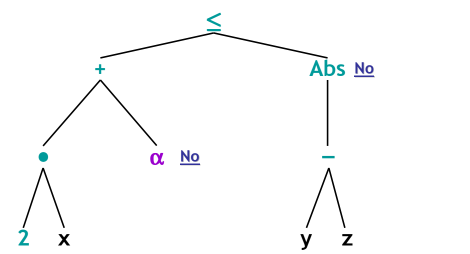
Understands : FunSymbol × Expr[] → bool



$2 \cdot x + sel(H,o,f) \leq Abs(y - z)$

Understandable to the Base Domain

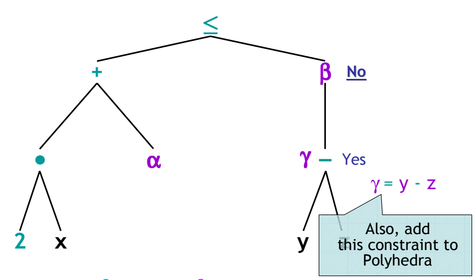
Understands : FunSymbol × Expr[] → bool



$2 \cdot x + \alpha \leq Abs(y - z)$

Understandable to the Base Domain

Understands : FunSymbol × Expr[] → bool



$2 \cdot x + \alpha \leq \beta$

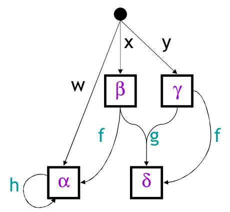
Congruence-Closure Domain

- Store mappings in an ***equivalence graph (e-graph)***
  - give the same symbolic value for equivalent expressions
- Tracks equalities of uninterpreted functions
  - an e-graph with abstract domain operations
  - symbolic values "name" equivalence classes of expressions
  - implements congruence closure

E-Graph

- $w = f(x) \land g(x,y) = f(y) \land w = h(w)$
- A set of mappings:

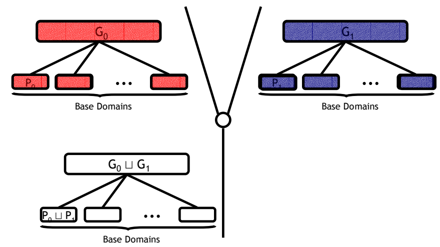
  $w \mapsto \alpha$
  $x \mapsto \beta$
  $f(\beta) \mapsto \alpha$
  $y \mapsto \gamma$
  $g(\beta,\gamma) \mapsto \delta$
  $f(\gamma) \mapsto \delta$
  $h(\alpha) \mapsto \alpha$



- Always congruence-closed

Join

- Roughly, join the e-graphs, then join the base domains



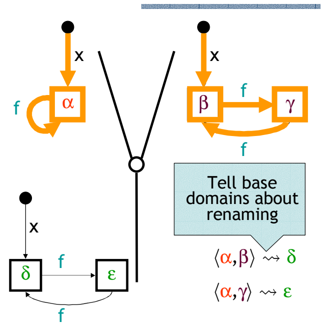
Join of E-Graphs

- Think of the lattice over conjunctions of equalities (including infinite ones)
- Let $G = \text{Join}(G_0, G_1)$

  $x \mapsto_G \langle\alpha',\beta'\rangle$
  if $x \mapsto_{G_0} \alpha'$ and $x \mapsto_{G_1} \beta$

  $f(\langle\alpha,\beta\rangle) \mapsto_G \langle\alpha',\beta'\rangle$
  if $f(\alpha) \mapsto_{G_0} \alpha'$ and
  $f(\beta) \mapsto_{G_1} \beta'$

- Rename distinct pairs to fresh symbolic values



Join of E-Graphs

- Complexity: $O(n \cdot m)$
- Complete? As precise as possible?
  - No, e-graphs do not form a lattice!

    $x = y \quad \sqcup \quad g(x) = g(y) \land x = f(x) \land y = f(y)$

    $= \quad \bigwedge_{i\,:\,i \geq 0} g(f^i(x)) = g(f^i(y))$
  - Only relatively complete

  [Gulwani et al. 2004]

Widen

- Widen the e-graphs, then widen the base domains
- Widen of e-graphs is a join of e-graphs that limits the number of new names introduced (see paper)

So Far We Have …

- Reasoning for uninterpreted functions
- Base domains that work with alien expressions transparently
- What we need for field reads
  - sel is alien to all base domains

Outline

- Overview
- Handling Alien Expressions
- Handling Heap Updates
- Concluding Remarks

Heap Updates

Java/C# `if (p.g == 8) { o.f = x; }`

Guarded Commands
assume H[p,g] == 8;
H := H′ *where*
sel(H′,o,f) = x *and*
$H' \equiv_{o,f} H$

Heap Updates

Guarded Commands
assume H[p,g] == 8;
H := H′ *where*
sel(H′,o,f) = x *and*
$H' \equiv_{o,f} H$

Abstract Interpreter
Constrain(sel(H,p,g) = 8)
Constrain(sel(H′,o,f) = x)
Constrain($H' \equiv_{o,f} H$)
Eliminate(H)
Rename(H′, H)

Tracked by a new base domain: *Heap Succession*

Heap Update Example

Heap Succession
$H' \equiv_{o,f} H$

E-Graph
sel(H,p,g) $\mapsto \alpha$
8 $\mapsto \alpha$
sel(H′,o,f) $\mapsto \beta$
x $\mapsto \beta$
H $\mapsto$ H   p $\mapsto$ p
H′ $\mapsto$ H′   g $\mapsto$ g
o $\mapsto$ o   f $\mapsto$ f

Constrain(sel(H,p,g) = 8)
Constrain(sel(H′,o,f) = x)
▷ Constrain($H' \equiv_{o,f} H$)
Eliminate(H)
Rename(H′, H)
ToPredicate()

Heap Update Example

Heap Succession
$H' \equiv_{o,f} H$

E-Graph
"Garbage values" remain
sel(H,p,g) $\mapsto \alpha$
8 $\mapsto \alpha$
sel(H′,o,f) $\mapsto \beta$
x $\mapsto \beta$
~~H $\mapsto$ H~~   p $\mapsto$ p
H′ $\mapsto$ H′   g $\mapsto$ g
o $\mapsto$ o   f $\mapsto$ f

Constrain(sel(H,p,g) = 8)
Constrain(sel(H′,o,f) = x)
Constrain($H' \equiv_{o,f} H$)
▷ Eliminate(H)
Rename(H′, H)
ToPredicate()

- Only removes mapping
- "Lazy quantifier elimination"

Heap Update Example

Heap Succession
$H' \equiv_{o,f} H$

E-Graph
sel(H,p,g) $\mapsto \alpha$
8 $\mapsto \alpha$
sel(H′,o,f) $\mapsto \beta$
x $\mapsto \beta$
~~H $\mapsto$ H~~   p $\mapsto$ p
H $\mapsto$ H′   g $\mapsto$ g
o $\mapsto$ o   f $\mapsto$ f

Constrain(sel(H,p,g) = 8)
Constrain(sel(H′,o,f) = x)
Constrain($H' \equiv_{o,f} H$)
Eliminate(H)
▷ Rename(H′, H)
ToPredicate()

Heap Update Example

Heap Succession

$H' \equiv_{o,f} H$

Can you give me an equivalent expression without H?

E-Graph

sel(H,p,g) $\mapsto \alpha$

8 $\mapsto \alpha$

sel(H′,o,f) $\mapsto \beta$

x $\mapsto \beta$

~~H $\mapsto$ H~~   p $\mapsto$ p

H $\mapsto$ H′   g $\mapsto$ g

o $\mapsto$ o   f $\mapsto$ f

Constrain(sel(H,p,g) = 8)
Constrain(sel(H′,o,f) = x)
Constrain($H' \equiv_{o,f}$ H)
Eliminate(H)
Rename(H′, H)
ToPredicate()

1. Do Eliminate (H)
   - EquivalentExpr : Queryable × Expr × Var → Expr option

Heap Update Example

Heap Succession

~~$H' \equiv_{o,f} H$~~

Yes, use H′

E-Graph

sel(H′,p,g) $\mapsto \alpha$

8 $\mapsto \alpha$

sel(H′,o,f) $\mapsto \beta$

x $\mapsto \beta$

~~H $\mapsto$ H~~   p $\mapsto$ p

H $\mapsto$ H′   g $\mapsto$ g

o $\mapsto$ o   f $\mapsto$ f

Constrain(sel(H,p,g) = 8)
Constrain(sel(H′,o,f) = x)
Constrain($H' \equiv_{o,f}$ H)
Eliminate(H)
Rename(H′, H)
ToPredicate()

1. Do Eliminate (H)
   - EquivalentExpr : Queryable × Expr × Var → Expr option
   - Eliminate(H) on Base
2. To ... nd
   Co ...
3. Conjoin Equalities

To query other abstract domains (*e.g.*, o ≠ p?)

Related Work

- Join for Uninterpreted Functions [Gulwani, Tiwari, Necula 2004]
  - same as our join for e-graphs
- Shape Analysis [many] and
  TVLA [Sagiv, Reps, Wilhelm, …]
  - they abstract heap nodes into summary nodes
  - they use special "instrumentation predicates" whereas we use "off-the-shelf" abstract domains
  - could use shape analysis as base domain?

Conclusion and Future Work

- Extended the power of abstract domains to work with alien expressions using the congruence-closure domain
- Added reasoning about heap updates with the heap succession domain
- Close to having "cooperating abstract interpreters"?
  - missing propagating back equalities inferred by base domains
- Implementation and experiments in progress

Thank you!

Questions? Comments?