Zoo Workshop

Research On Program Analysis System
Dept. of Computer Science
KAIST

□ An Ensemble

- abstract interpretation [CC77,CC92a,CC95b]
- conventional data flow analysis [KU76,KU77,Hec77,RP86]
- constraint-based analysis [Hei92,AH95]
- model checking [CGP99]

□ Use of Each Framework in Zoo

- specification variations

  - abstract interpretation

  - data flow analysis

  - constraint-based analysis

- query about analysis result

  - model checking: computation-tree-logic(CTL) formula over analysis results

□ Every Program Analysis

Given a program

- step 1: set-up equations
- step 2: solve the equations
  - solution = finite flow graph $\langle$abstract program states, flows$\rangle$
- step 3: make sense of the solution
  - checking some properties = model checking

□ Abstract Semantics

Skeleton for Data Flow Equations

Program:

$$
\begin{array}{rcll}
e & ::= & z \mid x & \text{integer/variable} \\
& \mid & e_1 \texttt{ + } e_2 & \text{primitive operation} \\
& \mid & x \texttt{ := } e & \text{assignment} \\
& \mid & e \texttt{ ; } e & \text{sequence} \\
& \mid & \texttt{if } e_1\ e_2\ e_3 & \text{choice}
\end{array}
$$

Abstract semantics:

$$\begin{array}{rcl} s & \in & \mathit{State} = \mathit{Var} \rightarrow \mathit{Sign} \\ E & \in & \mathit{Expr} \times \mathit{State} \rightarrow \mathit{Sign} \times \mathit{State} \end{array}$$

$$\begin{array}{rcl}
E(z, s) & = & (\widehat{z}, s) \\
E(x, s) & = & (s(x), s) \\
E(x\texttt{:=}e, s) & = & \mathit{let}\ (v_1, s_1) = E(e, s) \\
 & & \mathit{in}\ (v_1, s_1[v_1/x]) \\
E(e_1\texttt{;}e_2, s) & = & \mathit{let}\ (v_1, s_1) = E(e_1, s) \\
 & & \qquad (v_2, s_2) = E(e_2, s_1) \\
 & & \mathit{in}\ (v_2, s_2) \\
E(e_1\texttt{+}e_2, s) & = & \mathit{let}\ (v_1, s_1) = E(e_1, s) \\
 & & \qquad (v_2, s_2) = E(e_2, s_1) \\
 & & \mathit{in}\ (\mathit{add}(v_1, v_2), s_2) \\
E(\texttt{if}\ e_1\ e_2\ e_3, s) & = & \mathit{let}\ (v_1, s_1) = E(e_1, s) \\
 & & \qquad (v_2, s_2) = E(e_2, s_1) \\
 & & \qquad (v_3, s_3) = E(e_3, s_1) \\
 & & \mathit{in}\ (v_2, s_2) \sqcup (v_3, s_3)
\end{array}$$

$\square$ Side: Correctness

Analysis designer's job, not Zoo's:

$$\mathsf{fix}\mathcal{F} \;\underset{\alpha}{\overset{\gamma}{\leftrightarrows}}\; \mathsf{fix}F$$

where

$$\mathsf{fix}F = [\![E]\!] \quad \text{and} \quad \mathsf{fix}\mathcal{F} = [\![\mathcal{E}]\!]$$

of

$$\begin{array}{lll} F & \in & (\mathit{Expr} \times \mathit{State} \rightarrow \mathit{Sign} \times \mathit{State}) \rightarrow (\mathit{Expr} \times \mathit{State} \rightarrow \mathit{Sign} \times \mathit{State}) \\ \mathcal{F} & \in & (\mathit{Expr} \times \mathcal{S}\mathit{tate} \rightarrow \mathcal{I}\mathit{nt} \times \mathcal{S}\mathit{tate}) \rightarrow (\mathit{Expr} \times \mathcal{S}\mathit{tate} \rightarrow \mathcal{I}\mathit{nt} \times \mathcal{S}\mathit{tate}) \end{array}$$

□ Setting-up Equations

$$\overbrace{\underbrace{\texttt{x := 1;}}_{1}\ \underbrace{\texttt{y := x+1}}_{2}}^{0}$$

$$X_i^{\downarrow} \in \mathit{State} \qquad X_i^{\uparrow} \in \mathit{Sign} \times \mathit{State}$$

$$\begin{array}{lcl@{\qquad}lcl}
X_0^{\downarrow} & = & \perp & X_0^{\uparrow} & = & X_2^{\uparrow} \\
X_1^{\downarrow} & = & X_0^{\downarrow} & X_1^{\uparrow} & = & (X_{1a}^{\uparrow}.1, \quad X_{1a}^{\uparrow}.2[X_{1a}^{\uparrow}.1/x]) \\
X_2^{\downarrow} & = & X_1^{\uparrow}.2 & X_2^{\uparrow} & = & (X_{2a}^{\uparrow}.1, \quad X_{2a}^{\uparrow}.2[X_{2a}^{\uparrow}.1/y]) \\
X_{2a}^{\downarrow} & = & X_2^{\downarrow} & X_{2a}^{\uparrow} & = & (\mathit{add}(X_2^{\downarrow}.2(x), 1), \quad X_2^{\downarrow}.2)
\end{array}$$

□ Solution: Fixpoint and Flow Graph

Fixpoint: equation solution $(X_i^\downarrow, X_i^\uparrow)$.

Flow graph:

$$
\begin{array}{lcllcl}
 & & & X_0^\uparrow & \leftarrow & X_2^\uparrow \\
X_1^\downarrow & \leftarrow & X_0^\downarrow & X_1^\uparrow & \leftarrow & X_{1a}^\uparrow \\
X_2^\downarrow & \leftarrow & X_1^\uparrow.2 & X_2^\uparrow & \leftarrow & X_{2a}^\uparrow \\
X_{2a}^\downarrow & \leftarrow & X_2^\downarrow & X_{2a}^\uparrow & \leftarrow & X_2^\downarrow
\end{array}
$$

□ Query on Solution about Program Properties

Model checking

- model = the flow graph
- formula = CTL formula
  - modality $= \{\mathtt{A}, \mathtt{E}\} \times \{\mathtt{G}, \mathtt{F}, \mathtt{X}, \mathtt{U}\}$
  - body = first-order predicate over $X_i^{\downarrow}$ and $X_i^{\uparrow}$

Query examples:

$$X_i^{\uparrow} \in Sign \times State$$

- Does variable `v` remain positive?

$$\mathtt{AG}(X^{\uparrow}.2(\mathtt{v}) = \oplus)$$

- Can variable `v` be positive?

$$\mathtt{EF}(X^{\uparrow}.2(\mathtt{v}) = \oplus)$$

- Does variable `v` remain positive until `w` is negative?

$$\mathtt{AU}(X^{\uparrow}.2(\mathtt{v}) = \oplus,\;\; X^{\uparrow}.2(\mathtt{w}) = \ominus)$$

We can also query at a particular program point:

- annotate program text with CTL formula
  - "From here, does variable `v` remain positive?"

```
v := x+y;
## AG(X↑.2(v)=⊕)
if v > 0 then v := v-2 else v := v+1;
...
```

□ Higher-order Case

Program:

$$
\begin{array}{rcll}
e & ::= & x & \text{variable} \\
 & | & \lambda x.e & \text{abstraction} \\
 & | & e_1\ e_2 & \text{application}
\end{array}
$$

Abstract semantics:

$$
\begin{array}{rcl}
s & \in & \mathit{State} = \mathit{Var} \rightarrow 2^{\mathit{Expr}} \\
E & \in & \mathit{Expr} \times \mathit{State} \rightarrow 2^{\mathit{Expr}}
\end{array}
$$

$$
\begin{array}{rcl}
E(x, s) & = & s(x) \\
E(\lambda x.e, s) & = & \{\lambda x.e\} \\
E(e_1\ e_2, s) & = & \mathit{let}\ \{\lambda x_i.e_i'\} = E(e_1, s) \\
 & & \qquad v = E(e_2, s) \\
 & & \mathit{in}\ \sqcup_i\ E(e_i', s \sqcup \{x_i \mapsto v\})
\end{array}
$$

□ Setting-up Equations

$$\overbrace{\underbrace{(\lambda x.\,\overbrace{x\ 1}^{3})}_{1}\ \underbrace{(\lambda y.y)}_{2}}^{0}$$

$$X_i^{\downarrow} \in \mathit{State} \qquad X_i^{\uparrow} \in 2^{\mathit{Expr}}$$

$$\begin{array}{rclrcl}
X_0^{\downarrow} & = & \bot & X_0^{\uparrow} & = & \sqcup_{\lambda x_i.e_i \in X_1^{\uparrow}} X_{e_i}^{\uparrow} \\
X_1^{\downarrow} & = & X_0^{\downarrow} & X_1^{\uparrow} & = & (\lambda x.x\ 1) \\
X_2^{\downarrow} & = & X_0^{\downarrow} & X_2^{\uparrow} & = & (\lambda y.y)
\end{array}$$

$$X_{e_i}^{\downarrow} = X_0^{\downarrow} \sqcup \{x_i \mapsto X_2^{\uparrow}\} \qquad \text{for each } \lambda x_i.e_i \in X_1^{\uparrow}$$

□ Solution: Fixpoint and Flow Graph

As before, except that equations/flow edges are generated during fixpoint computation:

generated equations
while solving

$$
\begin{aligned}
X_0^{\uparrow} &= X_3^{\uparrow} \sqcup X_{2a}^{\uparrow} \\
X_3^{\downarrow} &= X_0^{\downarrow} \sqcup \{x \mapsto X_2^{\uparrow}\} \\
X_{2a}^{\downarrow} &= X_0^{\downarrow} \sqcup \{x \mapsto X_2^{\uparrow}\}
\end{aligned}
$$

□ Constraint-based Analysis

High-level skeleton for data flow equations

- setting-up constraints
- propagating constraints (constraint closure)
- solution: either
  - the set of "atomic" constraints, or
  - a model of the "atomic" constraints

□ Naive Style Example

Program:

$$\begin{array}{rcll} e & ::= & x & \text{variable} \\ & | & \lambda x.e & \text{abstraction} \\ & | & e_1\ e_2 & \text{application} \end{array}$$

Constraint set:

$$X \supset se$$

$$\begin{array}{rcll} se & ::= & \mathsf{lam}(x,e) & \quad atomic \\ & | & \mathsf{app}(X,X) & \\ & | & X & \end{array}$$

$$X \quad \text{at each expr or var} \quad \in 2^{Expr}$$

Setting-up constraints:

$$\frac{}{x \vdash \{\}} \qquad\qquad \frac{e' \vdash C}{\lambda x.e' \vdash \{X_e \supset \mathsf{lam}(x, e')\} \cup C}$$

$$\frac{e_1 \vdash C_1 \quad e_2 \vdash C_2}{e_1\ e_2 \vdash \{X_e \supset \mathsf{app}(X_{e_1}, X_{e_2})\} \cup C_1 \cup C_2}$$

□ Solution: Fixpoint and Flow Graph

By the constraint propagation(closure) rules:

$$\frac{X_a \supset \mathsf{app}(X_b, X_c),\ X_b \supset \mathsf{lam}(x, e)}{X_a \supset X_e,\ X_x \supset X_c}$$

$$\frac{X_a \supset X_b,\ X_b \supset \mathit{atomic}}{X_a \supset \mathit{atomic}}$$

- Solution: atomic constraints of $X_e \supset \mathsf{lam}(x, e)$ from the closure
- Flow graph: $X_e \leftarrow X_{e'}$ iff $X_e \supset X_{e'}$

□ Mixed Style: Constraint Rules + Equations

Atomic constraints with their interpretations = data flow equations

Program:

$$
\begin{array}{rcll}
e & ::= & z & \text{integer} \\
  & | & e + e & \text{addition} \\
  & | & x & \text{variable} \\
  & | & \lambda x.e & \text{abstraction} \\
  & | & e_1\ e_2 & \text{application}
\end{array}
$$

Constraint set:

$$X \supset se$$

$$\begin{array}{lcll} se & ::= & \mathsf{lam}(x, e') & atomic \\ & | & \mathsf{app}(X, X) & \\ & | & \mathsf{add}(X, X) & atomic \\ & | & \widehat{z} & atomic \\ & | & X & \end{array}$$

X for each expr or var $\in 2^{Expr} + 2^{Sign}$

Setting-up constraints:

$$\frac{}{z \vdash \{X_e \supset \widehat{z}\}} \qquad \frac{}{x \ \vdash \ \{\}}$$

$$\frac{e' \vdash C}{\lambda x.e' \ \vdash \ \{X_e \supset \mathsf{lam}(x, e')\} \cup C}$$

$$\frac{e_1 \vdash C_1 \quad e_2 \vdash C_2}{e_1 \ e_2 \ \vdash \ \{X_e \supset \mathsf{app}(X_{e_1}, X_{e_2})\} \cup C_1 \cup C_2}$$

$$\frac{e_1 \vdash C_1 \quad e_2 \vdash C_2}{e_1 \ + \ e_2 \ \vdash \ \{X_e \supset \mathsf{add}(X_{e_1}, X_{e_2})\} \cup C_1 \cup C_2}$$

□ Solution: 2×Fixpoint and Flow Graph

Constraint propagation:

$$\frac{X_a \supset \mathsf{app}(X_b, X_c),\ X_b \supset \mathsf{lam}(x, e)}{X_a \supset X_e,\ X_x \supset X_c}$$

$$\frac{X_a \supset X_b,\ X_b \supset atomic}{X_a \supset atomic}$$

As before, except that

- the atomic constraints of the closure as data flow equations to solve: (e.g.)

Atomic constraints

$$\begin{array}{ll} X_1 \supset \mathsf{add}(X_2, X_2) & X_1 \supset \mathsf{add}(X_1, X_2) \\ X_2 \supset \hat{z_1} & X_2 \supset \mathsf{add}(X_2, X_1) \\ X_3 \supset \mathsf{lam}(x, e) & X_3 \supset \mathsf{lam}(y, e') \end{array}$$

are

$$\begin{array}{l} X_1 = \mathsf{add}(X_2, X_2) \ \sqcup \ \mathsf{add}(X_1, X_2) \\ X_2 = \{\hat{z_1}\} \ \sqcup \ \mathsf{add}(X_2, X_1) \\ X_3 = \mathsf{lam}(x, e) \ \sqcup \ \mathsf{lam}(y, e') \end{array}$$

where

$$\begin{array}{lcl} X_i & \in & 2^{Expr} + 2^{Sign} \\ \mathsf{add}(X, X') & = & \{\text{pair-wise addition over } Sign\} \\ \mathsf{lam}(x, e) & = & \{\lambda x.e\} \end{array}$$

□ The Specification Language Rabbit

A language for expressing program analysis

- Sound typing: typed Rabbit spec $\Rightarrow$ typeful nML programs
  - monomorphic typing with overloading (`top`, `bottom`, `+`, `*`)
  - type-inference system
  - primitive types = int, bool, user-defined sets/lattices
  - compound types = tuple, sum, collection, function

- Module system: named/parameterized analyses
  - analysis module with/without a parameter analysis
  - to be compiled into nML functors and structures
- User-defined sets and lattices, with widening/narrowing
  - $\{1 ... 10\}$, $\{a, b, c\}$, $2^S, S_1 \times S_2$, $S_1 + S_2$, $S_1 \to S_2$, set of constraints
  - $S_\perp$, 2^S, $L_1 \times L_2$, $L_1 + L_2$, $S \to L$, $L_1 \to L_2$, set with explicit join
- Semantic functions: first-order

  - equations, operations
  - constraints

- Constraint closure rules

- Guards: first-order predicates
  - in patterns, set comprehensions, CTL formula, closure rules

- Inter-operation with nML exprs and patterns
  - via int, bool, user-declared structures for sets/lattices

```
analysis Eq =
 ana
   lattice A = power {a,b,c,d}

   eqn x1 = x2 + x3 * {a,b}
   and x2 = {b,c} * x3
   and x3 = x1 + x2

 end
```

□ Rabbit Example

```
analysis TinyCfa =
 ana
    set Var = /Exp.var/
    set Lam = /Exp.expr/
    lattice Val = power Lam
    lattice State = Var -> Val

    widen Val with {/Lam(x,Lam _)/ ...} => top

    eqn E(/x/,s) = s(x)
      | E(/Lam(x,e)/, s) = {/Lam(x,e)/}
      | E(/App(e1,e2)/, s) = let val lams = E(/e1/, s)
                                 val v = E(/e2/, s)
                             in
                              +{ E(e,s+bot[/x/=>v]) | /Lam(x,e)/ from lams }
                             end
 end
```

□ Rabbit Example

```
analysis Sba =
 ana
   set Var = /Ast.id/
   set Exp = /Ast.exp/
   set Val = power Exp
             constraint
               var = {X} index Var + Exp
               rhs = var
                   | app(var, var)
                   | lam(Var, Exp) : atomic
```

```
  eqn Col /Ast.Var(x)/ = {}
    | Col /Ast.Lam(x,body) as e/ = { X@/e/ <- lam(/x/,/body/) }
                                   + Col /body/
    | Col /Ast.App(e,e') as e/ = { X@/e/ <- app(X@/e/, X@/e'/) }
                                   + Col /e/ + Col /e'/

  ccr   X@a <- app(X@b, X@c), X@b <- lam(/x/, /body/)
        ---------------------------------------
        X@a <- X@/body/, X@/x/ <- X@c
end
```

□ Rabbit Example

```
signature CFA = sig
                  lattice Env
                  lattice Fns = power /Ast.exp/
                  eqn Lam: /Ast.exp/:index * Env -> Fns
                end

analysis ExnAnal(Cfa: CFA) =
 ana
   set Exp = /Ast.exp/        set Var = /Ast.var/       set Exn = /Ast.exn/
   set UncaughtExns = power Exn
                      constraint
                      var = {X, P} index Var + Exp
                      rhs = var
                          | app_x(/Ast.exp/, var)  |  app_p(/Ast.exp/, var)
                          | exn(Exn)                               : atomic
                          | minus(var, /Ast.exp/, power Exn) : atomic
                          | cap(var, /Ast.exp/, Exn)             : atomic
```

```
eqn Col /Ast.Var(x)/ = {}
  | Col /Ast.Const/ = {}
  | Col /Ast.Lam(x,e)/ = Col /e/
  | Col /e as Ast.Fix(f,x,e',e'')/ = Col /e'/ + Col /e''/
                                   + { X@/e/ <- X@/e''/, P@/e/ <- P@/e''/ }
  | Col /e as Ast.Con(e',k)/ = Col /e'/
                             + { X@/e/ <- exn(/k/), P@/e/ <- P@/e'/ }
  | Col /e as Ast.Decon(e')/ = Col /e'/
                             + { X@/e/ <- X@/e'/, P@/e/ <- P@/e'/ }
  | Col /e as Ast.Exn(k,e')/ = Col /e'/
                             + { X@/e/ <- exn /k/, X@/e/ <- X@/e'/ }
  | Col /e as Ast.App(e',e'')/ = Col /e'/ + Col /e''/
                             + { X@/e/ <- app_x(/e'/, X@/e''/),
                                 P@/e/ <- app_p(/e'/, X@/e''/),
                                 P@/e/ <- P@/e'/, P@/e/ <- P@/e''/ }
  | Col /e as Ast.Case(e',k,e'',e''')/ =
             Col /e'/ + Col /e''/ + Col /e'''/
             + { X@/e/ <- X@/e''/, X@/e/ <- X@/e'''/ }
             + { P@/e/ <- P@/e'/, P@/e/ <- P@/e''/, P@/e/ <- P@/e'''/ }
```

```
| Col /e as Ast.Raise(e')/ = Col /e'/ + { P@e <- X@/e'/ }
| Col /e as Ast.Mraise(e',Ks)/ =
          let
            val K = /Ast.list2set Ks/
          in
            Col /e'/ + { P@e <- minus(X@/e'/,/e'/, K) }
          end
| Col /e as Ast.Praise(e', k)/ =
          Col /e'/ + { P@/e/ <- cap(X@/e'/,/e'/,/k/) }
| Col /e as Ast.Handle(e', f as Ast.Lam(x,e''))/ =
          Col /e'/ + Col /e''/
          + { X@/e/ <- X@/e'/, X@/e/ <- app_x(/f/, P@/e'/) }
          + { X@/x/ <- P@/e'/, P@/e/ <- app_p(/f/, P@/e'/) }
```

```
(* constraint closure rules *)

ccr
     X@a <- app_x(/e/,X@b), /Ast.Lam(x,e')/ in post Cfa.Lam@/e/
    -----------------------------------------
          X@a <- X@/e'/, X@/x/ <- X@b
 |
     X@a <- app_x(/e/,P@b), /Ast.Lam(x,e')/ in post Cfa.Lam@/e/
    -----------------------------------------
         X@a <- X@/e'/, X@/x/ <- P@b
 |
     P@a <- app_p(/e/,X@b), /Ast.Lam(x,e')/ in post Cfa.Lam@/e/
    -----------------------------------------
         P@a <- P@/e'/, X@/x/ <- X@b
 |
     P@a <- app_p(/e/,P@b), /Ast.Lam(x,e')/ in post Cfa.Lam@/e/
    -----------------------------------------
         P@a <- P@/e'/, X@/x/ <- P@b
```

```
  (* constraint image definition *)

  cim exn(k) = {k}
    | minus(X,/e/,K) = if /Ast.exncarryexn(e)/ then X
                       else { x | x from X, not (x in K) }
    | cap(X,/e/,k)   = if /Ast.exncarryexn(e)/ then X
                       else { x | x from X, x = k }
end
```