Program Analysis Techniques:
System Zoo's Perspective

Kwangkeun Yi

Programming Research Laboratory
`ropas.snu.ac.kr`
SNU/KAIST

8/18/2003 @ LiComR Workshop, SonggwangSa Temple

□ Open Problem

automatic checking of bugs in softwares

□ 50-year Achievements (1/2)

1st generation: *syntax analysis*

- lexical analysis & parsing: `1+*^^*`
- checking in $\sim 10^4$ lines/sec
- context-free-grammar languages

□ 50-year Achievements (2/2)

2nd generation: *type checking/inference*

- simple typing, polymorphic typing, sub-typing: `1+''a''`
- inferencing in $\sim 10^3$ lines/sec
- HOT(higher-order & typed) languages (v.s. C, C++)

□ Need 3rd Gen. Debugging Technolgy

- correct programs in both syntax and type
  can still be incorrect.

- 1+2: correct in syntax and type, but does not compute 12 (our expectation)

□ Not Yet in 3rd Generation

- barely effective the-status-quo: testing, run-chase, code review, field manual, etc.

- not automatic, losing performance
  - AT&T: productivity = 10 lines/month (1995)
  - ETRI: 1-character bug/2 months (2000)
  - On-line game .com's: 24-hr monitoring under junk food

□ Badly Need 3rd Gen. Technology

impossible/difficult for manual debbugging

- complicated$^\infty$, large$^\infty$ softwares

- cost: big, low product quality
  - recall $k\times$million cars/zipels/phones?
  - Sony mobile phone: recall 420,000 units, 120 million dollars, 2001
  - Ariane rocket: 500 million dollars, 2 billion dollars, 1996

□ Position of Program Analysis

- 1st gen.(1970s): *syntax analysis*
- 2nd gen.(1990s): *type checking/inference*
- 3rd gen.(2000s): *program analysis*

□ Program Analysis

is statically understanding program behaviors

□ Facts about Program Analysis

- in principle: it's impossible
- in practice: it's impressive
- wisdom: *sound approximation, goal-specific accuracy-cost tradeoff, make use of statistics in programs*

□ Impressive Examples

not toys

- check for deadlock [CT95]
- check for overflow [Gu97]
- check for un-handled exceptions [YiRy97]
- check for resource requirements [Ba01]

- check for out-of-range buffer indices [CT03]

- transform memory allocation behavior [LeYaYi03]

- and many more

□ Program Analysis

a technology for static, automatic, and safe estimation of program's run-time behaviors

- "static" : before execution
- "automatic" : program analyzes programs
- "safe" : result must cover the reality
- "estimation" : cannot be exact in principle

"static analysis", "abstract interpretation", "data flow analysis", "model checking", "type system", ("program proof")

□ Obvious: Rising Industry Interest

- s/w companies experienced big failure
- they will ask/look for program analysis
- need be ready for the opportunity
- other apps too: s/w understanding, s/w optimization

□ Talk Outline

- program analysis frameworks and their roles
- one style: interpreter-based analysis
- another style: constraint-based analysis
- a mixed style
- program analyzer generator Zoo

□ Program Analysis Frameworks

- abstract interpretation [CC77,CC92a,CC95b]
- conventional data flow analysis [KU76,KU77,He77,RP86]
- constraint-based analysis [He92,AH95]
- model checking [CGP99]

□ Use of Each Framework

- design/specification frameworks

  - *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 = graph $\langle$abstract program states, flows$\rangle$
- step 3: make sense of the solution
  - checking some properties = *model checking*

□ One Style: Abstract Interpretation

Skeleton for Semantic(Data Flow) Equations

Program to analyze:

e	$::=$	$z \mid x$	integer/variable
	$\mid$	e_1 `+` e_2	primitive operation
	$\mid$	x `:=` e	assignment
	$\mid$	e `;` e	sequence
	$\mid$	`if` $e_1\ e_2\ e_3$	choice

Abstract semantics:

$$s \in \mathit{State} = \mathit{Var} \to \mathit{Sign}$$
$$\mathsf{E} \in \mathit{Expr} \times \mathit{State} \to \mathit{Sign} \times \mathit{State}$$

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

$$[\![\mathsf{E}]\!] \stackrel{\triangle}{=} \mathrm{fix} F$$

where $F : (\mathit{Expr} \times \mathit{State} \rightarrow \mathit{Sign} \times \mathit{State}) \rightarrow (\mathit{Expr} \times \mathit{State} \rightarrow \mathit{Sign} \times \mathit{State})$

where $F(\mathsf{E}) \stackrel{\triangle}{=} \lambda(e, s).\mathrm{case}\ e\ \mathrm{of}$

$$\begin{array}{l} z : ((\widehat{z}), s) \\ x : (s(x), s) \\ x := e : \cdots \mathsf{E}(e, s) \cdots \\ e_1 ; e_2 : \cdots \mathsf{E}(e_1, s) \cdots \mathsf{E}(e_2, s_1) \cdots \\ . \\ . \\ . \end{array}$$

□ Correctness

Analysis designer has to prove:

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

where

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

of

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

□ Analyzer Sets-up Equations from Programs

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

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

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

□ Analyzer Solves the Equations

$$\begin{pmatrix} X_1^{\downarrow} \\ \vdots \\ X_n^{\downarrow} \\ X_1^{\uparrow} \\ \vdots \\ X_n^{\uparrow} \end{pmatrix} = F \begin{pmatrix} X_1^{\downarrow} \\ \vdots \\ X_n^{\downarrow} \\ X_1^{\uparrow} \\ \vdots \\ X_n^{\uparrow} \end{pmatrix}$$

Solving

- $\bot, \quad F\bot, \quad F^2\bot, \cdots$
- $\bot, \quad \bot \oplus F\bot, \quad \bot \oplus F\bot \oplus F^2\bot, \cdots$

□ A Solution = (Fixpoint, Flow Graph)

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

Flow graph:

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

□ 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?

$$\mathtt{AG}(\mathtt{v} = \oplus)$$

- Can variable `v` be positive?

$$\mathtt{EF}(\mathtt{v} = \oplus)$$

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

$$\mathtt{AU}(\mathtt{v} = \oplus,\ \ \mathtt{w} = \ominus)$$

May query at a particular program point:

- annotate program text with CTL formula

– "From here, does variable `v` remain positive?"

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

□ Higher-order Case: Analyzing Java or ML Programs

Program:

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

Abstract semantics:

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

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

□ Analyzer Sets-up Equations from Programs

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

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

$$X_0^{\downarrow} = \bot \qquad X_0^{\uparrow} = \sqcup_{\lambda x_i.e_i \in X_1^{\uparrow}} X_{e_i}^{\uparrow}$$

$$X_1^{\downarrow} = X_0^{\downarrow} \qquad X_1^{\uparrow} = (\lambda x.x\ 1)$$

$$X_2^{\downarrow} = X_0^{\downarrow} \qquad X_2^{\uparrow} = (\lambda y.y)$$

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

□ Solution: Fixpoint and Flow Graph

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

generated equations
while solving

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

□ Another Style: Constraint-based Analysis

A high-level skeleton for data flow equations

- setting-up constraints
- propagating constraints (constraint closure)
- solution: either
  - the set of “atomic” constraints, or
  - solution/model of the “atomic” constraints

□ Naive Style Example

Program:

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

Constraint set:

$$X \supset se$$

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

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

Setting-up constraints:

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

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

□ Solution: Fixpoint and Flow Graph

By the constraint propagation(closure) rules:

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

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

- 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:

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

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

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

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

□ Solution: Fixpoint of 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

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

□ System Zoo (ropas.snu.ac.kr/zoo)

program analyzer generator

- to transfer technology to the industry (int'l/domestic)
- as "realistic/routine" as `lex` and `yacc`
- work in s l o w progress

□ Inputs In Rabbit

Rabbit: a language for writing inputs to Zoo

- how-to-set-up equations in Rabbit: *abstract interpreters*, *data flow equations*, *constraints*

- what-to-query in Rabbit: CTL formula

□ Rabbit

- Type-inference: monomorphic typing, overloading, castings
  - primitive types $\ni$ user-defined sets/lattices
  - compound types $\ni$ tuple, sum, collection, function

- Module system
  - analysis module with/without a parameter analysis

- User-defined sets and lattices

– $\{1...10\}$, $\{a, b, c\}$, $2^S, S_1 \times S_2$, $S_1 + S_2$, $S_1 \rightarrow S_2$, constraint set

– $S_\perp$, 2^S, $L_1 \times L_2$, $L_1 + L_2$, $S \rightarrow L$, $L_1 \rightarrow L_2$, set with an order

- First-order functions

□ 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,_)/ ...} => 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

```
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

   (* constraint closure rule *)
```

```
  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

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

end
```

□ Issue I: Not a Blind Zoo

Zoo generates analyzers only when

- Rabbit exprs are monotonic or extensive: to guarantee termination of generated analyzers
- Rabbit exprs are typeful: well-formedness, efficiency
- Rabbit domains are lattices
- CTL formula are meaningful

□ Monotonicity and Extensionality Check [MuYi'02,YiEo'02]

Static check of F

- so that $\bot, F\bot, F^2\bot, \cdots$ terminates
- monotonicity: $\forall X \sqsubseteq Y. F\ X \sqsubseteq F\ Y$
- extensionality: $\forall X. X \sqsubseteq F\ X$

□ Issue II: Clever Fixpoint Algorithms [EoYi'02,Ahn'03]

Some redundancies in:

$$\bot,\ F\bot,\ F^2\bot, \cdots$$

Differential algorithm with $F' = \partial F / \partial X$:

$$\sqcup\{\bot,\ \ F' \triangle_0,\ \ F' \triangle_1, \cdots\}$$

□ Summing Up

- program analysis has a real motivation:
- program analysis area is rich and reaching the peak.
- program anlaysis area needs talents in both practice and theory.
- high time for a realistic program analyzer generator/library: e.g. Zoo

Thank you