System Zoo

(work-in-progress)

Kwangkeun Yi

Research On Program Analysis System
National Creative Research Initiative Center
Dept. of Computer Science
KAIST

11/11/2002@SNU

□ System Zoo

a software tool to make softwares safe

□ A Shame

unsafe softwares

□ Unsafe Softwares

- bugs : everywhere

- cost: big

  - recall $k\times$million cars/zipels/phones?

  - Ariane rocket: 500 million dollars, 2 billion dollars

- mass anxiety $\Rightarrow$ new legislations $\Rightarrow$ insurances $\Rightarrow$ high cost

□ Technology for Safe Softwares

very primitive the-status-quo

- ad-hoc/cowboy approaches:

  testing, debugging, code review, simulations, testing, field manual, etc.

- performance:

  - AT&T: productivity = 10 lines/month (1995)

  - ETRI: 1-character bug/2 months (2000)

□ Badly Need Better Technology

difficult/impossible for manual debugging

- complicated$^{\infty}$, large$^{\infty}$ softwares
- dynamic$^{\infty}$ computing: earth = computer = oxygen

□ Open Research Problem

Goal = automatic checking of bugs

Bugs = program runs unexpectedly

□ 50-Year Achievements: in retrospect

revolved in 3 steps

- step 1) Definition of bugs (logic)
- step 2) Checking system (logic)
- step 3) Implementation (logic and computation)

□ Automatic Checking of Bugs: 1st gen.

syntax analysis: lexical analysis & parsing (70s)

- step 1) bug = program's shape is wrong "`{intt x = 8*)}`"
- step 2) **Thm.** "no bugs" $\iff$ correct shape
- step 3) **Thm.** "`YES`" $\iff$ "no bugs"
  - checking in $\sim 10^4$ lines/sec
  - CFG languages

□ Automatic Checking of Bugs: 2nd gen.

type checking/inference (90s, a pride of pgm'ng language area)

- step 1) bug = program's execution is untypeful "`free(x);`"
- step 2) **Thm.** "no bugs" $\Longrightarrow$ typeful exec.
- step 3) **Thm.** "`YES`" $\Longleftrightarrow$ "no bugs"
  - checking/inferencing in $\sim 10^3$ lines/sec
  - HOT(higher-order & typed) languages v.s. C, C++, Java

□ Automatic Checking of Bugs: (3+k)th gen.

under way

- step 1) bug = program's execution is not "as required"
- step 2) by program analysis/program logics/language technologies
- step 3) implementation

□ System Zoo is a

tool for the generation-3 debugging technology (LET Project)

□ LET Project `ropas.kaist.ac.kr` (simplified)

- use *static analysis*
- step 1) bug = program's execution is not "as required"
- step 2) static analysis of programs against requirements
- step 3) implementation
- System Zoo automates step 2 and 3

□ Static Analysis

a general technology for compile-time, automatic, and safe estimation of program's run-time properties

- "general" : no limit on languages and properties
- "compile-time" : before execution
- "automatic" : program analyzes programs
- "safe" : result must subsume the reality
- "estimation" : cannot be exact in principle

□ Example: exception analysis [Yi94,YiRy97,Yi98,YiRy02]

- bug = uncaught exceptions

- analysis = statically analyzing every possible uncaught exceptions

- requirement = the result must be the empty set

□ Example: KAIST SatRec's Science Satellite (under way)

- bug = C module's index variable is beyond [0,127]
- analysis = statically estimating index variable's values
- requirement = the result must be within [0,127]

□ System Zoo

- a program analyzer generator

- a language for program properties/requirements

and ...

□ System Zoo

- to integrate with our nML compiler system (`ropas.kaist.ac.kr/n`)
  - a Korean dialect of Standard ML and OCaml: HOT family
- to transfer technology to the industry (int'l/domestic)
  - as "realistic/routine" as `lex` and `yacc`

□ Zoo Supports An Ensemble

- *abstract interpretation*
- *conventional data flow analysis*
- *constraint-based analysis*
- *model checking*

□ Use of Each Framework in Zoo

- variations in static analysis specification

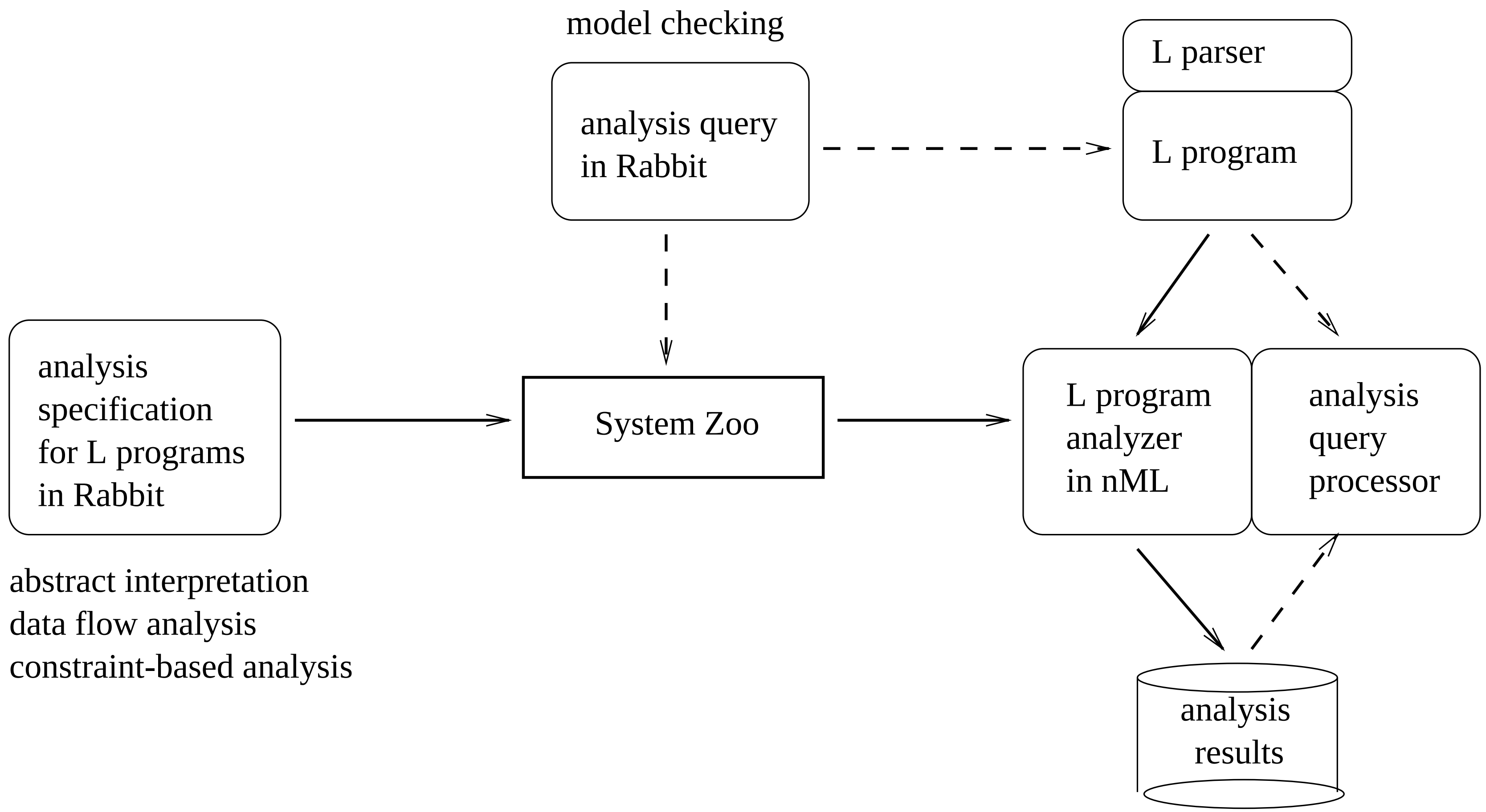
  - *abstract interpretation*

  - *data flow analysis*

  - *constraint-based analysis*

- query about analysis result

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


model checking
L parser
analysis query
in Rabbit
L program
analysis
specification
for L programs
in Rabbit
System Zoo
L program
analyzer
in nML
analysis
query
processor
abstract interpretation
data flow analysis
constraint-based analysis
analysis
results

□ Talk Plan

1. Zoo's viewpoint to program analysis

2. Rabbit: Zoo's programming language

3. Unique issues

□ Program Analysis: Views from Zoo

Given a program

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

□ Input to Zoo

How to set-up equations: *abstract interpretation* style

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

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

□ Correctness

Zoo users have 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

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

□ Generated Analyzer Sets Up Equations

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

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

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

□ Generated Analyzer Solves an Equation

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

- The F is derived from the input Rabbit program
- Solution: $\sqcup\{\bot, F\bot, F^2\bot, \cdots\}$

□ Solution: Fixpoint and Flow Graph

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

Flow graph:

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

□ Generated Analyzer Answers to Query

- program behavior = analysis result, the flow graph
- query = Computation-Tree-Logic formula (a modal logic)
  - modality = $\{\mathtt{A}, \mathtt{E}\} \times \{\mathtt{G}, \mathtt{F}, \mathtt{X}, \mathtt{U}\}$
  - body = first-order predicate over $X_i^{\downarrow}$ and $X_i^{\uparrow}$

Examples:

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

- Does variable v remain positive?

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

- Can variable v be positive?

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

- Does variable v remain positive until w is negative?

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

- 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;
...
```

□ All 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,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

```
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
```

```
(* equation set-up rule *)

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

  ccr  P@a <- app_p(/e/,X@b), /Ast.Lam(x,e')/ in post Cfa.Lam@/e/
      -----------------------------------------
           P@a <- P@/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 [MuYi02,YiEo02]

Static check of F

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

□ Issue II: Fixpoint Algorithm [EoYi02]

Some redundancies in:

$$\sqcup\{\bot, F\bot, F^2\bot, \cdots\}$$

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

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

Achieved a linear scale-up of the algorithm cost

□ Issue III: Rabbit's Use in Fixpoint-Carrying Code

FCC = a technology to check the safety of anonymous programs

Code provider sends	Code consumer checks
$\langle \text{code}, \vec{S} \rangle$	- Scan(code) = ($\vec{X} = F\vec{X}$)
	- Check($\vec{S} = F\vec{S}$)

Rabbit: the language for expressing the F and $\vec{S}$.

- property to check = a Rabbit program
- Rabbit translation from for ML/C to for x86/MSIL/JVM

□ Summing Up

- System Zoo will be useful for building/checking safe programs

- System Zoo is not a toy: practice $\leftrightarrows$ theory

- nML programming system is ready for the Zoo technology

May thy be proud of it, or not.

□ Links

- Papers/details/concrete numbers/other works:

  `ropas.kaist.ac.kr/~kwang/paper`
  `ropas.kaist.ac.kr/memo`

- Softwares:
  - The Zoo/Rabbit definition: `ropas.kaist.ac.kr/zoo`
  - The nML definition/compiler: `ropas.kaist.ac.kr/n`

Thank you.