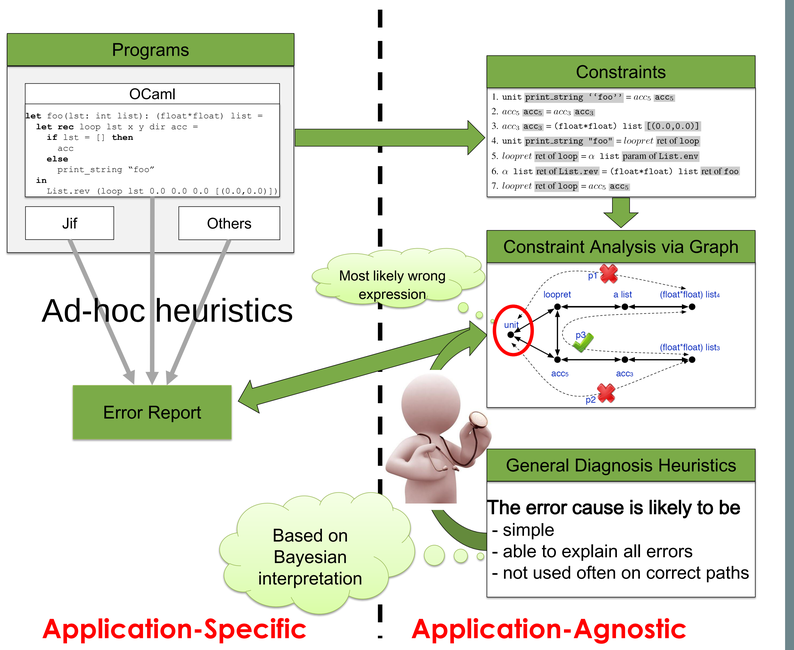
Toward General Diagnosis of Static Errors

Danfeng Zhang and Andrew C. Myers (Cornell University)

A general, concise and accurate diagnostic method for static errors

Overview



Static Program Analyses

- **Many flavors**
  - Type systems
  - Dataflow analysis
  - Information-flow analysis
- **Useful properties**
  - Type safety
  - Memory safety
  - Information-flow security
- But, sometimes hard to localize the cause of static errors

Examples

Better error report is needed

Type Inference (OCaml)

```
1 let foo(lst: int list): (float*float) list =
2   …
3   let rec loop lst x y dir acc =
4     if lst = [] then
5       acc
6     else
7       print_string "foo"
8   in
9     List.rev (loop lst 0.0 0.0 0.0 [(0.0,0.0)])
```

Mistake

OCaml: This expression has type 'a list but is here used with type unit

Information-Flow Analysis (Jif)

```
1 public final byte[{}] {this} encText;
2 …
3 public void m(FileOutputStream[{this}]{this} encFos)
4   throws (IOException) {
5   try {
6     for (int i=0; i<encText.length; i++)
7       encFos.write(encText[i]);
8   } catch (IoException e) {}
9 }
```

Mistake

Jif: This label is too conservative

Constraints

General representation of program analyses

Syntax of Constraints

$$G ::= G_1 \wedge G_2 \mid A \qquad A ::= C_1 \vdash C_2$$
$$C ::= I_1 \wedge \ldots \wedge I_n \ ^{(n\geq 0)} \qquad I ::= E_1 \leq E_2$$
$$E ::= \alpha \mid c(E_1, \ldots, E_{a(c)}) \mid \overline{c}^i(E) \mid E_1 \sqcup E_2 \mid E_1 \sqcap E_2 \mid \bot \mid \top$$

OCaml

```
1 let foo(lst: int list): (float*float) list =
2   let rec loop lst x y dir acc =
3     if lst = [] then
4       acc
5     else
6       print_string "foo"
7   in
8     List.rev (loop lst 0.0 0.0 0.0 [(0.0,0.0)])
```

Constraints

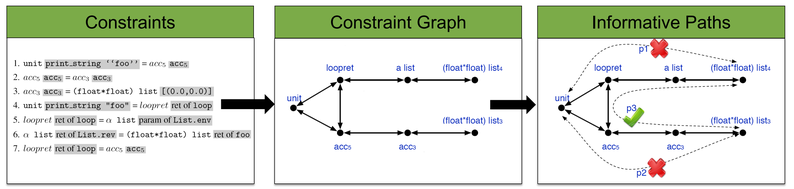
1. unit print_string ''foo'' = acc_5 acc_5
2. acc_5 acc_5 = acc_3 acc_3
3. acc_3 acc_3 = (float*float) list [(0.0,0.0)]
4. unit print_string "foo" = $loopret$ ret of loop
5. $loopret$ ret of loop = α list param of List.env
6. α list ret of List.rev = (float*float) list ret of foo
7. $loopret$ ret of loop = acc_5 acc_5

Can also express information-flow and dataflow analysis

Constraint Graph

Efficient constraint analysis via CFG-reachability

- **Graph construction**
  - Node: constraint elements
  - Edge: partial orders on elements
- **Finding informative paths**
  - Saturate constraint graph via CFG-reachability
  - Test the satisfiability of a partial order on end nodes
  - Trivial paths are ignored (e.g., one end node is a variable)



Error Diagnosis

Idea: maximum a posteriori (MAP) estimation

- **The likelihood of being the cause of error**
  - o: observation (satisfiablility of LEQ paths)
  - $\mathcal{G}$: pair of elements and hypothesis that explains error
  - k_E: # satisfiable paths with elements of E

$$\underset{(E,H)\in\mathcal{G}}{\operatorname{argmax}} P(E,H|o) \rightarrow \underset{(E,H)\in\mathcal{G}}{\operatorname{argmax}} P_1^{|E|}\left(\frac{P_2}{1-P_2}\right)^{k_E} P_\Psi(H)$$

Bayes' theorem & simplifying assumptions

Refinement is likely to improve precision

Simplifying Assumptions

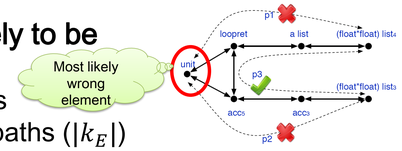
All elements are equally likely to be wrong (with P_1)
Errors are unlikely (with $P_2 < 0.5$) to appear on satisfiable paths

Inferring Likely Wrong Elements

Idea: heuristic search

$$\underset{E}{\operatorname{argmax}} P_1^{|E|}\left(\frac{P_2}{1-P_2}\right)^{k_E}$$

- **A* search**
  - Optimal: return all most likely wrong elements
  - Efficient: ~10 seconds when the search space is over 2^{1000}
- **The wrong elements are likely to be**
  - simple ($|E|$)
  - used on all unsatisfiable paths
  - not used often on satisfiable paths ($|k_E|$)



Inferring Likely Missing Hypothesis

Idea: find minimal weakest hypothesis

$$\underset{H}{\operatorname{argmax}} P_\Psi(H)$$

- **Simplicity is not the only metric**
  - $\top \leq \bot$ "explains" all errors
- **Likely missing hypothesis is weak and simple**

$\mathrm{Bob} \leq \mathrm{Carol} \vdash \mathrm{Alice} \leq \mathrm{Bob}$
$\mathrm{Bob} \leq \mathrm{Carol} \vdash \mathrm{Alice} \leq \mathrm{Carol}$
$\mathrm{Bob} \leq \mathrm{Carol} \vdash \mathrm{Alice} \leq \mathrm{Carol} \sqcup \bot$

Minimal weakest hypothesis: $\mathrm{Alice} \leq \mathrm{Bob}$

Evaluation

Our tool identifies error locations more accurately

Correctness metric
- OCaml: user's fix with larger time stamp
- Jif: errors marked by the programmer

OCaml

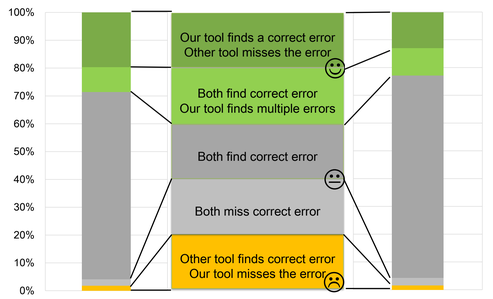
Same corpus used by the Seminal tool [Lerner et al.'07]

Analyzed 336 programs with type mismatch errors

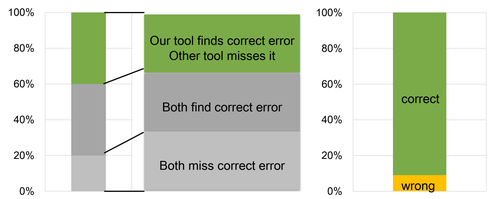
Jif

16 previous collected buggy programs

Contains both error types



The OCaml compiler — Seminal [Lerner et al.'07]



The Jif compiler (Wrong elements) — Missing hypothesis