Snugglebug: A Powerful Approach To Weakest Preconditions (PLDI'09)

Authors: Satish Chandra, Stephen J. Fink, Manu Sridharan

Presenter: Chiao Hsieh

Problem of finding a precondition

Entrypoint m

```
27 public static void entrypoint(
28   checkValid(c, c instanceof N
29 }
30 private static void checkValic
31   boolean newCarsOnly) throws
32   Iterator<Car> it = c.iterator();
33   while (it.hasNext()) {
34     Car x = it.next();
35     if (newCarsOnly) {
36       int y = x.getYear();
37       if (y != 2009) {
38         throw new MyException(); // GOAL
39       }
40     }
41     // ... other checks.
42   }
43 }
```

Precondition φ:
Satisfying φ must drive execution from m to g

Goal state g

Example 1

Why finding the precondition φ at m

- Specification discovery and API hardening
  - Avoid exceptional state represented by g via adding φ to documentation or changing code
- Test case generation
  - Test cases that produce (entry) states satisfying φ could help generate method sequences to g
- Bug (report) validation
  - Reduce false-positives reported by existing tools via finding φ for these bug reports

Background

- Backward Symbolic Analysis
  - Going backward from goal location to the entry point
  - Compute *weakest precondition*[1] over each statement
- Difference from SE/DSE
  - \+ Explore only paths and parts of program that are relevant to the goal
  - − No help from concrete execution path and variable value
  - − Not only input variables

```
Iterator<Car> it = c.iterator();
while (it.hasNext()) {
  Car x = it.next();
  if (x.year != 2009 && newCarsOnly) {
    int y = x.getYear();
    if (y != 2009) {
      throw new MyException(); // GOAL
    }
  }
```

(x.year != 2009 && newCarsOnly)
(x.year != 2009)
(y != 2009)
(true)

Snippet from Example 1

[1] E. W. Dijkstra., “A Discipline of Programming”, 1997

Analysis Basics - Intraprocedure

statement	edge condition	$wp(statement, \phi)$
$v = w$		$\phi[w/v]$
$v = v_1\ op\ v_2$		$\phi[(v_1\ op\ v_2)/v]$
$v = w.f$	normal successor	$(w \neq \texttt{null}) \wedge \phi[\texttt{read}(f$
	NPE successor	$(w = \texttt{null}) \wedge \phi[fresh(\texttt{N}$
$v.f = w$	normal successor	$(v \neq \texttt{null}) \wedge \phi[\texttt{update}$
	NPE successor	$(v = \texttt{null}) \wedge \phi[fresh(\texttt{N}$
$v = w[i]$	normal successor	$(w \neq \texttt{null}) \wedge (i < \texttt{rea}$
	NPE successor	$(w = \texttt{null}) \wedge \phi[fresh(\texttt{N}$
	OOB successor	$(w \neq \texttt{null}) \wedge (i < 0 \vee$
$w[i] = v$	normal successor	$(w \neq \texttt{null}) \wedge (i < \texttt{rea}$
	NPE successor	$(w = \texttt{null}) \wedge \phi[fresh(\texttt{N}$
	OOB successor	$(w \neq \texttt{null}) \wedge (i < 0 \vee$
$v = new\ T$		$\phi[fresh(T)/v]$
$assume\ c$		$\phi \wedge c$
$v_1 = (T)\ v_2$	CCE successor	$v_2 \neq \texttt{null} \wedge \neg(\texttt{subType}$
	normal successor	$(v_2 = \texttt{null} \vee \texttt{subType}($
$return\ v$		$\phi[\texttt{null}/\ exc][v/\ ret]$
$w = v.m()$	normal successor, callee $meth$	$meth = \texttt{dispatch}(\texttt{type}$
	NPE successor	$(v = \texttt{null}) \wedge \phi[fresh(\texttt{N}$

Table 1. Specification of wp for representative Java statements. v, w and c variab

Functions and Constants	
T_i	type constants (one per concrete type)
M_i	method constants (one per concrete method)
F_i	field constants (one per declared field)
sig_i	constant corresponding to a method signature
$\texttt{read}(f, v)$	$f(v)$, where f is $Val \rightarrow Val$ (a relational model of some declared field)
$\texttt{update}(f, v, w)$	functional update of f, i.e. $f[v \mapsto w]$
$\texttt{aread}(a, v, i)$	$a(v, i)$, where a is $Val \times Index \rightarrow Val$
$\texttt{aupdate}(a, w, i, v)$	functional update of a, i.e. $a[(w, i) \mapsto v]$
$\texttt{typeOf}(v)$	the type of object to which v points
$\texttt{dispatch}(t, sig)$	method to which signature sig will dispatch to on receiver of type t
$\texttt{subType}(t_1, t_2)$	true iff type t_1 is subtype of type t_2 in Java

Axioms

$this \neq \texttt{null}$

$\forall f.\forall v.\forall w.\texttt{read}(\texttt{update}(f, v, w), v) = w$

$\forall f.\forall v.\forall w.\forall u.u \neq v \Rightarrow \texttt{read}(\texttt{update}(f, v, w), u) = \texttt{read}(f, u)$

$\texttt{subType}(T_1, T_2) \Leftrightarrow \texttt{T}_1$ is a subtype of $\texttt{T}_2$ in Java

$\texttt{dispatch}(T_1, sig_{A.m}) = M_{B.m} \Leftrightarrow$
$\forall \texttt{x}.(\texttt{x.class} = \texttt{T}_1 \Rightarrow \texttt{x.m()}$ dispatches to target method $\texttt{B.m()})$

$\forall x.\texttt{read}(length, x) \geq 0$

Table 2. Some representative symbols and axioms in our theory

Backward derivation rules and extra predicates

Analysis Basics - Interprocedure

- Handle method call `A.m()`
  - Compute wp backward from exit point to the entry point of `A.m()`
- Problems in Example 1
  - Iterator from which class? List? Set?
  - Compute wp for `getYear()` every time encountered?

```
public class Car {
  int year;
  void setYear(int y) { this.year = y; }
  int getYear() { return year; }
}
```

???

```
Iterator<Car> it = c.iterator();
while (it.hasNext()) {
  Car x = it.next();
  if (newCarsOnly) {
    int y = x.getYear();
    if (y != 2009) {
      throw new MyException(); // GOAL
    }
  }
```

Snippet from Example 1

Challenges addressed

Optimization & Design Decisions

Directed Call Graph Construction

Generalization
(Generalized Procedure Summaries)

[2] M. Sharir and A. Pnueli, “Two approaches to interprocedural data flow analysis”, 1981

Idea of Directed Call Graph Construction

Skip computing wp for calls via adding *skolem* constants.

Search for only classes/methods that satisfy current version of precondition

(*Feedback*)

See Section 4

```
27 public static void entrypoint(Collection<Car> c)
28   checkValid(c, c instanceof NewCarList);
29 }
30 private static void checkValid(Collection<Car> c,
31   boolean newCarsOnly) throws MyException {
32   Iterator<Car> it = c.iterator();
33   while (it.hasNext()) {
34     Car x = it.next();
35     if (newCarsOnly) {
36       int y = x.getYear();
37       if (y != 2009) {
38         throw new MyException(); // GOAL
39       }
40     }
41     // ... other checks.
42   }
43 }
```

Skipped

Must be `NewCarList.iterator()`

Example 1

Idea of Generalization

Postcondition from backward analysis,

4: $\varphi 4$

5: $\varphi 5$

...

7: $\varphi 7$

8: $\varphi 8$

Summarize `Car.setYear()` : $\varphi 8 \xrightarrow{\texttt{setYear()}} \varphi 7$

If $\varphi 5 = \varphi 8$ we can derive $\varphi 4$ using this summary

```
1 public static void test(NewCarList l)
2   Car c1 = new Car();
3   l.set(0,c1);
4   c1.setYear(2008); // a bad car
5   Car c2 = new Car();
6   l.set(1,c2);
7   c2.setYear(2009); // a good car
8   entrypoint(l);
9 }
```

Example 2

Idea of Generalization

φ from backward analysis,

5: l.elems[0].year != 2009 && l.elems.length > 1

8: l.elems[0].year != 2009 && l.elems.length > 0

Summary will be too specific

Cannot reuse due to difference on l.elems.length

Solution: generalize and use related only

5: y != 2009 && y=read(year, x) && x = l.elems[0] && l.elems.length > 1

8: y != 2009 && y=read(year, x) && x = l.elems[0] && l.elems.length > 0

See Section 5.

```
1 public static void test(NewCarList l)
2   Car c1 = new Car();
3   l.set(0,c1);
4   c1.setYear(2008); // a bad car
5   Car c2 = new Car();
6   l.set(1,c2);
7   c2.setYear(2009); // a good car
8   entrypoint(l);
9 }
```

Example 2

Optimizations and Design Decisions

- Disjunctive formula propagation
  - Maintain computed formula as a set of minterms
- On-the-fly simplification
  - Custom domain specific SMT simplifier to rewrite formulas
- Loop, Recursion, and Search Heuristics
  - Tend to search for less looping and locations less reached
- Constraining Search with Over-approximation
  - Use abstract interpretation to cheaply prune infeasible paths

Evaluation - Bug report validation

[3] http://findbugs.sourceforge.net/

- 5 open source Java programs with over 750k lines
- 56 findings of possible NPE reported by FindBugs[3]
  - 56 = 29 validated + 9 not validated but feasible + 18 infeasible
  - 29 / 56 = 52% validated, each case in 30 mins
  - 29 / (29 + 9) = 76% feasible cases are validated

Benchmark	Version	Source kLOC
`ant`	1.7.0	88
`antlr`	2.7.2	38
`batik`	1.6	157
`tomcat`	6.0.16	163
`eclipse.ui`	3.3.1	305

Table 3. Information about our benchmarks, popular open-source Java programs. `eclipse.ui` consists of the plugins from Eclipse

Configuration	Validated	Not Validated	Avg. Time Per Goal (s)
Production	29	0	93
NoGeneralization	27	2	193
NoSimplification	11	18	1122
NoFeedback-CHA	12	17	1057
NoFeedback-Andersen	8	21	1331

Table 4. Comparison of results across five configurations.

Evaluation - Efficiency of Techniques

Production: All techniques

NoGeneralization: Generalization Disabled

NoSimplification: Custom Simplifier Disabled

NoFeedback-CHA: Call Graph via Class Hierarchy Analysis

NoFeedback-Anderson: Call Graph via Anderson's Analysis[4]

Configuration	Validated	Not Validated	Avg. Time Per Goal (s)
Production	29	0	93
NoGeneralization	27	2	193
NoSimplification	11	18	1122
NoFeedback-CHA	12	17	1057
NoFeedback-Andersen	8	21	1331

Table 4. Comparison of results across five configurations.

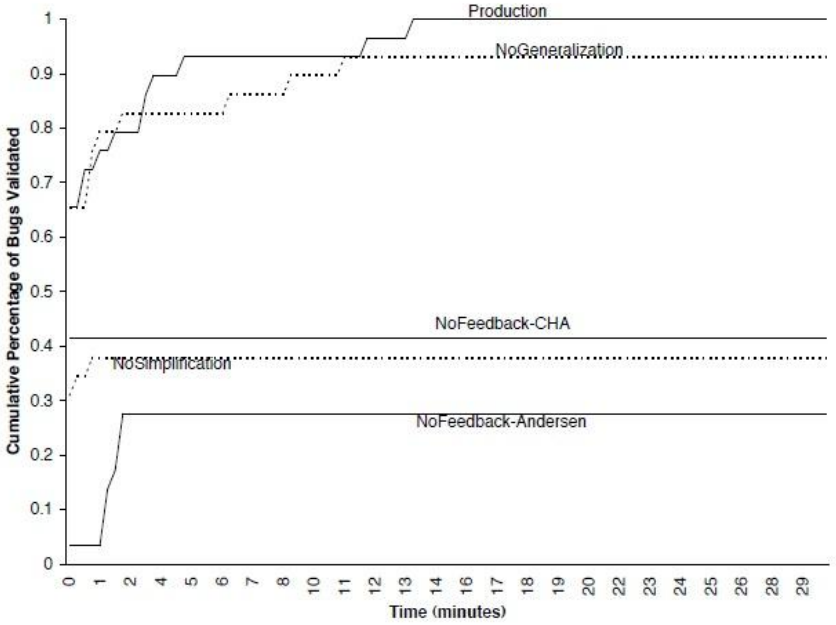

Figure 7. Comparison of running times across configurations.

[4] L. O. Andersen. "Program Analysis and Specialization for the C Programming Language", 1994

Evaluation - Directed Call Graph Construction

Sizes of call graphs from other 2 analyses are much larger.

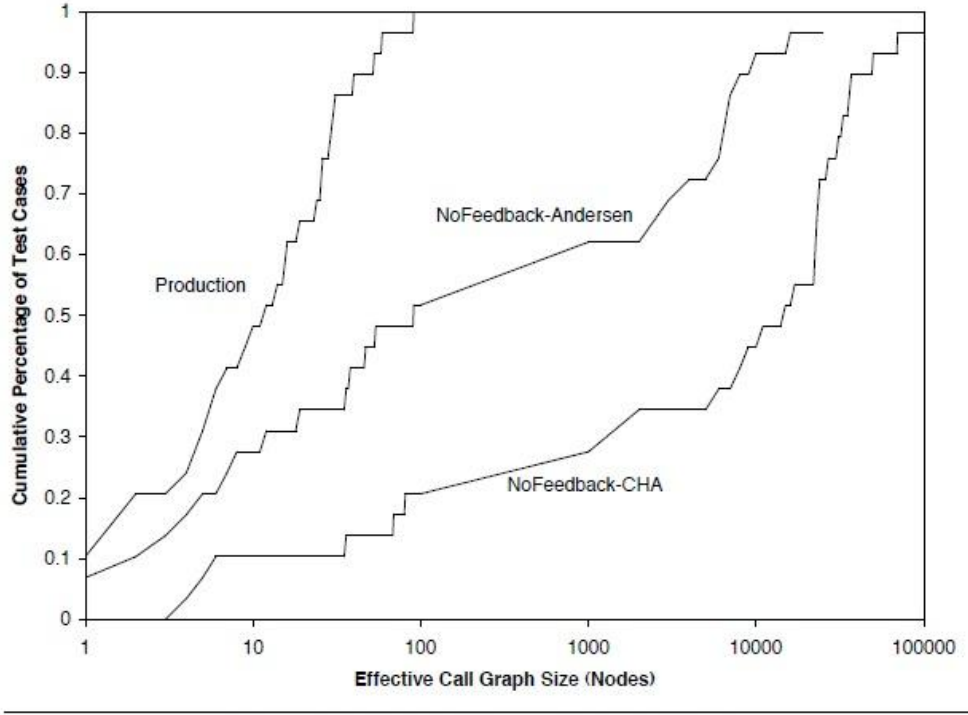

Figure 8. Effective call graph sizes.

Conclusion

- Directed Call Graph Construction
  - Explore much less possible call graphs
- Generalization
  - Improve summary reuse
- Other optimization techniques
  - On-the-fly domain specific simplification saves time usage
- Evaluation on large Java libraries

Thank you ~

Question 1

In other details of Section 6, it states, “For any native method m in a client program, we assume that

(1) m cannot throw an exception,

(2) m can return an arbitrary value, and

(3) m does not modify the heap.”

Where did these assumptions come from and what are their reasonings for these assumptions?

Question 2

From the graph given (Figure 7),

the technique with generalization disabled seems to beat the production strategy at early stage of analysis,

do you think that is a typical pattern?