Program Calling Context

Motivation

- Calling context enhances program understanding and dynamic analyses by providing a rich representation of program location
- Collecting calling context can be expensive
- The resulting representation of contexts can be bulky

2

Overview

- Probabilistic Calling Context [2]
- Precise Calling Context Encoding [3]

3

Probabilistic Calling Context[2]

Based on Mike Bond's slides[1]

Why Context Sensitivity?

- Static program location not enough

```
at com.mckoi.db.jdbcserver.JDBCInterface.execQuery():213
```


Why Context Sensitivity?

- Static program location not enough

```
at com.mckoi.db.jdbcserver.JDBCInterface.execQuery():213
at com.mckoi.db.jdbc.MConnection.executeQuery():348
at com.mckoi.db.jdbc.MStatement.executeQuery():110
at com.mckoi.db.jdbc.MStatement.executeQuery():127
at Test.main():48
```

Why Context Sensitivity?

- Static program location not enough

```
at com.mckoi.db.jdbcserver.JDBCInterface.execQuery():213
at com.mckoi.db.jdbc.MConnection.executeQuery():348
at com.mckoi.db.jdbc.MStatement.executeQuery():110
at com.mckoi.db.jdbc.MStatement.executeQuery():127
at Test.main():48
```

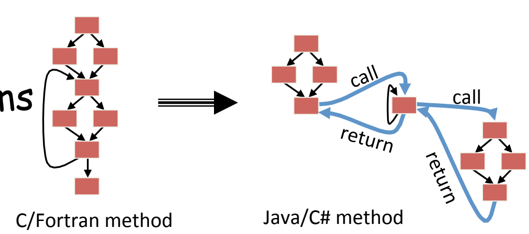
- Motivated by
  - Complex programs
  - Small methods
  - Virtual dispatch

Why Context Sensitivity?

- Static program location not enough

```
at com.mckoi.db.jdbcserver.JDBCInterface.execQuery():213
at com.mckoi.db.jdbc.MConnection.executeQuery():348
at com.mckoi.db.jdbc.MStatement.executeQuery():110
at com.mckoi.db.jdbc.MStatement.executeQuery():127
at Test.main():48
```

- Motivated by
  - Complex programs
  - Small methods
  - Virtual dispatch



Context Is Nontrivial

Program	API calls: Call sites	API calls: Distinct contexts
antlr	4,184	128,627
bloat	3,306	600,947
chart	2,335	202,603
eclipse	9,611	226,020
fop	2,225	37,710
hsqldb	947	16,050
jython	1,830	628,048
luindex	654	102,556
lusearch	507	905
pmd	1,890	847,108
xalan	1,530	17,905

Example: Residual Testing

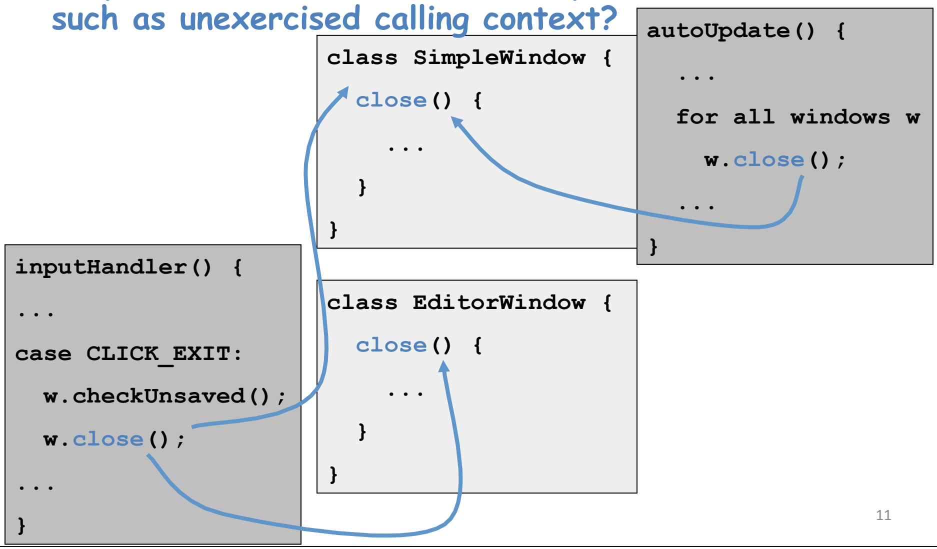
At production time, are there any untested behaviors, such as unexercised calling context?

```
class SimpleWindow {
  close() {
    ...
  }
}
```

```
class EditorWindow {
  close() {
    ...
  }
}
```

Example: Residual Testing

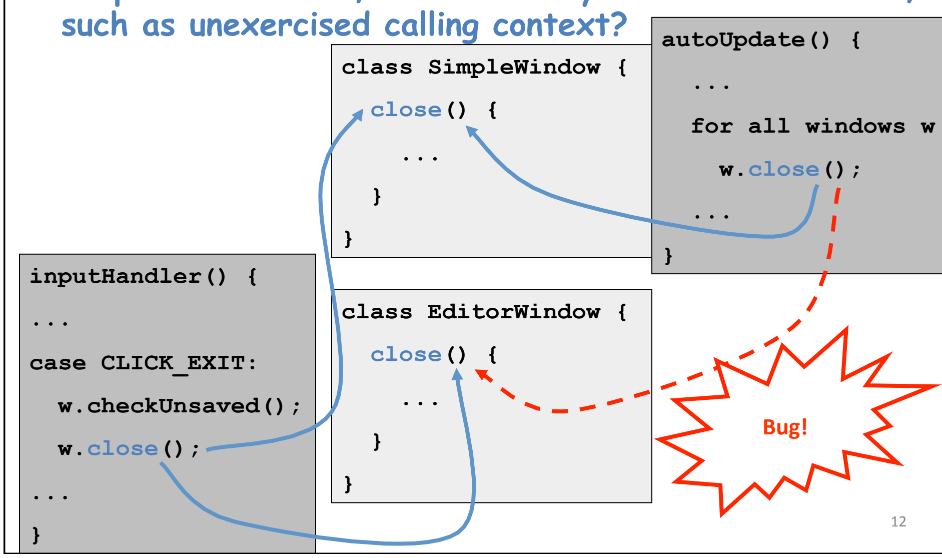
At production time, are there any untested behaviors, such as unexercised calling context?



11

Example: Residual Testing

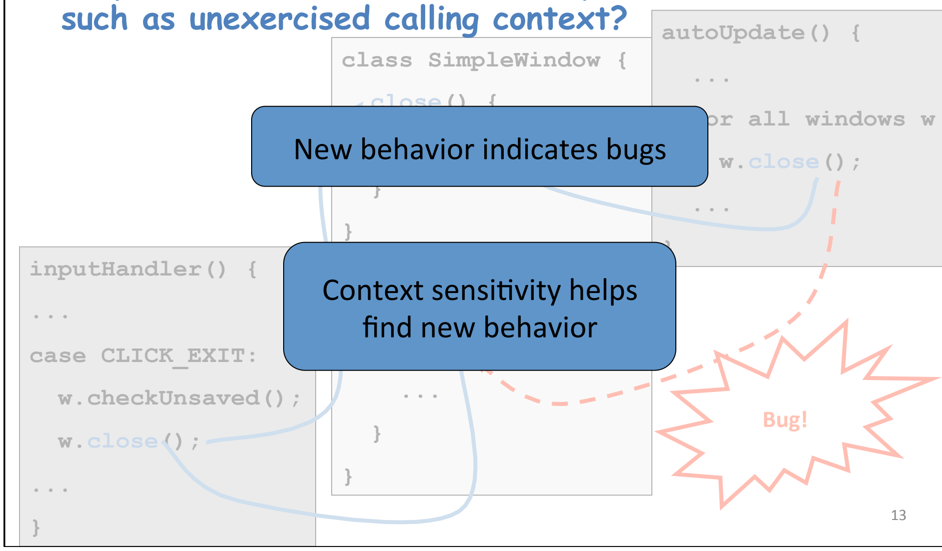
At production time, are there any untested behaviors, such as unexercised calling context?



12

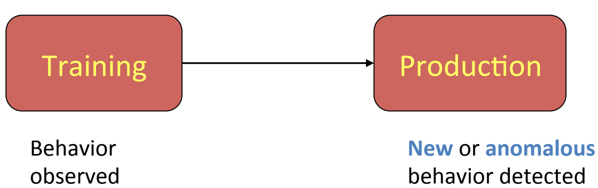
Example: Residual Testing

At production time, are there any untested behaviors, such as unexercised calling context?



13

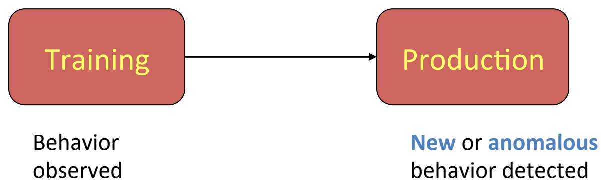
Two-Phase Dynamic Analyses



14

Probabilistic Calling Context

- *Adds context sensitivity to dynamic analyses*
- Maintains value representing context
  - Unique with high probability
  - New value → new context → walk stack
- High accuracy: <0.1% false negatives
- Low overhead: 3% overhead, 0-8% for clients



15

PCC Function

$$f\,(\,V\,,\,cs\,)$$

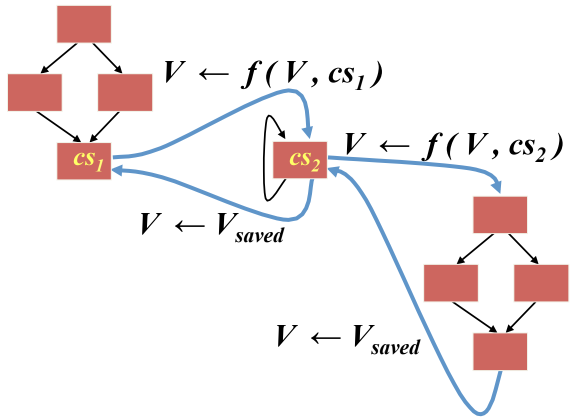
- V is PCC value
- cs is call site ID

16

PCC Function

$f(V, cs)$

- V is PCC value
- cs is call site ID



17

PCC Function

$$f(V, cs) \equiv 3V + cs \pmod{2^{32}}$$

- V is PCC value
- cs is call site ID

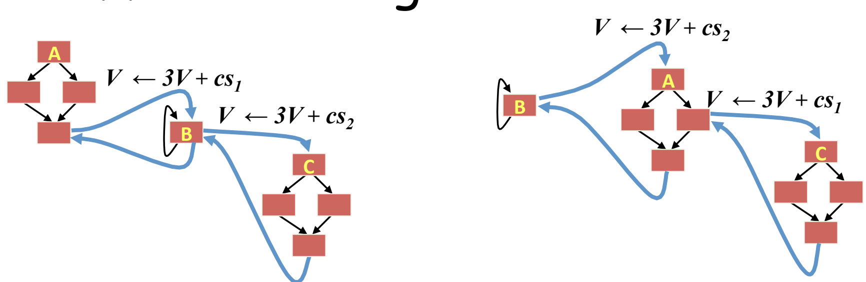
18

PCC Function

$$f(V, cs) \equiv 3V + cs \pmod{2^{32}}$$

- Cheap to compute
- Desirable properties:
  - Non-commutative
  - Composition efficient to compute

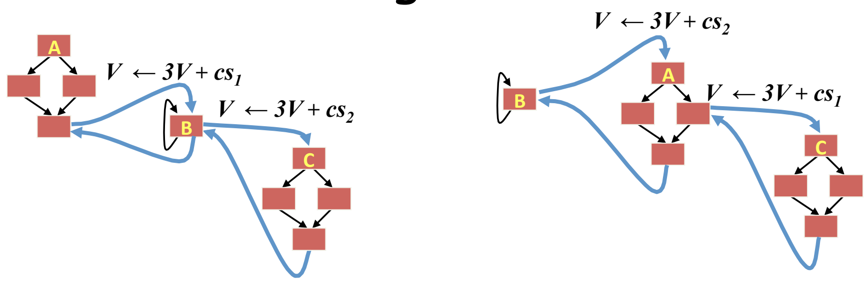
Differentiating Similar Contexts



… → A() → B() → …

… → B() → A() → …

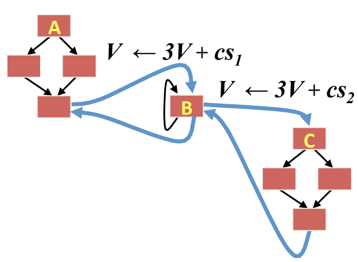
Differentiating Similar Contexts



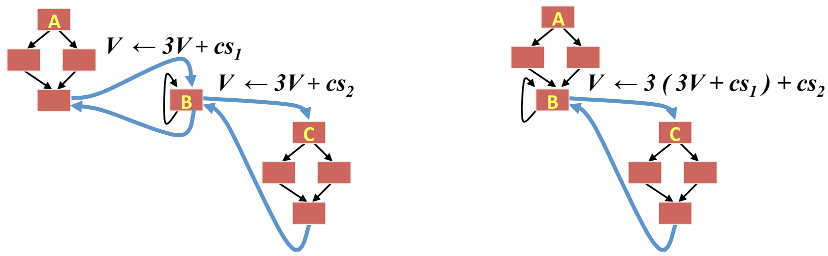
- Non-commutative

$$f(f(V, cs_1), cs_2) \neq$$
$$f(f(V, cs_2), cs_1)$$

Efficiency at Inlined Calls

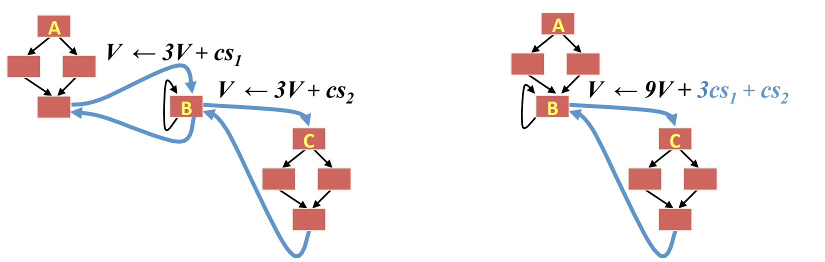


Efficiency at Inlined Calls



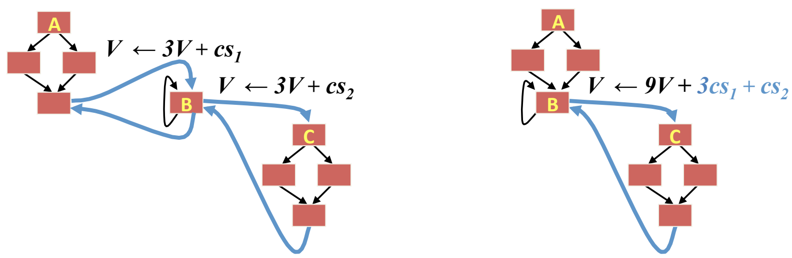
23

Efficiency at Inlined Calls



24

Efficiency at Inlined Calls



- Composition efficient to compute

$$f^n(V, cs_i) = 3^n V + \sum_i 3^i cs_i$$

Outline

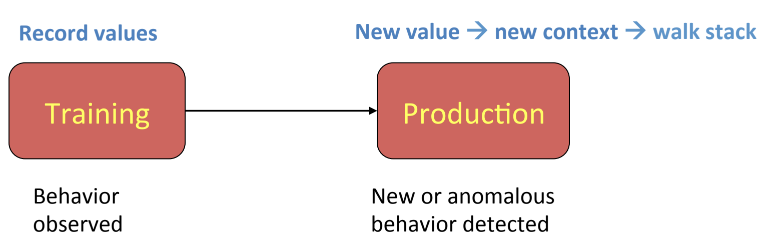
- Introduction
- Previous approaches
- Maintaining the PCC value
- Evaluation
  - Methodology
  - Evaluating potential clients
  - Accuracy
  - Performance

Methodology

- Implementation in Jikes RVM 2.4.6
  - Available on Jikes RVM Research Archive
- Deterministic calling context profiling
  - Maintains CCT node at each call & return
- Benchmarks: DaCapo, SPEC JBB2000, SPEC JVM98
- Platform: 3.6 GHz Pentium 4 w/Linux

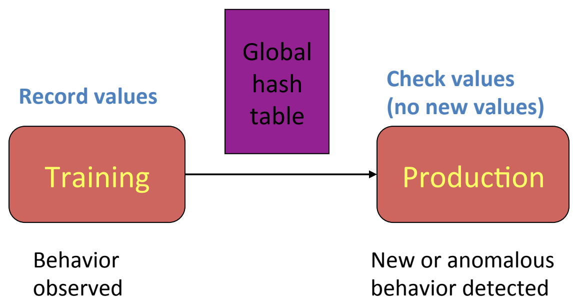
27

How Clients Use PCC



28

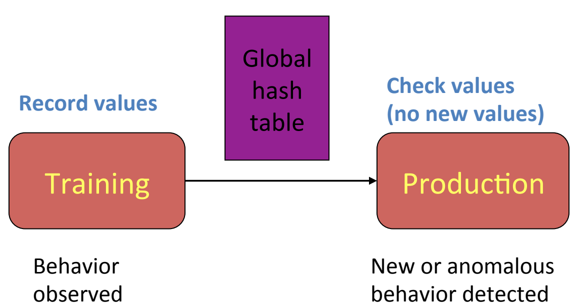
Evaluating Potential Clients



29

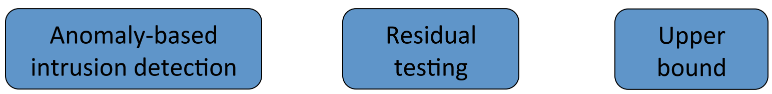
Evaluating Potential Clients

Memory overhead:
proportional to contexts



30

Evaluating Potential Clients



Anomaly-based intrusion detection	Residual testing	Upper bound
Check PCC value at **system calls** (Network, I/O, OS)	Check PCC value at **Java API calls** (calls to `java.*`)	Check PCC value at **all calls**

Ideal Accuracy

- PCC maps context to value
  - New PCC value → new context
  - Familiar PCC value → probably familiar context

Ideal Accuracy

- PCC maps context to value
  - New PCC value → new context
  - Familiar PCC value → probably familiar context

	Expected conflicts (false negatives)	
Distinct contexts	32-bit values	64-bit values
100,000	1 (0.0%)	0 (0.0%)
1,000,000	116 (0.0%)	0 (0.0%)
10,000,000	11,632 (0.1%)	0 (0.0%)
100,000,000	1,155,170 (1.2%)	0 (0.0%)
1,000,000,000	107,882,641 (10.8%)	0 (0.0%)
10,000,000,000	6,123,623,065 (61.2%)	3 (0.0%)

33

Ideal Accuracy

- PCC maps context to value
  - New PCC value → new context
  - Familiar PCC value → probably familiar context

	Expected conflicts (false negatives)	
Distinct contexts	32-bit values	64-bit values
100,000	1 (0.0%)	0 (0.0%)
1,000,000	116 (0.0%)	0 (0.0%)
10,000,000	11,632 (0.1%)	0 (0.0%)
100,000,000	1,155,170 (1.2%)	0 (0.0%)
1,000,000,000	107,882,641 (10.8%)	0 (0.0%)
10,000,000,000	6,123,623,065 (61.2%)	3 (0.0%)

API calls

34

Ideal Accuracy

- PCC maps context to value
  - New PCC value → new context
  - Familiar PCC value → probably familiar context

	Expected conflicts (false negatives)	
Distinct contexts	32-bit values	64-bit values
100,000	1 (0.0%)	0 (0.0%)
1,000,000	116 (0.0%)	0 (0.0%)
10,000,000	11,632 (0.1%)	%)
100,000,000	1,155,170 (1.2%)	0 (0.0%)
1,000,000,000	107,882,641 (10.8%)	0 (0.0%)
10,000,000,000	6,123,623,065 (61.2%)	3 (0.0%)

All calls

35

Ideal Accuracy

- PCC maps context to value
  - New PCC value → new context
  - Familiar PCC value → probably familiar context

	Expected conflicts (false negatives)	
Distinct contexts	32-bit values	64-bit values
100,000	1 (0.0%)	0 (0.0%)
1,000,000	116 (0.0%)	0 (0.0%)
10,000,000	11,	0 (0.0%)
100,000,000	1,155,170 (1.2%)	0 (0.0%)
1,000,000,000	107,882,641 (10.8%)	0 (0.0%)
10,000,000,000	6,123,623,065 (61.2%)	3 (0.0%)

Near-perfect accuracy

36

PCC's Accuracy

	System calls		
Program	Dynamic	Distinct	Conf.
antlr	211,490	1,567	0
bloat	12	10	0
chart	63	62	0
eclipse	14,110	197	0
fop	18	17	0
hsqldb	12	12	0
jython	5,929	4,289	0
luindex	2,615	14	0
lusearch	141	11	0
pmd	1,045	25	0
xalan	137,895	59	0

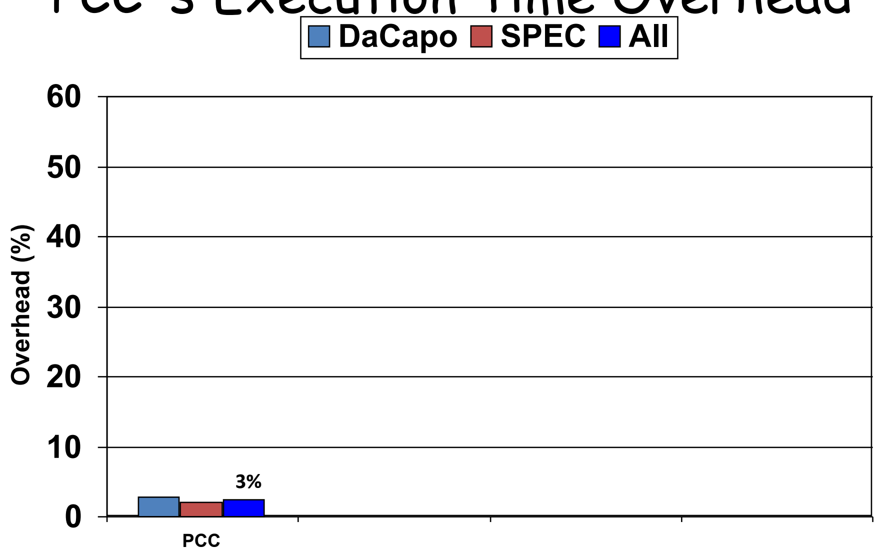
PCC's Accuracy

	System calls			Java API calls		
Program	Dynamic	Distinct	Conf.	Dynamic	Distinct	Conf.
antlr	211,490	1,567	0	24,422,013	128,627	3
bloat	12	10	0	1,159,281,573	600,947	40
chart	63	62	0	258,891,525	202,603	4
eclipse	14,110	197	0	132,507,343	226,020	5
fop	18	17	0	9,918,275	37,710	0
hsqldb	12	12	0	81,161,541	16,050	0
jython	5,929	4,289	0	543,845,772	628,048	48
luindex	2,615	14	0	39,733,214	102,556	0
lusearch	141	11	0	113,511,311	905	0
pmd	1,045	25	0	537,017,118	847,108	79
xalan	137,895	59	0	2,105,838,670	17,905	0

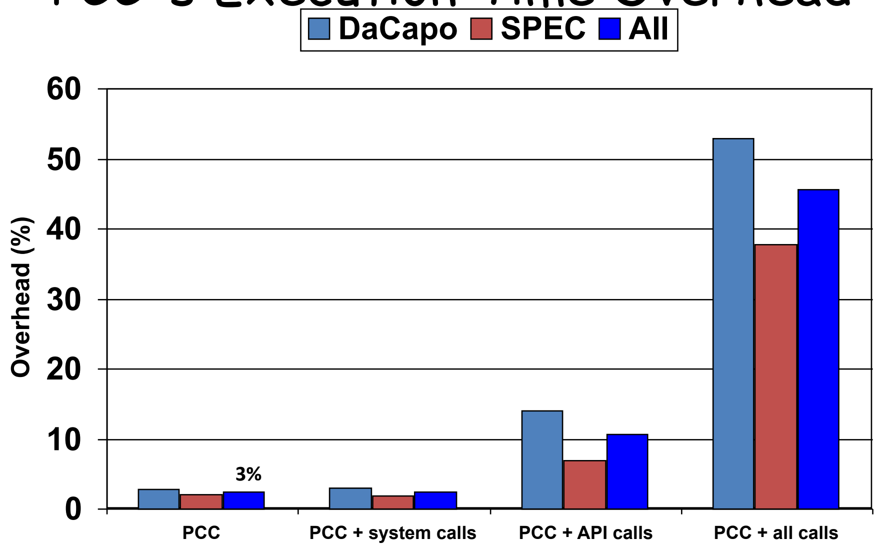
PCC' s Accuracy

Program	All calls Dynamic	Distinct	Conf.
antlr	490,363,211	1,006,578	118
bloat	6,276,446,059	1,980,205	453
chart	908,459,469	845,432	91
eclipse	1,266,810,504	4,815,901	2,652
fop	44,200,446	174,955	2
hsqldb	877,680,667	110,795	1
jython	5,326,949,158	3,859,545	1,738
luindex	740,053,104	374,201	12
lusearch	1,439,034,336	6,039	0
pmd	2,726,876,957	8,043,096	7,653
xalan	10,083,858,546	163,205	6

PCC' s Execution Time Overhead



PCC's Execution Time Overhead



Summary

- PCC maintains calling context value
  - New value indicates new behavior
- Low overhead
  - Maintaining PCC value adds 3%
  - Checking PCC value 0-8%
  - Memory overhead proportional to contexts
- High accuracy
  - Less than 0.1% false negative rate
- PCC adds context sensitivity to clients that detect anomalous behavior

42

Precise Calling Context Encoding [3]

Based on Slides[5]

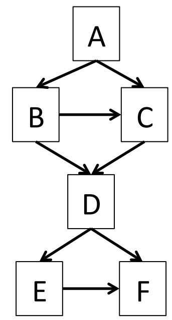
Motivation

- Calling context is important for profiling, debugging, and event logging
- The prior work encodes each calling context into a hash value, but cannot decode the context out of the value
- Is it possible to encode calling contexts into numbers from which the contexts can be restored?

44

Background

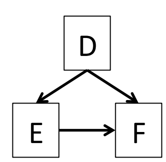
- Efficient Path Profiling [4]
  - Naively, where do you want to instrument to get the path profile?
  - Is there any way to reduce instrumentation edges?



45

The Basic Idea

- If one node has only one outgoing edge, you don’t need to instrument the edge
- If different paths can be encoded to different numbers, you can restore the whole paths purely based on the number values
- A naïve example: Given the CFG, how do you map the values to paths?



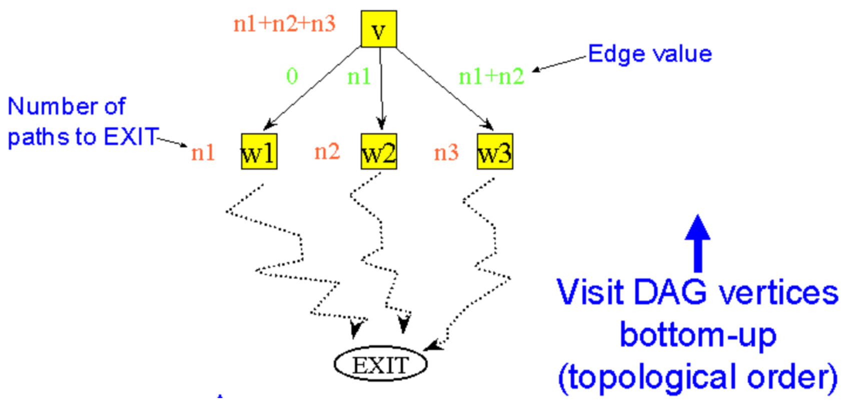
46

Edge Value Assignment

- Assign each edge a value such that:
  - Sum of values along DAG path is unique, non-negative integer
  - Sum lies in range 0 … num_paths – 1
- Simple linear-time algorithm

47

Edge Value Assignment

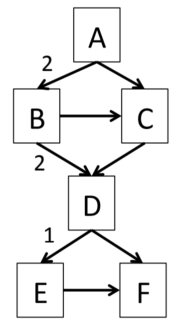


48

Edge Value Assignment

- Algorithm

```
foreach vertex v in reverse topological order {
   if v is a leaf vertex {
      NumPaths(v) = 1;
   } else {
      NumPaths(v) = 0;
      for each edge e = v - > w {
         Val(e) = NumPaths(v);
         NumPaths(v) = NumPaths(v) + NumPaths(w);
      }
   }
}
```

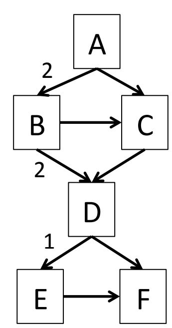


49

How to decode a path from a number?

- Algorithm

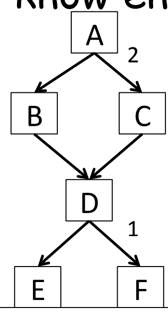
```
Given a path id Idx, set current node cur = A
while (cur != F) {
   take the path whose value val(e) is
         (1) smaller than Idx, and
         (2) the maximum value among all the
             values smaller than Idx
   Idx = Idx – val(e);
   cur = tgt(e);
}
```



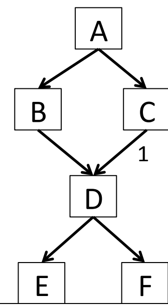
50

Is the Ball-Larus algorithm applicable to Call Graph encoding?

- Yes. The BL algorithm uniquely identifies each path ending at different methods. However, paths ending at different methods can have the same values, if we know ending points.



Context	**id**
ABDE	0
ABDF	1
ACDE	2
ACDF	3



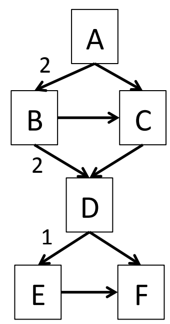
Context	**id**
ABDE	0
ABDF	0
ACDE	1
ACDF	1

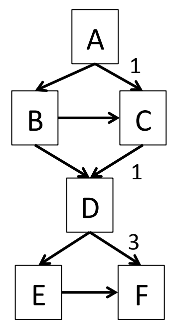
Edge Value Assignment

- Algorithm

```
foreach vertex v in topological order {
    if v is a root vertex {
        NumPaths(v) = 1;
    } else {
        NumPaths(v) = 0;
        for each edge e = w - > v {
            Val(e) = NumPaths(v);
            NumPaths(v) = NumPaths(w) + Val(e);
        }
    }
}
```

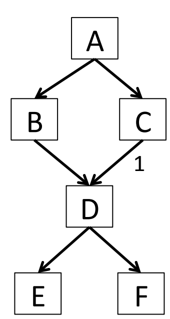
BL vs. New Algorithm





53

How to decode a context from a number?

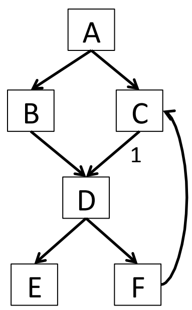


Context	**id**
ABDE	0
ABDF	0
ACDE	1
ACDF	1

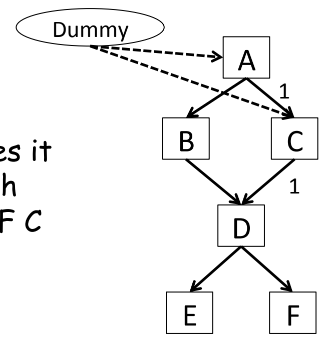
54

Encoding Cyclic CGs

- Introduce dummy edges



Given encoded context (F, 1), does it correspond to path (A C D F), (A B D F C D F), etc?



Path	Encoding
A C D F	(1, F)
A B D F C D F	(<0, F to C>, 1, F)

55

Selective Reduction

- To fit IDs of contexts in 32-bit integers,
  - Profiling is used to identify hot and code call edges
  - Code edges are replaced with dummy edges such as sub-paths starting with these code edges can be separately encoded

56

Evaluation

- 19 C programs from SPECint 2000 benchmarks, and a set of open source programs

Comparison between BL and the new algorithm

Table 1: Static program characteristics.

programs	LOC	CG nodes	CG edges	recursions	fun pointers	our max ID	BL max ID
cmp 2.8.7	6681	68	162	0	5	44	156
diff 2.8.7	15835	147	465	6	8	645	3140
sdiff 2.8.7	7428	90	281	0	5	242	684
find 4.4.0	39531	567	1362	28	33	1682	5020
locate 4.4.0	28393	320	688	3	19	251	1029
grep 2.5.4	26821	193	665	17	10	17437	44558
tar 1.16	58301	791	2697	19	46	1865752	4033519
make 3.80	29882	271	1294	61	7	551654	1543113
alpine 2.0	556283	2880	26315	302	1570	4294967295*	4.5e+18
vim 6.0	154450	2365	15822	1124	27	4291329441*	8.7e+18
164.gzip	11121	154	426	0	2	536	1283
175.vpr	29807	327	1328	0	2	1848	13047
176.gcc	340501	2255	22982	1801	128	4294938599*	9.1e+18
181.mcf	4820	93	208	2	0	6	96
186.crafty	42203	179	1533	17	0	188589	650779
197.parser	27371	381	1676	125	0	2734	14066
255.vortex	102987	980	7697	41	15	4294966803*	1.5e+13
256.bzip2	8014	133	396	0	0	131	609
300.twolf	49372	240	1386	9	0	1051	3766

*Selective reduction is applied to 288 nodes in `alpine 2.0`, 300 in `176.gcc`, 33 in `255.vortex`, and 877 in `vim`.

Observations

- BL usually instruments outgoing edges when there are multiple
- The new algorithm usually instruments incoming edges when there are multiple
- It seems that the latter instrumentation is more efficient than the former one
  - The intuition behind is there are more methods with multi-method callees than methods with multi-method callers
- No experiment focuses on the instrumentation comparison

59

Stack Depth & Encountered Dynamic Contexts

programs	Max Depth		90% Depth		dynamic contexts
	ours	plain	ours	plain	
cmp 2.8.7	0	3	0	3	9
diff 2.8.7	0	7	0	5	34
sdiff 2.8.7	0	5	0	4	44
find 4.4.0	2	12	1	12	186
locate 4.4.0	0	9	0	9	65
grep 2.5.4	0	11	0	8	117
tar 1.16	3	40	2	31	1346
make-3.80	6	82	3	43	1789
alpine 2.0	11	29	6	18	7575
vim 6.0	10	31	5	10	3226
164.gzip	0	9	0	7	258
175.vpr	0	9	0	6	1553
176.gcc	19	136	2	15	169090
181.mcf	14	42	0	2	12920
186.crafty	34	41	10	23	27103471
197.parser	36	73	11	28	3023011
255.vortex	7	43	2	12	205004
256.bzip2	1	8	0	8	96
300.twolf	4	11	0	5	971
Average	8.78	39.22	2.17	13.72	1795292

Table 2: Dynamic context characteristics.

60

Summary

- Calling context is important for program analysis and debugging
- Both instrumentation overhead and memory overhead (how to encode the context) are important
- Can you think about any approach to further reduce instrumentation effort?

61

References

[1] Mike Bond, Probabilistic Calling Context, PPT, http://web.cse.ohio-state.edu/~mikebond/papers.html

[2] Mike D. Bond, Kathryn S. McKinley, Probabilistic Calling Context, OOPSLA '07.

[3] William N. Sumner, Yunhui Zheng, Dasarath Weeratunge, Xiangyu Zhang, Precise calling context encoding, ICSE '10

[4] Thomas Ball, James Larus, Efficient Path Profiling, Micro '96

[5] James Larus, Efficient Path Profiling, Pages, http://pages.cs.wisc.edu/~larus/Talks/path_talk/

62