Advanced Program Analyses for Object-oriented Systems

Dr. Barbara G. Ryder
Rutgers University
http://www.cs.rutgers.edu/~ryder
http://prolangs.rutgers.edu/
July 2007

Lecture 3 - Outline

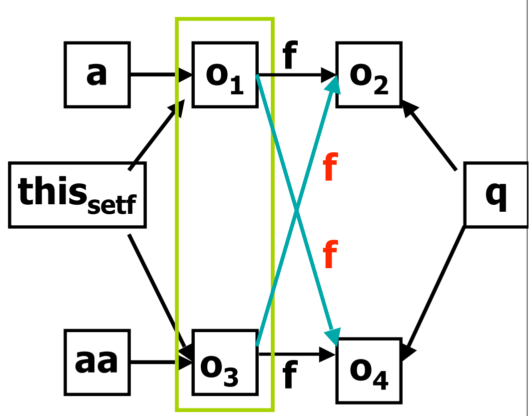
- Context-sensitive reference analysis
  - K-CFA vs. Object-sensitive analysis
  - Clients: run-time cast check removal, side effect analysis
- Dealing with the 'closed world' assumption
  - Modeling libraries
  - Incremental points-to analysis
- Dynamic analysis for Feedback-directed Optimization (FDO)

Imprecision of Context-insensitive Analysis

- Does not distinguish contexts for instance methods and constructors
  - States of distinct objects are merged
- Common OOPL features and idioms result in imprecision
  - Encapsulation
    - set() method conflates all instances with same field
  - Inheritance
    - Initialized fields in superclass constructor conflates points-to sets of subclass objects created
  - Containers, maps and iterators
    - Same creation site results in apparent unioning of all contents

Example: Imprecision

```
class Y extends X { … }
class A {
  X f;
  void setf(X q) {
      this.f=q ; }
  }
A a = new A() ;
a.setf(new X()) ;
A aa = new A() ;
aa.setf(new Y()) ;
}
```



Example - Imprecision

```
class X {void n(){}}
class Y extends X{ void n(){}}
class Z extends X{ void n(){}}
class A { X f;
  A(X xa) {this.f = xa;}}
Class B extends A{
  B(X xb) {super(xb);..}
  void m(){
    X xb = this.f;  xb.n();}}
Class C extends A{
  C(X xc) {super(xc);..}
  void m(){
    X xc = this.f; xc.n();}}
//in main()
{Y y = new Y(); Z z = new Z();
B b = new B(y); C c = new C(z);
b.m();  c.m();
}
```

What is target of the red call?
What is the target of the blue call?

Context Sensitivity

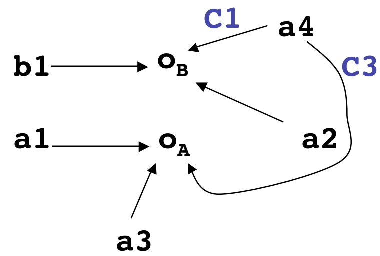
- **Keeping calling contexts distinct during the analysis**
- **Classically two approaches**
  - *Call string* - distinguish analysis result by (truncated) call stack on which it is obtained
    - e.g., K-CFA
  - *Functional* - distinguish analysis result by (partial) program state at call
    - e.g., receiver identity, argument types

M. Sharir, A. Pneuli, "Two Approaches to Interprocedural Dataflow Analysis". Ch 7 in Program Flow Analysis, Edited by S. Muchnick, N. Jones, Prentice-Hall 1981

1-CFA

- Calling context is tail of *call string* (1-CFA is last call site)

```
static void main(){
       B b1 = new B();//O_B
       A a1 = new A();//O_A
       A a2,a3;
C1:    a2 = f(b1);
C2:    a2.foo();
C3:    a3 = f(a1);
C4:    a3.foo();
}
public static A f(A a4){return a4;}
```



Points-to Graph

at C2, main calls B.foo()
at C4, main calls A.foo()

1-CFA Characteristics

- Call-string approach to context sensitivity
- Only analyzes methods *reachable* from main()
- Keeps track of individual reference variables and fields
- Groups objects by their creation site
- Incorporates reference value flow in assignments and method calls
- Differentiates points-to relations for different calling contexts

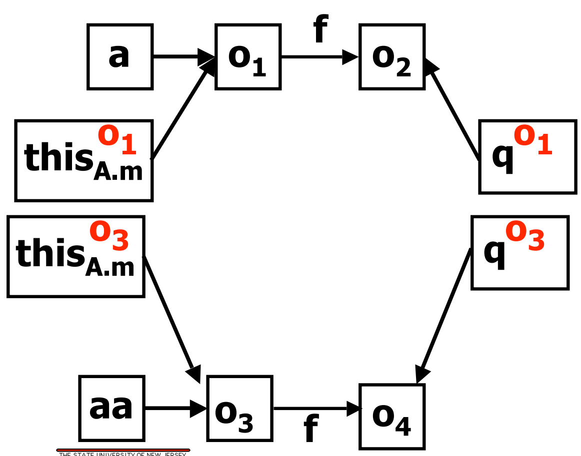
Object-sensitive Analysis (ObjSens)

- Receiver objects used as calling context
- Instance methods and constructors analyzed for different contexts
- Multiple copies of local reference variables

this.f=q $\xrightarrow{o_1}$ $this^{o_1}_{A.m}.f=q^{o_1}$

Example: Object-sensitive Analysis

```
class A {
  X f;
  void m(X q) {
this^o3_A.m.f=q^o3; }
  }
A a = new A() ;
a.m(new X()) ;
A aa = new A() ;
aa.m(new Y()) ;
```



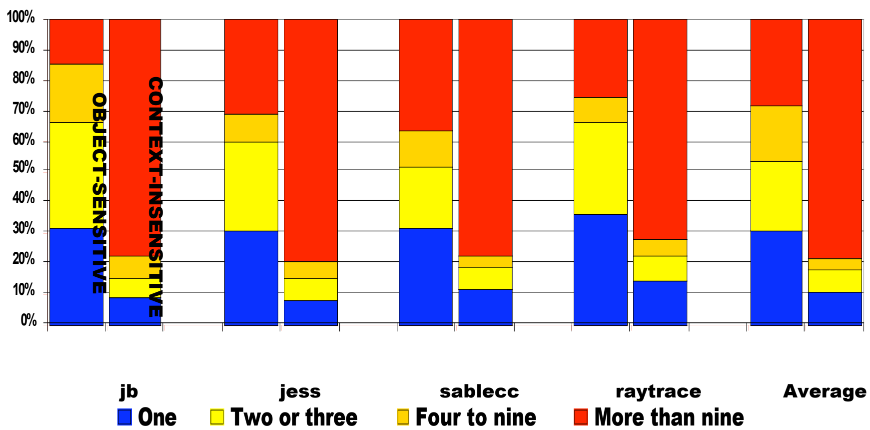
ObjSens Analysis

- Based on Andersen's points-to for C
- Uses receiver object to distinguish different calling contexts
- Groups objects by creation sites
- Represents reference variables and fields by name program-wide
- Flow-insensitive, context-sensitive, field-sensitive

Milanova, A. Rountev, B. G. Ryder, "Practical Points-to Analyses for Java", ISSTA'02;"Parameterized Object Sensitivity for Points-to Analysis for Java", TOSEM, Jan 2005

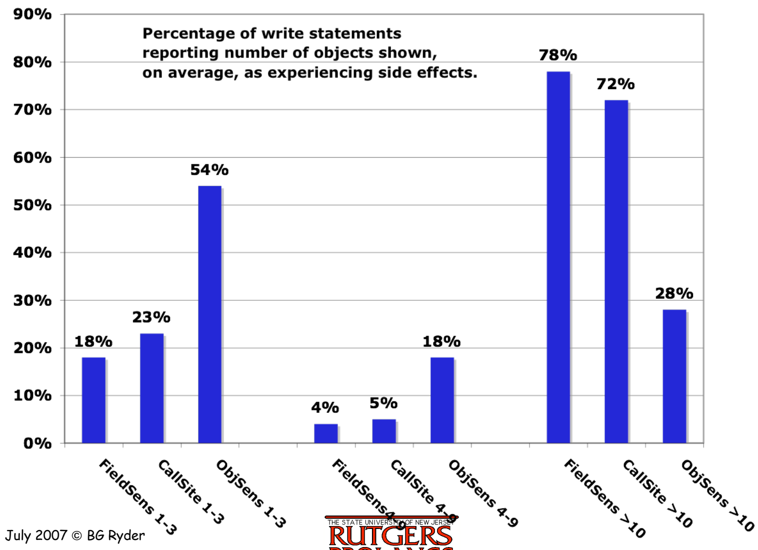
Side-effect Analysis: Modified Objects Per Statement

Milanova et.al, ISSTA'02



Side Effect Analysis Comparison

Milanova, et. al TOSEM05



ACACES-3 July 2007 © BG Ryder 13

Comparison ObjSens vs 1-CFA

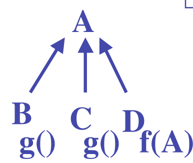
- The call string and functional approaches to context sensitivity are incomparable!
- Neither is more powerful than the other
- Recent papers cite ObjSens as better on clients: race detection and cast check elimination (Aiken et. al, PLDI'06; Lhotak & Hendren, CC'06)

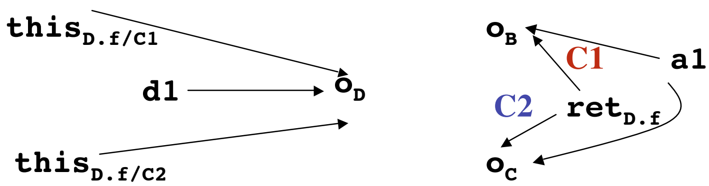
ACACES-3 July 2007 © BG Ryder 14

1-CFA more precise than ObjSens

1-CFA

```
static void main(){
 D d1 = new D();
 if (…)C1: (d1.f(new B())).g();
 else  C2: (d1.f(new C())).g();
}
public class D
{ public A f(A a1){return a1;}
}
```



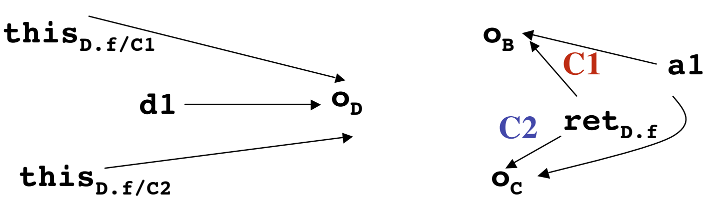


1-CFA more precise than ObjSens

1-CFA

```
static void main(){
 D d1 = new D();
 if (…)C1: (d1.f(new B())).g();
 else  C2: (d1.f(new C())).g();
}
public class D
{ public A f(A a1){return a1;}
}
```

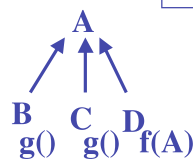
1-CFA distinguishes the two calling contexts of D.f at C1 and C2;
At C1, B.g() called;
At C2, C.g() called;

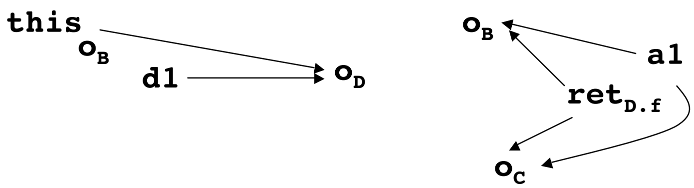


1-CFA more precise than ObjSens

ObjSens

```
static void main(){
 D d1 = new D();
 if (…)C1: (d1.f(new B())).g();
 else  C2: (d1.f(new C())).g();
}
public class D
{ public A f(A a1){return a1;}
}
```



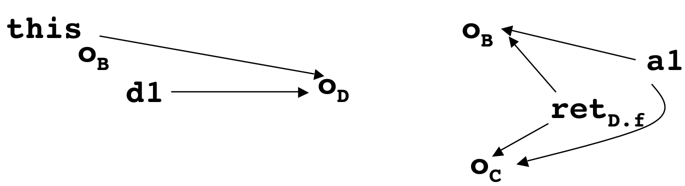


1-CFA more precise than ObjSens

ObjSens

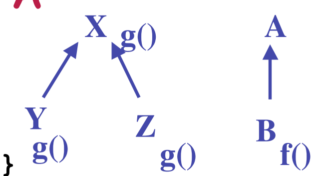
```
static void main(){
 D d1 = new D();
 if (…)C1: (d1.f(new B())).g();
 else  C2: (d1.f(new C())).g();
}
public class D
{ public A f(A a1){return a1;}
}
```

ObjSens groups the two calling contexts of D.f with the same receiver at C1 and C2;
Both B.g(),C.g() are called at C1 and C2;

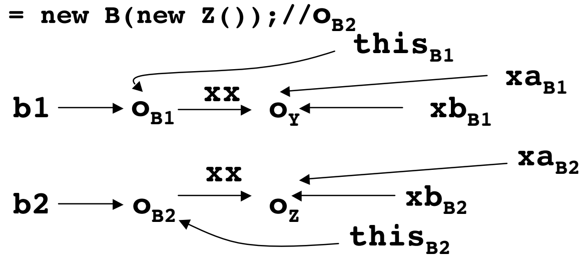


ObjSens more precise than 1-CFA

```
public class A
{ X xx;
  A (X xa){ this.xx=xa;}
}
public class B extends A
{ B (X xb){C3: super(xb);}
  public X f() {return this.xx;}
  static void main(){
    X x1,x2;
  C1: B b1 = new B(new Y());//oB1
  C2: B b2 = new B(new Z());//oB2
```



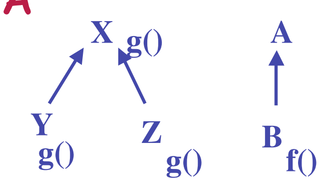
ObjSens

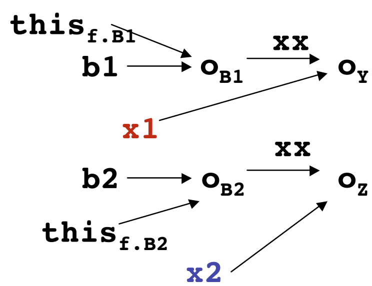


ObjSens more precise than 1-CFA

```
public class A
{ X xx;
  A (X xa){ this.xx=xa;}
}
public class B extends A
{ B (X xb){C3: super(xb);}
  public X f() {return this.xx;}
  static void main(){
   X x1,x2;
  C1: B b1 = new B(new Y());//oB1
  C2: B b2 = new B(new Z());//oB2
     x1=b1.f();
C4: x1.g();
     x2=b2.f();
C5: x2.g();
}
```

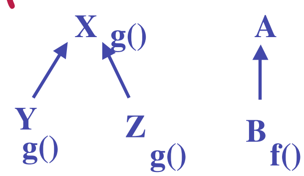
ObjSens





ObjSens more precise than 1-CFA

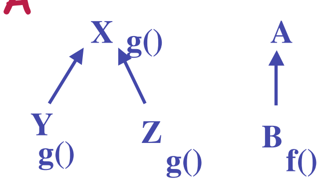
```
public class A
{ X xx;
  A (X xa){ this.xx=xa;}
}
public class B extends A
{ B (X xb){C3: super(xb);}
  public X f() {return this.xx;}
  static void main(){
   X x1,x2;
  C1: B b1 = new B(new Y());//oB1
  C2: B b2 = new B(new Z());//oB2
     x1=b1.f();
C4: x1.g();
     x2=b2.f();
C5: x2.g();
}
```



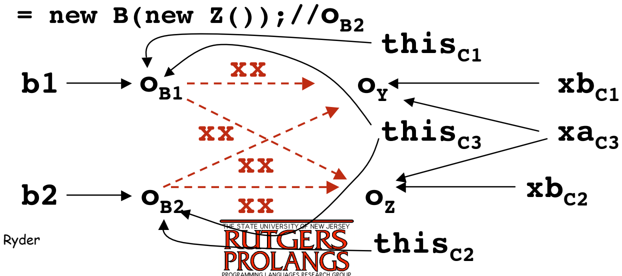
ObjSens finds
C4 calls Y.g() and
C5 calls Z.g()

ObjSens more precise than 1-CFA

```
public class A
{ X xx;
  A (X xa){ this.xx=xa;}
}
public class B extends A
{ B (X xb){C3: super(xb);}
  public X f() {return this.xx;}
  static void main(){
    X x1,x2;
  C1: B b1 = new B(new Y());//oB1
  C2: B b2 = new B(new Z());//oB2
```

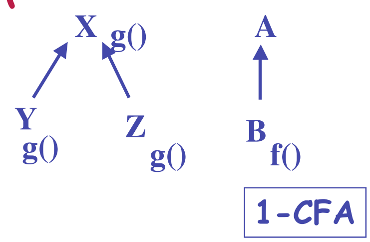


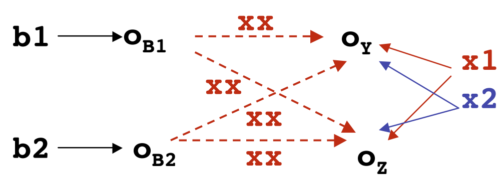
1-CFA



ObjSens more precise than 1-CFA

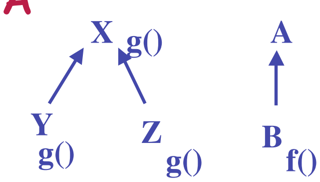
```
public class A
{ X xx;
  A (X xa){ this.xx=xa;}
}
public class B extends A
{ B (X xb){C3: super(xb);}
  public X f() {return this.xx;}
  static void main(){
   X x1,x2;
  C1: B b1 = new B(new Y());//oB1
  C2: B b2 = new B(new Z());//oB2
     x1=b1.f();
C4:  x1.g();
     x2=b2.f();
C5:  x2.g();
}
```





ObjSens more precise than 1-CFA

```
public class A
{ X xx;
  A (X xa){ this.xx=xa;}
}
public class B extends A
{ B (X xb){c3: super(xb);}
  public X f() {return this.xx;}
  static void main(){
   X x1,x2;
  C1: B b1 = new B(new Y());//oB1
  C2: B b2 = new B(new Z());//oB2
     x1=b1.f();
C4:  x1.g();
     x2=b2.f();
C5:  x2.g();
}
```



1-CFA finds
C4 calls Y.g(), Z.g() and
C5 calls Y.g(), Z.g()

Empirical Comparisons CC'06

- Reports on a comparison of 4 different context-sensitive analyses
  - Run on same 16 benchmarks
  - Implemented on the same framework (JEDD in Soot)
  - Combined with context-sensitive object naming schemes
  - Effectiveness measured on devirtualization, redundant cast removal, call graph size
- *Bottom line:* object-sensitive analysis shown to be superior, in terms of scalability and precision, on points-to analysis and cast elimination

"Context-sensitive analysis - Is it worth it?", O. Lhotak, L. Hendren, CC'06

Context-sensitive Points-to Algorithms in Study

Lhotak & Hendren, CC'06

- Informal algorithm is flow- and context-insensitive
- Call-site-string-based uses a string of the *k* most recent actual call sites on the runtime stack as the 'calling context'
- Receiver object-based (object-sensitive) uses the sequence of the *k* most recent receiver objects as the 'calling context'
- Cloning-based (with BDDs) actually makes one copy per method instantiation
  - Corresponding to call edges that DO NOT participate in a cycle in the context-insensitive call graph (ZCWL, PLDI'04)

Questions to answer

Lhotak & Hendren, CC'06

1. Which contexts are actually useful to improve analysis precision?
   - How often contexts have identical points-to info?
   - How much context can be saved for practical cost?
   - Does more context help precision?
2. Why can BBDs do so well in representing large numbers of contexts?
   - How poorly would non-BDD representations do for context-sensitive analyses?
3. How well do the algorithms do on client problems?
   - Call graph construction, devirtualization, unnecessary cast elimination

Findings - #Equiv Contexts

Lhotak & Hendren, CC'06

Benchmark	insens.	object-sensitive 1	object-sensitive 2	object-sensitive 3	object-sensitive 1H	call site 1	call site 2	call site 1H	ZCWL
compress	2597	8.4	9.9	11.3	12.1	2.4	3.9	4.9	3.3
db	2614	8.5	9.9	11.4	12.1	2.4	3.9	5.0	3.3
jack	2870	8.6	10.2	11.6	11.9	2.4	3.9	5.0	3.4
javac	3781	10.4	17.7	33.8	14.3	2.7	5.3	5.4	
jess	3217	8.9	10.6	12.0	13.9	2.6	4.2	5.0	3.9
mpegaudio	2794	8.1	9.4	10.8	11.5	2.4	3.8	4.8	3.3
mtrt	2739	8.3	9.7	11.1	11.8	2.5	4.0	4.9	3.4
soot-c	4838	7.1	13.7	18.4	9.8	2.6	4.2	4.8	
sablecc-j	5609	6.9	8.4	9.6	9.5	2.3	3.6	3.9	
polyglot	5617	7.9	9.4	10.8	10.2	2.4	3.7	4.7	3.3
antlr	3898	9.4	12.1	13.8	13.2	2.5	4.1	5.2	4.3
bloat	5238	10.2	44.6		12.9	2.8	4.9	5.2	6.7
chart	7070	10.0	17.4		18.2	2.7	4.8		
jython	4402	9.9	55.9		15.6	2.5	4.3	4.6	4.0
pmd	7220	7.6	14.6	17.0	11.0	2.4	4.2	4.2	
ps	3875	8.7	9.9	11.0	12.0	2.6	4.0	5.2	4.4

Table III: Number of equivalence classes of abstract contexts

#Distinct Points-to Sets

Lhotak & Hendren, CC'06

- Found fairly equivalent numbers of distinct points-to sets across all algorithms with all levels of context.
- Means the problem for a non-BDD solution procedure for context-sensitive analysis is not points-to set size, but rather how to efficiently store contexts.

Run-time Cast Checks Needed

Lhotak & Hendren, CC'06

		object-sensitive				call site			
Benchmark	insens.	1	2	3	1H	1	2	1H	ZCWL
compress	**18**	18	18	18	18	18	18	18	18
db	27	27	27	27	**21**	27	27	27	27
jack	146	145	145	145	**104**	146	145	146	146
javac	405	370	370	370	**363**	391	370	391	
jess	130	130	130	130	**86**	130	130	130	130
mpegaudio	42	**38**	38	38	38	40	40	40	42
mtrt	31	**27**	27	27	27	27	27	27	29
soot-c	955	932	932	932	**878**	932	932	932	
sablecc-j	375	369	369	369	**331**	370	370	370	
polyglot	3539	3307	3306	3306	**1017**	3526	3443	3526	3318
antlr	295	275	275	275	**237**	276	275	276	276
bloat	1241	1207	1207		**1160**	1233	1207	1233	1234
chart	1097	1086	1085		**934**	1070	1070		
jython	501	499	499		**471**	499	499	499	499
pmd	1427	1376	1375	1375	**1300**	1393	1391	1393	
ps	641	612	612	612	**421**	612	612	612	612

Difficult Issues

- Need a whole program for a safe analysis
  - For reflection and dynamic class loading must estimate possible effects
- Java native methods
  - Need to model possible effects
- Exceptions
  - Need to approximate possible control flow
- Incomplete programs (e.g., analyzing libraries)
- Lack of benchmarks for comparing analyses

Handling Dynamic Class Loading

- Dynamic class loading, reflection, native libraries present problems to whole-program analysis
- New algorithm incrementally accounts for classes loaded and *performs analysis updates online at runtime*
    - Generates constraints at runtime and propagates them when a client needs valid points-to results

M.Hirzel, A. Diwan, M. Hind, "Pointer Analysis in the Presence of Dynamic Class Loading", ECOOP 2004;
M. Hirzel, D. VanDincklage, A. Diwan, M. Hind, "Fast Online Pointer Analysis", ACM TOPLAS, April 2007.

Online Points-to Algorithm

Hirzel et al, ECOOP'04

- Andersen's analysis with field-sensitive object representation, objects represented by their creation sites, and static call graph (CHA)
- Two stages (can be iterated when get new constraints)
  - Constraint generation
  - Constraint propagation with type filtering (producing points-to sets through fixed-point iteration)
- Use CHA call graph (generated online) to get call edges
  - Process constraints from an edge only after have seen both source and target

Online Points-to Algorithm

Hirzel et al, ECOOP'04

- Uses deferred evaluation to handle unresolved references
  - From native code, reflection, JIT compilation of a method, type resolution, class loading, VM startup
- Handle reflection through instrumenting the JVM to add constraints dynamically
  - Need to re-propagate at runtime as new constraints are added
  - Use JVM to catch reflection and add appropriate constraints when it occurs
  - Native code with returned heap value assumed to return any allocated object
  - Initial prototype assumed that any exception throw could hit any catch

Online Points-to Algorithm

Hirzel et al, ECOOP'04

- Showed efficacy through use in new connectivity-based GC algorithm
    - Used Jikes RVM 2.2.1 on Specjvm98 benchmarks with good results; claimed need long-running programs for the incremental computation cost to be amortized
- Validation:
  - Need to make sure points-to solution is updated before do a GC.
  - Then GC verifies the points-to solution by making sure the dynamically observed points-to's are in the solution.

Dynamic Analysis of OOPLs

- Collection of full call traces
  - May also collect specific events such as object creations
  - Useful for debugging (e.g., slicing) and performance diagnosis
- Sampling for recognition of "hot methods"
  - Useful for online optimizations in JITs
    - Method inlining and specialization

Feedback Directed Optimization

- Commonly, JITs compile a method method at first use with fixed set of optimizations
- *Feedback directed optimization* (FDO) for longer-running applications
    - Profiling used to choose *what* and *how* to optimize
    - Offline profiles used since online profile collection often degraded performance due to cost of code instrumentation
  - ☹ Translation incurs runtime overhead
  - ☺ Allows compiler to make judgments using run-time information

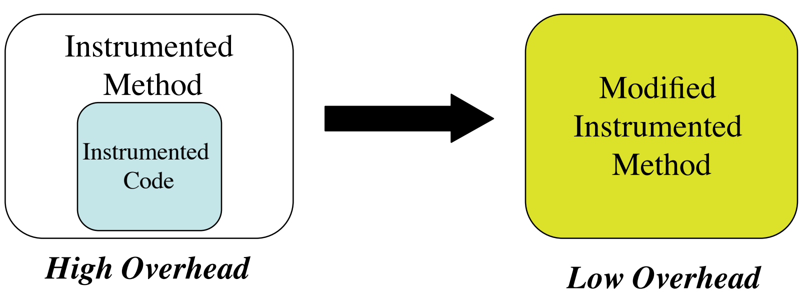
Problems with Online FDO

- What is *instrumentation?*
  - e.g., recording object field accesses, method calls
- Instrumentation overhead
  - Profiling interval must be short, but then may not be representative
  - Need a way to stop instrumented execution
    - Dynamic instrumentation
- ***General framework for instrumentation sampling and experiments with it.***

M. Arnold & B.G. Ryder, Reducing the Cost of Instrumenting Code Via Sampling, PLDI'01; M. Arnold, M. Hind, B.G. Ryder, Online Feedback-directed Optimization of Java, OOPSLA'02

Key Idea

Arnold & Ryder, PLDI'01
Arnold, Hind, Ryder, OOPSLA'02



Achieved through our new sampling framework, independent of architecture or operating system.

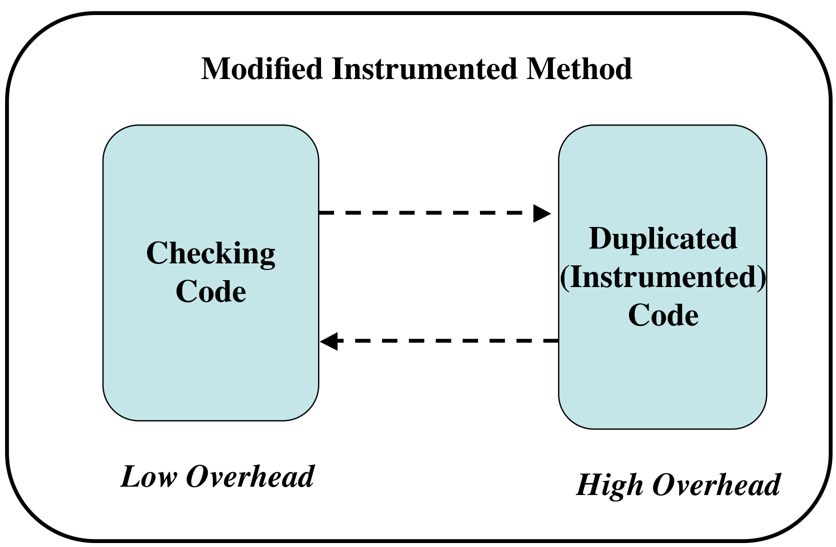
Advantages

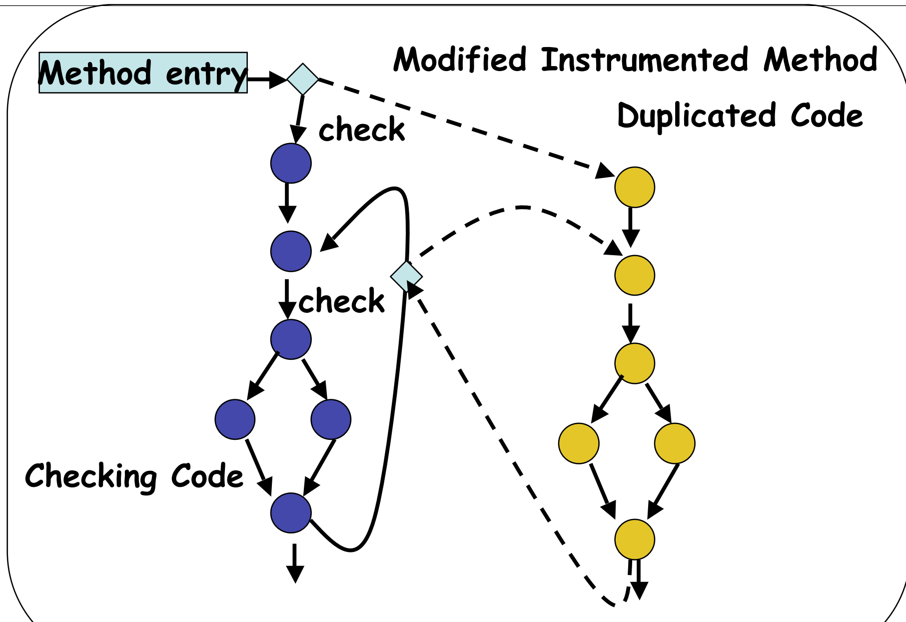
A low overhead sampling framework

- Instrumentation can be run longer for greater accuracy
- Can apply multiple instrumentations at same time without framework modification;
- Most instrumentation incorporated without modification
- Framework is *tunable* allowing tradeoffs between overhead and accuracy (i.e., adjustable sampling rates)
- *Deterministic sampling* simplifies debugging

Arnold & Ryder, PLDI'01
Arnold, Hind, Ryder, OOPSLA'02

Framework





Arnold & Ryder, PLDI'01
Arnold, Hind, Ryder, OOPSLA'02

Arnold & Ryder, PLDI'01
Arnold, Hind, Ryder, OOPSLA'02

Potential Disadvantages

- Code space may be doubled
  - VM will apply instrumentation selectively
    - Only in frequently executing methods
  - Designed other space-saving versions of framework
  - Empirical results show space usage is not a problem
- Sampled profile not same as exhaustive profile
  - Can't determine that an event DID NOT OCCUR
  - Can't check "for every iteration" assertions

Counter-based Sampling

- Take a sample after executing *n* checks
- Each check is:

```
globalCounter --;
If (globalCounter ==0) {
  takeSample();
  globalCounter = resetValue;
}
```

- Advantages
  - Simple, but effective
  - Hardware independent
  - Tunable, flexible sampling rate
  - Can be used with any VM

Arnold & Ryder, PLDI'01
Arnold, Hind, Ryder, OOPSLA'02

Framework Measurment

- Implemented in IBM's Jalapeno JVM
- 10 benchmarks
  - *SPECjvm98*(input size 10), *Volano, pBob, opt-compiler*
  - Running times from 1.1-4.8 seconds
  - Class file sizes from 10K-1,517K bytes
  - Machine 333Mz IBM RS/6000 powerPC 604e with 2096Mb RAM running AIX 4.3
- Instrumented all methods in applications and libraries

Arnold & Ryder, PLDI'01
Arnold, Hind, Ryder, OOPSLA'02

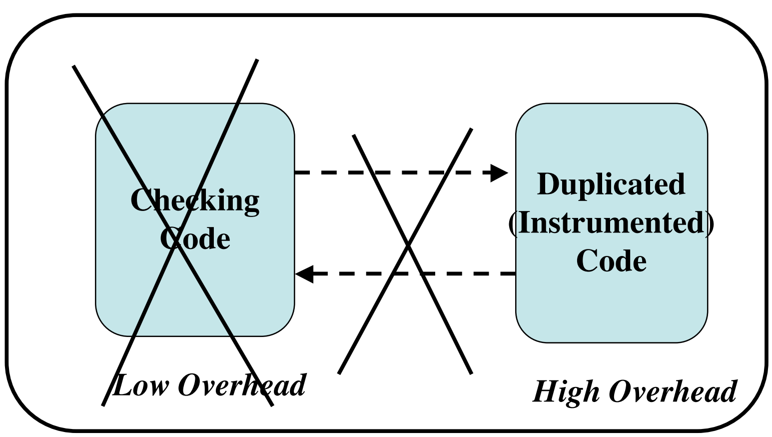
Instrumentation

- **Call-edge:**
  - Collect caller, callee, call-site within caller at method entry
  - One counter per call edge
- **Field-access:**
  - One counter per field of each class
  - Each *putfield, getfield* access instrumented

Arnold & Ryder, PLDI'01
Arnold, Hind, Ryder, OOPSLA'02

Exhaustive Instrumentation Overhead

On average, 88% call-edge and 60% field-access



Findings - #Contexts

Benchmark	insens.	object-sensitive 1	object-sensitive 2	object-sensitive 3	object-sensitive 1H	call site 1	call site 2	call site 1H	ZCWL
compress	2596	13.7	113	1517	13.4	6.5	237	6.5	2.9×10^4
db	2613	13.7	115	1555	13.4	6.5	236	6.5	7.9×10^4
jack	2869	13.8	156	1872	13.2	6.8	220	6.8	2.7×10^7
javac	3780	15.8	297	13289	15.6	8.4	244	8.4	
jess	3216	19.0	305	5394	18.6	6.7	207	6.7	6.1×10^6
mpegaudio	2793	13.0	107	1419	12.7	6.3	221	6.3	4.4×10^5
mtrt	2738	13.3	108	1447	13.1	6.6	226	6.6	1.2×10^5
soot-c	4837	11.1	168	4010	10.9	8.2	198	8.2	
sablecc-j	5608	10.8	116	1792	10.5	5.5	126	5.5	
polyglot	5616	11.7	149	2011	11.2	7.1	144	7.1	10130
antlr	3897	15.0	309	8110	14.7	9.6	191	9.6	4.8×10^9
bloat	5237	14.3	291		14.0	8.9	159	8.9	3.0×10^8
chart	7069	22.3	500		21.9	7.0	335		
jython	4401	18.8	384		18.3	6.7	162	6.7	2.1×10^{15}
pmd	7219	13.4	283	5607	12.9	6.6	239	6.6	
ps	3874	13.3	271	24967	13.1	9.0	224	9.0	2.0×10^8

Table II: Total number of abstract contexts

Findings - #Equiv Contexts

- Given <m1,c1> and <m1,c2>, if every local reference has same points-to set in these 2 contexts, they are *equivalent*
- Found many equivalent abstract contexts in the data
- In general, there are more equiv classes of contexts with ObjSens than with CallSite abstractions
  - Expect better precision from this
- In both ObjSens and CallSite, increasing k increases the #equiv classes only slightly while increasing the absolute #contexts significantly (little precision improvement for a large cost)
- #contexts of ZCWL is very small because of the merges on the large SCCs in the benchmark initial call graphs; effectively ZCWL models much of the call graph context-insensitively

"Context-sensitive analysis - Is it worth it?", O. Lhotak, L. Hendren, CC'06