

Dimensions of Precision in Reference Analysis of Object-oriented Programming Languages

Dr. Barbara G. Ryder
Rutgers University

<http://www.cs.rutgers.edu/~ryder>

<http://prolangs.rutgers.edu/>

Research supported, in part, by NSF CCR9900988

ETAPS 4/03, BG Ryder



1

Outline

- What is reference analysis of OOPLs and what is it used for?
- Examples of reference analyses
 - Hierarchy-based: CHA, RTA
 - Incorporating flow: FieldSens
 - Context-sensitive: 1-CFA
- Dimensions of analysis
 - More examples and categorization of analysis algorithm choices
- Conclusions

ETAPS 4/03, BG Ryder



2

Object-oriented PL

- Characterized by data abstraction, inheritance, polymorphism
- Allows dynamic binding of method calls
- Allows dynamic loading of classes
- Allows querying of program semantics at run-time through reflection

Reference Analysis

- *Determines information about the set of objects to which a reference variable or field may point during program execution*

Reference Analysis enables...

- **Construction of possible calling structure of program**
 - Dynamic dispatch of methods based on runtime type of receiver `x.f();`
- **Understanding of possible dataflow in program**
 - Indirect side effects through reference variables and fields `r.g=`

Uses of Reference Analysis Information in Compilers

- Side-effect analysis, dependence analysis (flow)
- Devirtualization & inlining (hierarchy)
- Synchronization removal (flow)
- Stack-based object allocation (flow)

Uses of Reference Analysis Information in Software Tools

- **Program understanding tools (flow)**
 - Semantic browsers
 - Program slicers
- **Software maintenance tools (hier, flow)**
 - Change impact analysis tools
- **Testing tools (flow)**
 - Coverage metrics

ETAPS 4/03, BG Ryder



7

Example Analyses

- **Hierarchy-based**
 - CHA, RTA
- **Incorporating flow**
 - FieldSens (Andersen-based points-to)
- **Incorporating flow and calling context**
 - 1-CFA

ETAPS 4/03, BG Ryder



8

Example

cf Frank Tip, OOPSLA'00

```

static void main(){
  B b1 = new B();
  A a1 = new A();
  f(b1);
  g(b1);
}

static void f(A a2){
  a2.foo();
}

static void g(B b2){
  B b3 = b2;
  b3 = new C();
  b3.foo();
}

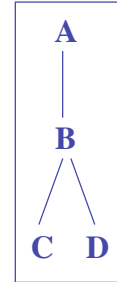
class A {
  foo(){..}
}

class B extends A{
  foo() {...}
}

class C extends B{
  foo() {...}
}

class D extends B{
  foo() {...}
}

```



ETAPS 4/03, BG Ryder



9

CHA Example

cf Frank Tip, OOPSLA'00

```

static void main(){
  B b1 = new B();
  A a1 = new A();
  f(b1);
  g(b1);
}

static void f(A a2){
  a2.foo();
}

static void g(B b2){
  B b3 = b2;
  b3 = new C();
  b3.foo();
}

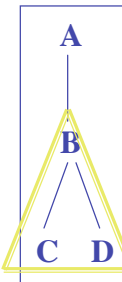
class A {
  foo(){..}
}

class B extends A{
  foo() {...}
}

class C extends B{
  foo() {...}
}

class D extends B{
  foo() {...}
}

```



ETAPS 4/03, BG Ryder

Cone(Declared_type(receiver))



10

CHA Characteristics

- Ignores program flow
- Calculates types that a reference variable can point to
- Uses 1 abstract reference variable per class throughout program
- Uses 1 abstract object to represent all possible instantiations of a class

J. Dean, D. Grove, C. Chambers, *Optimization of OO Programs Using Static Class Hierarchy*, ECOOP'95

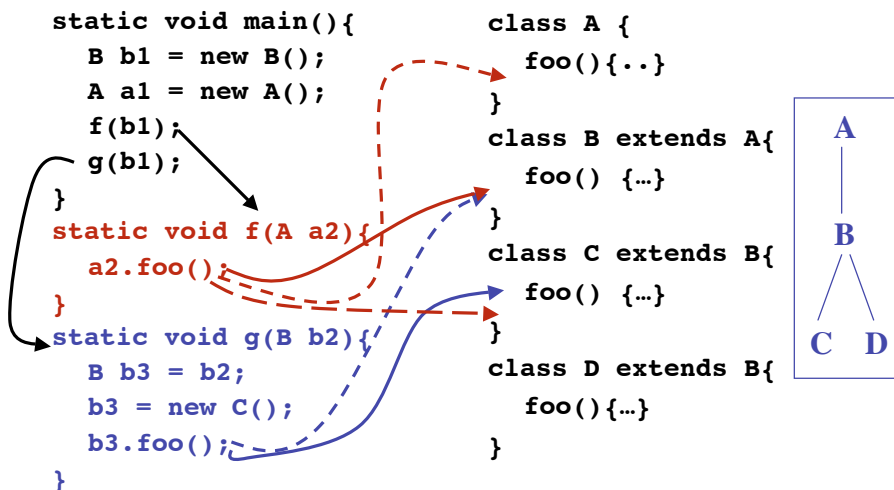
ETAPS 4/03, BG Ryder



11

RTA Example

cf Frank Tip, OOPSLA'00



ETAPS 4/03, BG Ryder



12

RTA Characteristics

- Only analyzes methods *reachable* from `main()`, on-the-fly
- Ignores classes which have not been instantiated as possible receiver types
- Uses 1 abstract reference variable per class throughout program
- Uses 1 abstract object to represent all possible instantiations of a class

ETAPS 4/03, BG Ryder

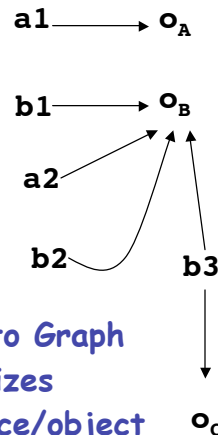
D. Bacon and P. Sweeney, "Fast Static Analysis of C++ Virtual Function Calls", OOPSLA'96



13

FieldSens Example

```
static void main(){
  B b1 = new B();
  A a1 = new A();
  f(b1);
  g(b1);
}
static void f(A a2){
  a2.foo();
}
static void g(B b2){
  B b3 = b2;
  b3 = new C();
  b3.foo();
}
```



Points-to Graph
summarizes
reference/object
relationships

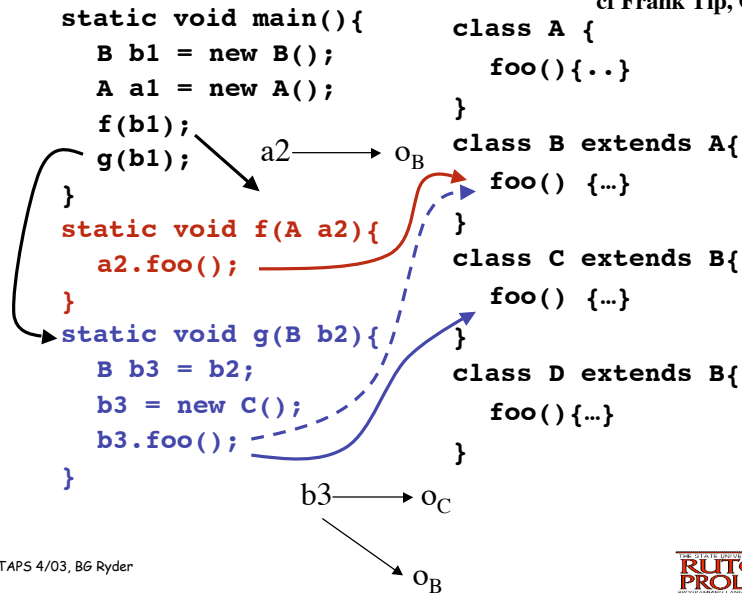
ETAPS 4/03, BG Ryder



14

FieldSens Example

cf Frank Tip, OOPSLA'00



ETAPS 4/03, BG Ryder



15

FieldSens Characteristics

- 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

Rountev, A. Milnova, B. Ryder, "Points-to Analysis for Java Using Annotated Constraints" OOPSLA'01

ETAPS 4/03, BG Ryder



16

1-CFA Analysis

- Improves on FieldSens by keeping track of calling context

```
static void main(){
```

```
    B b1 = new B();//OB
```

```
    A a1 = new A();//OA
```

```
    A a2,a3;
```

```
    C1: a2 = f(b1);
```

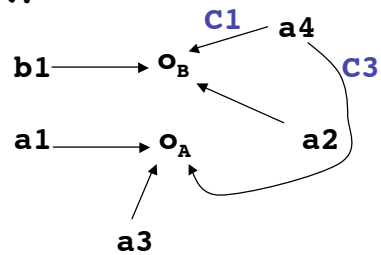
```
    C2: a2.foo();
```

```
    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()

ETAPS 4/03, BG Ryder

RUTGERS
PROLANGS 17

1-CFA Characteristics

- 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

ETAPS 4/03, BG Ryder

RUTGERS
PROLANGS 18

Dimensions of Precision

- Independent characteristics of a reference analysis which determines its precision
- Different combinations of these dimensions have already been explored in algorithms
- Need to understand what choices are available to design new analyses of appropriate precision for clients

ETAPS 4/03, BG Ryder



19

Dimensions of Precision

- **Program representation - Call graph**
 - Use hierarchy-based approximation
 - Do reference analysis based on an already built approximate call graph - Palsberg'91, Chatterjee'99, Sundaresan'00, Liang'01
 - Lazy on-the-fly construction
 - Only explore methods which are statically reachable from the main() (especially library methods)
 - Interleave reference analysis and call graph construction - Oxhoj'92, Razafimahefa'99, Rountev'01, Grove'01, Liang'01, Milanova'02, Whaley'02

ETAPS 4/03, BG Ryder



20

Dimensions of Precision

• Object Representation

- Use one abstract object per class -
Hierarchy-based analyses: Dean'95, Bacon'96; Flow-based analyses:
Diwan'96, Palsberg'91, Sundaresan'00, Tip'00
- Group object instantiations by creation
site - Points-to analyses: Grove'01, Liang'01, Rountev'01,
Milanova'02, Whaley'02
- Finer-grained object naming - Oxhoj'92,
Plevyak'94, Grove'01, Liang'02, Milanova'02

• Field Sensitivity

- May or may not distinguish fields - Rountev'01

ETAPS 4/03, BG Ryder



21

Dimensions of Precision

• Reference representation

- Use one abstract reference per class - Dean'95,
Bacon'96, Sundaresan'00
- Use one abstract reference for each class per
method - Tip'00
- Represent reference variables or fields by their
names program-wide - Sundaresan'00, Liang'01, Rountev'01,
Milanova'02
- **XTA**: example of one abstract reference
for each class per method

ETAPS 4/03, BG Ryder



22

XTA Analysis

- Calculates set of classes that reach a method, incorporating (limited) flow
- Uses an on-the-fly constructed call graph
- Uses one abstract object per class with distinct fields
- Uses one abstract reference per class in each method

Tip and Palsberg, "Scalable Propagation-based Call Graph Construction Algorithms", OOPSLA'00

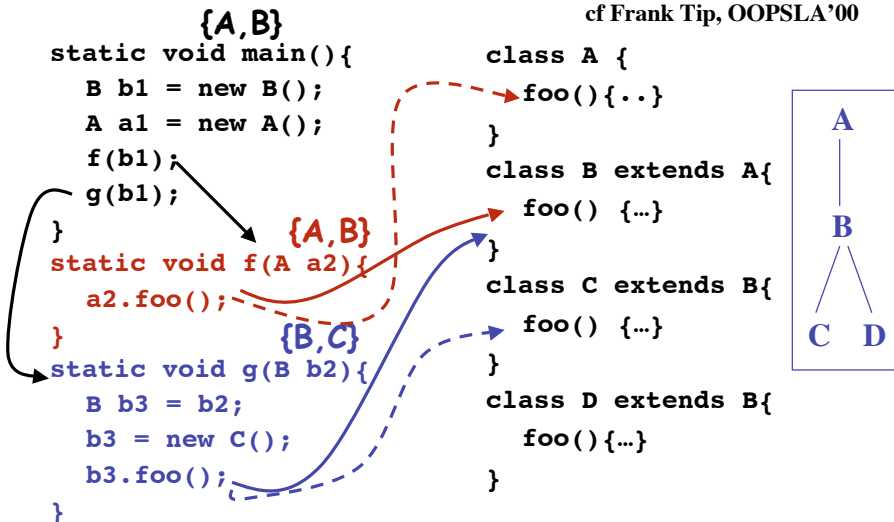
ETAPS 4/03, BG Ryder



23

Example of XTA

cf Frank Tip, OOPSLA'00



ETAPS 4/03, BG Ryder



24

Dimensions of Precision

• Directionality

- How flow in reference assignments ($r=s$) is interpreted by the analysis
 - Symmetric (Unification): r and s have same points-to set after the assignment - Ruf'00, Liang'01
 - Directional (Inclusion): r 's points-to set includes s 's points-to set after the assignment - Sundarassen'00, Rountev'01, Liang'01, Milanova'02, Whaley'02

ETAPS 4/03, BG Ryder



25

Dimensions of Precision

• Flow sensitivity

- Analyses which capture the sequential order of execution of program statements - Diwan'96, Chatterjee'99, Whaley'02

- E.G.,

1. A s, t;
2. s = new A(); //o ₁
3. t = s;
4. s = new A(); //o ₂

flow-sensitive:

at 2., s refers to o_1
at 3., s, t refer to o_1
at 4., s refers to o_2
 t refers to o_1

flow-insensitive:

s, t refer to $\{o_1, o_2\}$

ETAPS 4/03, BG Ryder



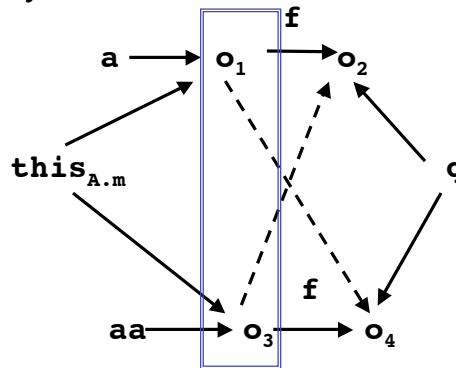
26

Imprecision of Context Insensitivity

```
class Y extends X{ ... }
```

```
class A{
  X f;
  void m(X q)
  { this.f=q; }
}
```

```
A a = new A(); //o1
a.m(new X()); //o2
A aa = new A(); //o3
aa.m(new Y()); //o4
```



ETAPS 4/03, BG Ryder

RUTGERS
PROLANGS

27

Dimensions of Precision

- **Context sensitivity**
 - Analyses which distinguish different calling contexts - Sharir/Pnueli'81
 - Call string - Palsberg'91, Grove'01
 - Functional approach- Plevyak'94, Agesen'95, Milanova'02
 - **1-CFA**, example of call string approach
 - **ObjSens**, example of functional approach

ETAPS 4/03, BG Ryder

RUTGERS
PROLANGS

28

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

Milanova, Rountev, Ryder, "Parameterized Object-sensitivity for Points-to and Side-effect Analyses for Java" ISSTA'02

ETAPS 4/03, BG Ryder

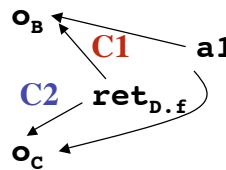
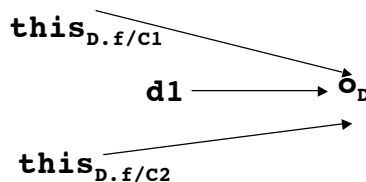
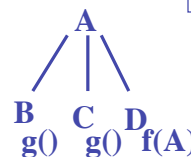


29

1-CFA more precise than 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



ETAPS 4/03, BG Ryder



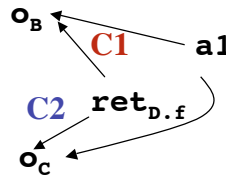
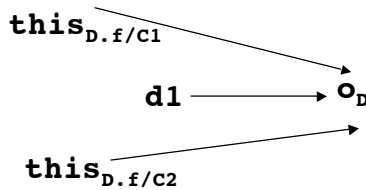
30

1-CFA more precise than 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

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



ETAPS 4/03, BG Ryder

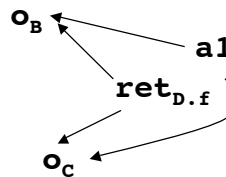
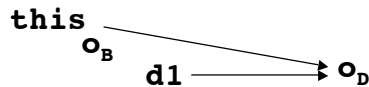
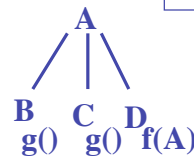
RUTGERS
PROLANGS

31

1-CFA more precise than 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



ETAPS 4/03, BG Ryder

RUTGERS
PROLANGS

32

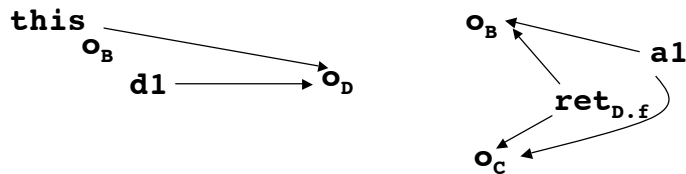
1-CFA more precise than ObjSens

```

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

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;



ETAPS 4/03, BG Ryder

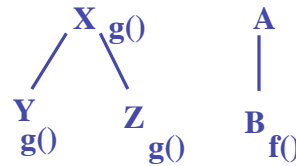


33

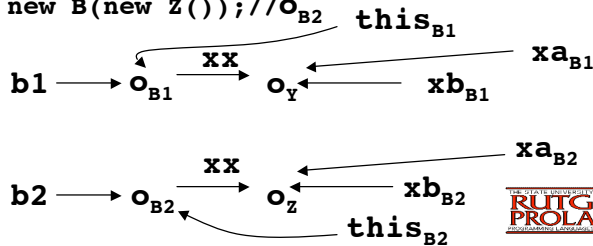
ObjSens more precise than 1-CFA

```

public class A
{ X xx;
  A (X xa){ this.xx=xa;}
}
public class B
{ 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()); // o_{B1}
    C2: B b2 = new B(new Z()); // o_{B2}
  }
}
    
```



ObjSens



ETAPS 4/03, BG Ryder



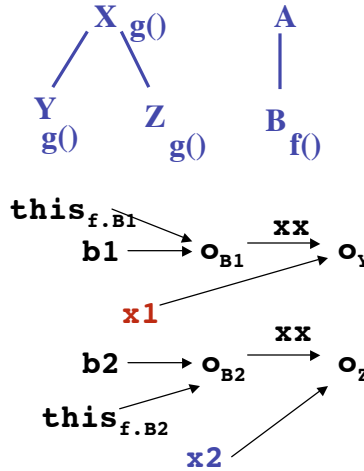
34

ObjSens more precise than 1-CFA

```

public class A
{ X xx;
  A (X xa){ this.xx=xa;}
}
public class B
{ 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

ETAPS 4/03, BG Ryder

RUTGERS
PROLANGS

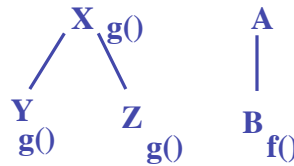
35

ObjSens more precise than 1-CFA

```

public class A
{ X xx;
  A (X xa){ this.xx=xa;}
}
public class B
{ 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()

ETAPS 4/03, BG Ryder

RUTGERS
PROLANGS

36

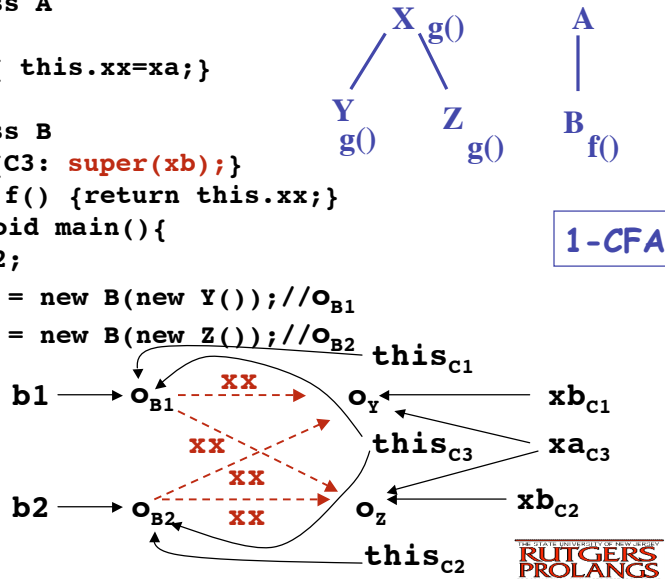
ObjSens more precise than 1-CFA

```
public class A
{ X xx;
  A (X xa){ this.xx=xa;}
}
```

```
public class B
{ 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
```



ETAPS 4/03, BG Ryder

RUTGERS
PROLANGS

37

ObjSens more precise than 1-CFA

```
public class A
{ X xx;
  A (X xa){ this.xx=xa;}
}
```

```
public class B
{ 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
```

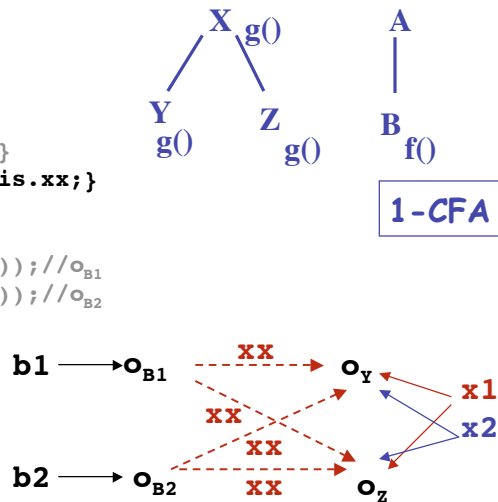
```
x1=b1.f();
```

```
C4: x1.g();
```

```
x2=b2.f();
```

```
C5: x2.g();
```

```
}
```



ETAPS 4/03, BG Ryder

RUTGERS
PROLANGS

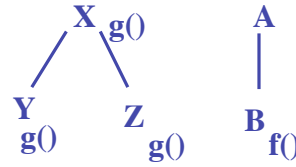
38

ObjSens more precise than 1-CFA

```

public class A
{ X xx;
  A (X xa){ this.xx=xa;}
}
public class B
{ 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()

Comparison Conclusion

- The call string and functional approaches to context sensitivity are incomparable!
- Neither is more powerful than the other

Open Issues

- Reflection
- Java native methods
- Exceptions
- Dynamic class loading
- Incomplete programs
- Benchmarks

Summary

- Challenge in reference analysis of OOPLs is to match the right analysis to a specific client
 - Experimentation with analysis clients is necessary to validate good analysis
- Good analyses for call graph construction exist
- Analyses capable of finding data dependences are being developed now
 - Context sensitivity may give significant precision improvement for acceptable cost

Summary

- **Dimensions of precision define the design space of new reference analyses**
 - Program, object (w/wo fields) and reference representations
 - Directionality
 - Flow and context sensitivity
- **Given the use of method calls as a primitive in OOPLs, context sensitivity should be exploited to add analysis precision**

Thank You

References

- O. Agesen. *The cartesian product algorithm: Simple and precise type inference of parametric polymorphism*. In European Conference on Object Oriented Programming, pages 2–26, 1995
- L. O. Andersen. *Program analysis and specialization for the C programming language*. PhD thesis, DIKU, University of Copenhagen, 1994. Also available as DIKU report 94/19.
- D. Bacon and P. Sweeney, *Fast Static Analysis of C++ Virtual Function Calls*, OOPSLA'96
- R. Chatterjee. *Modular Data-flow Analysis of Statically Typed Object-oriented Programming Languages*. PhD thesis, Department of Computer Science, Rutgers University, October 1999.
- R. Chatterjee, B. G. Ryder, and W. Landi. *Relevant context inference*. In Symposium on Principles of Programming Languages, pages 133–146, 1999.
- J. Dean, D. Grove, C. Chambers, *Optimization of OO Programs Using Static Class Hierarchy*, ECOOP'95

References

- D. Grove and C. Chambers. *A framework for call graph construction algorithms*. ACM Transactions on Programming Languages and Systems (TOPLAS), 23(6), 2001.
- D. Liang, M. Pennings, and M.J. Harrold. *Evaluating the precision of static reference analysis using profiling*. In Proceedings of the international symposium on Software testing and analysis, pages 22–32. ACM Press, 2002
- A. Milanova, A. Rountev, and B. G. Ryder. *Parameterized object-sensitivity for points-to and side-effect analyses for Java*. In International Symposium on Software Testing and Analysis, pages 1–11, 2002.
- N. Oxhoj, J. Palsberg, and M. Schwartzbach. *Making type inference practical*. In European Conference on Object-Oriented Programming, pages 329–349, 1992.
- J. Palsberg and M. Schwartzbach. *Object-oriented type inference*. In Conference on Object-Oriented Programming Systems, Languages, and Applications, pages 146–161, 1991.
- J. Plevyak and A. Chien. *Precise concrete type inference for object oriented languages*. In Proceeding of Conference on Object-Oriented Programming Systems, Languages and Applications.(OOPSLA '94), pages 324–340, October 1994.

References

- A. Rountev, A. Milnova, B. Ryder, *Points-to Analysis for Java Using Annotated Constraints*, OOPSLA'01
- E. Ruf. Effective synchronization removal for Java. In *Conference on Programming Language Design and Implementation*, pages 208–218, 2000.
- M. Sharir and A. Pnueli. *Two approaches to interprocedural data flow analysis*. In S. Muchnick and N. Jones, editors, *Program Flow Analysis: Theory and Applications*, pages 189–234, Prentice Hall, 1981.
- B. Steensgaard. *Points-to analysis in almost linear time*. In *Conference Record of the Twenty-third Annual ACM SIGACT/SIGPLAN Symposium on Principles of Programming Languages*, pages 32–41, 1996.
- V. Sundaresan, L. Hendren, C. Razafimahefa, R. Vallee-Rai, P. Lam, E. Gagnon, and C. Godin., *Practical Virtual Method Call Resolution for Java*, OOPSLA'00
- F. Tip and J. Palsberg. *Scalable Propagation-based Call Graph Construction Algorithms*, OOPSLA'00
- J. Whaley and M. Lam. *An efficient inclusion-based points-to analysis for strictly-typed languages*. In *Static Analysis Symposium*, 2002.