Evaluating the Impact of Context-Sensitivity on Andersen's Algorithm for Java Programs

Donglin Liang
University of Minnesota
UMSEC University of Minnesota Software Engineering Center

Collaborate with Maikel Pennings and Mary Jean Harrold

Overview

Andersen's algorithm
- How it works
- Why it is imprecise

A context-sensitive extension to Andersen's algorithm
- Call-string contexts
- Receiver contexts

Implementation and empirical studies
- Comparing the information computed by different algorithms with that collected during the execution

Andersen's Algorithm

- One points-to graph for the whole program
- Flow-insensitive and context-insensitive
  - flow-insensitive
    - process statements in an arbitrary order
  - context-insensitive
    - name objects by allocation sites
    - simulate parameter passing and method return using assignments

An Example

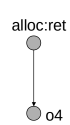
```
class A {
  Object f;
  Object alloc() {
      return new Object();      →  alloc:ret= o4;
  }
  A() {
    this.f = this.alloc();      →  alloc:this = A:this;
                                    A:this.f = alloc:ret;
  }
  static main() {
    A a1 = new A();             →  A:this=o10; a1=o10;
    A a2 = new A();             →  A:this=o11; a2=o11;
    A p = a2.alloc();           →  alloc:this=a2; p =alloc:ret;
  }
}
```


An Example

$\boxed{\text{alloc:ret= o4;}}$

alloc:this = A:this;
A:this.f = alloc:ret;

A:this=o10; a1=o10;
A:this=o11; a2=o11;
alloc:this=a2; p =alloc:ret;

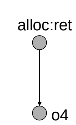


An Example

alloc:ret= o4;

$\boxed{\text{alloc:this = A:this; A:this.f = alloc:ret;}}$

A:this=o10; a1=o10;
A:this=o11; a2=o11;
alloc:this=a2; p =alloc:ret;

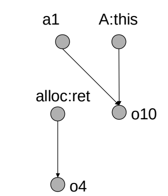


An Example

alloc:ret= o4;

alloc:this = A:this;
A:this.f = alloc:ret;

A:this=o10; a1=o10;
A:this=o11; a2=o11;
alloc:this=a2; p =alloc:ret;

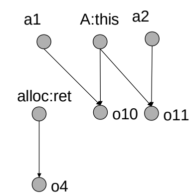


An Example

alloc:ret= o4;

alloc:this = A:this;
A:this.f = alloc:ret;

A:this=o10; a1=o10;
A:this=o11; a2=o11;
alloc:this=a2; p =alloc:ret;

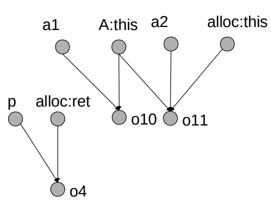


An Example

alloc:ret= o4;

alloc:this = A:this;
A:this.f = alloc:ret;

A:this=o10; a1=o10;
A:this=o11; a2=o11;
alloc:this=a2; p =alloc:ret;

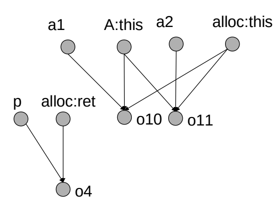


An Example

alloc:ret= o4;

alloc:this = A:this;
A:this.f = alloc:ret;

A:this=o10; a1=o10;
A:this=o11; a2=o11;
alloc:this=a2; p =alloc:ret;

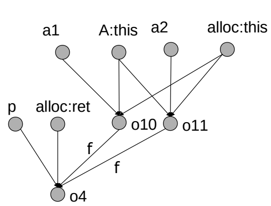


An Example

alloc:ret= o4;

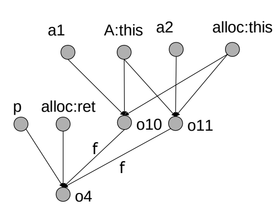
alloc:this = A:this;
A:this.f = alloc:ret;

A:this=o10; a1=o10;
A:this=o11; a2=o11;
alloc:this=a2; p =alloc:ret;



Imprecise Results

```
class A {
  Object f;
  Object alloc() {
      return new Object();
  }
  A() {
     this.f = this.alloc();
  }
  static main() {
     A a1 = new A();
     A a2 = new A();
     A p = a2.alloc();
  }
}
```



A Reason for Imprecision: Context-Insensitivity

```
1 class A {
2    Object f;
3    Object alloc() {
4         return new Object();  ->  alloc:ret= o4;
5    }
6    A() {
7       this.f = this.alloc();  ->  alloc:this = A:this;
8    }                              A:this.f = alloc:ret;
9    static main() {
10      A a1 = new A();         ->  A:this=o10; a1=o10;
11      A a2 = new A();         ->  A:this=o11; a2=o11;
12      A p = a2.alloc();       ->  alloc:this=a2; p =alloc:ret;
13  }
14 }
```


Overview

Andersen's algorithm
- How it works
- Why it is imprecise

A context-sensitive extension to Andersen's algorithm
- Call-string contexts
- Receiver contexts

Implementation and empirical studies
- Comparing the information computed by different algorithms with that collected during the execution

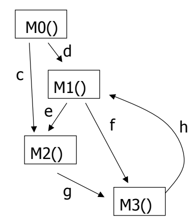
A Context-sensitive Extension to Andersen's Algorithm

- Start the analysis from "main()"
- During the analysis, iteratively
  - Discover the contexts under which a method may be invoked in the program
  - Analyze each method specifically under each context
    - Clone the local variables and parameters
    - Clone the assignments
    - Name the instances allocated in the method by annotating the allocation site with the context

Challenge: There may be intractable (or even infinite) number of possible contexts.

Abstracting the Contexts: a traditional approach

- Bounded call-strings
  - Using the string of statement numbers of the k top-most callsites
    - Level-2 call-string contexts for M3()
      - "cg", "eg", "df", "hf"



Abstracting the Contexts: an "object-oriented" approach

- Receiver contexts
  - Identify the contexts with the string names that identifies the receiver instances
    - The receiver contexts for alloc() would be "o10" and "o11"

```
1 class A {
2    Object f;
3    Object alloc() {
4         return new Object();
5    }
6    A() {
7       this.f = this.alloc();
8    }
9    static main() {
10      A a1 = new A();
11      A a2 = new A();
12      A p = a2.alloc();
13  }
14 }
```

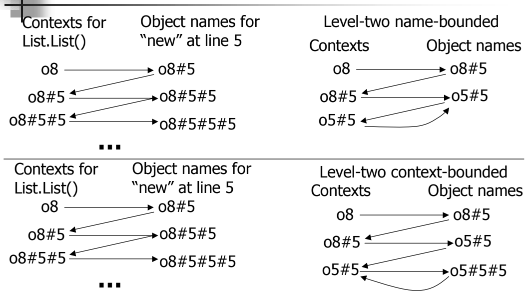

Growing Object Names and Receiver Contexts

- Receiver contexts come from object names
- Object names are created by annotating the allocation sites with receiver contexts

```
1 class List {
2   List next;
3   List(int i) {
4     if( i>0 )
5       next = new List(i-1);
6   }
7   static main() {
8     List p = new List(100);
9   }
10 }
```

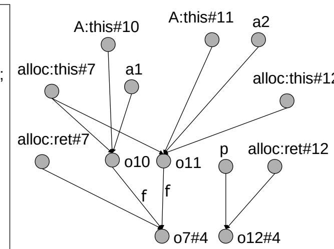
Contexts for List.List()	Object names for "new" at line 5
o8	o8#5
o8#5	o8#5#5
o8#5#5	o8#5#5#5
…	

Bounding Receiver Contexts



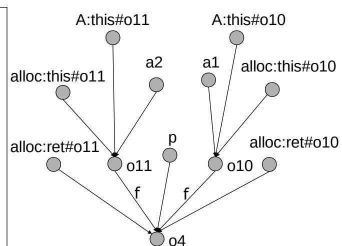
Using Level-1 Call-string Contexts

```
1 class A {
2   Object f;
3   Object alloc() {
4      return new Object();
5   }
6   A() {
7      this.f = this.alloc();
8   }
9   static main() {
10    A a1 = new A();
11    A a2 = new A();
12    A p = a2.alloc();
13  }
14 }
```



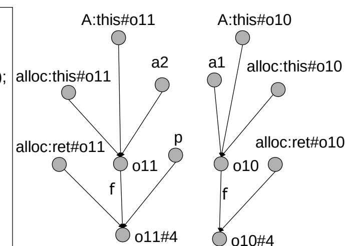
Using Level-1 Name-Bounded Receiver Contexts

```
1 class A {
2   Object f;
3   Object alloc() {
4      return new Object();
5   }
6   A() {
7      this.f = this.alloc();
8   }
9   static main() {
10    A a1 = new A();
11    A a2 = new A();
12    A p = a2.alloc();
13  }
14 }
```



Using Level-1 Context-Bounded Receiver Contexts

```
1 class A {
2   Object f;
3   Object alloc() {
4      return new Object();
5   }
6   A() {
7      this.f = this.alloc();
8   }
9   static main() {
10    A a1 = new A();
11    A a2 = new A();
12    A p = a2.alloc();
13  }
14 }
```



Overview

Andersen's algorithm
- How it works
- Why it is imprecise

A context-sensitive extension to Andersen's algorithm
- Call-string contexts
- Receiver contexts

Implementation and empirical studies
- Comparing the information computed by different algorithms with that collected during the execution

Questions to Answer about the Context-Sensitive Andersen's Algorithms

- How does a context-sensitive version of the algorithm compare with the context-insensitive version?
- How does the calling-string context approach compare with the receiver context approach?
- How would increasing the allowing size of contexts affect the performance?
- How would the client analyses benefit from the context-sensitive reference information?

How to design a reference analysis algorithm for Java that achieves the best trade-off between precision and efficiency?

Challenges in Evaluating Reference Analysis Algorithms

- No unique metrics
- No absolutely precise information to compare with
- No well-accepted benchmarks

An additional challenge for our studies

Results of different algorithms not directly comparable:
- The instances are identified with different naming schemes.

25

Our Evaluation Approach

- Compare the static results with the dynamic information
  - Static results: computed by each algorithm
    - Provide names for each possible instance
    - Estimate how a potential instance identified by a name may be used at the callsites
  - Dynamic info: collected by a profiler
    - Provide info for mapping an instance to a static name
    - Record how a real instance is actually used at the callsites
- Measure the accuracy in estimating how each instance will be used
  - Compute a precision reference value (PRV) for each instance

26

Computing PRVs

$C[i]$ = {callsites recorded for instance i by profiler}

$C_A[n]$ = {callsites recorded for name n according to the info computed by algorithm A}

$Cover$ = {callsites hit at least once during some execution}

If n is the name for i in A, then the PRV for i according to A is

$$r_A[i] = \frac{|C[i]|}{|C_A[n] \cap Cover|}$$

27

An Example

```
...
10 foo(int flag) {
11    A p = new A(); // o11
12    if( flag> 0 )
13        p.m1();
15    p.m2();
17 }
```

Static info:

C[o11]={11,13,15}

Assuming that instance 100 is created by 11 when flag is >0 and instance 200 is created by 11 when flag is =0

C[100]={11,13,15}

C[200]={11,15}

r[100]=3/3=1.0

r[200]=2/3=0.66

28

Settings

- Algorithms to evaluate
  - Without using contexts (CI)
  - Levels 1, 2, 3 call-string (CS1,CS2,CS3) contexts
  - Level 3 call-string contexts but without using contexts for naming objects (SemiCS3)
  - Levels 1, 2 name-bounded (NB1,NB2) or context-bounded (CB1,CB2) receiver contexts
- Implementation: focus attention on application code
  - Use models to avoid analyzing the library code
- Subjects: 12 Java programs collected from various sources

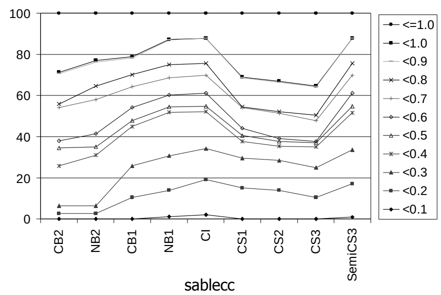
29

	Subject Size			*Method Coverage*		
program	*Locs†*	*Cls*	*Meth*	*And*	*Dyn*	*%*
antlr	32030	148	1858	950	682	72%
jar	1839	8	89	36	24	67%
jas	4620	116	413	295	264	89%
java_cup	10155	35	372	269	206	77%
jb	6025	45	548	184	100	54%
jess	19317	207	1132	974	757	78%
jfe	31636	310	1837	830	726	87%
jlex	7351	20	134	105	92	88%
jtar	11904	40	202	147	97	66%
raja	6369	65	391	37	32	86%
sablecc	44521	295	2025	1598	1376	86%
toba	6417	26	196	150	105	70%

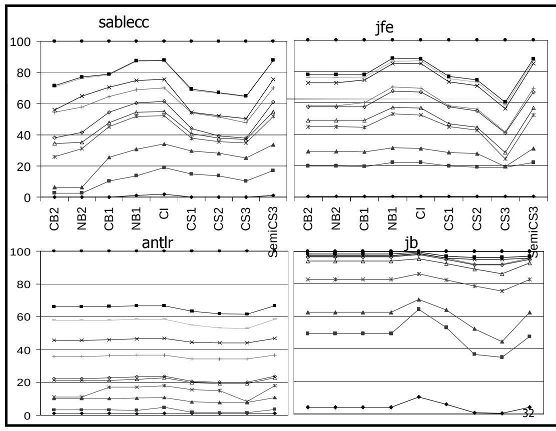
30

Comparing the PRVs for Instances

Percentage of instances whose PRVs are below a certain value



31



32

Compare Average PRVs for Allocation Sites

I_a = { Instances allocated at allocation site a}

Average PRV for a: $\bar{r}_A[a] = \dfrac{\sum_{i \in I_a} r_A[i]}{|I_a|}$

Compare two algorithms A and B:

$$Diff_{A/B}[a] = \bar{r}_A[a] - \bar{r}_B[a]$$

33

The distribution of $Diff_{A/B}[]$

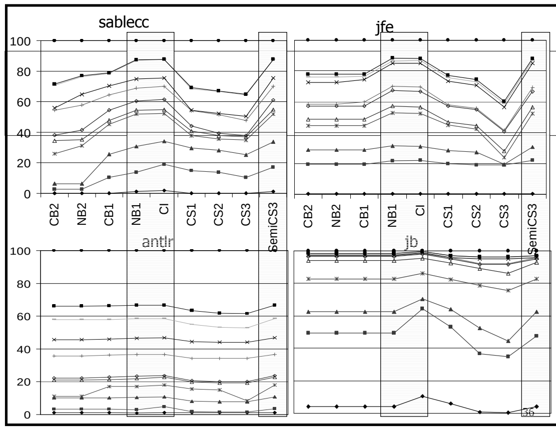
program	*Diff*	(0, 1.0)	(0, 0.1)	[0.1, 0.2)	[0.2, 0.3)	[0.3, 0.4)	[0.4, 0.5)	[0.5, 0.6)	[0.6, 0.7)	[0.7, 0.8)	[0.8, 0.9)	[0.9, 1.0)
antlr(254)	CS3/CI	19	5	5	1	3	1	1	1	–	2	–
	CB2/CI	26	14	6	2	2	1	–	–	–	1	–
	CS3/CB2	8	–	3	1	1	1	1	–	–	1	–
	CB2/CS3	18	14	3	1	–	–	–	–	–	–	–
jb(46)	CS3/CI	21	2	11	5	1	–	–	–	1	–	1
	CB2/CI	21	7	10	3	–	–	–	–	–	–	1
	CS3/CB2	10	5	3	–	1	–	–	–	1	–	–
	CB2/CS3	7	7	–	–	–	–	–	–	–	–	–
jess(386)	CS1/CI	33	25	3	1	–	1	1	1	1	–	–
	CB1/CI	41	22	12	4	–	1	1	–	1	–	–
	CS1/CB1	19	16	3	–	–	–	–	–	–	–	–
	CB1/CS1	25	13	8	4	–	–	–	–	–	–	–
jfe(284)	CS3/CI	80	22	8	3	7	13	6	9	7	4	1
	CB2/CI	87	38	10	6	7	8	4	5	5	1	3
	CS3/CB2	28	9	5	–	1	3	1	2	4	3	–
	CB2/CS3	49	27	7	6	6	1	–	1	1	–	–
sablecc(504)	CS3/CI	103	10	7	14	19	19	18	5	9	1	1
	CB2/CI	213	77	13	19	25	24	25	10	13	6	1
	CS3/CB2	12	5	3	2	1	1	–	–	–	–	–
	CB2/CS3	162	109	16	9	4	2	8	5	3	5	1

34

Observations

- For some subjects, considering contexts have insignificant effects on the precision of the computed information
- For some other subjects, considering contexts can improve the precision of the computed information
  - The two different types of contexts seem to offer their unique improvements
  - More context information does not always mean significant better precision

35



36

Observations

- Distinguishing the instances allocated at the same statement with contexts is crucial for getting more precise reference information.

37

Deriving Context-Insensitive Information

- Many (traditional) analyses compute information that will be true for all contexts
  - Slicing, modification side-effect
- Two approaches

Andersen: analyze methods under each context → Client: analyze methods under each context → Summarize: merge info computed for all contexts

Andersen: analyze methods under each context → Summarize: merge info computed for all contexts → Client: analyze methods without distinguishing context

38

Comparing with Info Computed by the Context-Insensitive Algorithm

Andersen: analyze methods under each context → Summarize: merge info computed for all contexts

Andersen: the context-insensitive version

→ Client: analyze methods without distinguishing context

$C_A[a]$ = {callsites recorded for allocation site a according to the info computed by algorithm A}

$$Diff_{A-B}[a] = \frac{(|C_A[a] - C_B[a]|) * 100}{|C_A[a]|}$$

39

The Distribution of Diff[]

program	compare	(0%, 100%)	(0%, 20%)	[20%, 40%)	[40%, 60%)	[60%, 80%)	[80%, 100%)
Antlr (254)	CI-CS3	19	8	9	1	1	-
	CI-CB2	15	6	8	-	1	-
	CI-CS3CB2	19	8	9	1	1	-
Jb (46)	CI-CS3	27	7	-	-	17	3
	CI-CB2	18	-	-	-	15	3
	CI-CS3CB2	27	7	-	-	17	3
Jfe (284)	CI-CS3	72	17	23	3	14	15
	CI-CB2	47	4	22	2	11	8
	CI-CS3CB2	72	17	23	3	14	15
Sablecc (504)	CI-CS3	123	23	1	12	13	74
	CI-CB2	159	9	18	20	20	92
	CI-CS3CB2	166	10	18	23	23	92

40