Shanghai Jiao Tong University
Software Theory and Practice Group

Aspect-Aware Points-to Analysis

Qiang Sun   Jianjun Zhao

Shanghai Jiao Tong University

http://stap.sjtu.edu.cn/

Outline

- Introduction
- Points-to analysis for Java
- Motivating examples
- A semantics for points-to analysis of AspectJ
- Evaluation
- Related work
- Concluding Remarks

Introduction of AOP

- Aspect-Oriented Programming (AOP)
  - AOP has been proposed as a technique for improving separation of concerns in software design and implementation.
  - AspectJ (a seamless aspect-oriented extension to Java)

Introduction of AOP

- ***Weaving***

  The process of combining aspect and object models to create the desired runtime behavior

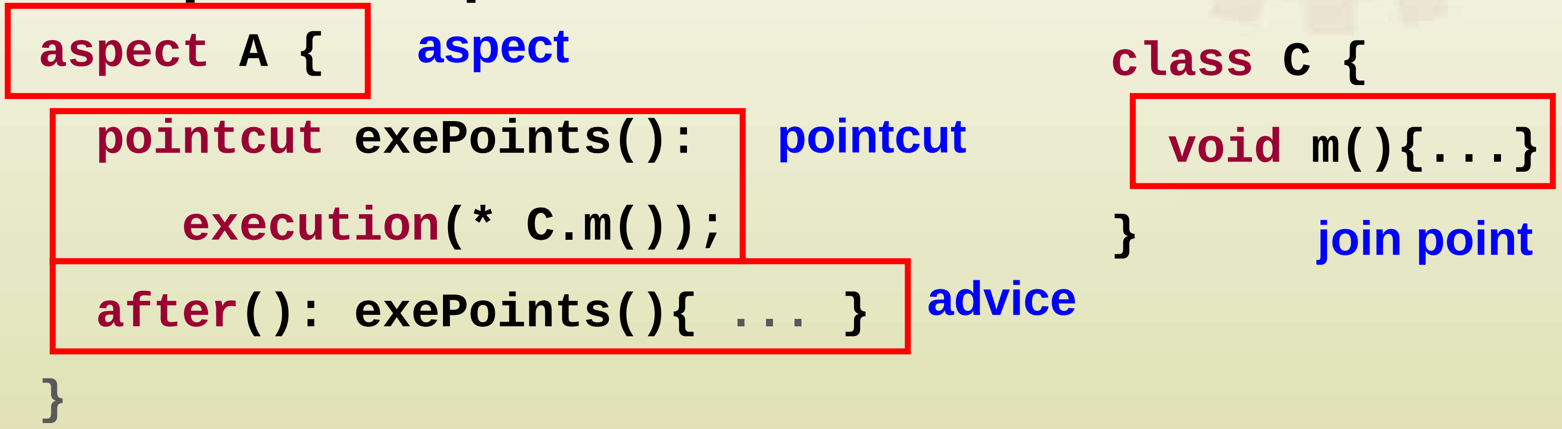
  – Can happen at build time or dynamically during runtime

  – Depending on technology, ***what*** to weave specified programmatically or declaratively

AspectJ Semantic

- **An AspectJ program can be divided into two parts:**
  - *Base code*, that is, language constructs as in Java
  - *Aspect code*, includes aspectual constructs, like *join points*, *pointcuts*, *pieces of advice*, *intertype declarations*.

- **A Simple Example:**



```
aspect A {
  pointcut exePoints():
     execution(* C.m());
  after(): exePoints(){ ... }
}
```

```
class C {
  void m(){...}
}
```

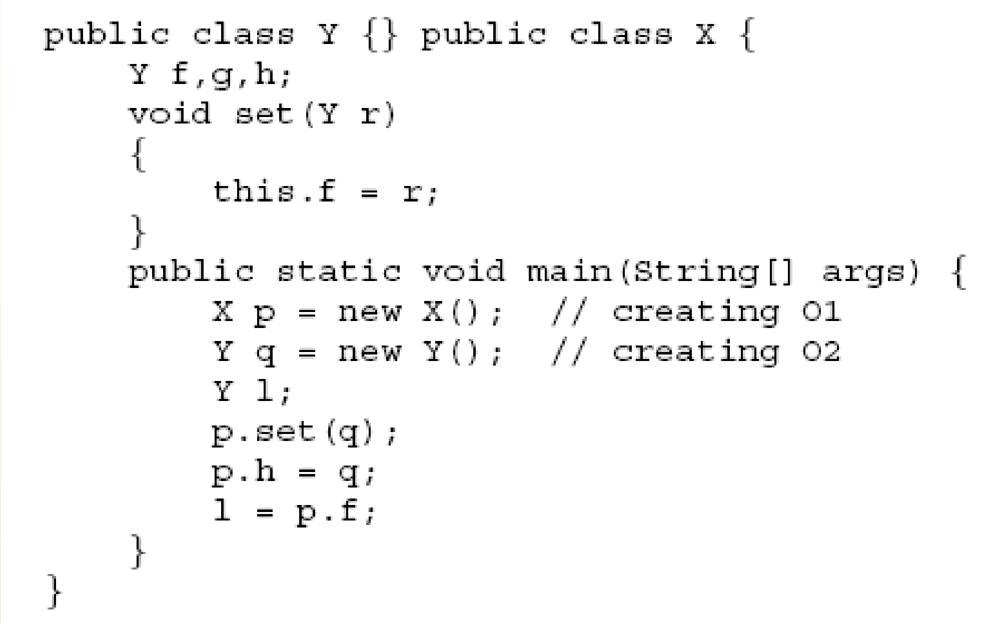
Points-To Analysis

- **Compute objects each variable can point to**
  - For each variable x, points-to set pt(x)
- **Andersen′s Analysis** **(Andersen, PhD thesis 1994)**
  - One abstract location for each allocation site

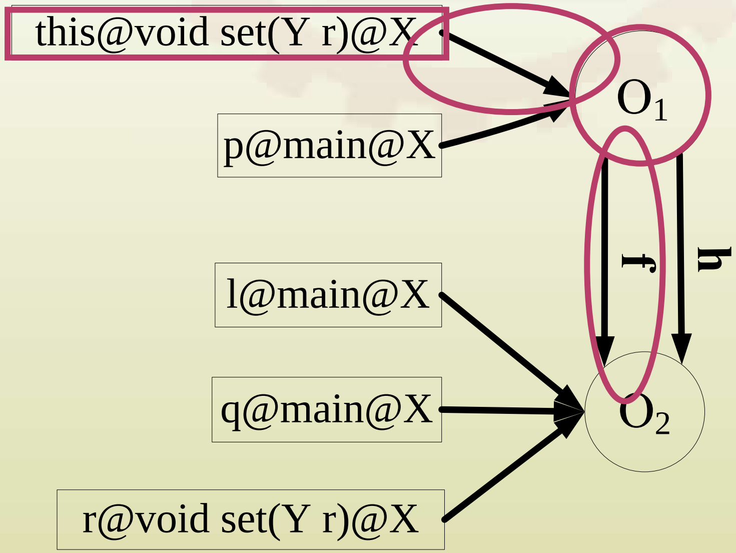
    `x = new C()` yields pt(x) = { o1 }
  - View pointer assignments using a constraint graph
  - Propagate points-to relations along the edges of the constraint graph, adding new edges as indirect constraints are resolved
- **Context- and flow-insensitive**
  - Context-insensitive: different invocation contexts without separation
  - Flow-insensitive: statements in any order

Flow- and Context-Insensitive Points-to Analysis for Java

- Points-to information representation in RMR* analysis



```
public class Y {} public class X {
    Y f,g,h;
    void set(Y r)
    {
        this.f = r;
    }
    public static void main(String[] args) {
        X p = new X();  // creating O1
        Y q = new Y();  // creating O2
        Y l;
        p.set(q);
        p.h = q;
        l = p.f;
    }
}
```

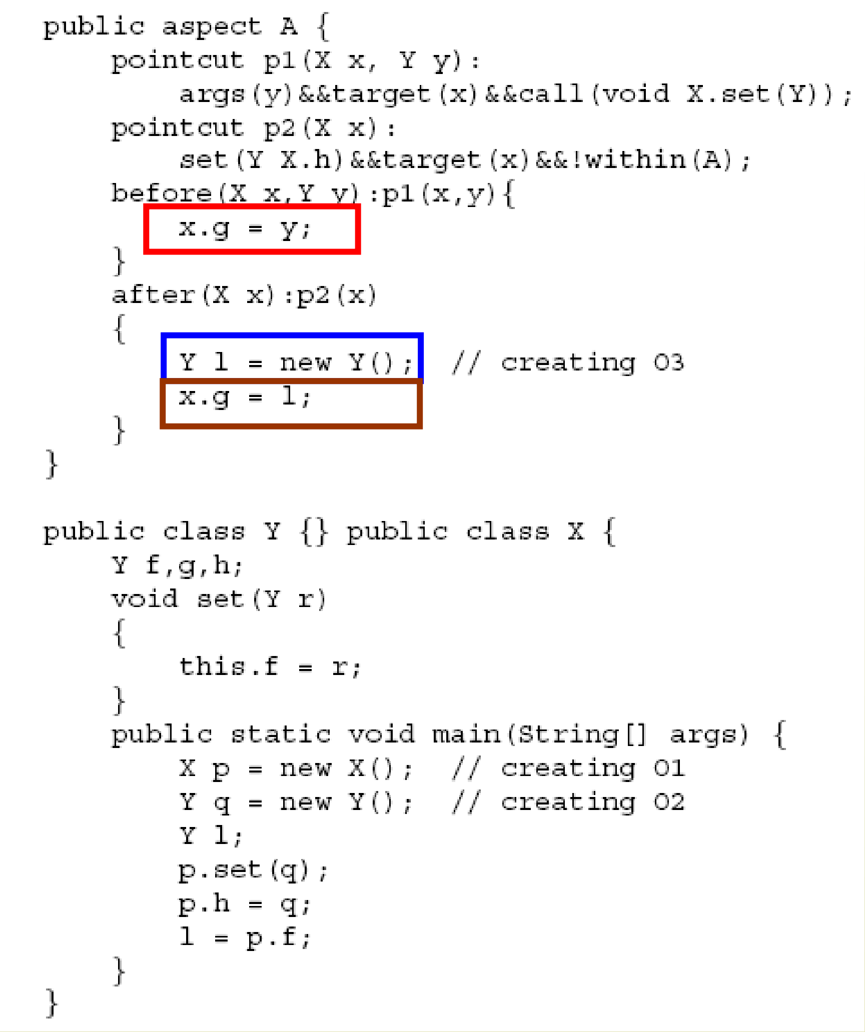


*A. Rountev, A. Milanova, and B. Ryder (OOPSLA 2001)

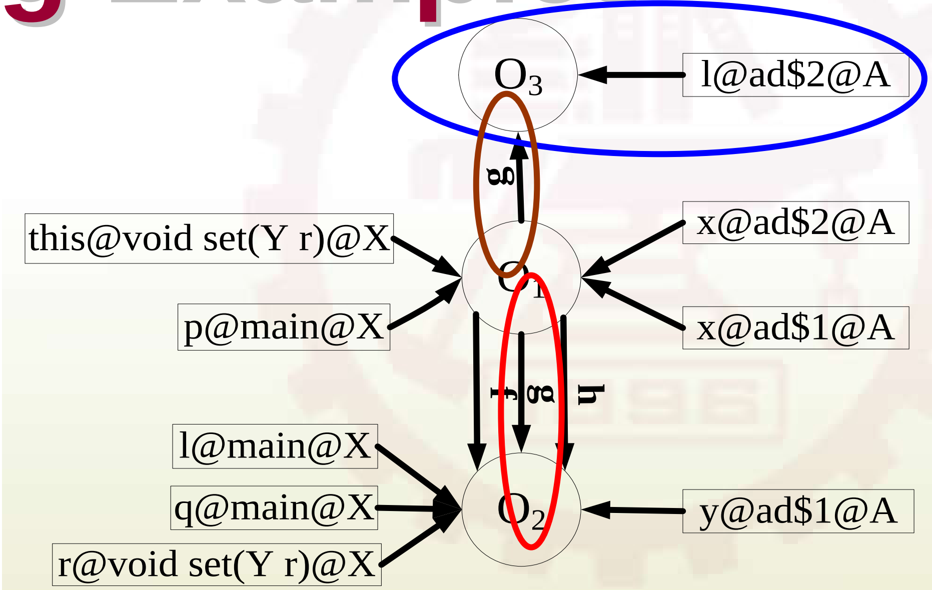
Motivating Examples

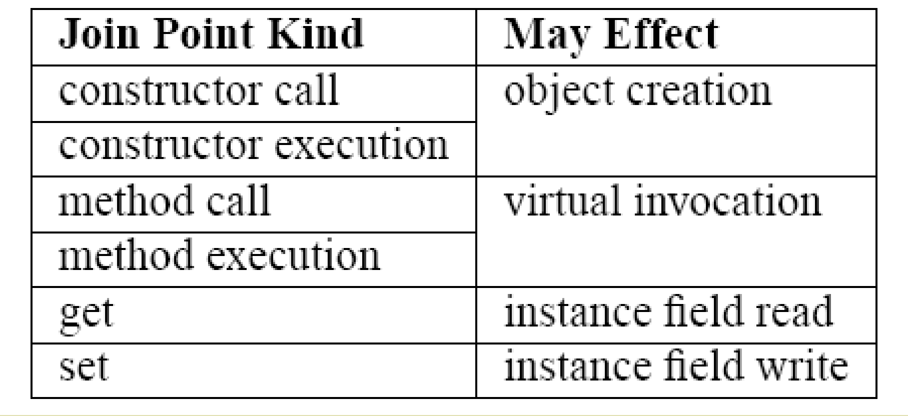
- Aspects in AspectJ are transparent to classes
  - From the source of a class, you can not know which aspect should be woven into the class.
  - ***If we directly use the existing analysis for the AspectJ program, the result will be imprecise*.**
- In order to handle the unique aspectual features, a new points-to analysis technique is needed.

Motivating Examples



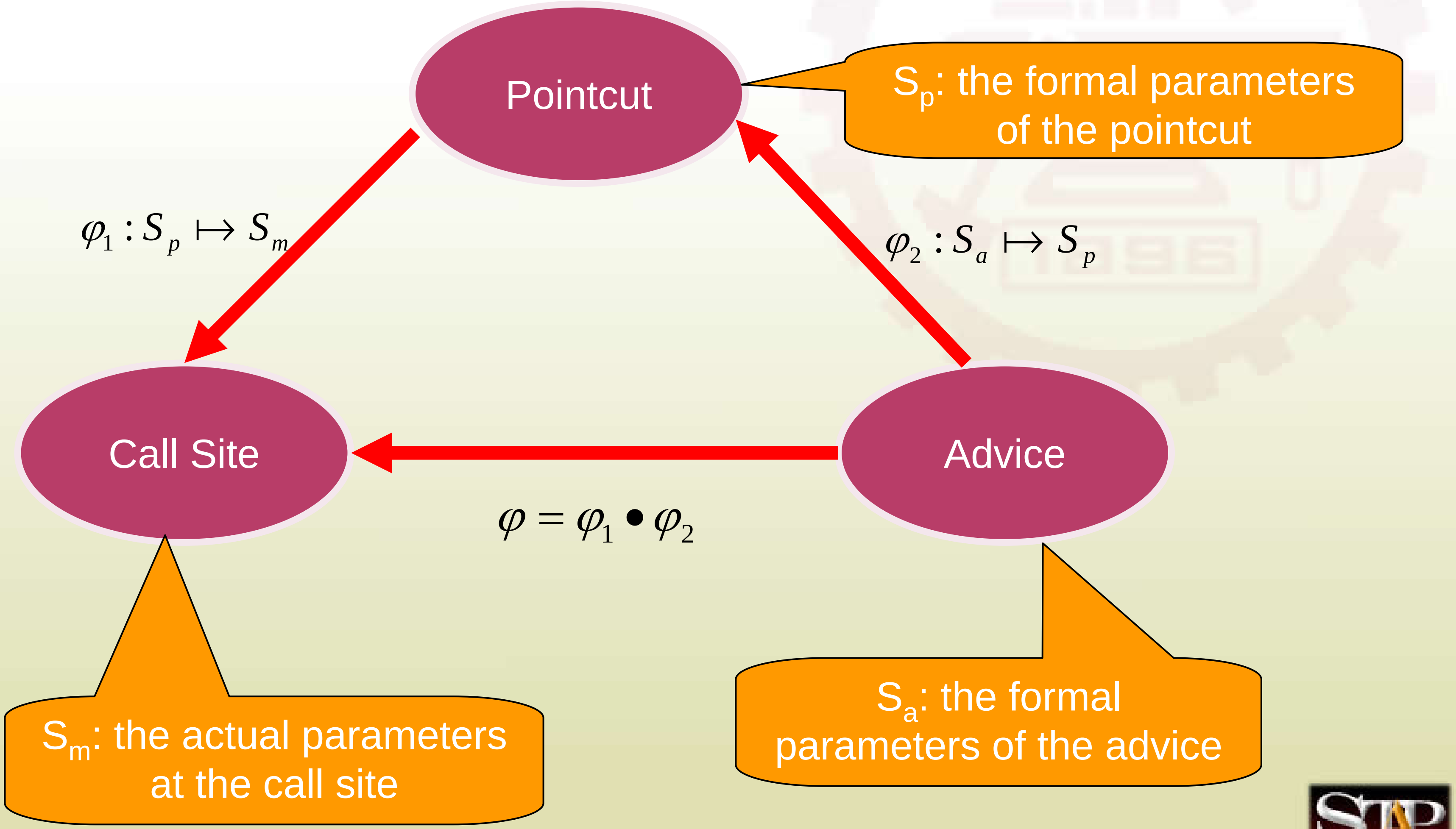
```
public aspect A {
    pointcut p1(X x, Y y):
        args(y)&&target(x)&&call(void X.set(Y));
    pointcut p2(X x):
        set(Y X.h)&&target(x)&&!within(A);
    before(X x,Y y):p1(x,y){
        x.g = y;
    }
    after(X x):p2(x)
    {
        Y l = new Y();  // creating O3
        x.g = l;
    }
}

public class Y {} public class X {
    Y f,g,h;
    void set(Y r)
    {
        this.f = r;
    }
    public static void main(String[] args) {
        X p = new X();  // creating O1
        Y q = new Y();  // creating O2
        Y l;
        p.set(q);
        p.h = q;
        l = p.f;
    }
}
```





Join Point Kind	May Effect
constructor call	object creation
constructor execution	
method call	virtual invocation
method execution	
get	instance field read
set	instance field write

A Semantics for Points-to Analysis of AspectJ



$\varphi_1 : S_p \mapsto S_m$

$\varphi_2 : S_a \mapsto S_p$

$\varphi = \varphi_1 \bullet \varphi_2$

S_p: the formal parameters of the pointcut

S_m: the actual parameters at the call site

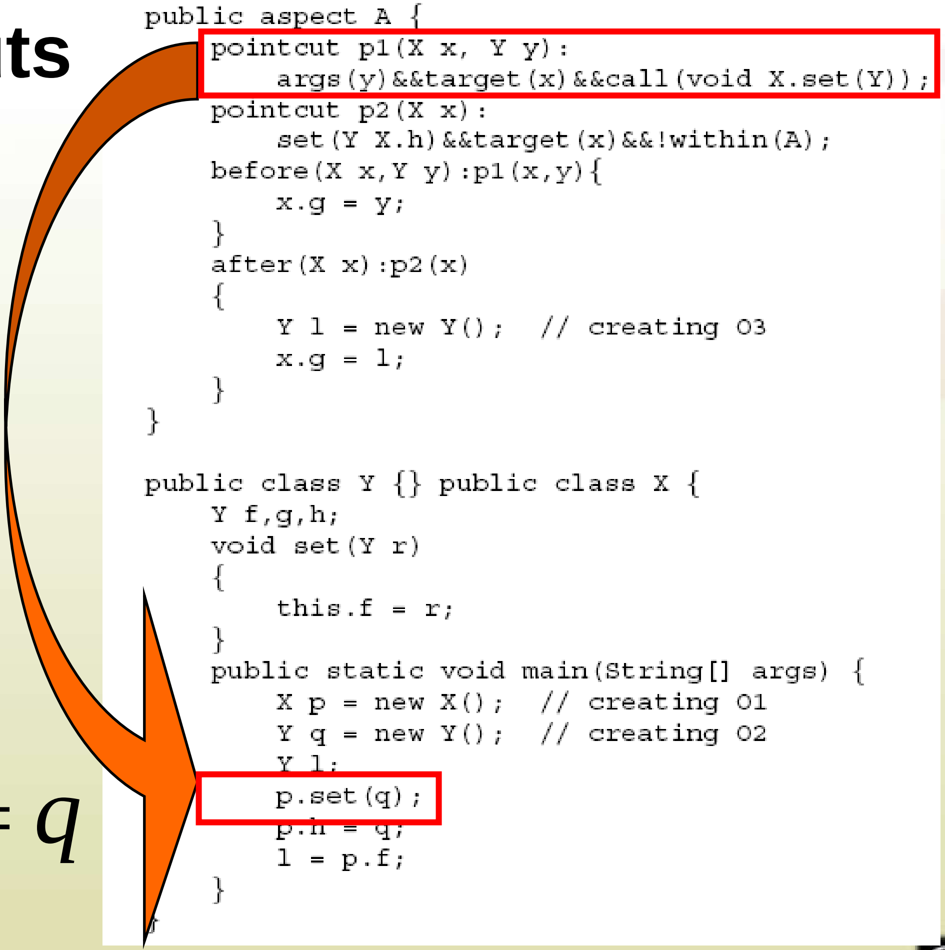
S_a: the formal parameters of the advice

A Semantics for Points-to Analysis of AspectJ

- **To handle pointcuts**

$$S_p = \{x, y\}$$

$$\varphi_1(x) = p, \varphi_1(y) = q$$



```
public aspect A {
    pointcut p1(X x, Y y):
        args(y)&&target(x)&&call(void X.set(Y));
    pointcut p2(X x):
        set(Y X.h)&&target(x)&&!within(A);
    before(X x,Y y):p1(x,y){
        x.g = y;
    }
    after(X x):p2(x)
    {
        Y l = new Y();  // creating O3
        x.g = l;
    }
}

public class Y {} public class X {
    Y f,g,h;
    void set(Y r)
    {
        this.f = r;
    }
    public static void main(String[] args) {
        X p = new X();  // creating O1
        Y q = new Y();  // creating O2
        Y l;
        p.set(q);
        p.h = q;
        l = p.f;
    }
}
```

A Semantics for Points-to Analysis of AspectJ

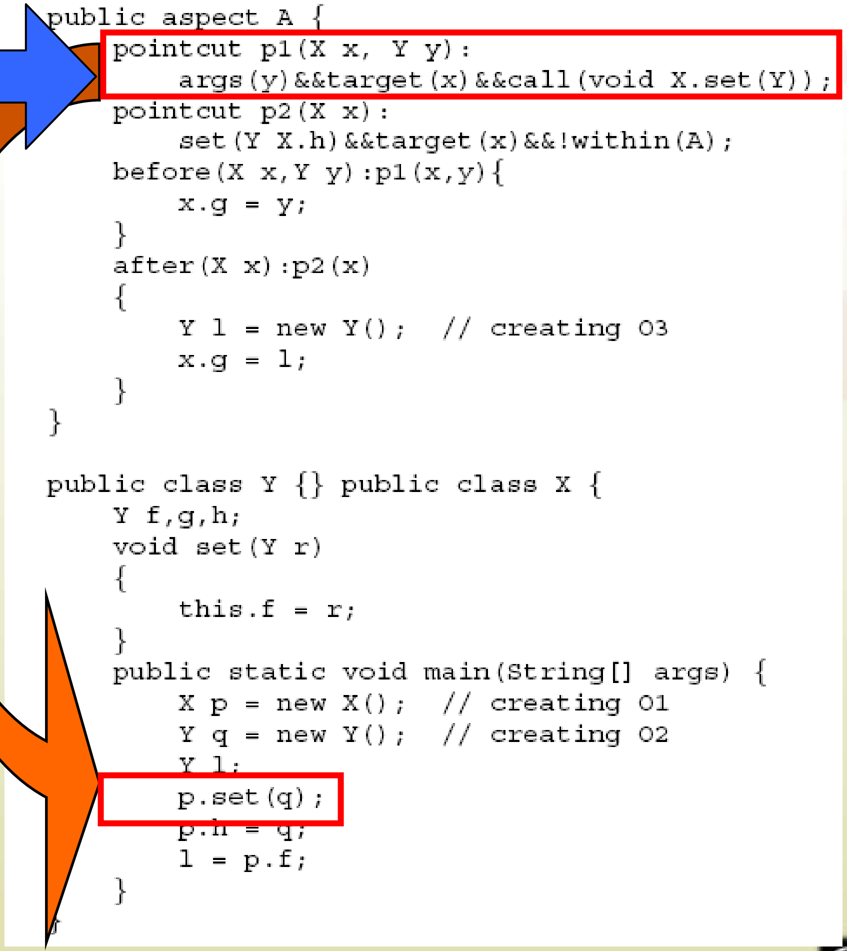
- **To handle advice**

$$after(X\ xx,\ Y\ yy)\ :\ p1(xx,\ yy)$$

$$S_a = \{xx, yy\}$$

$$\varphi_2(xx) = x, \varphi_2(yy) = y$$

$$\underline{\varphi(xx) = p, \varphi(yy) = q}$$



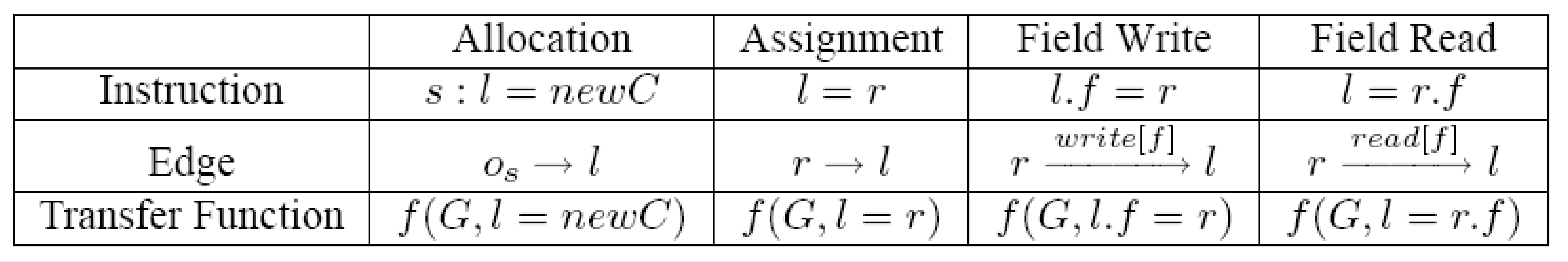
```
public aspect A {
    pointcut p1(X x, Y y):
        args(y)&&target(x)&&call(void X.set(Y));
    pointcut p2(X x):
        set(Y X.h)&&target(x)&&!within(A);
    before(X x,Y y):p1(x,y){
        x.g = y;
    }
    after(X x):p2(x)
    {
        Y l = new Y();  // creating O3
        x.g = l;
    }
}

public class Y {} public class X {
    Y f,g,h;
    void set(Y r)
    {
        this.f = r;
    }
    public static void main(String[] args) {
        X p = new X();  // creating O1
        Y q = new Y();  // creating O2
        Y l;
        p.set(q);
        p.h = q;
        l = p.f;
    }
}
```

Performing Points-to Analysis for AspectJ

- To construct call graph and pointer assignment graph
- To build up points-to graph based on the former stage

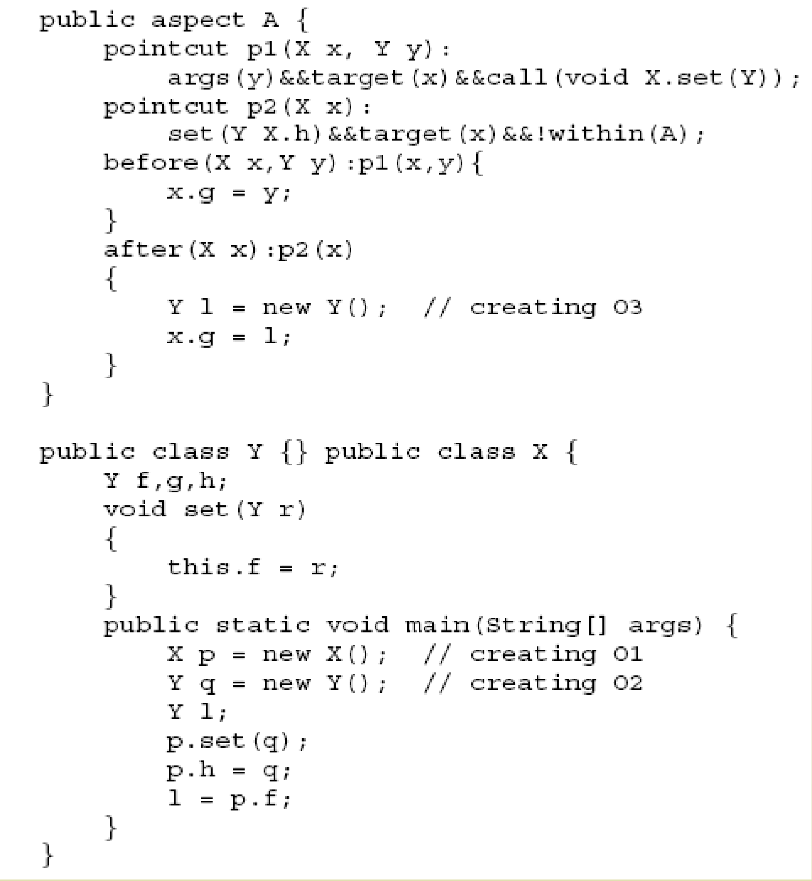
Performing Points-to Analysis for AspectJ

- **Pointer assignment graph**
  - – Allocation site nodes
  - – Variable nodes
  - – Field dereference nodes

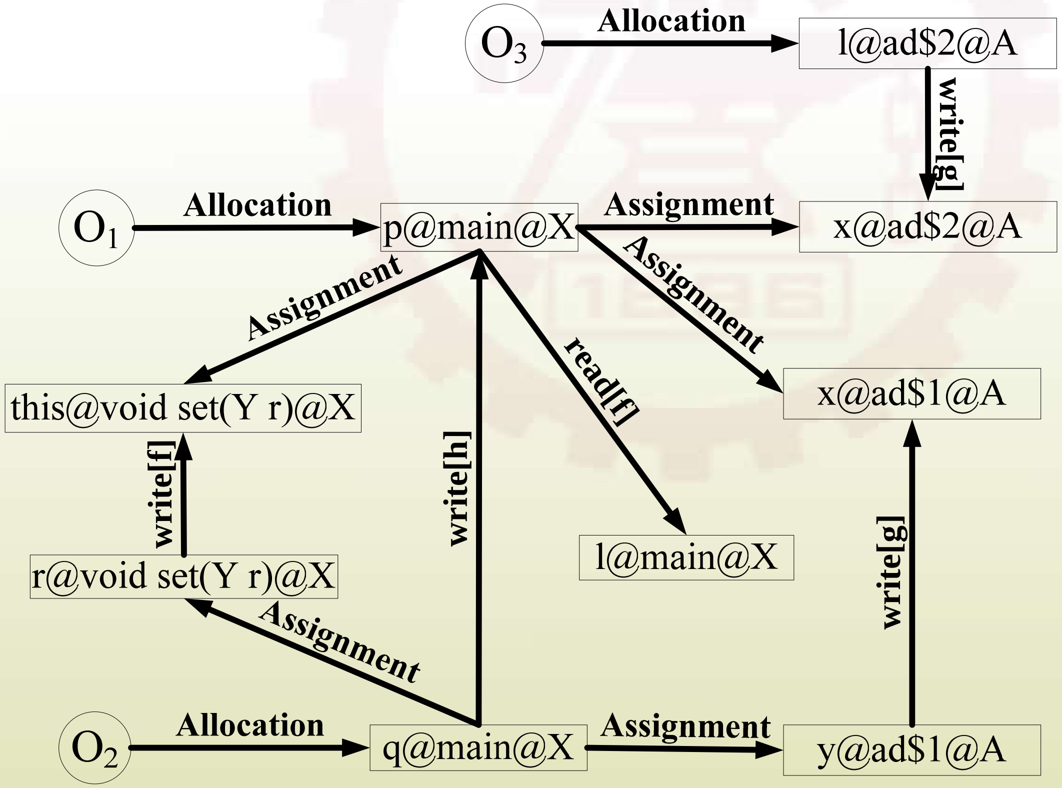


	Allocation	Assignment	Field Write	Field Read
Instruction	$s : l = newC$	$l = r$	$l.f = r$	$l = r.f$
Edge	$o_s \rightarrow l$	$r \rightarrow l$	$r \xrightarrow{write[f]} l$	$r \xrightarrow{read[f]} l$
Transfer Function	$f(G, l = newC)$	$f(G, l = r)$	$f(G, l.f = r)$	$f(G, l = r.f)$

Performing Points-to Analysis for AspectJ

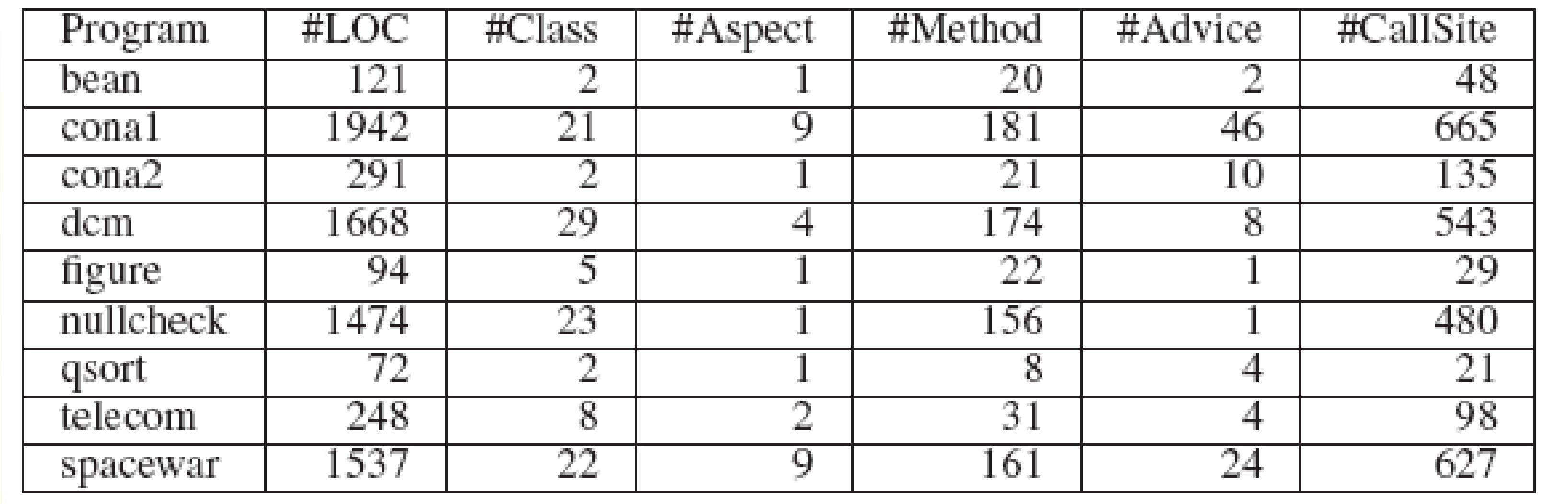


```
public aspect A {
    pointcut p1(X x, Y y):
        args(y)&&target(x)&&call(void X.set(Y));
    pointcut p2(X x):
        set(Y X.h)&&target(x)&&!within(A);
    before(X x,Y y):p1(x,y){
        x.g = y;
    }
    after(X x):p2(x)
    {
        Y l = new Y();  // creating O3
        x.g = l;
    }
}

public class Y {} public class X {
    Y f,g,h;
    void set(Y r)
    {
        this.f = r;
    }
    public static void main(String[] args) {
        X p = new X();  // creating O1
        Y q = new Y();  // creating O2
        Y l;
        p.set(q);
        p.h = q;
        l = p.f;
    }
}
```



Evaluation

- Subject programs



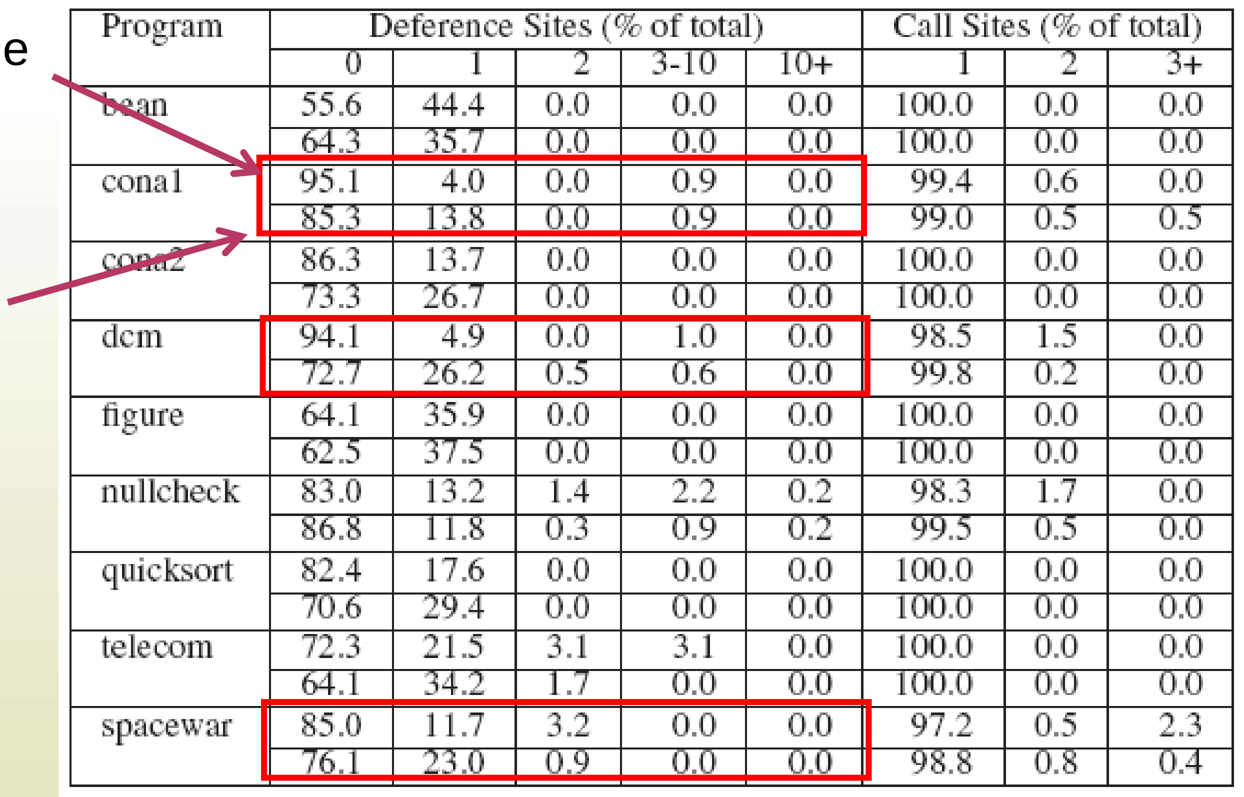
Program	#LOC	#Class	#Aspect	#Method	#Advice	#CallSite
bean	121	2	1	20	2	48
cona1	1942	21	9	181	46	665
cona2	291	2	1	21	10	135
dcm	1668	29	4	174	8	543
figure	94	5	1	22	1	29
nullcheck	1474	23	1	156	1	480
qsort	72	2	1	8	4	21
telecom	248	8	2	31	4	98
spacewar	1537	22	9	161	24	627

- Experiment procedure
  - Compare the precision between the existing byte-code-level Java approach and our source-code-level approach.

Precision Analysis

Source-code approach

Byte-code approach

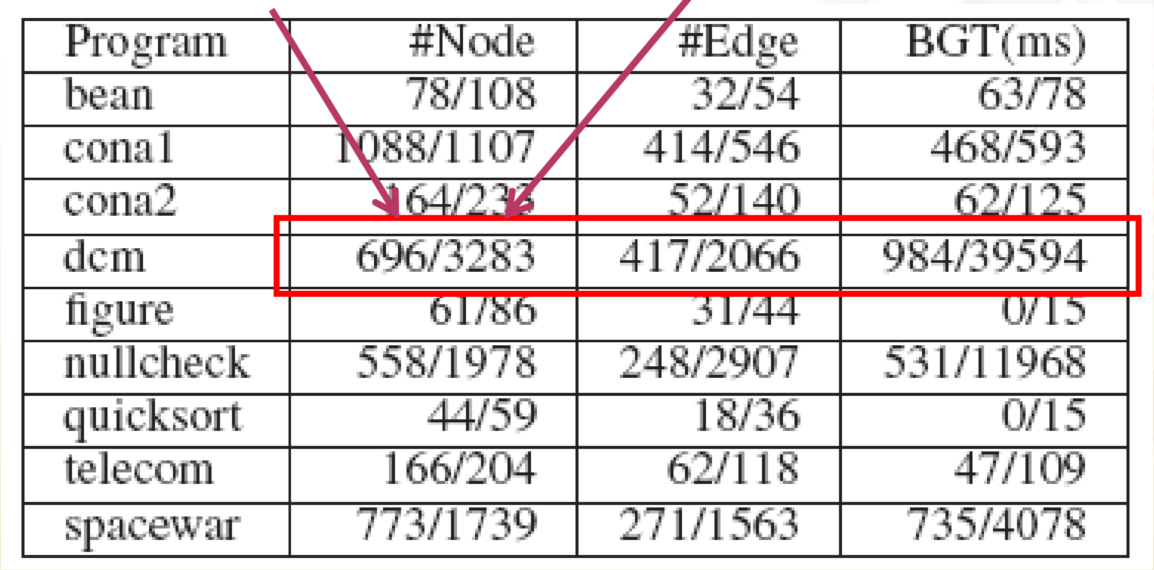


Program	Deference Sites (% of total)					Call Sites (% of total)		
	0	1	2	3-10	10+	1	2	3+
bean	55.6	44.4	0.0	0.0	0.0	100.0	0.0	0.0
	64.3	35.7	0.0	0.0	0.0	100.0	0.0	0.0
cona1	95.1	4.0	0.0	0.9	0.0	99.4	0.6	0.0
	85.3	13.8	0.0	0.9	0.0	99.0	0.5	0.5
cona2	86.3	13.7	0.0	0.0	0.0	100.0	0.0	0.0
	73.3	26.7	0.0	0.0	0.0	100.0	0.0	0.0
dcm	94.1	4.9	0.0	1.0	0.0	98.5	1.5	0.0
	72.7	26.2	0.5	0.6	0.0	99.8	0.2	0.0
figure	64.1	35.9	0.0	0.0	0.0	100.0	0.0	0.0
	62.5	37.5	0.0	0.0	0.0	100.0	0.0	0.0
nullcheck	83.0	13.2	1.4	2.2	0.2	98.3	1.7	0.0
	86.8	11.8	0.3	0.9	0.2	99.5	0.5	0.0
quicksort	82.4	17.6	0.0	0.0	0.0	100.0	0.0	0.0
	70.6	29.4	0.0	0.0	0.0	100.0	0.0	0.0
telecom	72.3	21.5	3.1	3.1	0.0	100.0	0.0	0.0
	64.1	34.2	1.7	0.0	0.0	100.0	0.0	0.0
spacewar	85.0	11.7	3.2	0.0	0.0	97.2	0.5	2.3
	76.1	23.0	0.9	0.0	0.0	98.8	0.8	0.4

Performance Analysis

Source-code approach

Byte-code approach



Program	#Node	#Edge	BGT(ms)
bean	78/108	32/54	63/78
cona1	1088/1107	414/546	468/593
cona2	164/233	52/140	62/125
dcm	696/3283	417/2066	984/39594
figure	61/86	31/44	0/15
nullcheck	558/1978	248/2907	531/11968
quicksort	44/59	18/36	0/15
telecom	166/204	62/118	47/109
spacewar	773/1739	271/1563	735/4078

Our aspect-aware points-to analysis approach contains fewer nodes and edges, and improve the precision significantly.

Related Work

- Points-to analysis
  - A. Rountev, A. Milanova, and B. Ryder (OOPSLA 2001)
  - O. Lhotak and L. Hendren (CC 2003)
  - David F. Bacon and Peter F. Sweeney (OOPSLA 1996)
  - B. Steensgaard (POPL 1996)
  - L. Andersen (PhD thesis 1994)
  - J. Whaley and M. S. Lam (SAS 2002)

Concluding Remarks

- **Conclusions**
  - We proposed an aspect-aware points-to analysis
  - The feature of our analysis
    - AspectJ source-code-level
    - Flow-insensitive and context-insensitive
- **Future work**
  - Library code
  - Flow-sensitive & context-sensitive

Thank you!