State-Sensitive Points-to Analysis for the Dynamic Behavior of JavaScript Objects

Shiyi Wei and Barbara G. Ryder

ECOOP 2014

Presented by ke tian

Outlines:

- What is the problem?

  track the changes of object properties

- What is the solution/contribution?

  state-sensitive points-to analysis

  + a new control-flow graph representation

- How efficient is the solution?

  significant improvement (+11% precision)

Background (__proto__)

Edit This Code:

See Result »

```
<!DOCTYPE html>
<html>
<body>

<p>Test By Ke</br></p>
<p id="demo"></p>

<script>

function Foo(y){this.y = y;}

var b = new Foo(20);
b.c =  30;

document.getElementById("demo").innerHTML =
(b.__proto__ ) + '</br>'+
(b.c) + '</br>'+
(b.y) + '</br>' +
(b.__proto__== Foo.prototype) +'</br>' +
(b.constructor)
;

</script>

</body>
</html>
```

Result:

Test By Ke

[object Object]
30
20
true
function Foo(y){this.y = y;}

b	
c	30
__proto__	

Foo.prototype	
y	20
constructor	function Function() { [native code] }
__proto__	

Object.prototype

How to (formally) describe a JavaScript Object?

Definition 1. The *obj-ref state* at a program point denotes all of its accessible properties and their non-primitive values.

Def 1. is used to describe a type (not constant) of a JavaScript object

Definition 2. *State-update statements* are: (1) property write statement (i.e., $x.p = y$ or $x['p'] = y$), (2) property delete statement (i.e., *delete* $x.p$ or *delete* $x['p']$), and (3) an invocation that directly or indirectly results in execution of (1) and/or (2).

Def 2. Write and delete operations can result in state-update, affect obj-ref states

An example of obj-ref state

Local O3 override the property p2, thus O6 is not accessible

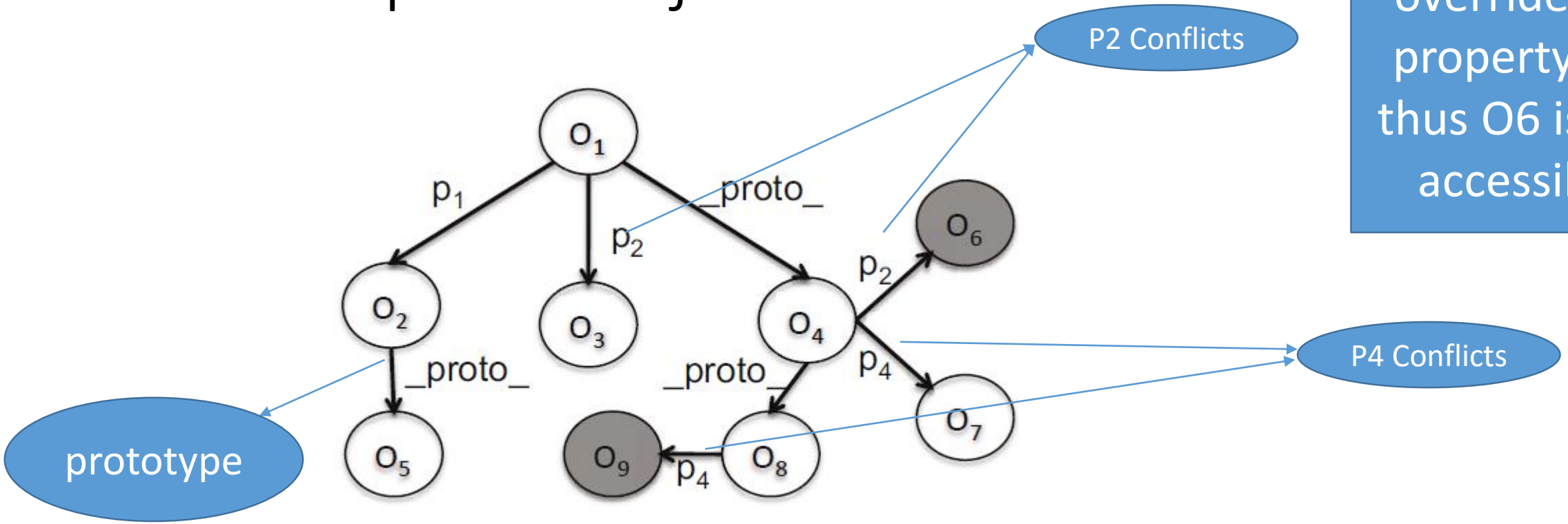


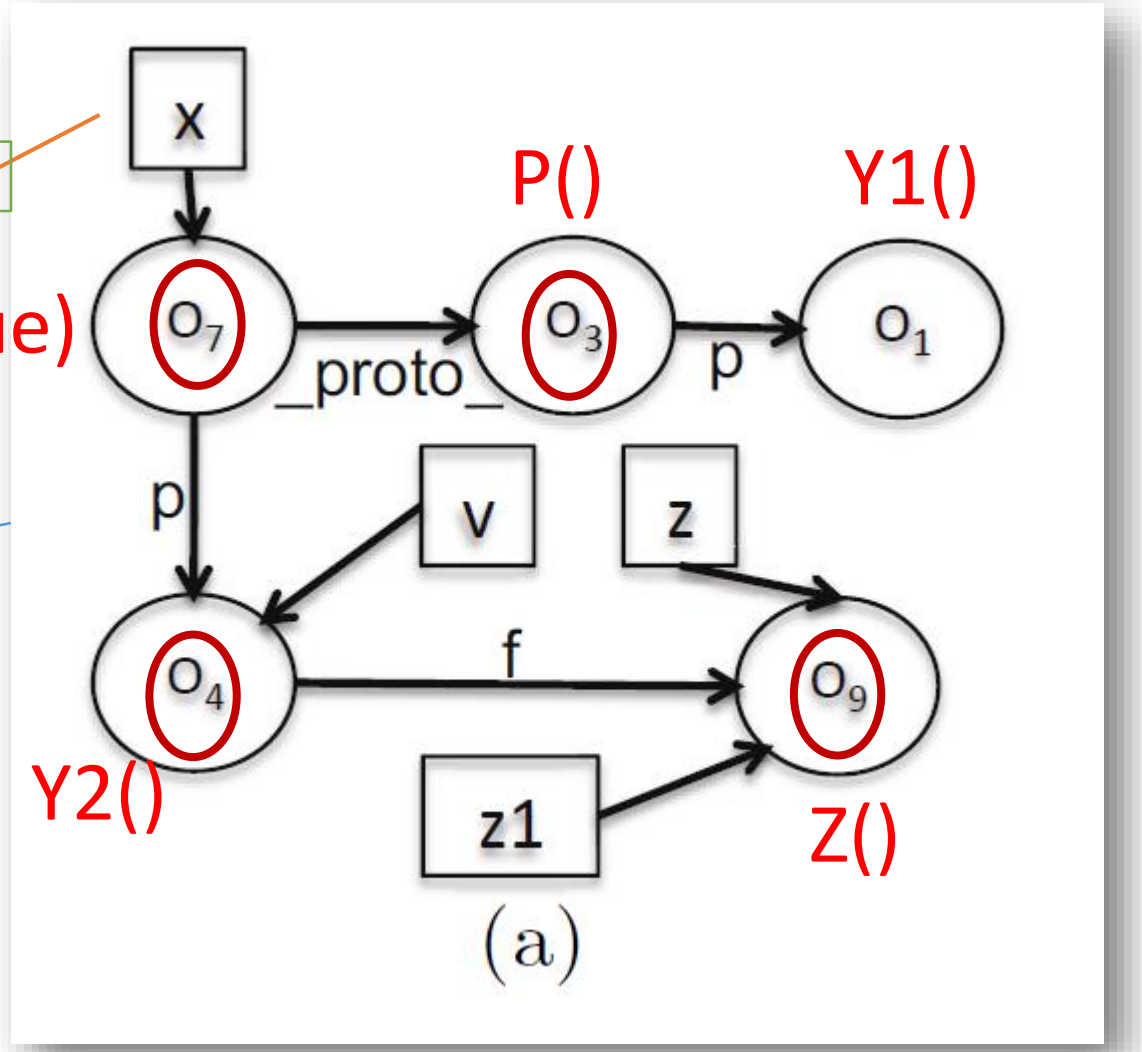
Fig. 1. *obj-ref state* for O_1. (Unshaded nodes only)

Obj-Ref(O1)={O1,O2,O3,O4,O5,O7,O8}

Motivating example (the problem)

```
1   function P(){ this.p = new Y1(); }
2   function X(b){
3       this.__proto__ = new P();
4       if(b) { this.p = new Y2(); }
5       else this.q = new Y3();
6   }
7   var x = new X(true);
8   x.bar = function(v, z){ v.f = z; }
9   var z1 = new Z();
10  x.bar(x.p, z1);
11  ...
12  x.p = new A();
13  ...
14  var z2 = new Z();
15  x.bar(x.p,z2);
```

Line 10

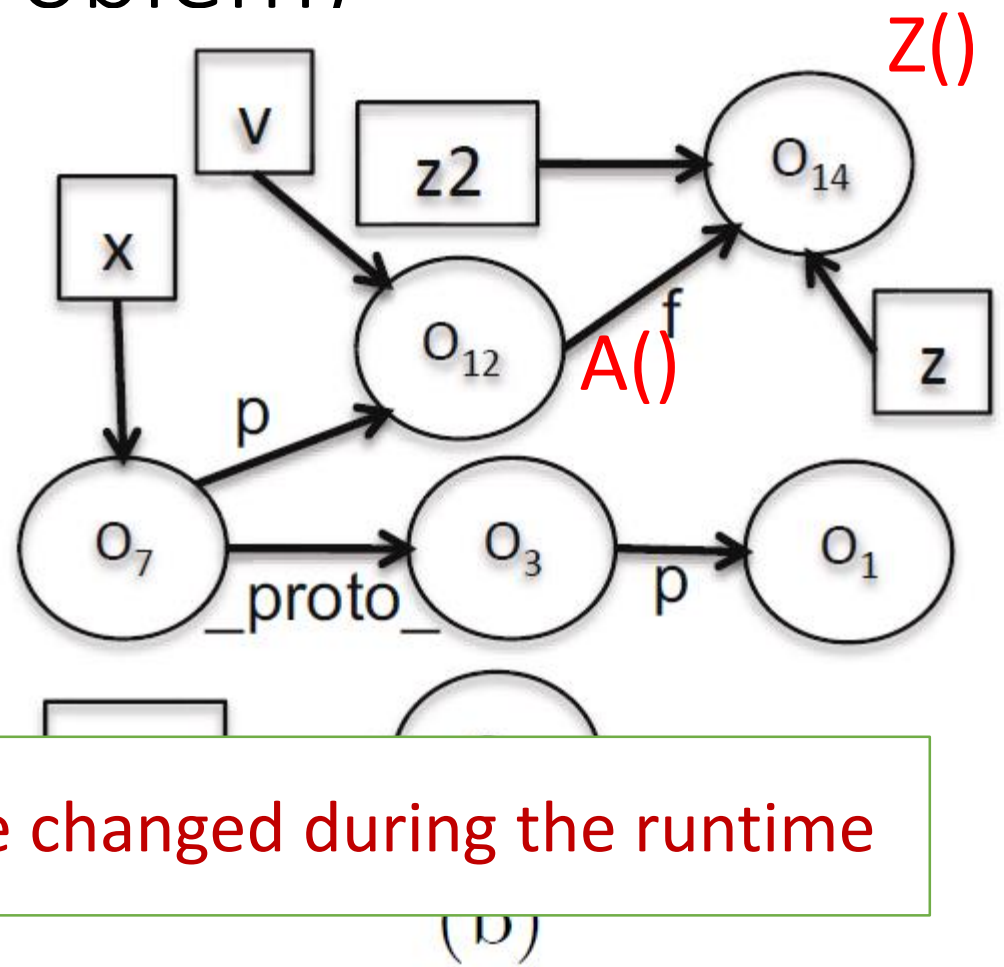


Obj-Ref(O7)={O7, O3,O4,O9}

Run-time points-to graph at line 10

Motivating example (the problem)

```
1   function P(){ this.p = new Y1(); }
2   function X(b){
3       this.__proto__ = new P();
4       if(b) { this.p = new Y2(); }
5       else this.q = new Y3();
6   }
7   var x = new X(true);
8   x.bar = function(v, z){ v.f = z; }
9   var z1 = new Z();
10  x.bar(x.p, z1);
11  ...
12  x.p =
13  ...
14  var z2 = new Z();
15  x.bar(x.p,z2);
```



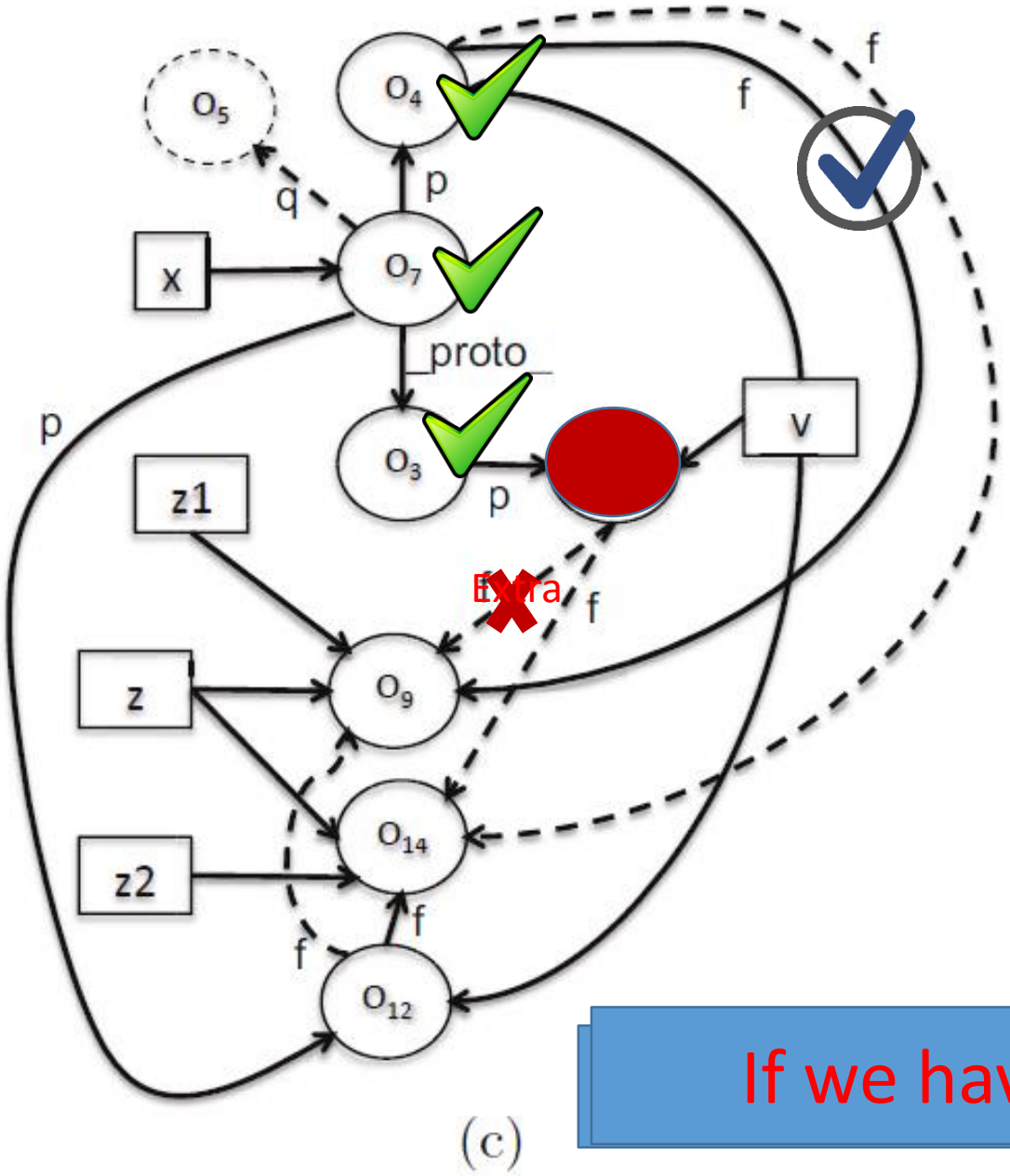
X(true)

Conclusion: Obj-Ref(O7) can be changed during the runtime

Obj-Ref(O7)={O7, O12,O3,O14}

Run-time points-to graph at line 15

What is the problem? (imprecision)



- 1 constructor polymorphism (O7.q)
- 2 Object property change
- **3 Function invocation**

  x.bar(x.p,z1) [line 10]

  x.p.f = z1(O9)

  E.g.,not knowing O4 exists extra : (O1 -> O9)

If we have Obj-Ref(O7)={O7, O3,O4} line 9

What is the Solution? (state-preserving block graph)

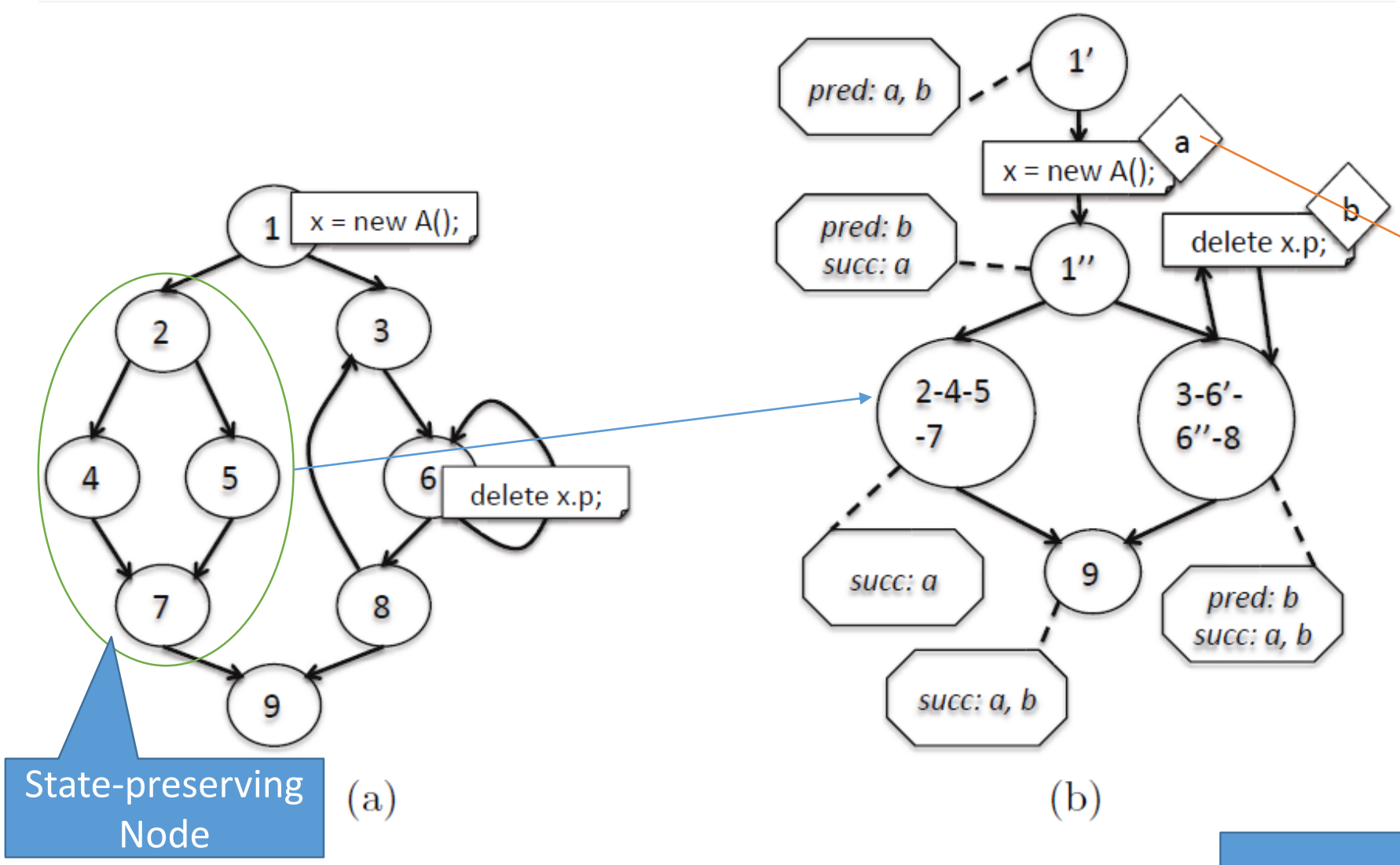


Fig. 4. *SPBG* generation. (a) CFG. (b) *SPBG*.

- Split the CFG based on state-update

Statement (new/delete)

1->

(1',x=newA(),1'')

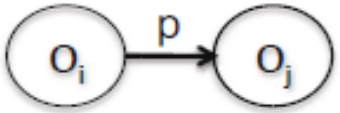
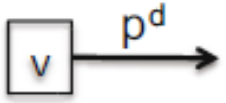
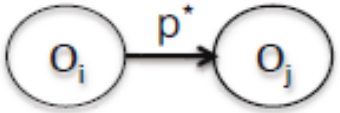
- Aggregate the

state-preserving nodes in the graph

Partial flow sensitive

Points-to graph representation

Table 1. Expanded points-to graph with annotations

points-to graph G	node N	variable v	
		abstract object o	
		in-construction object $@o$	
	edge E	variable reference $(v, \phi o)$	
		property reference $(< \phi o_i, p >, \phi o_j)$	
		access path $(< v, p >, \phi o)$	
	annotation A	d annotation p^d	
		* annotation p^*	

Used for object creation (in transfer function)

$\Phi o = \{@o, o\}$

Traditional points-to graph

Help us to (strongly) determine the property of variable

P* means the relation MAY not exist (safety)

Transfer functions (update points-to graph)

- Object creation (x =new X(a1,a2,…, an))
- Property write (x.p = y) [example]
- Property delete (delete x.p)
- Direct write (x = y)
- Property read (x = y.p)
- Method invocation (x =y.m(a1,a2,…an))

Complex Algebra

Transfer rule (example x.p = y)

Strong update:
Delete (<x,p> ,O4)
Add(<x,p>, O3)
Add(<x,p>, O2)
Weak update:
Add(<O1,p*>, O3)
Add(<O1,p*>, O2)

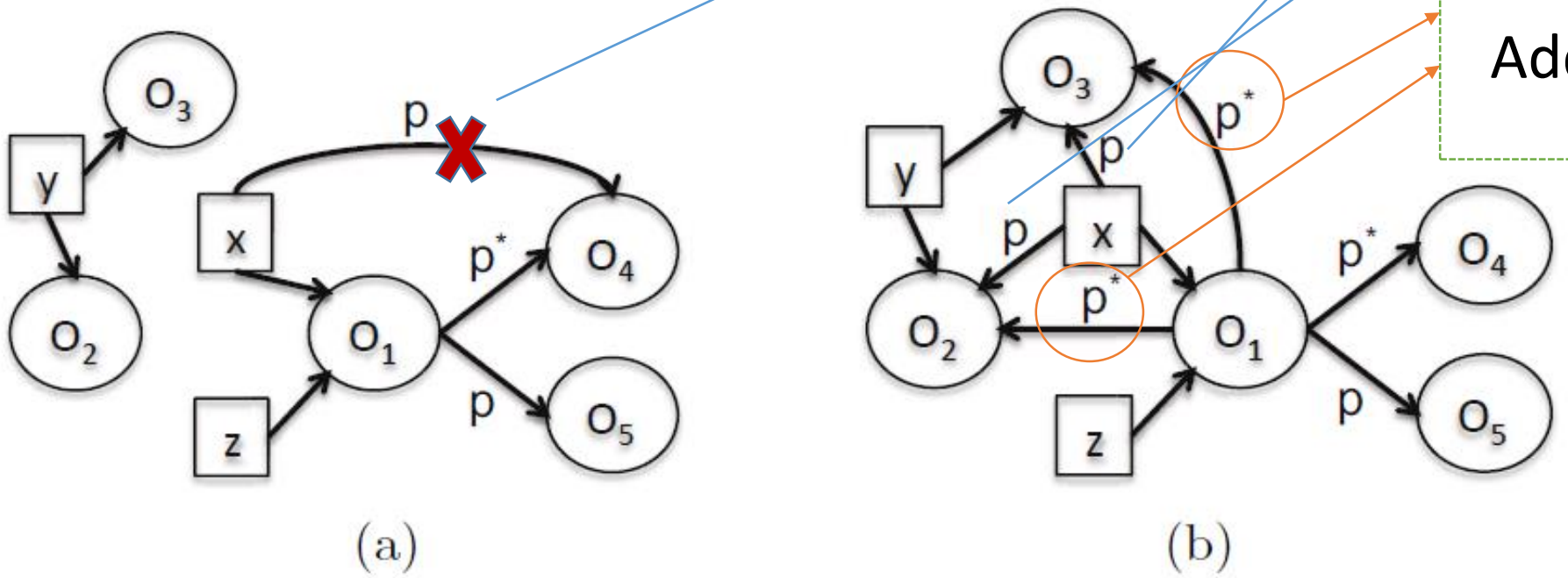


Fig. 5. Property write example. (a) Input points-to graph. (b) Updated points-to graph.

Approximation (reduce analysis overhead) use obj-ref state as context

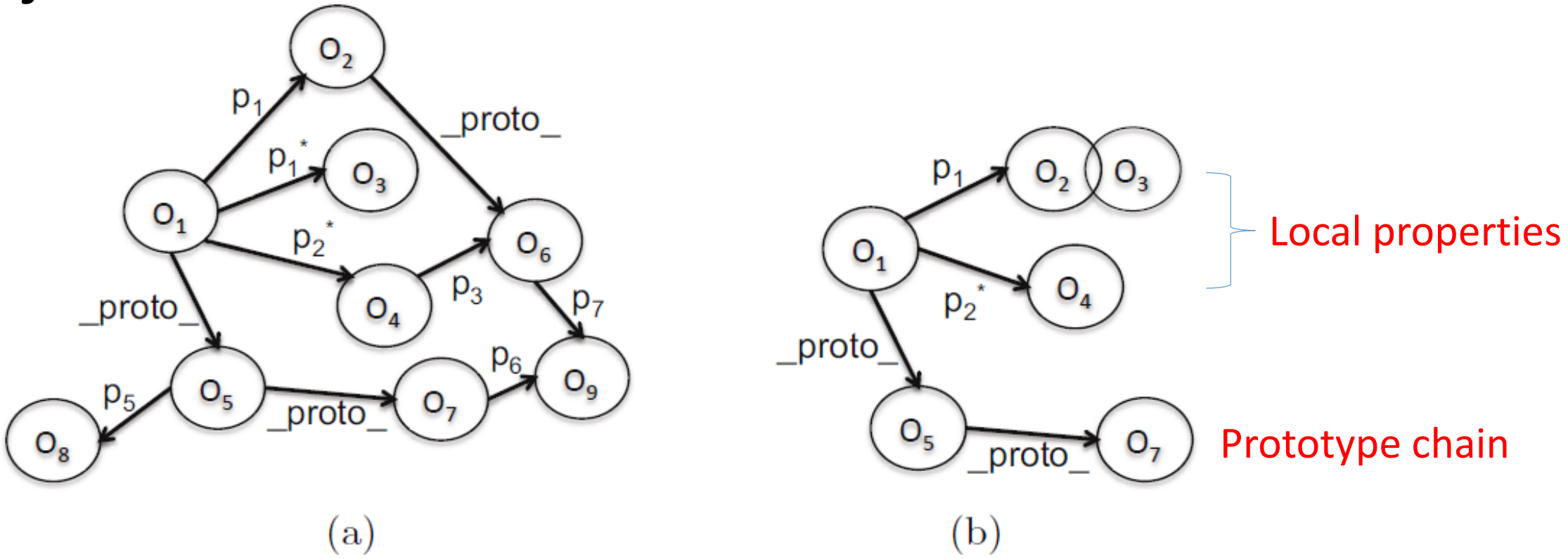
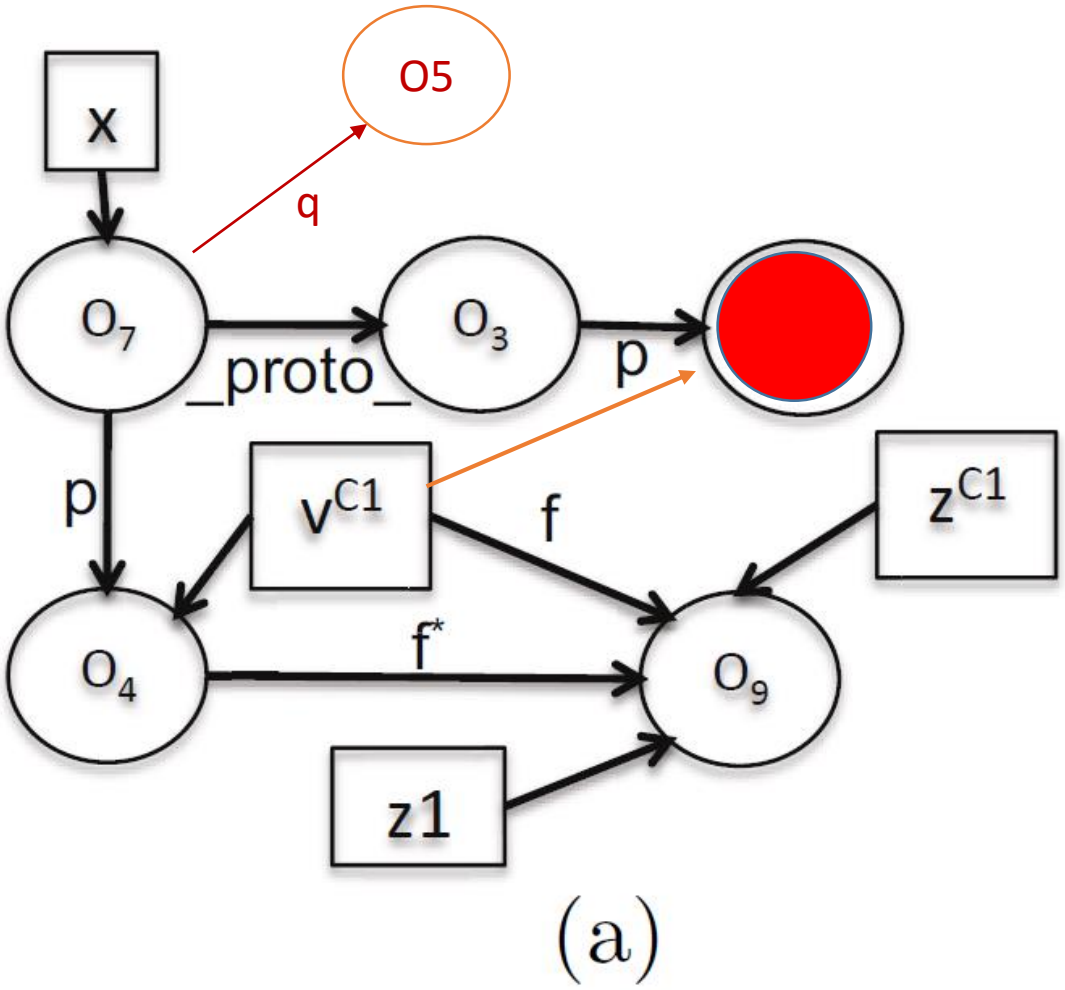


Fig. 6. Approximate *obj-ref state* as a context. (a) *obj-ref state* of O_1. (b) Approximate *obj-ref state* of O_1.

Trade-off :Lose the (some) precision but increase scalability

How efficient is the solution?



(a)

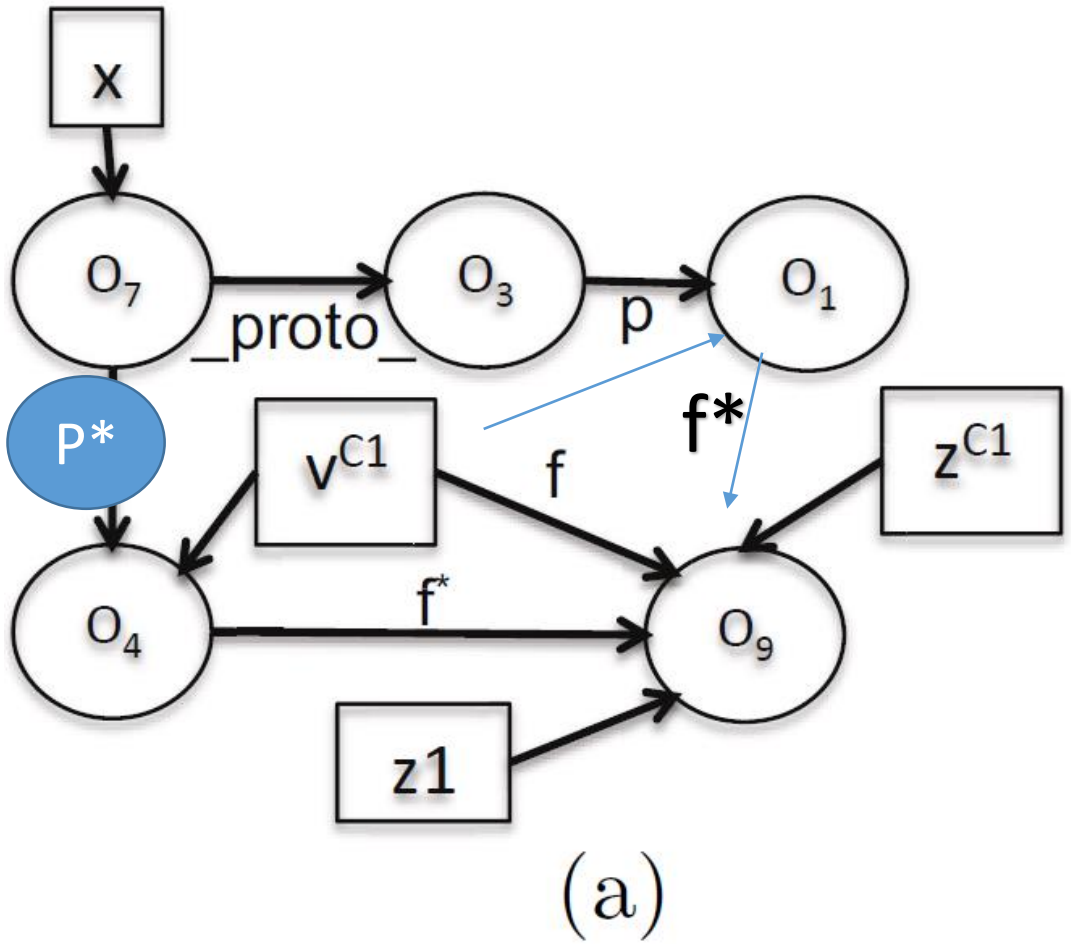
insensitive Points-to graph at line 10

Approximate Obj-Ref(O7)
C1= {O7,
p:O4, (NO O1 here!!!)
__proto__ : O3
}

```
1   ; }
2
3
4   }
5
6
7
8   x.bar = function(v, z){ v.f = z; }
9   var z1 = new Z();
10  x.bar(x.p, z1);
11  ...
12  x.p = new A();
13  ...
14  var z2 = new Z();
15  x.bar(x.p,z2);
```

Fig. 2. JavaScript example

How efficient is the solution?



Points-to graph at line 10 (if p*)

What if O7 -(p*)-> O4?

C1={O7,

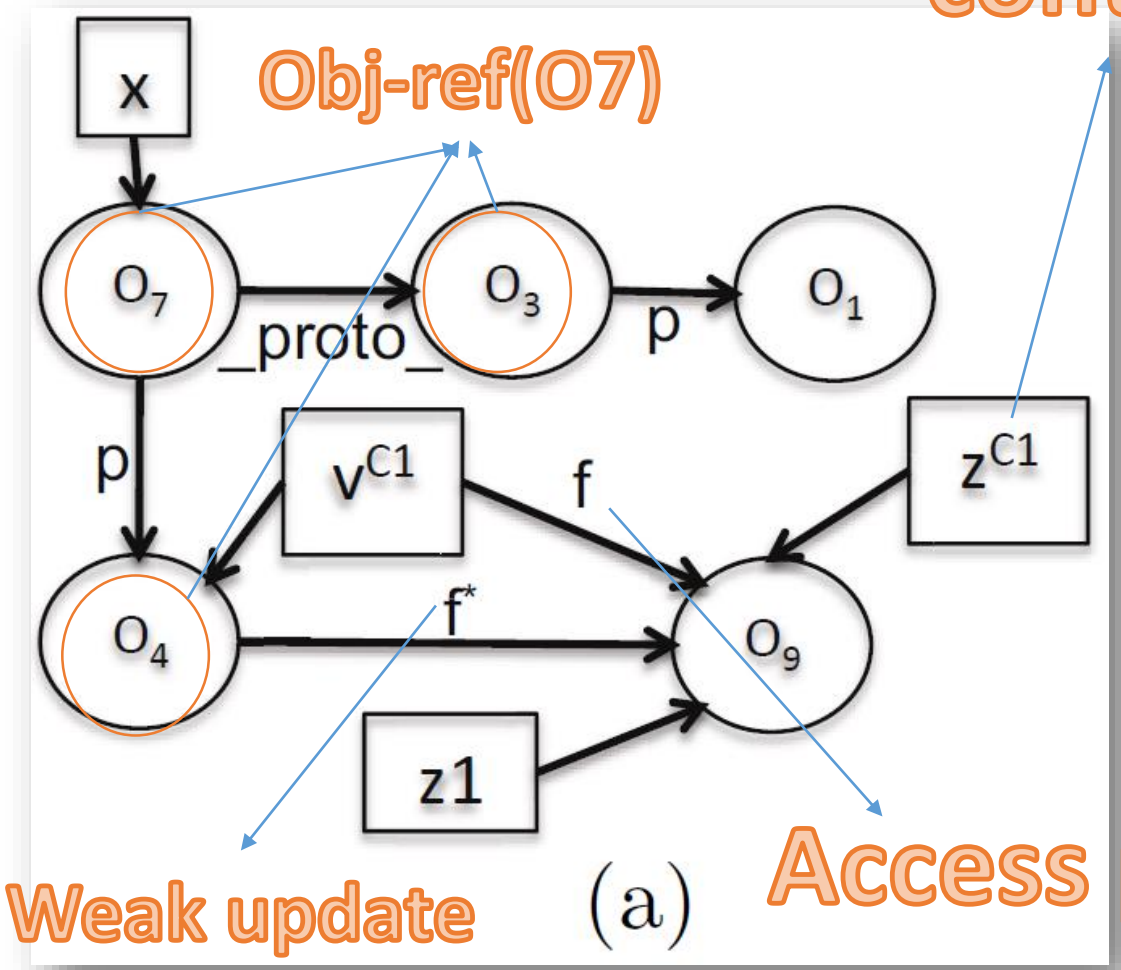
p:O4,O1,

__proto__:O3}

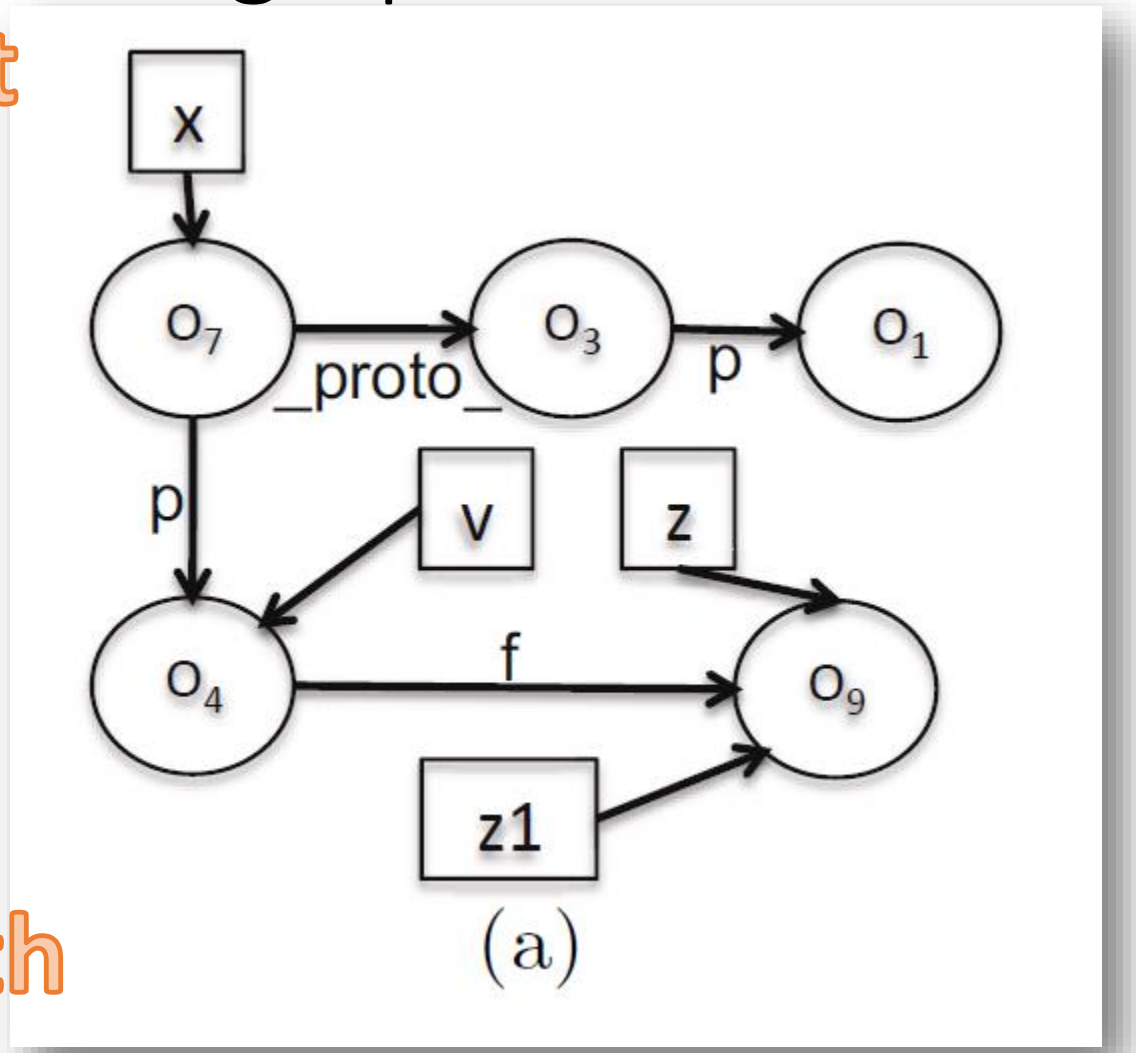
```
 8  x.bar = function(v, z){ v.f = z; }
 9  var z1 = new Z();
10  x.bar(x.p, z1);
11  ...
12  x.p = new A();
13  ...
14  var z2 = new Z();
15  x.bar(x.p,z2);
```

Fig. 2. JavaScript example

Compare with the ground truth graph



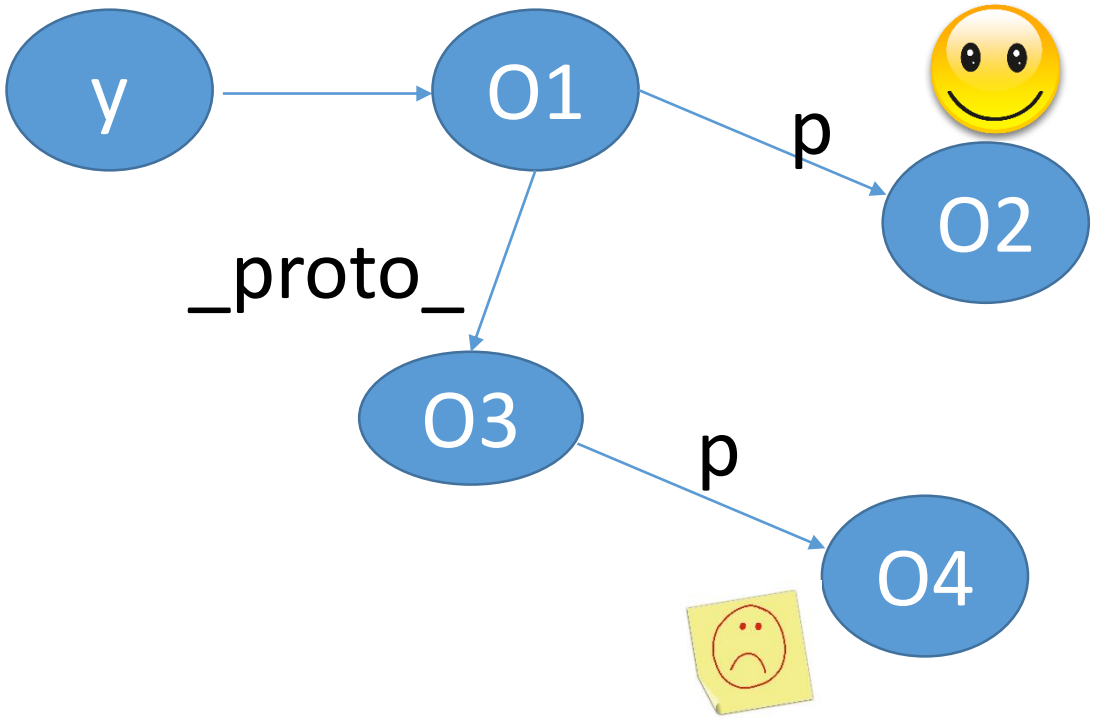
New Points-to graph at line 10 (the authors' approach) Figure 7(a)



Run-time Points-to graph at line 10 Fig 3(a)

Measurement (# of return objects)

x = y.p:
Return {O2} better than return {O2,O4}



x = y.p/x=y.p(…)
[statement s/context c]
REF(s,c) = How many objects will be returned Through lookuping

The smaller #, the better precision

Table 4. *REF* analysis precision

Website	*Corr*			*CorrBSSS*		
	1	2-4	≥ 5	1	2-4	≥ 5
facebook.com	38%	52%	10%	50%	47%	3%
google.com	32%	51%	17%	53%	42%	5%
youtube.com	41%	47%	12%	54%	41%	5%
yahoo.com	48%	46%	6%	52%	45%	3%
wikipedia.org	29%	45%	26%	43%	39%	18%
amazon.com	45%	52%	3%	46%	51%	3%
twitter.com	32%	53%	15%	39%	49%	12%
blogspot.com	35%	34%	31%	53%	36%	11%
linkedin.com	34%	49%	17%	44%	50%	6%
msn.com	40%	36%	24%	48%	37%	15%
ebay.com	30%	40%	30%	46%	40%	14%
bing.com	41%	34%	25%	54%	37%	9%
Geom. Mean	**37%**	**44%**	**15%**	**48%**	**43%**	**7%**

Corr: correlation tracking …
CorrBSSS: the authors' appraoch

Better performance
48-37 % =11%

Overhead (acceptable)

Table 5. *REF* analysis cost (in seconds) on average per webpage

Website	*Corr*	*CorrBSSS*	overhead
facebook	17.4	45.9	163%
google	13.0	30.4	134%
youtube	31.2	75.3	141%
yahoo	28.5	54.1	90%
wiki	16.0	40.1	151%
amazon	15.1	24.2	61%
twitter	38.1	94.5	148%
blog	15.9	42.4	137%
linkedin	27.8	62.0	167%
msn	34.4	57.9	68%
ebay	8.3	27.2	227%
bing	22.1	50.4	128%
Geom. Mean	**20.4**	**46.7**	**127%**

Average overhead

Q&A

- What is “state-sensitive”? and the relation with context sensitivity.

Context sensitivity
Object ….
Call-site …
State …

- Others….