

Incremental Algorithms for Inter-procedural Analysis of Safety Properties

Christopher L. Conway
Columbia University

Joint work with:
Kedar S. Namjoshi (Bell Labs),
Dennis Dams (Bell Labs), and
Stephen A. Edwards (Columbia University)

Why Incremental Analysis?

- ◆ Programming is incremental: analyze early and often

Why Incremental Analysis?

- ◆ Programming is incremental: analyze early and often
- ◆ Speed: faster, cheaper, better analysis

Why Incremental Analysis?

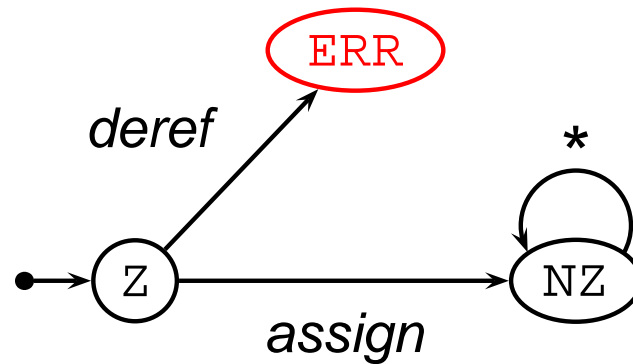
- ◆ Programming is incremental: analyze early and often
- ◆ Speed: faster, cheaper, better analysis
- ◆ “Continuous verification” (cf. continuous testing [Saff & Ernst 2003])

Safety Properties

- ◆ Properties of the form “Always not ERROR”

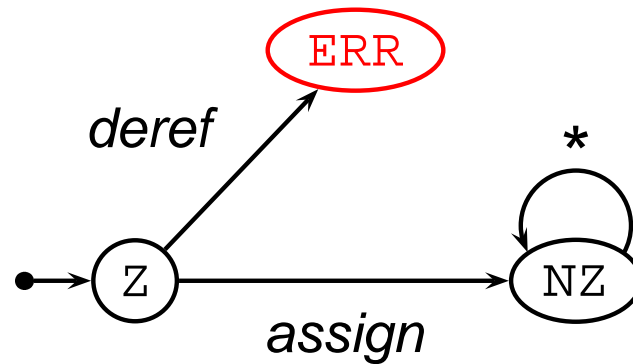
Safety Properties

- ◆ Properties of the form “Always not ERROR”
- ◆ Modelled as an observer automaton.



Safety Properties

- ◆ Properties of the form “Always not ERROR”
- ◆ Modelled as an observer automaton.



- ◆ Inter-procedural program analysis ==
reachability on Recursive State Machines

Previous Work

- ◆ Push-down model checking [Schmidt & Steffen 1998;
Alur, Etessami & Yannakakis 2001; Benedikt, Godefroid & Reps 2001]
- ◆ Incremental analysis
 - Model checking
[Sokolsky & Smolka 1993; Henzinger, Jhala, Majumdar & Sanvido 2003]
 - Dataflow analysis
[Ryder & Marlowe 1990; Yur, Ryder, Landi & Stocks 1997; ...]
 - Logic programs [Saha & Ramakrishnan 2003]

Previous Work

- ◆ Push-down model checking [Schmidt & Steffen 1998; Alur, Etessami & Yannakakis 2001; Benedikt, Godefroid & Reps 2001]
- ◆ Incremental analysis
 - Model checking [Sokolsky & Smolka 1993; Henzinger, Jhala, Majumdar & Sanvido 2003]
 - Dataflow analysis [Ryder & Marlowe 1990; Yur, Ryder, Landi & Stocks 1997; ...]
 - Logic programs [Saha & Ramakrishnan 2003]

Our Contributions

- ◆ Two incremental algorithms for RSM reachability
 - “forward” (top-down from entry function)
 - “backward” (inside-out from program changes)
 - Both algorithms precise

Previous Work

- ◆ Push-down model checking [Schmidt & Steffen 1998; Alur, Etessami & Yannakakis 2001; Benedikt, Godefroid & Reps 2001]
- ◆ Incremental analysis
 - Model checking [Sokolsky & Smolka 1993; Henzinger, Jhala, Majumdar & Sanvido 2003]
 - Dataflow analysis [Ryder & Marlowe 1990; Yur, Ryder, Landi & Stocks 1997; ...]
 - Logic programs [Saha & Ramakrishnan 2003]

Our Contributions

- ◆ Two incremental algorithms for RSM reachability
 - “forward” (top-down from entry function)
 - “backward” (inside-out from program changes)
 - Both algorithms precise
- ◆ Implementation in the Orion static analyzer (Bell Labs)

Previous Work

- ◆ Push-down model checking [Schmidt & Steffen 1998; Alur, Etessami & Yannakakis 2001; Benedikt, Godefroid & Reps 2001]
- ◆ Incremental analysis
 - Model checking [Sokolsky & Smolka 1993; Henzinger, Jhala, Majumdar & Sanvido 2003]
 - Dataflow analysis [Ryder & Marlowe 1990; Yur, Ryder, Landi & Stocks 1997; ...]
 - Logic programs [Saha & Ramakrishnan 2003]

Our Contributions

- ◆ Two incremental algorithms for RSM reachability
 - “forward” (top-down from entry function)
 - “backward” (inside-out from program changes)
 - Both algorithms precise
- ◆ Implementation in the Orion static analyzer (Bell Labs)
- ◆ Speedups to 17.5 (avg. 8.2)

The Full (Non-Incremental) Algorithm

Inter-procedural Safety Analysis

[Reps, Horwitz & Sagiv 1995, Alur, Etassami & Yannakakis 2001]

The Full (Non-Incremental) Algorithm

Inter-procedural Safety Analysis

[Reps, Horwitz & Sagiv 1995, Alur, Etassami & Yannakakis 2001]

- ♦ A state = a control location + an automaton state

The Full (Non-Incremental) Algorithm

Inter-procedural Safety Analysis

[Reps, Horwitz & Sagiv 1995, Alur, Etassami & Yannakakis 2001]

- ◆ A state = a control location + an automaton state
- ◆ Inside a procedure, straight-forward state space reachability.

The Full (Non-Incremental) Algorithm

Inter-procedural Safety Analysis

[Reps, Horwitz & Sagiv 1995, Alur, Etassami & Yannakakis 2001]

- ◆ A state = a control location + an automaton state
- ◆ Inside a procedure, straight-forward state space reachability.
- ◆ At a function call:
 - Add state to function call site table.
 - Explore the called function.

The Full (Non-Incremental) Algorithm

Inter-procedural Safety Analysis

[Reps, Horwitz & Sagiv 1995, Alur, Etassami & Yannakakis 2001]

- ◆ A state = a control location + an automaton state
- ◆ Inside a procedure, straight-forward state space reachability.
- ◆ At a function call:
 - Add state to function call site table.
 - Explore the called function.
- ◆ At a function exit:
 - Add automaton state pair $\langle entry, exit \rangle$ to function summary table.
 - Notify call sites of new exit state.

The Full (Non-Incremental) Algorithm

Inter-procedural Safety Analysis

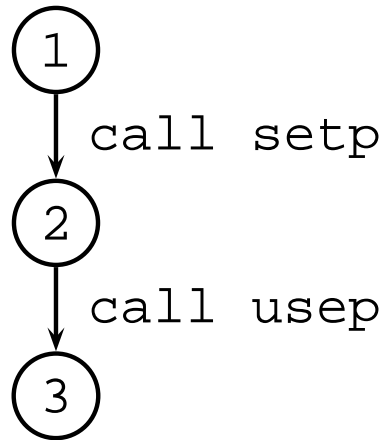
[Reps, Horwitz & Sagiv 1995, Alur, Etassami & Yannakakis 2001]

- ◆ A state = a control location + an automaton state
- ◆ Inside a procedure, straight-forward state space reachability.
- ◆ At a function call:
 - Add state to function call site table.
 - Explore the called function.
- ◆ At a function exit:
 - Add automaton state pair $\langle entry, exit \rangle$ to function summary table.
 - Notify call sites of new exit state.
- ◆ Iterate to a fixpoint.

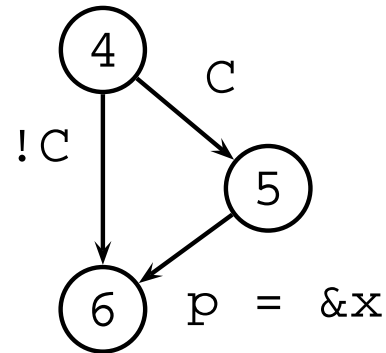
The Full Algorithm (Example)

```
int *p, x, y ;  
setp() { if C then p = &x ; }  
usep() { y = *p ; }  
main() { setp() ; usep() ; }
```

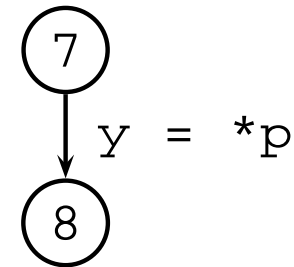
main:



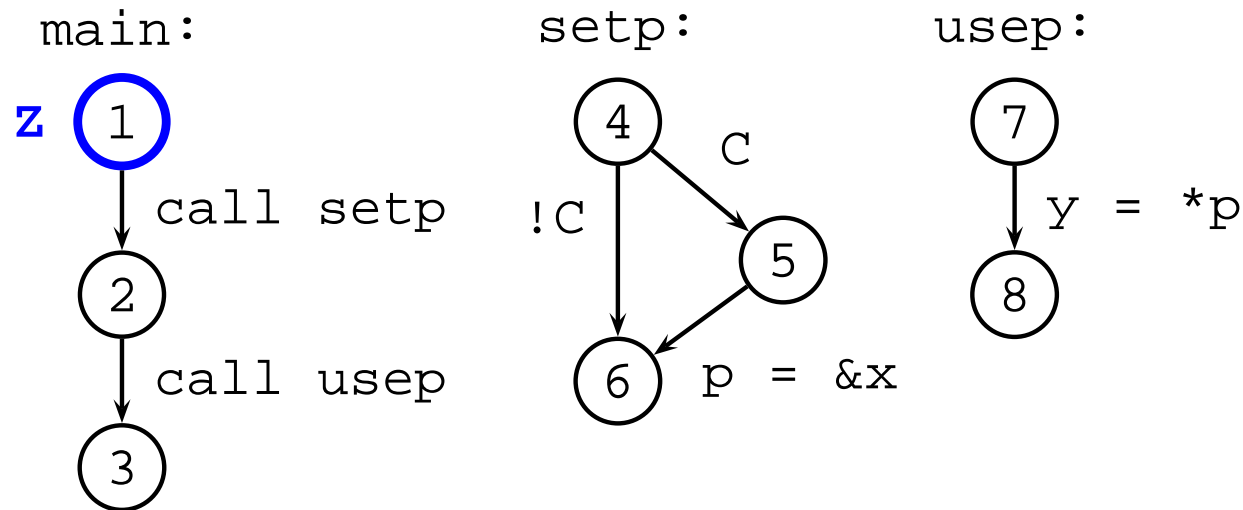
setp:



usep:

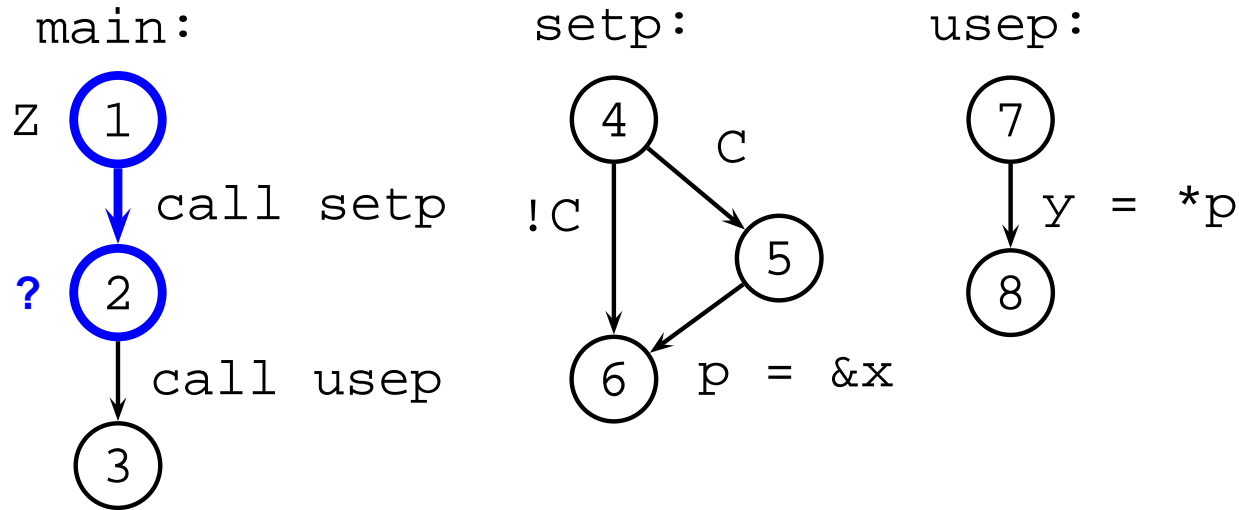


The Full Algorithm (Example)



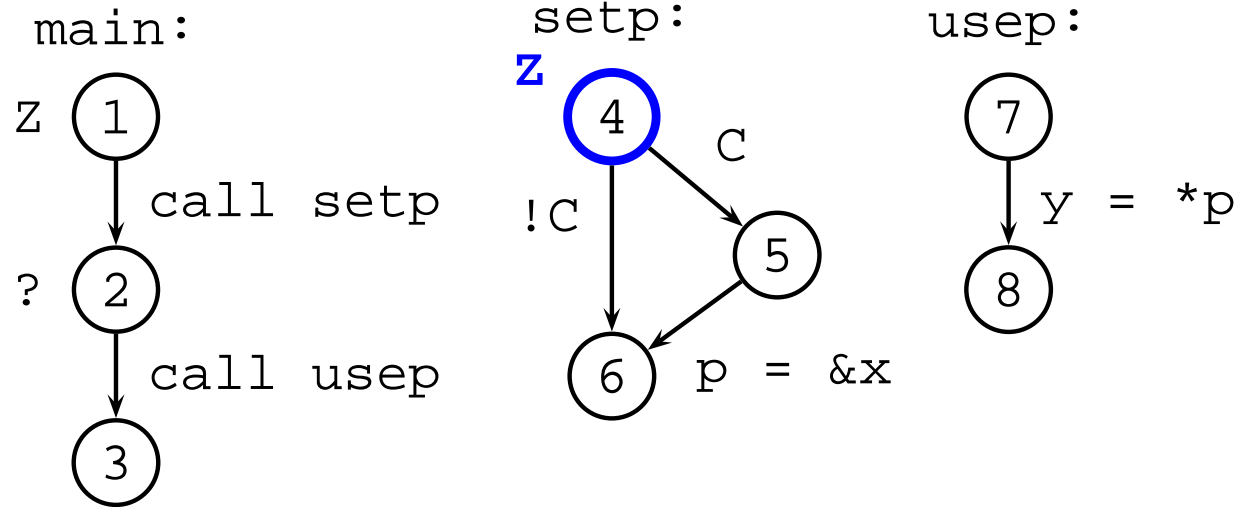
	setp	usep
Summaries:		
Call Sites:		

The Full Algorithm (Example)



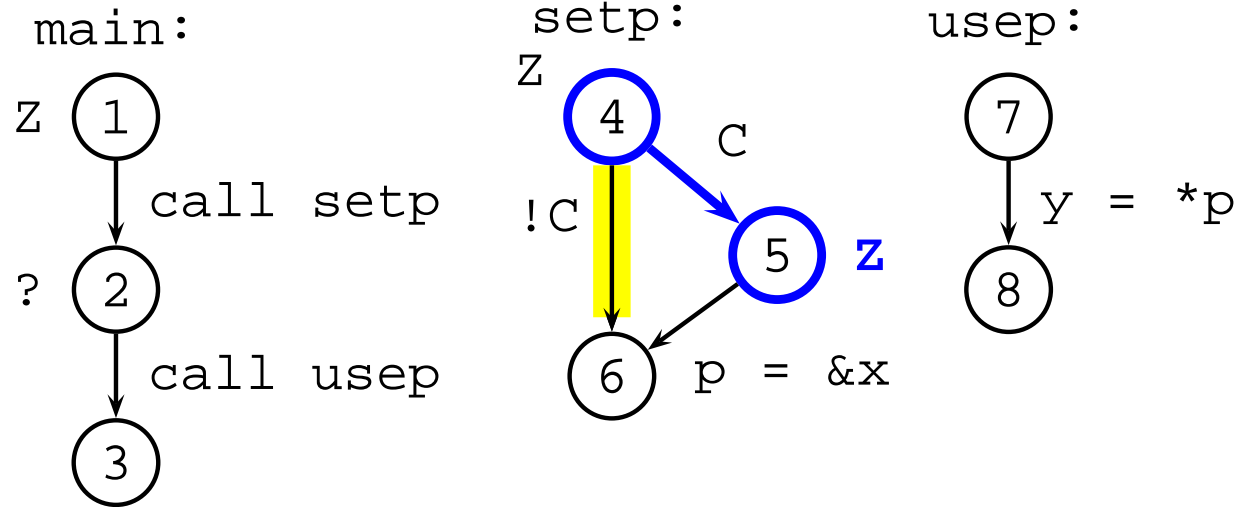
	setp	usep
Summaries:		
Call Sites:		

The Full Algorithm (Example)



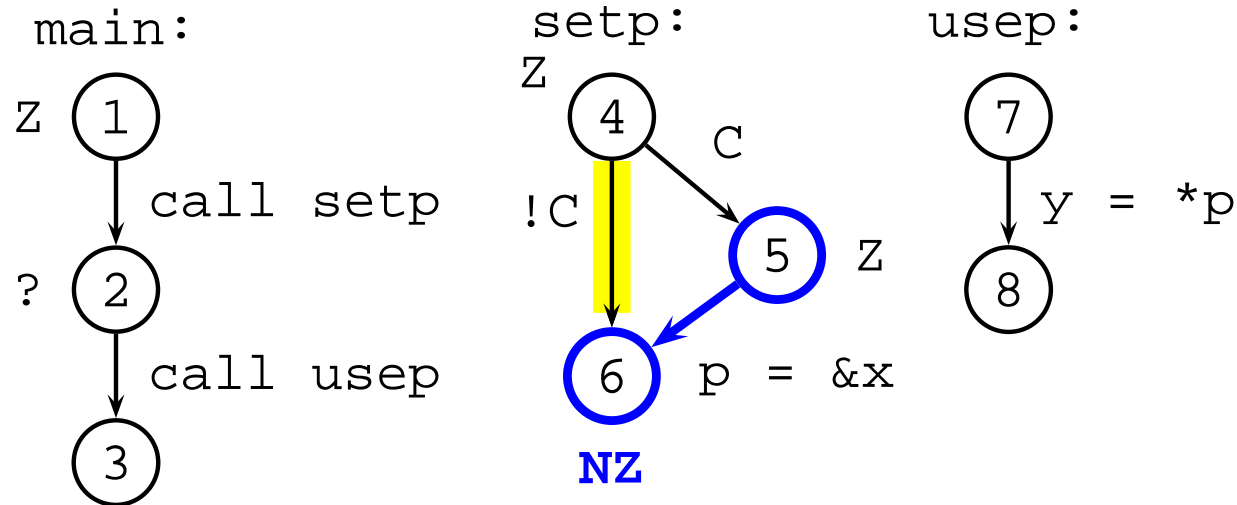
	setp	usep
Summaries:		
Call Sites:	(1, z)	

The Full Algorithm (Example)



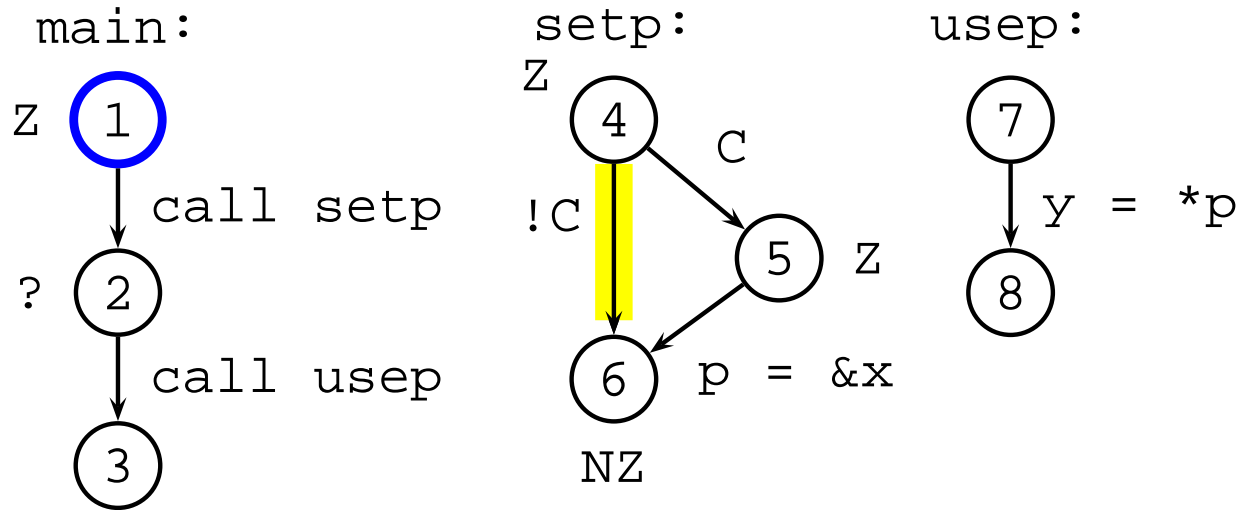
	setp	usep
Summaries:		
Call Sites:	(1, z)	

The Full Algorithm (Example)



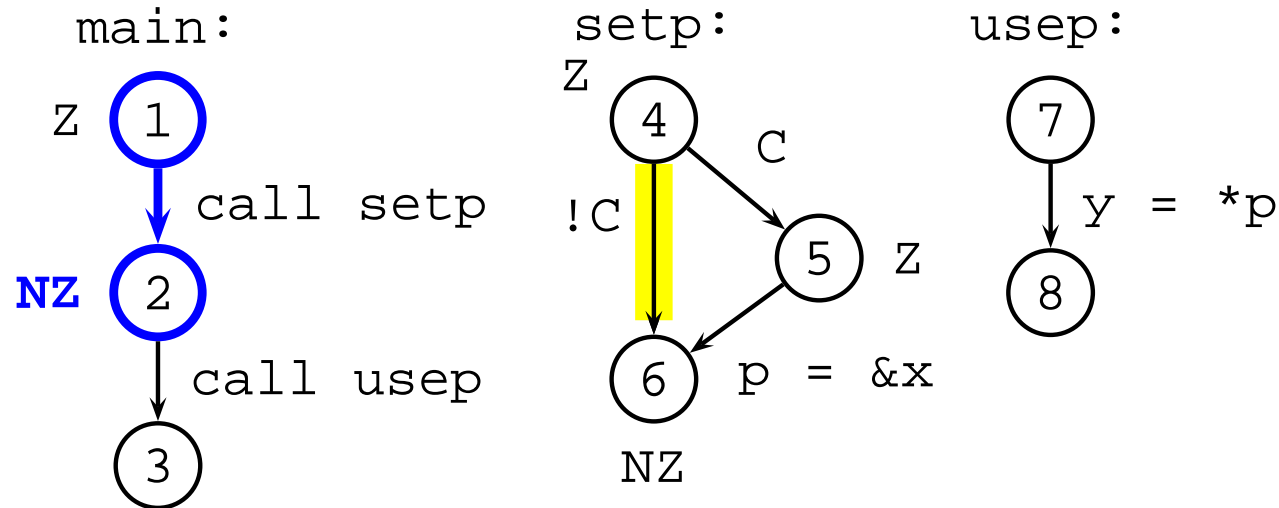
	setp	usep
Summaries:	$\langle \mathbf{z}, \mathbf{NZ} \rangle$	
Call Sites:	(1, z)	

The Full Algorithm (Example)



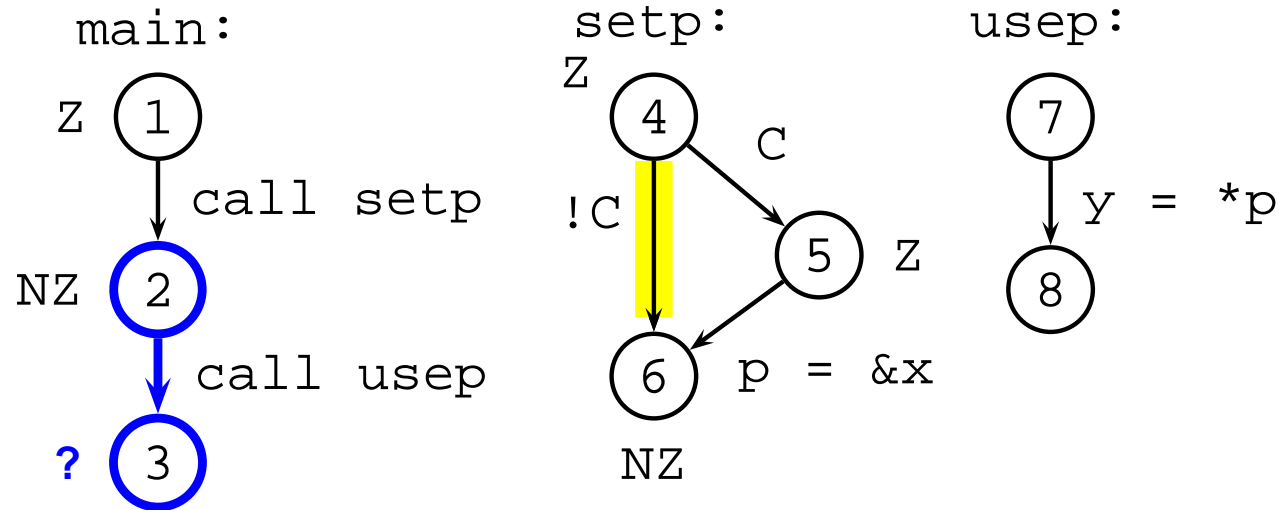
	setp	usep
Summaries:	$\langle Z, NZ \rangle$	
Call Sites:	(1, Z)	

The Full Algorithm (Example)



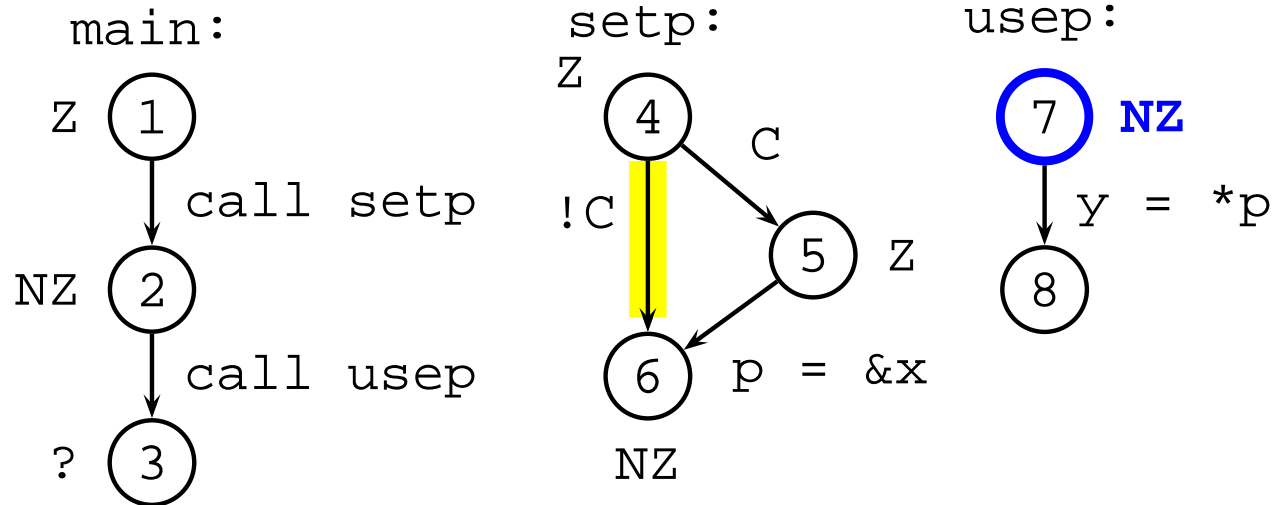
	setp	usep
Summaries:	$\langle Z, NZ \rangle$	
Call Sites:	(1, Z)	

The Full Algorithm (Example)



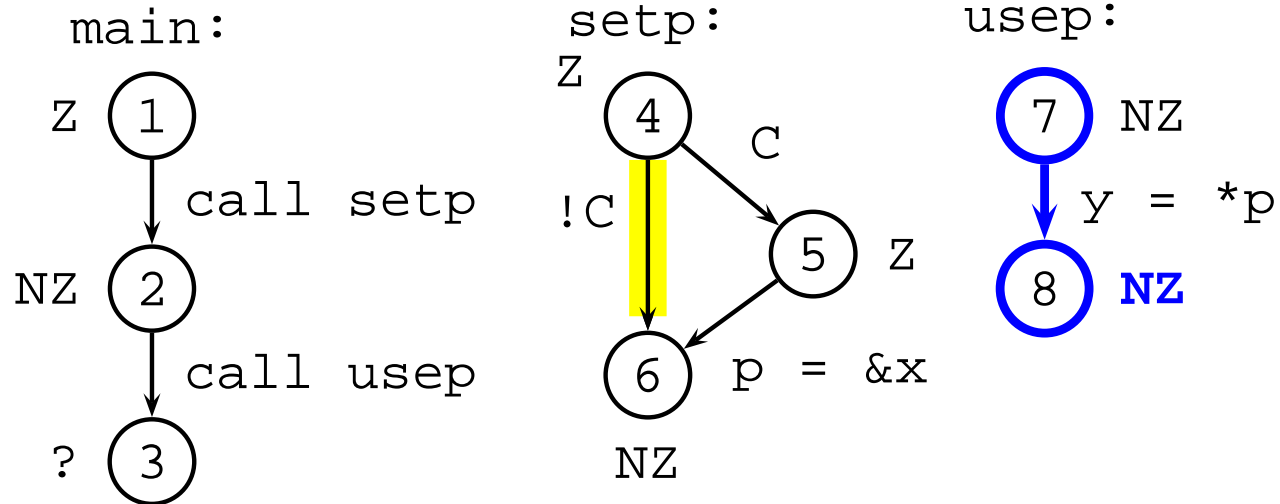
	setp	usep
Summaries:	$\langle Z, NZ \rangle$	
Call Sites:	$(1, Z)$	

The Full Algorithm (Example)



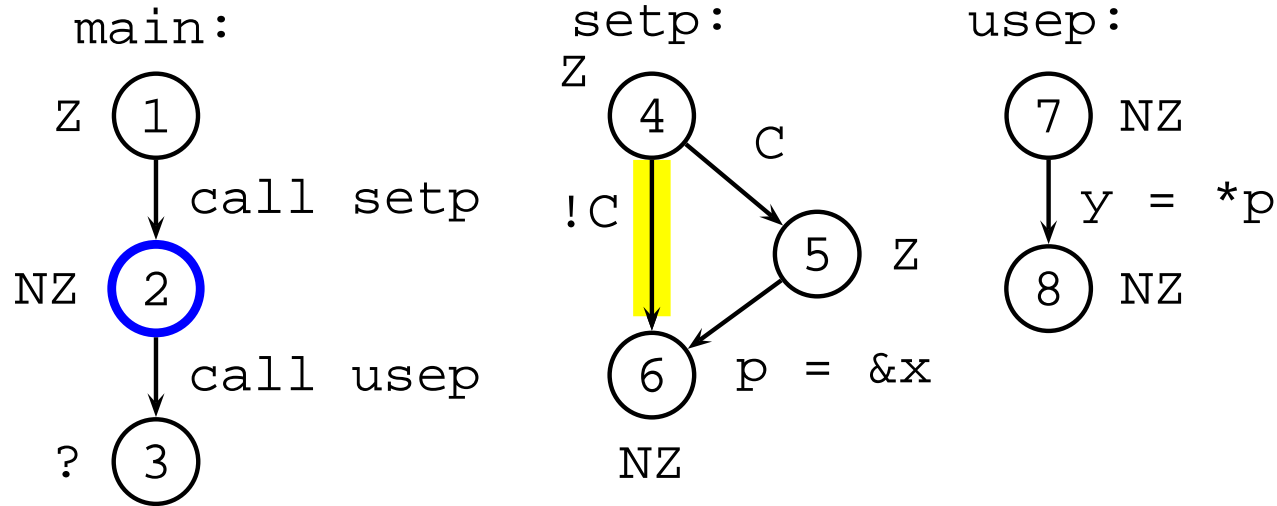
	setp	usep
Summaries:	$\langle Z, NZ \rangle$	
Call Sites:	$(1, Z)$	$(2, \mathbf{NZ})$

The Full Algorithm (Example)



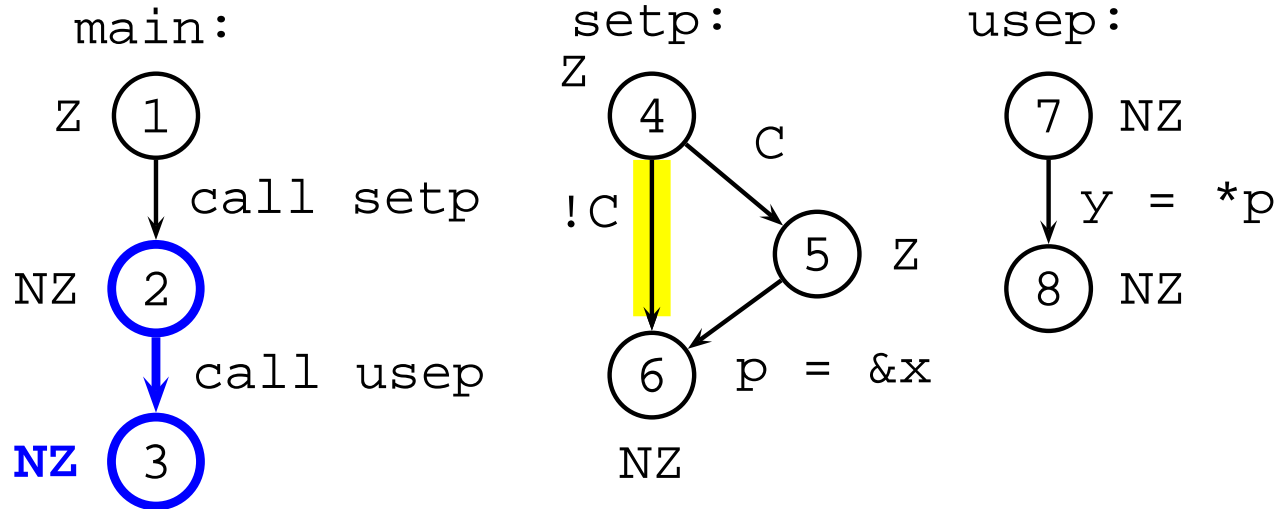
	setp	usep
Summaries:	$\langle Z, NZ \rangle$	$\langle \mathbf{NZ}, \mathbf{NZ} \rangle$
Call Sites:	(1, Z)	(2, NZ)

The Full Algorithm (Example)



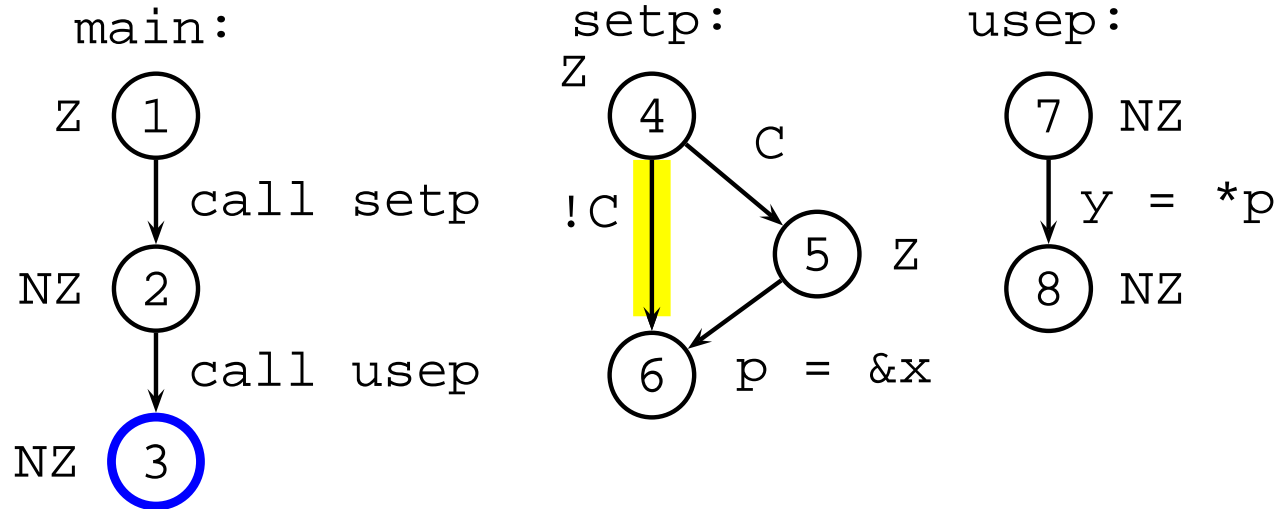
	setp	usep
Summaries:	$\langle Z, NZ \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	$(1, Z)$	$(2, NZ)$

The Full Algorithm (Example)



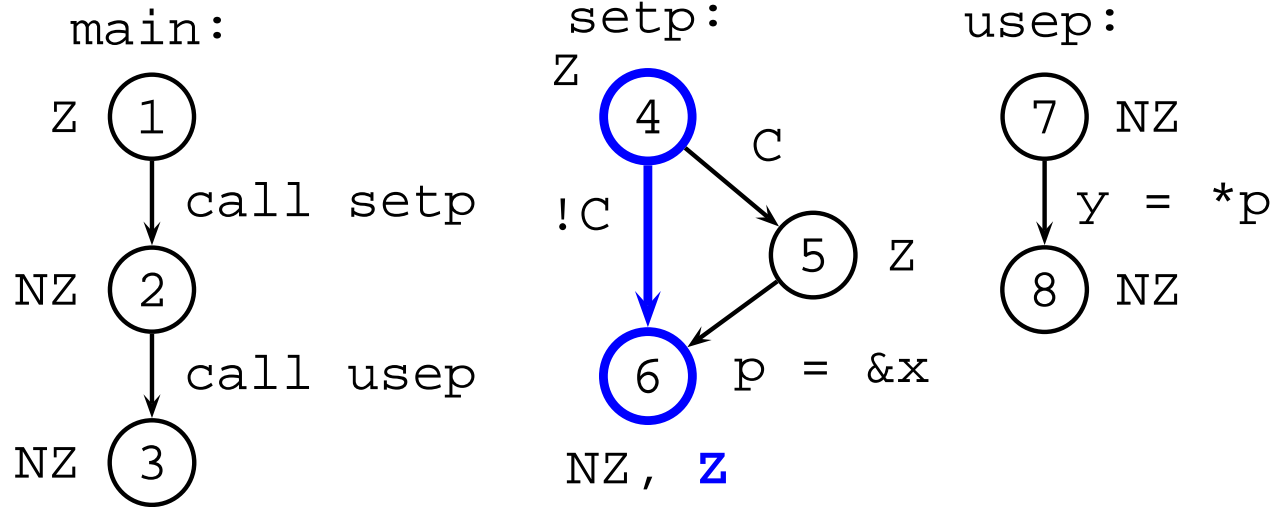
	setp	usep
Summaries:	$\langle Z, NZ \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	(1, Z)	(2, NZ)

The Full Algorithm (Example)



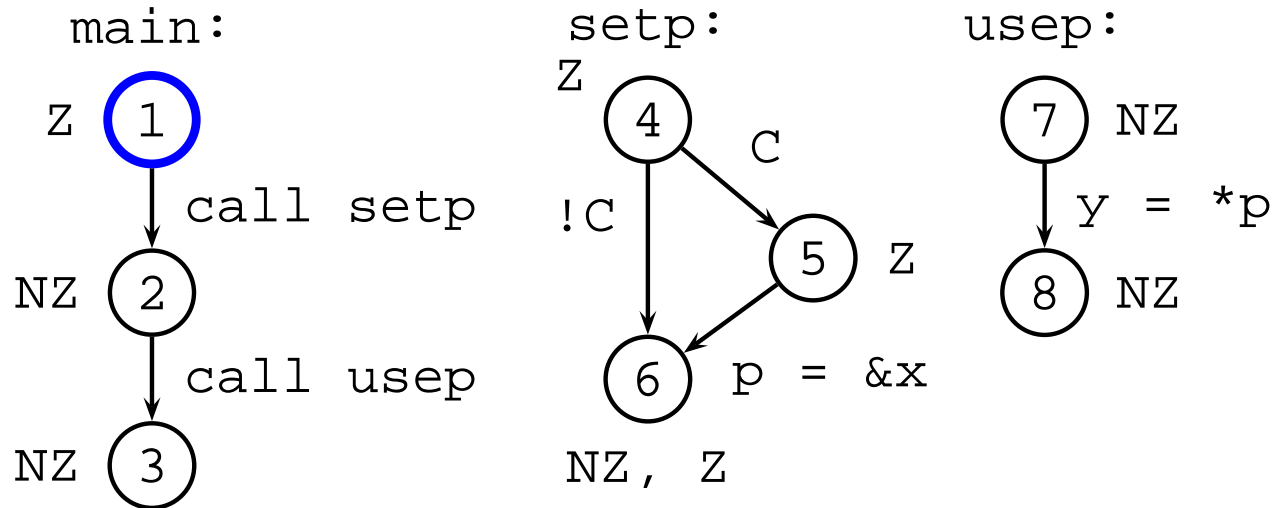
	setp	usep
Summaries:	$\langle Z, NZ \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	(1, Z)	(2, NZ)

The Full Algorithm (Example)



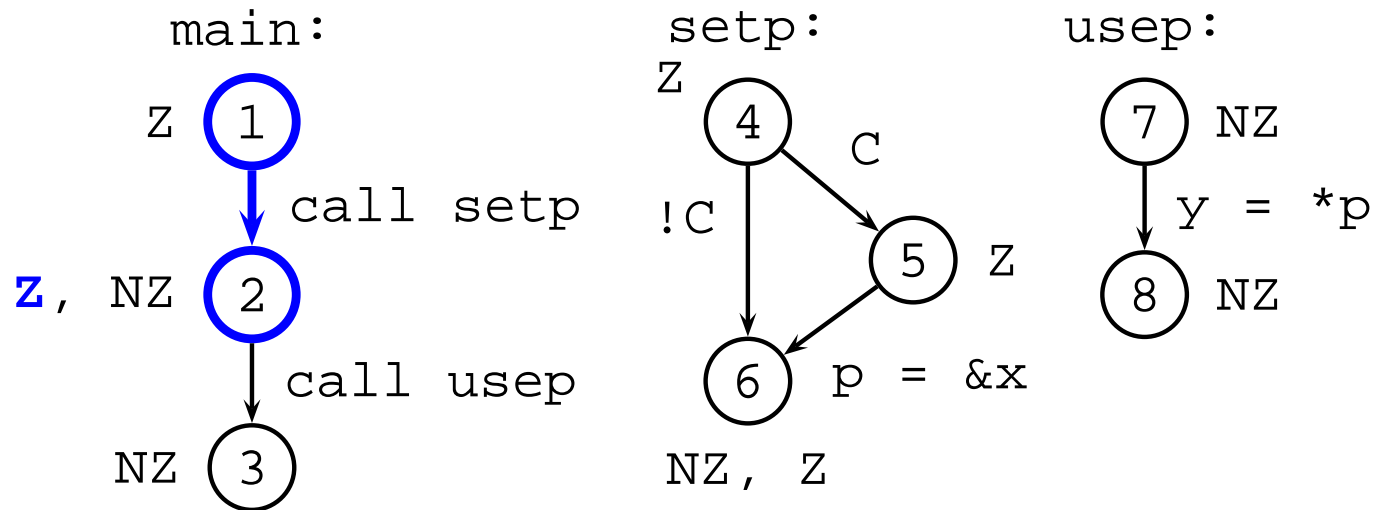
	setp	usep
Summaries:	$\langle z, NZ \rangle$ $\langle \mathbf{z}, \mathbf{z} \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	(1, z)	(2, NZ)

The Full Algorithm (Example)



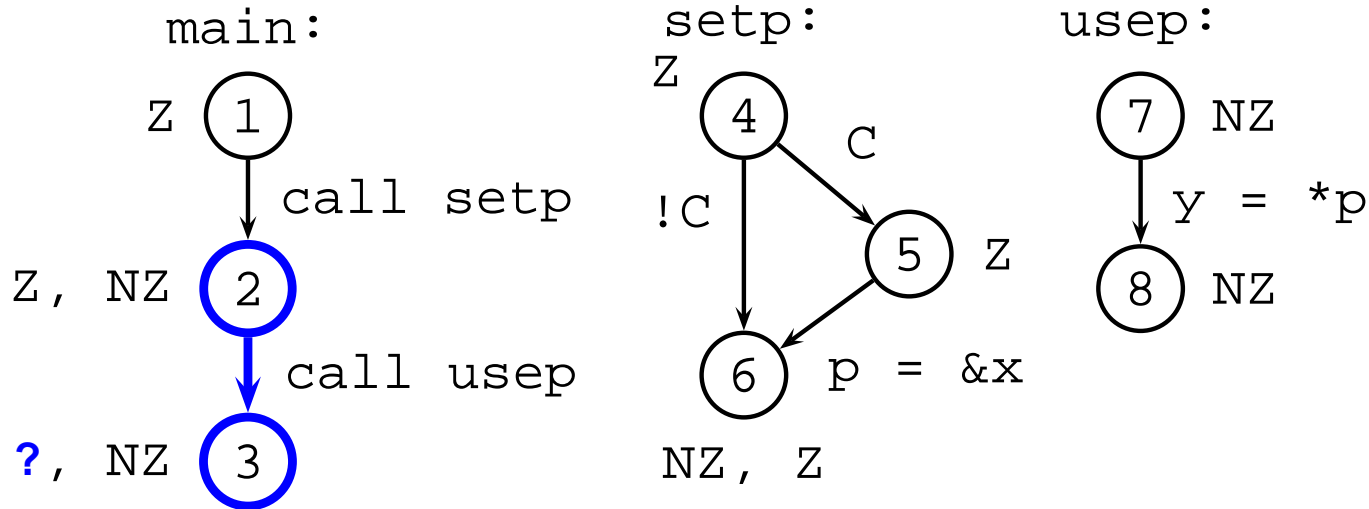
	setp	usep
Summaries:	$\langle Z, NZ \rangle$ $\langle Z, Z \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	$(1, Z)$	$(2, NZ)$

The Full Algorithm (Example)



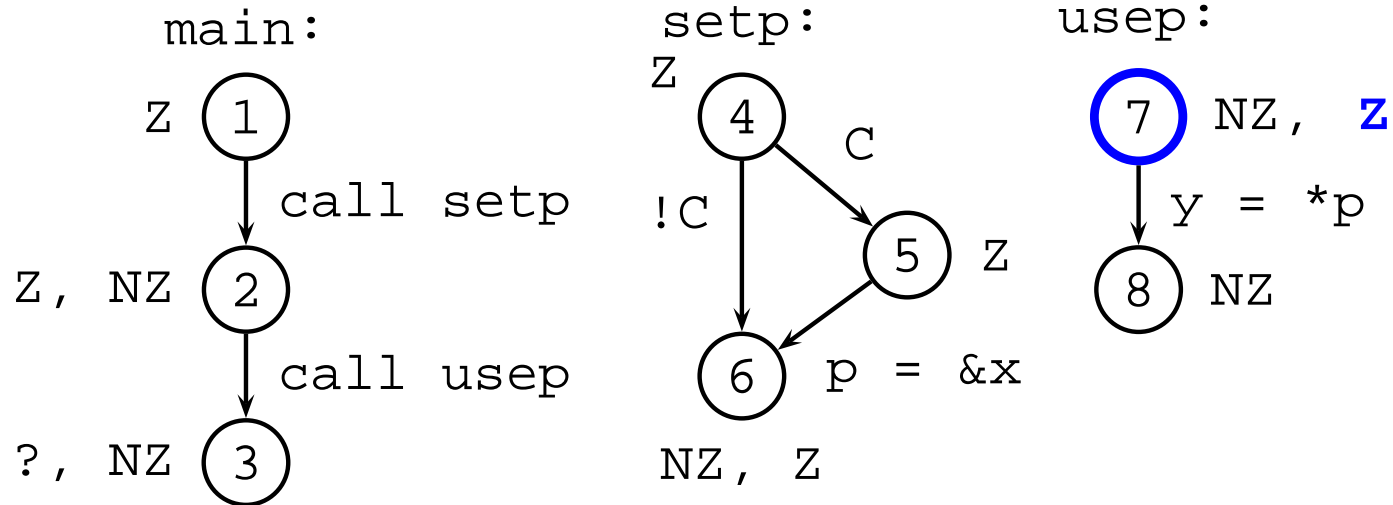
	setp	usep
Summaries:	$\langle Z, NZ \rangle$ $\langle Z, Z \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	(1, Z)	(2, NZ)

The Full Algorithm (Example)



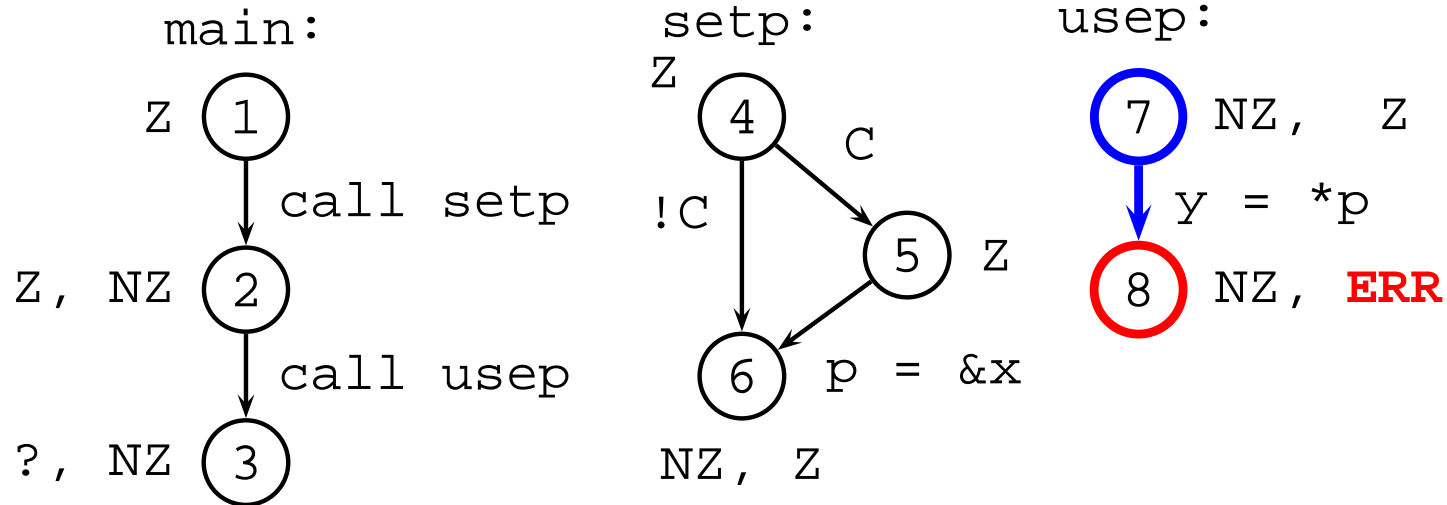
	setp	usep
Summaries:	$\langle z, NZ \rangle$ $\langle z, z \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	(1, z)	(2, NZ)

The Full Algorithm (Example)



	setp	usep
Summaries:	$\langle z, NZ \rangle$ $\langle z, z \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	(1, z)	(2, NZ) (2, z)

The Full Algorithm (Example)

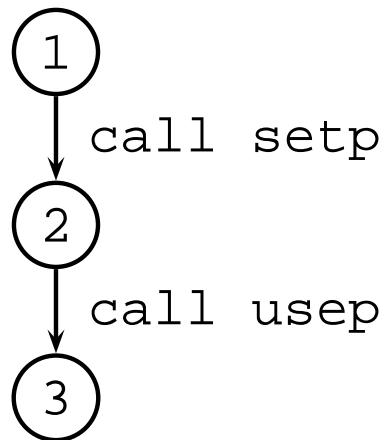


	setp	usep
Summaries:	$\langle z, NZ \rangle$ $\langle z, z \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	(1, z)	(2, NZ) (2, z)

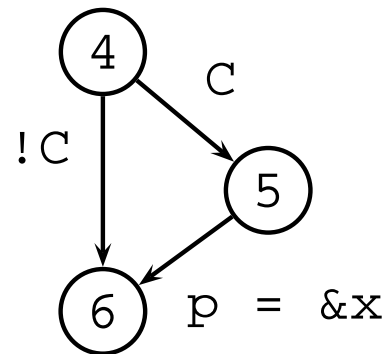
Changing Code

```
int *p, x, y ;  
setp() { if C then p = &x ; }  
usep() { y = *p ; }  
main() { setp() ; usep() ; }
```

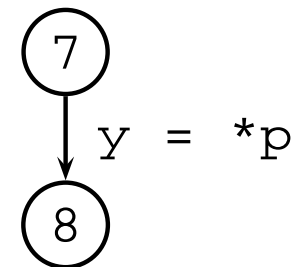
main:



setp:



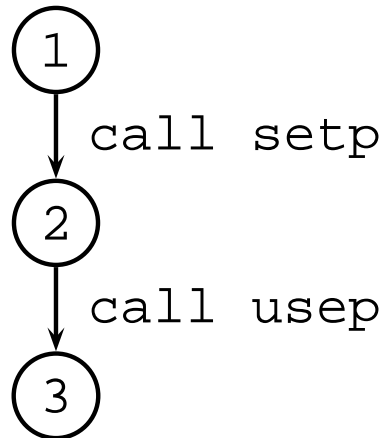
usep:



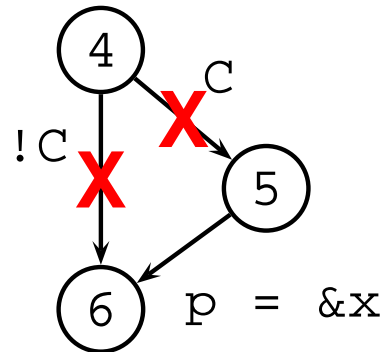
Changing Code

```
int *p, x, y ;  
setp() {           p = &x ; }  
usep() { y = *p ; }  
main() { setp() ; usep() ; }
```

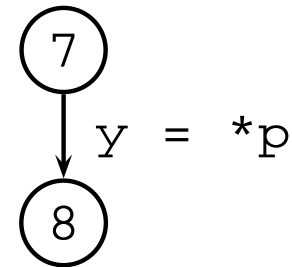
main:



setp:



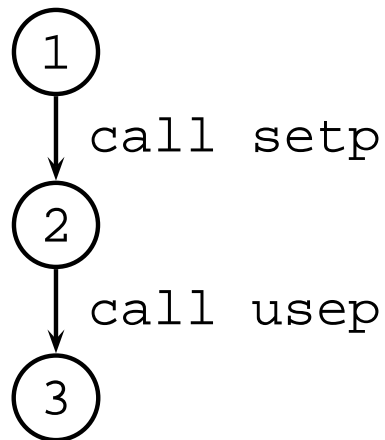
usep:



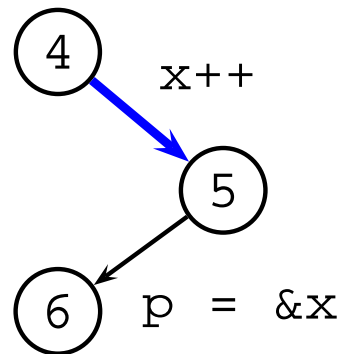
Changing Code

```
int *p, x, y ;  
setp() { x++ ; p = &x ; }  
usep() { y = *p ; }  
main() { setp() ; usep() ; }
```

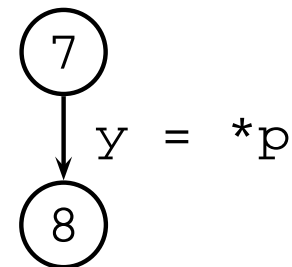
main:



setp:

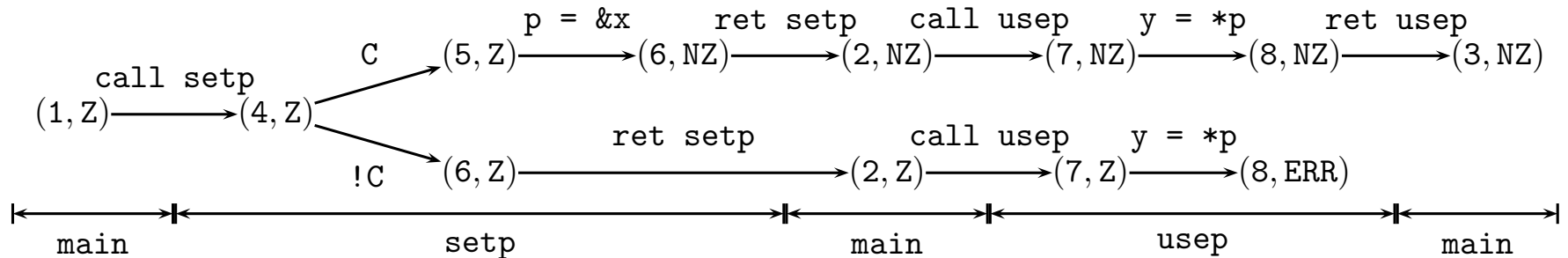


usep:



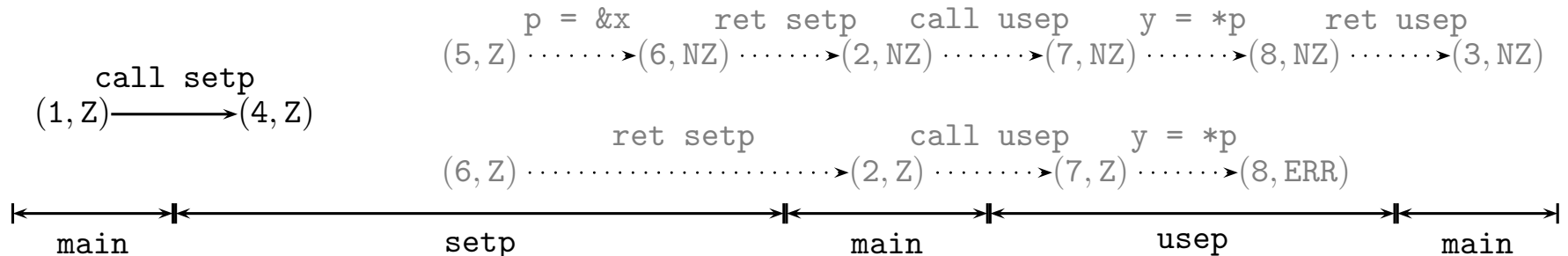
Handling Changes

- ◆ Store a “derivation graph” (all the data we computed in the analysis)



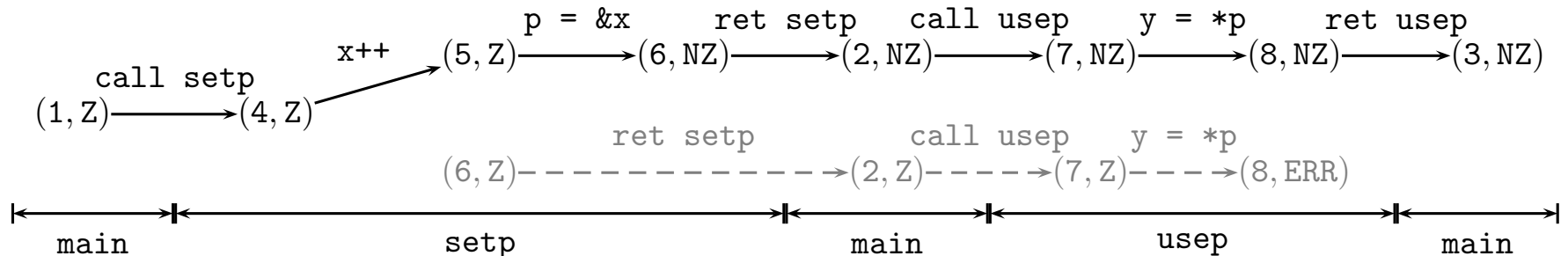
Handling Changes

- ◆ Store a “derivation graph” (all the data we computed in the analysis)
- ◆ On deletion:
 - portions of the graph are pruned



Handling Changes

- ◆ Store a “derivation graph” (all the data we computed in the analysis)
- ◆ On deletion:
 - portions of the graph are pruned
- ◆ Effects of addition:
 - new states are derived
 - pruned derivations are re-attached



The Forward (Incremental) Algorithm

- ◆ Check the previously computed derivation graph.
- ◆ Save time on re-computing transitions.
- ◆ A “mark and sweep” approach to reachability.

The Forward (Incremental) Algorithm

- ◆ Check the previously computed derivation graph.
 - ◆ Save time on re-computing transitions.
 - ◆ A “mark and sweep” approach to reachability.
1. Mark all states, summaries, and call sites unreachable.

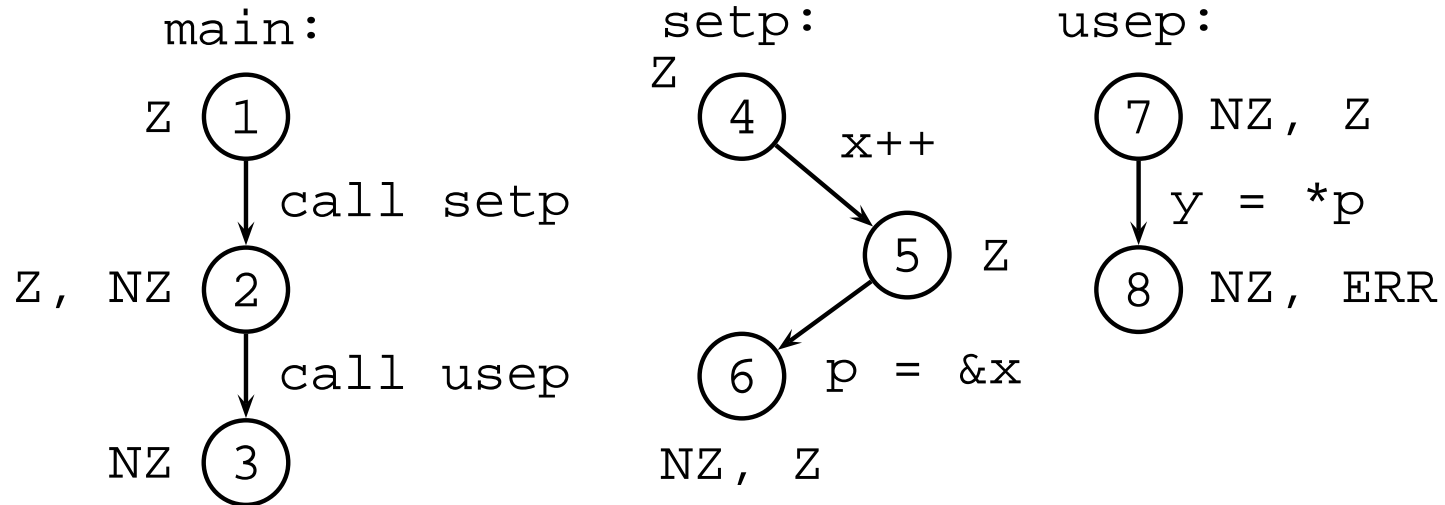
The Forward (Incremental) Algorithm

- ◆ Check the previously computed derivation graph.
 - ◆ Save time on re-computing transitions.
 - ◆ A “mark and sweep” approach to reachability.
1. Mark all states, summaries, and call sites unreachable.
 2. Re-explore state space starting at main, marking items re-discovered as reachable.

The Forward (Incremental) Algorithm

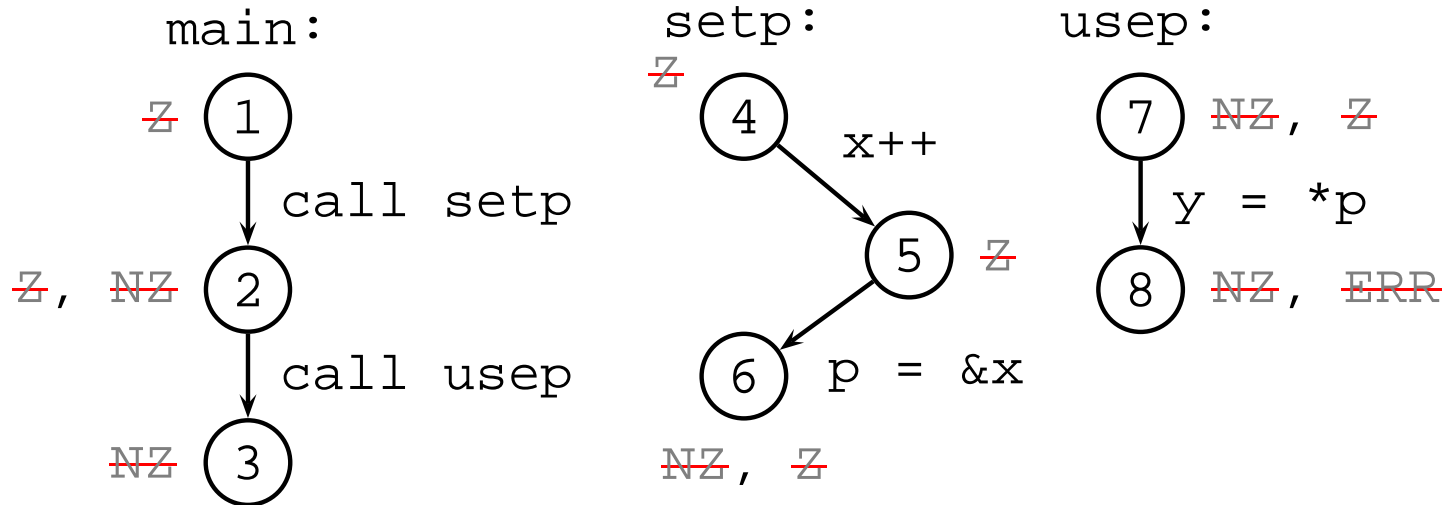
- ◆ Check the previously computed derivation graph.
 - ◆ Save time on re-computing transitions.
 - ◆ A “mark and sweep” approach to reachability.
1. Mark all states, summaries, and call sites unreachable.
 2. Re-explore state space starting at main, marking items re-discovered as reachable.
 3. Remove unreachable items.

Forward Algorithm (Example)



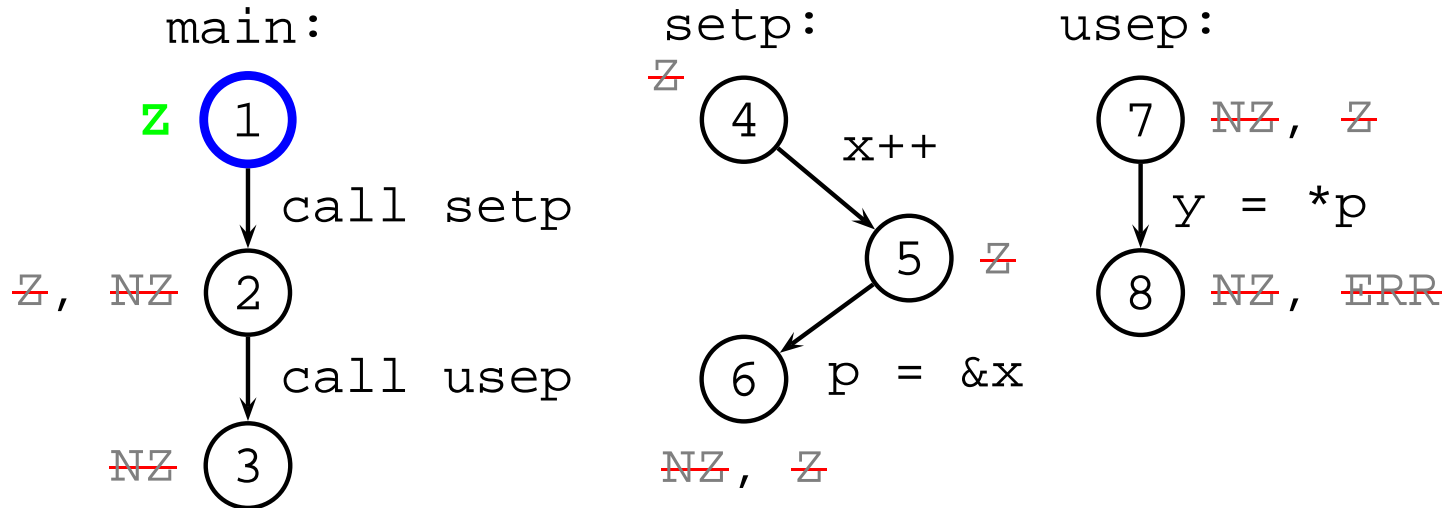
	setp	usep
Summaries:	$\langle z, NZ \rangle$ $\langle z, z \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	(1, z)	(2, NZ) (2, z)

Forward Algorithm (Example)



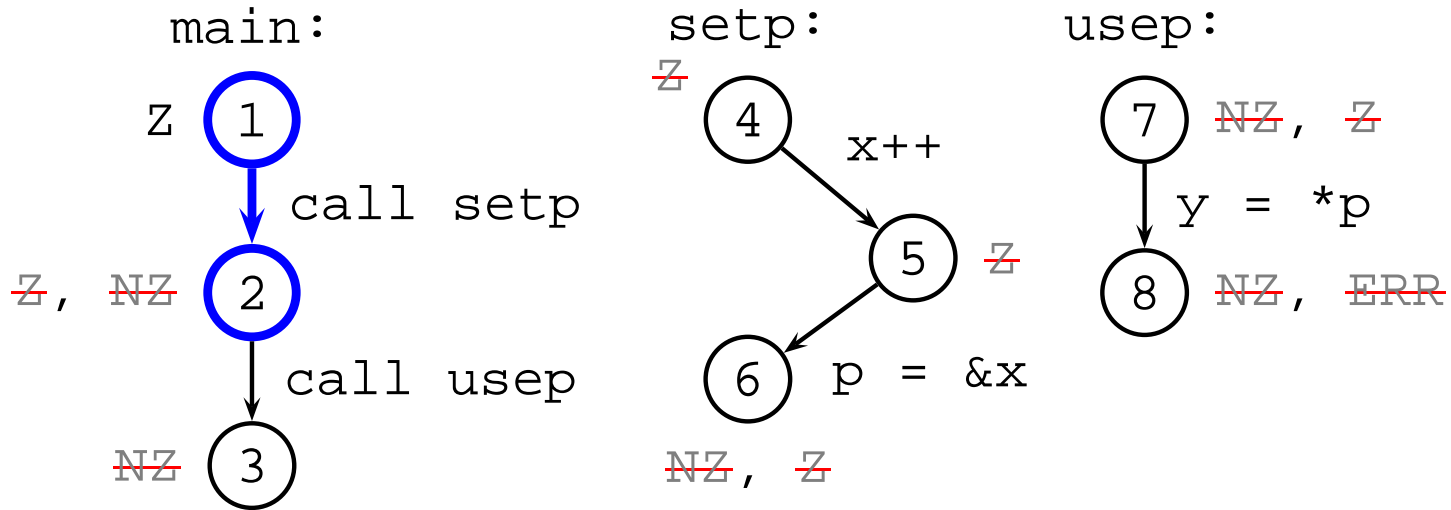
	setp	usep
Summaries:	<z, NZ> <z, z>	<NZ, NZ>
Call Sites:	(1, z)	(2, NZ) (2, z)

Forward Algorithm (Example)



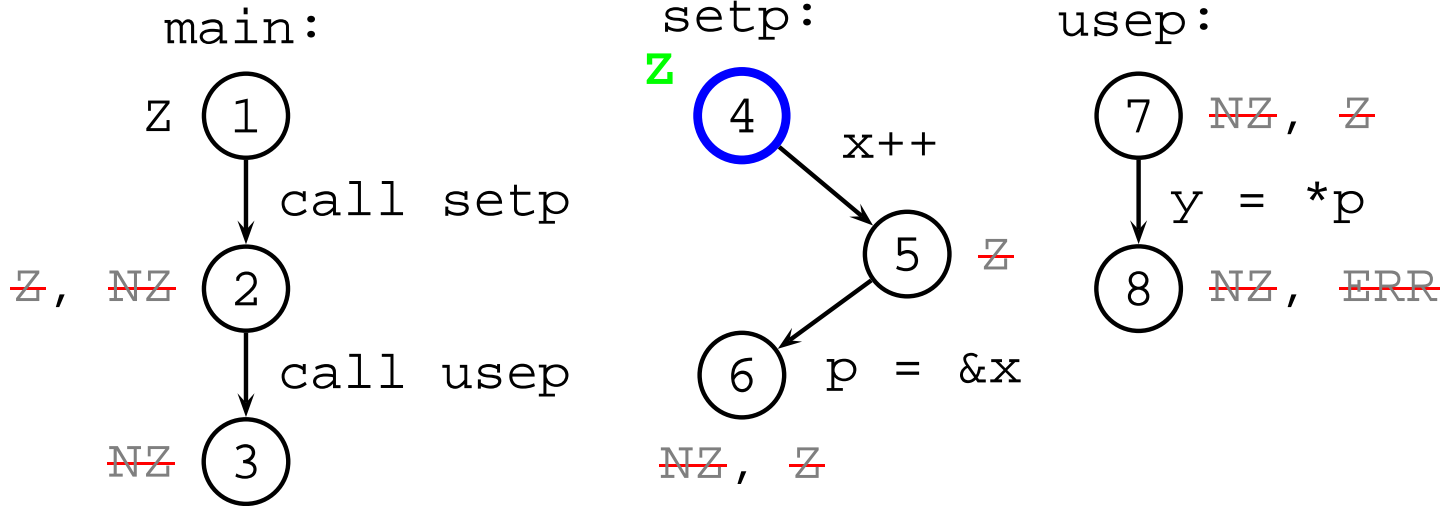
	setp	usep
Summaries:	$\langle \cancel{z}, \cancel{NZ} \rangle$ $\langle \cancel{z}, \cancel{z} \rangle$	$\langle \cancel{NZ}, \cancel{NZ} \rangle$
Call Sites:	$\langle \cancel{1}, \cancel{z} \rangle$	$\langle \cancel{2}, \cancel{NZ} \rangle$ $\langle \cancel{2}, \cancel{z} \rangle$

Forward Algorithm (Example)



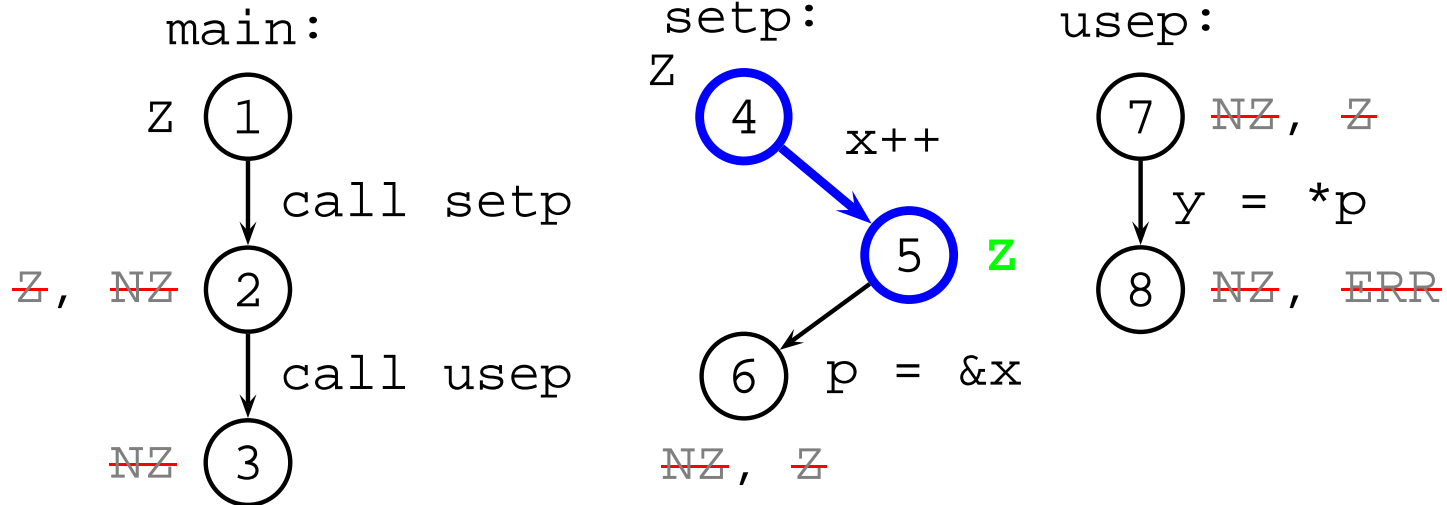
	setp	usep
Summaries:	$\langle \cancel{z}, \cancel{NZ} \rangle \quad \langle \cancel{z}, \cancel{z} \rangle$	$\langle \cancel{NZ}, \cancel{NZ} \rangle$
Call Sites:	$\langle \cancel{1}, \cancel{z} \rangle$	$\langle \cancel{2}, \cancel{NZ} \rangle \quad \langle \cancel{2}, \cancel{z} \rangle$

Forward Algorithm (Example)



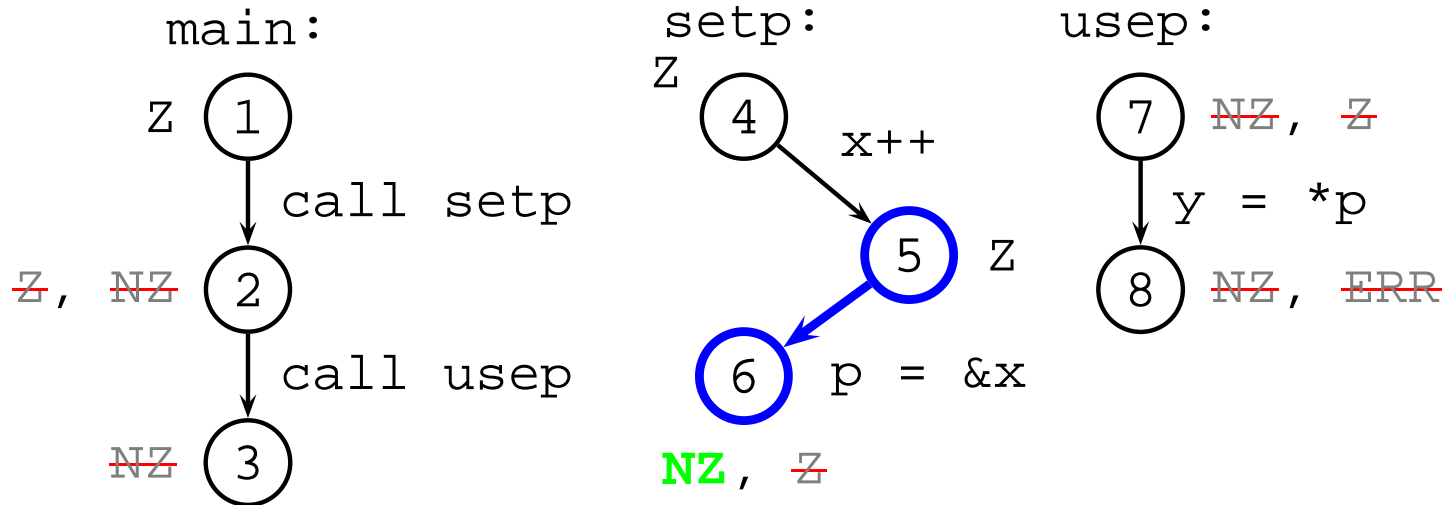
	setp	usep
Summaries:	<z, NZ> <z, z>	<NZ, NZ>
Call Sites:	(1, z)	(2, NZ) (2, z)

Forward Algorithm (Example)



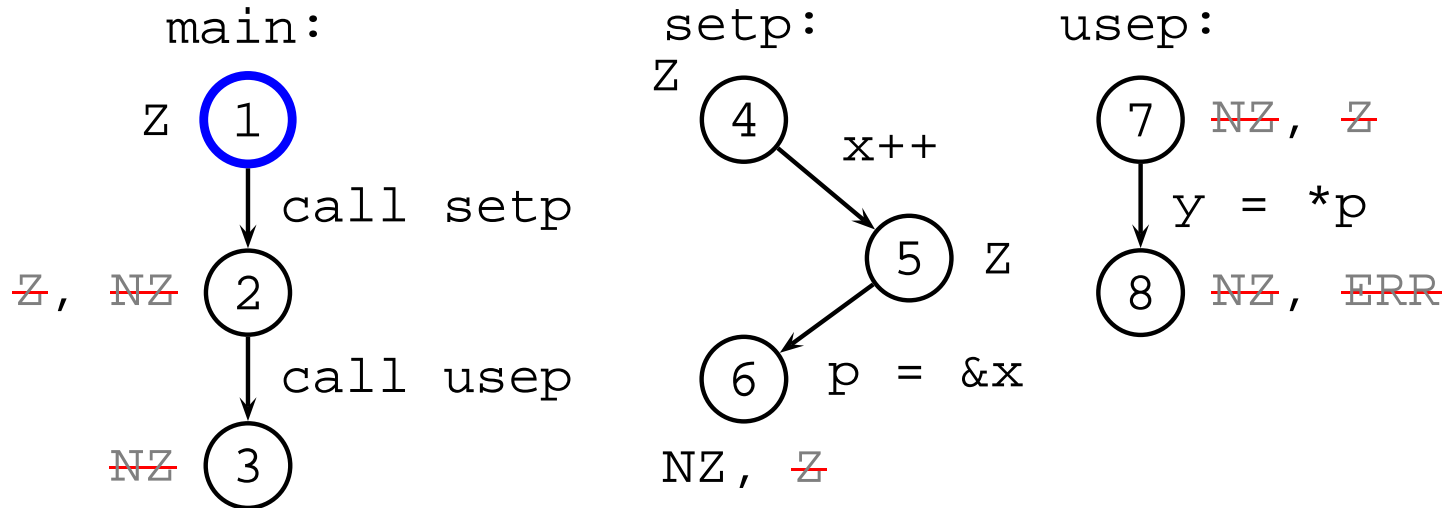
	setp	usep
Summaries:	<z, NZ> <z, z>	<NZ, NZ>
Call Sites:	(1, z)	(2, NZ) (2, z)

Forward Algorithm (Example)



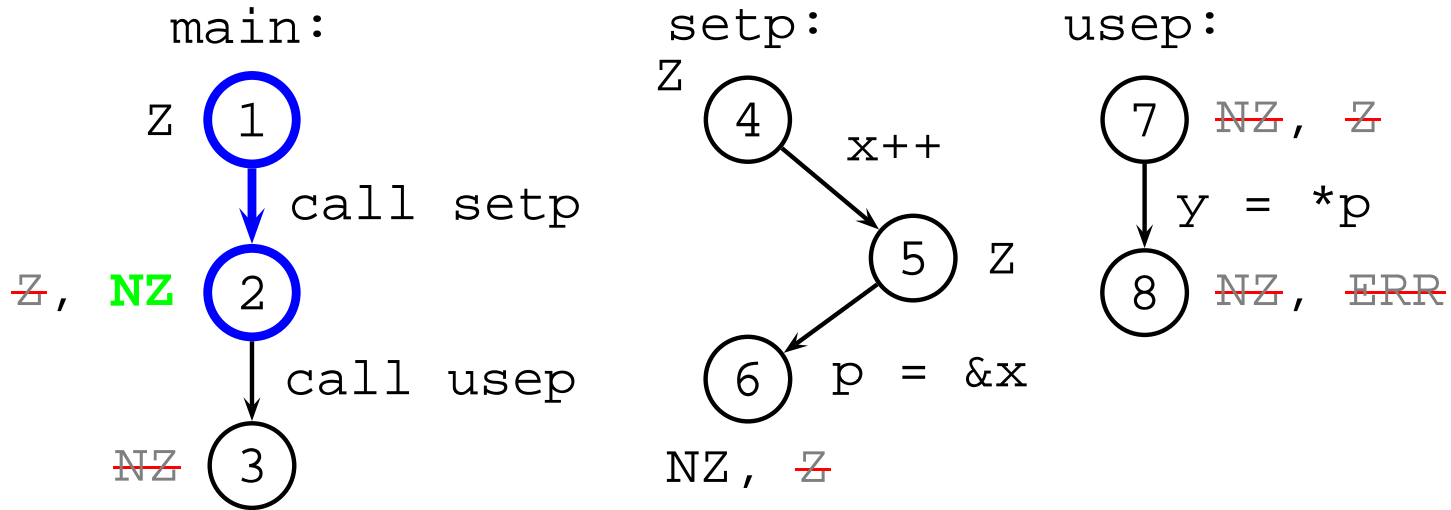
	setp	usep
Summaries:	$\langle z, NZ \rangle$ $\langle z, z \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	(1, z)	(2, NZ) (2, z)

Forward Algorithm (Example)



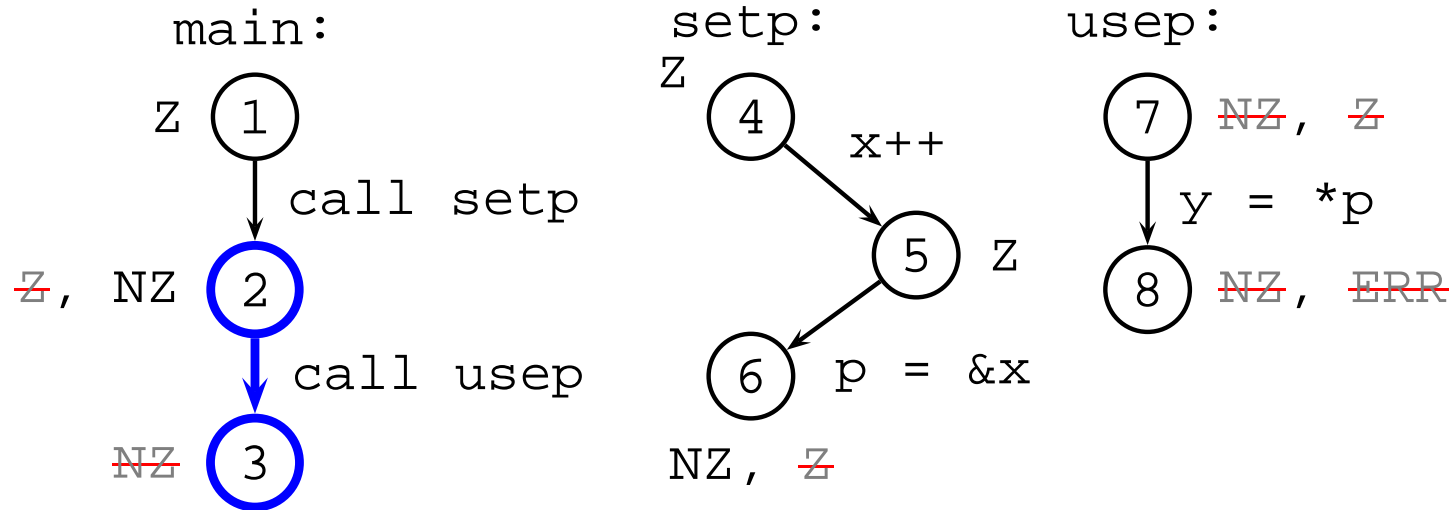
	setp	usep
Summaries:	$\langle z, NZ \rangle$ $\langle \cancel{z}, \cancel{z} \rangle$	$\langle \cancel{NZ}, \cancel{NZ} \rangle$
Call Sites:	$(1, z)$	$\langle \cancel{2}, \cancel{NZ} \rangle$ $\langle \cancel{2}, \cancel{z} \rangle$

Forward Algorithm (Example)



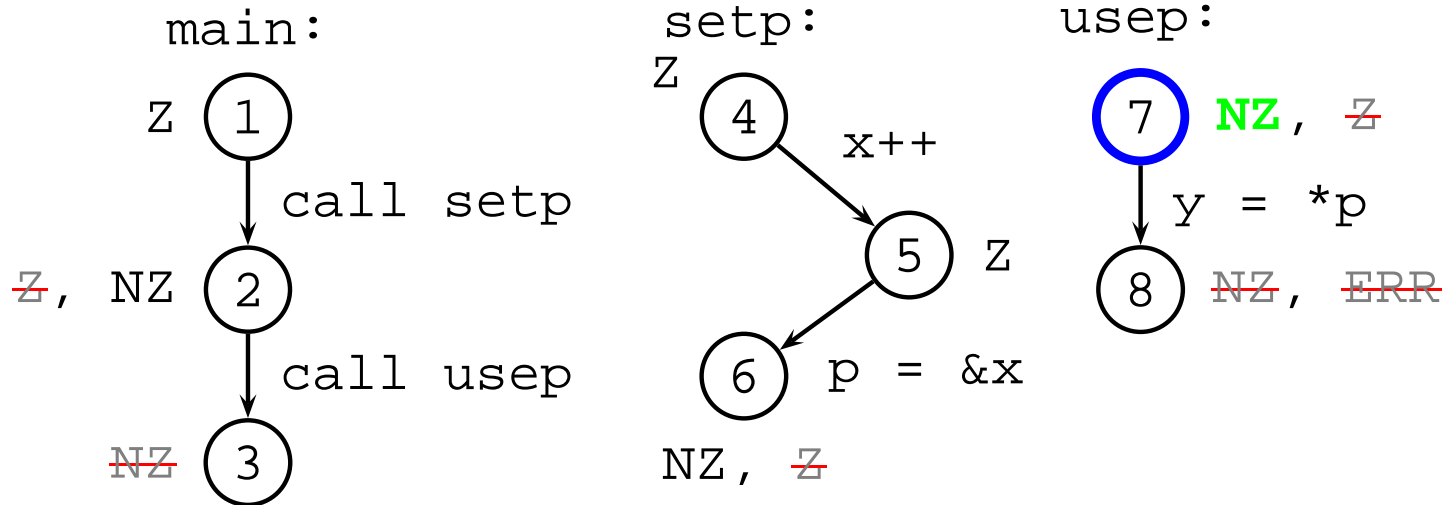
	setp	usep
Summaries:	$\langle z, NZ \rangle$ $\langle z, z \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	(1, z)	(2, NZ) (2, z)

Forward Algorithm (Example)



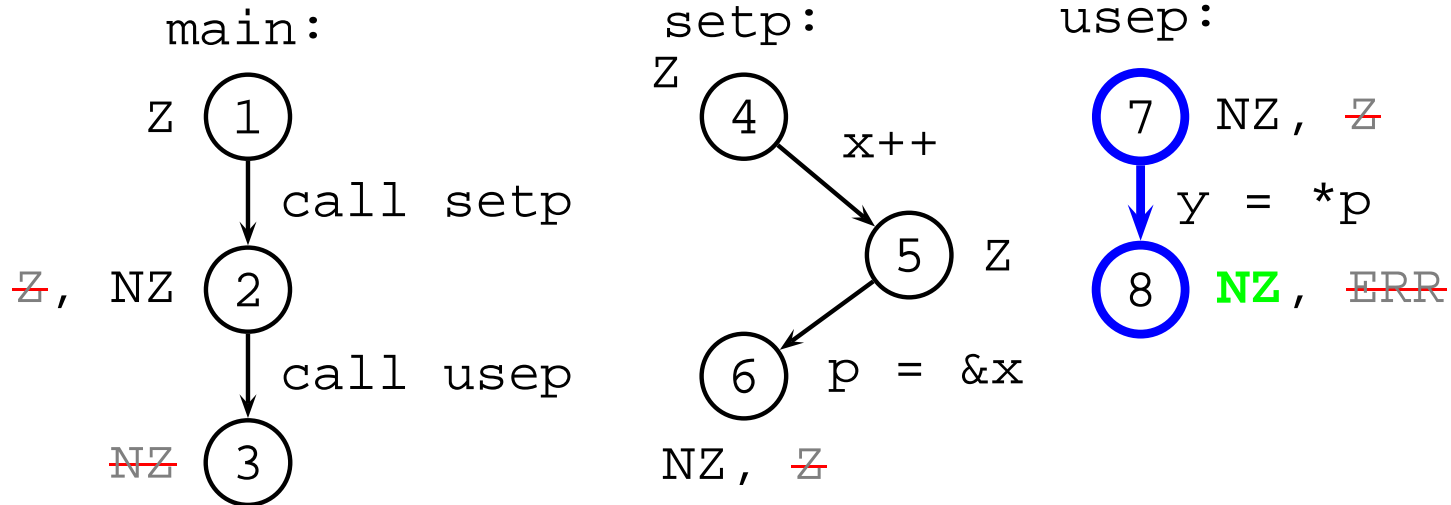
	setp	usep
Summaries:	$\langle z, NZ \rangle$ $\langle z, z \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	$(1, z)$	$(2, NZ)$ $(2, z)$

Forward Algorithm (Example)



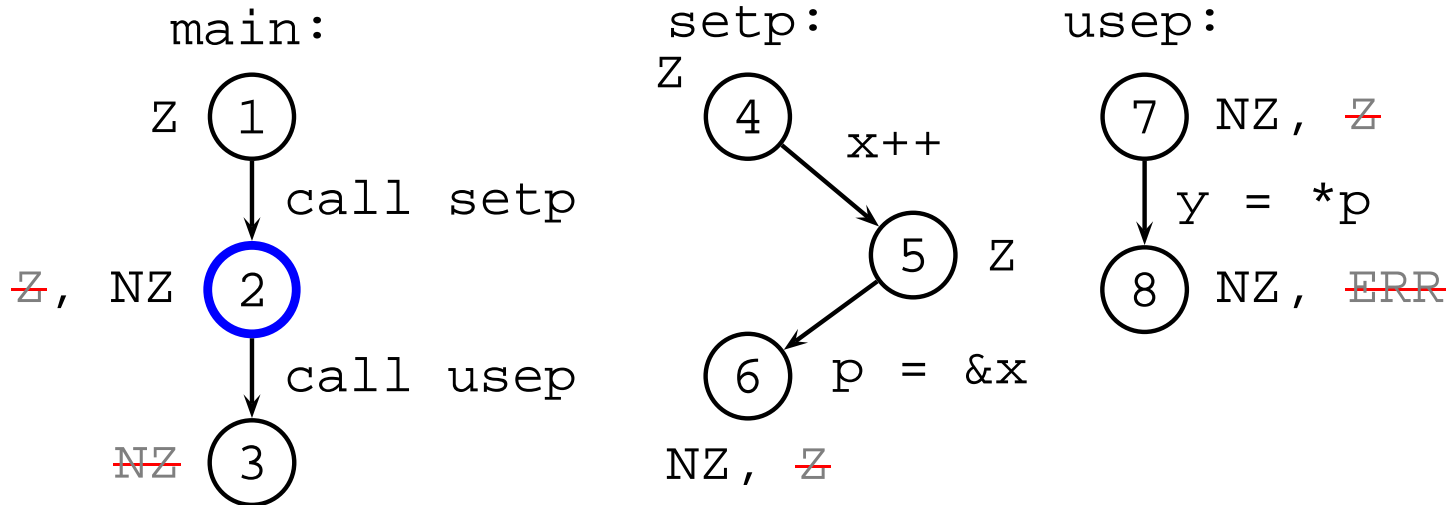
	setp	usep
Summaries:	$\langle z, NZ \rangle$ $\langle \cancel{z}, z \rangle$	$\langle \cancel{NZ}, NZ \rangle$
Call Sites:	$(1, z)$	$(2, NZ)$ $(\cancel{2}, z)$

Forward Algorithm (Example)



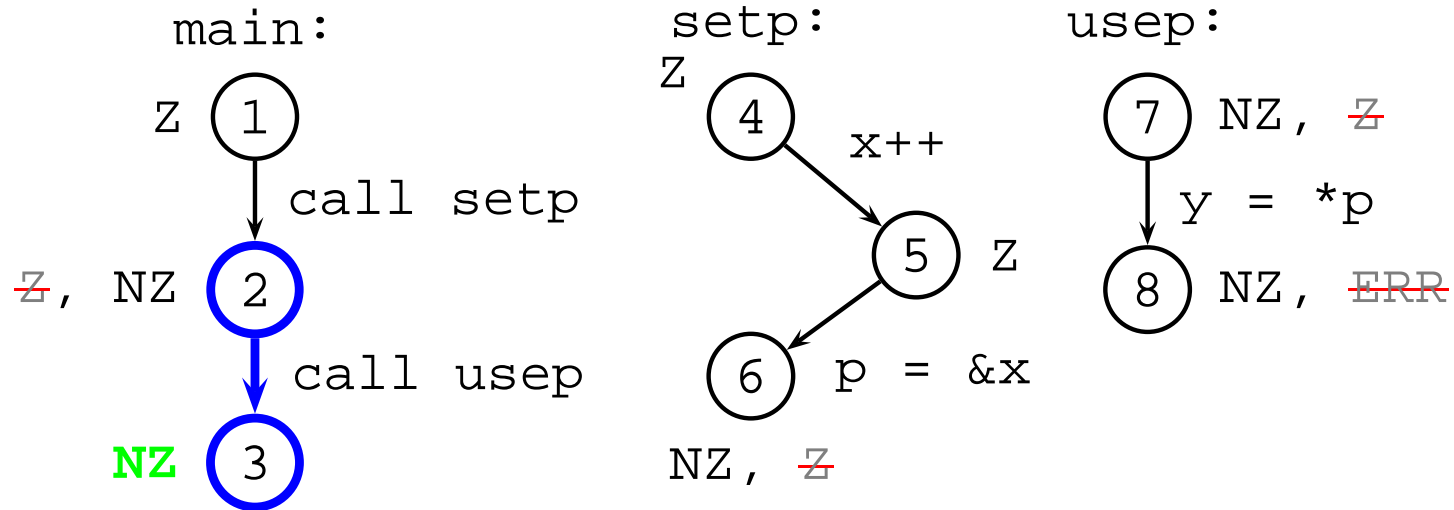
	setp	usep
Summaries:	$\langle z, NZ \rangle$ $\langle z, z \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	(1, z)	(2, NZ) (2, z)

Forward Algorithm (Example)



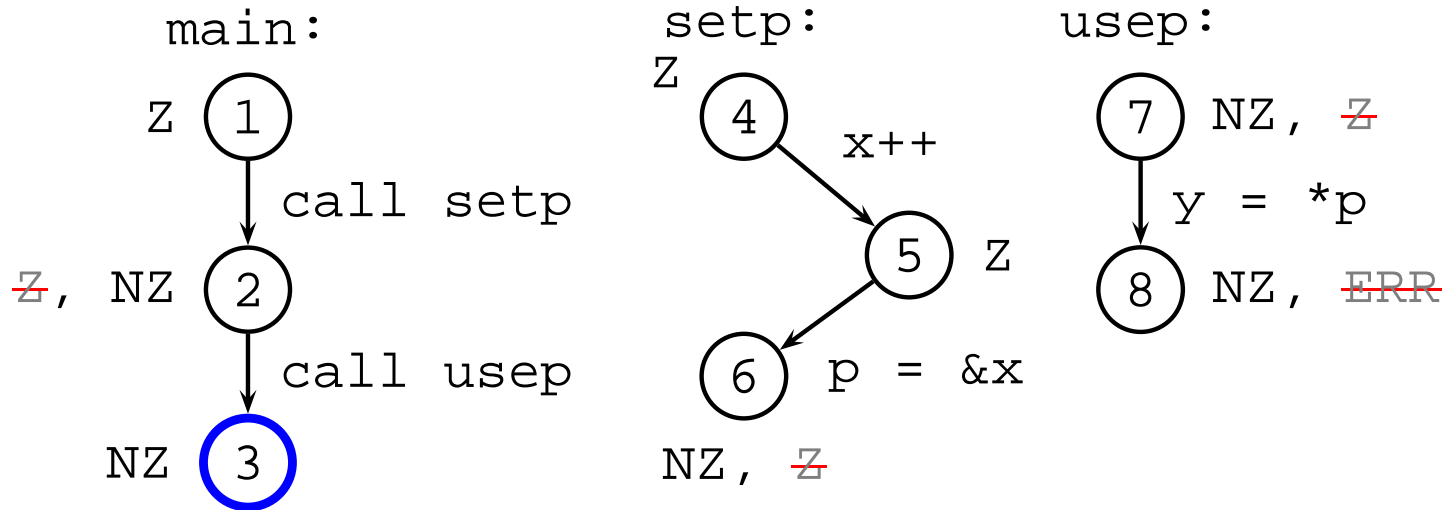
	setp	usep
Summaries:	$\langle z, NZ \rangle$ $\langle z, z \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	(1, z)	(2, NZ) (2, z)

Forward Algorithm (Example)



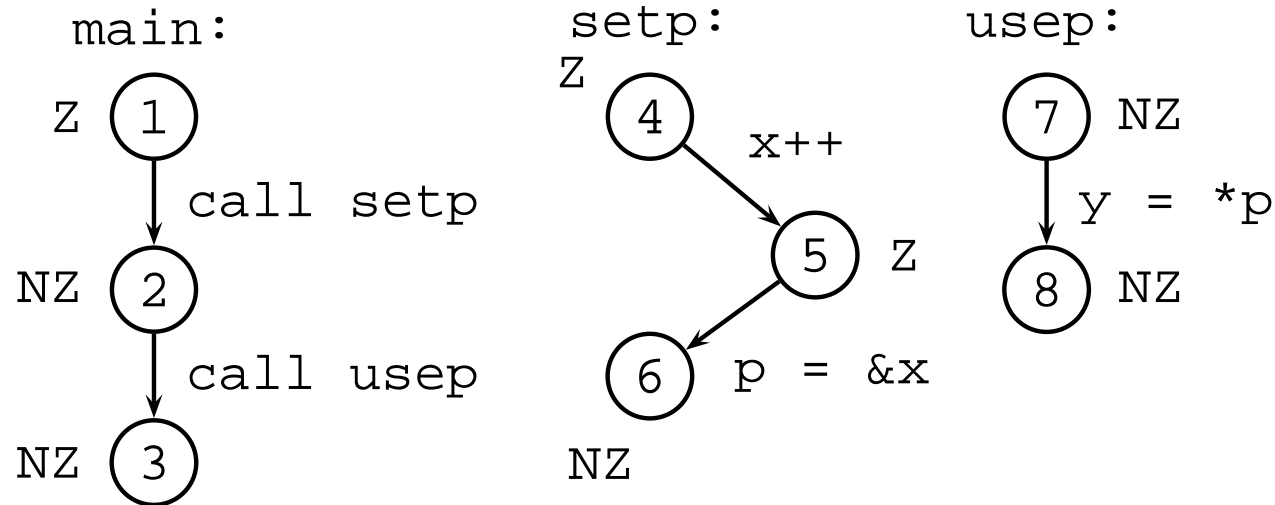
	setp	usep
Summaries:	$\langle z, NZ \rangle$ $\langle z, z \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	(1, z)	(2, NZ) (2, z)

Forward Algorithm (Example)



	setp	usep
Summaries:	$\langle z, NZ \rangle$ $\langle z, z \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	(1, z)	(2, NZ) (2, z)

Forward Algorithm (Example)



	setp	usep
Summaries:	$\langle Z, NZ \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	$(1, Z)$	$(2, NZ)$

Forward Algorithm: Thoughts

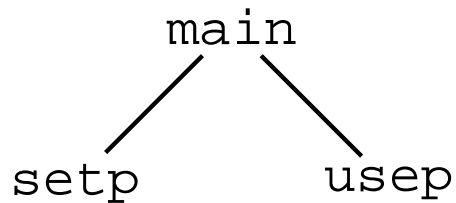
- ◆ Average speedup 40% (max 60%)

Forward Algorithm: Thoughts

- ◆ Average speedup 40% (max 60%)
- ◆ Complexity equal to Full algorithm; saves on re-computation.

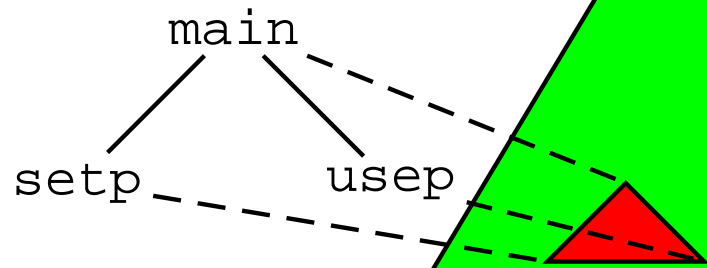
Forward Algorithm: Thoughts

- ◆ Average speedup 40% (max 60%)
- ◆ Complexity equal to Full algorithm; saves on re-computation.

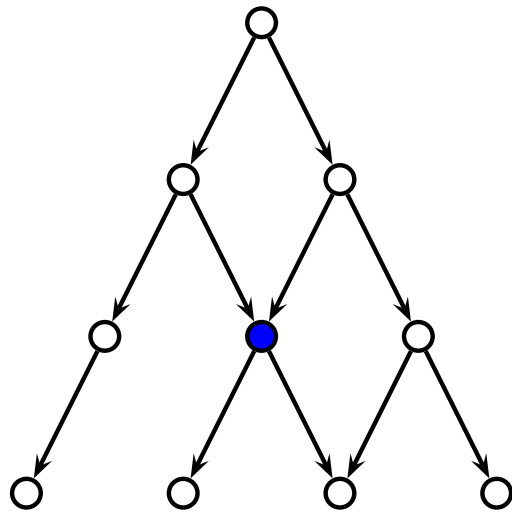


Forward Algorithm: thoughts

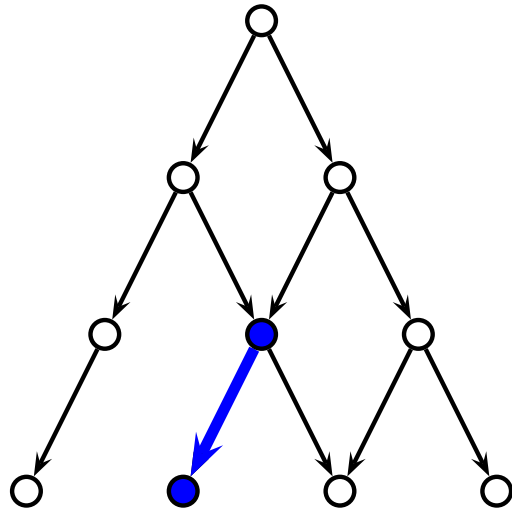
- ◆ Average speedup 40% (m)
- ◆ Complexity equal to Full leaves on re-computation.



Effects of Changes on the Call Graph

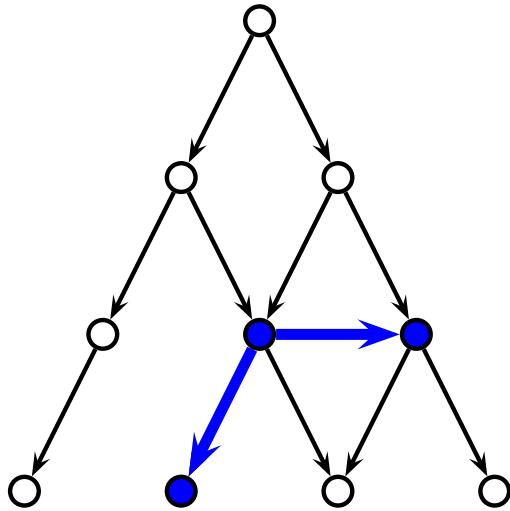


Effects of Changes on the Call Graph



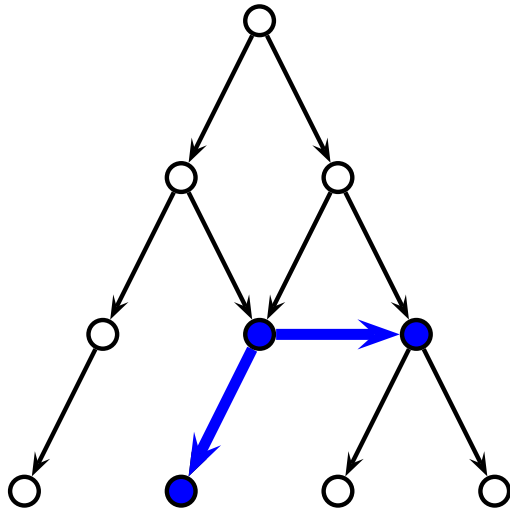
- ◆ New function entries

Effects of Changes on the Call Graph



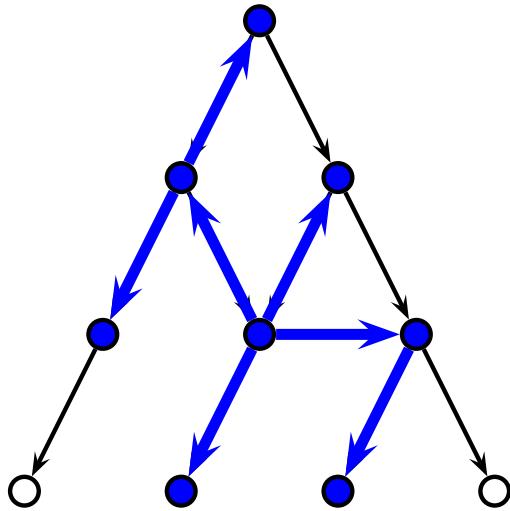
- ◆ New function entries
- ◆ Newly reachable functions

Effects of Changes on the Call Graph



- ◆ New function entries
- ◆ Newly reachable functions
- ◆ Newly unreachable functions

Effects of Changes on the Call Graph



- ◆ New function entries
- ◆ Newly reachable functions
- ◆ Newly unreachable functions
- ◆ New input/output pairs for callers
- ◆ And so on...

The Backward (Inside-Out) Algorithm

1. Decompose call graph into SCCs.

The Backward (Inside-Out) Algorithm

1. Decompose call graph into SCCs.
2. For each SCC C (in reverse topological order):

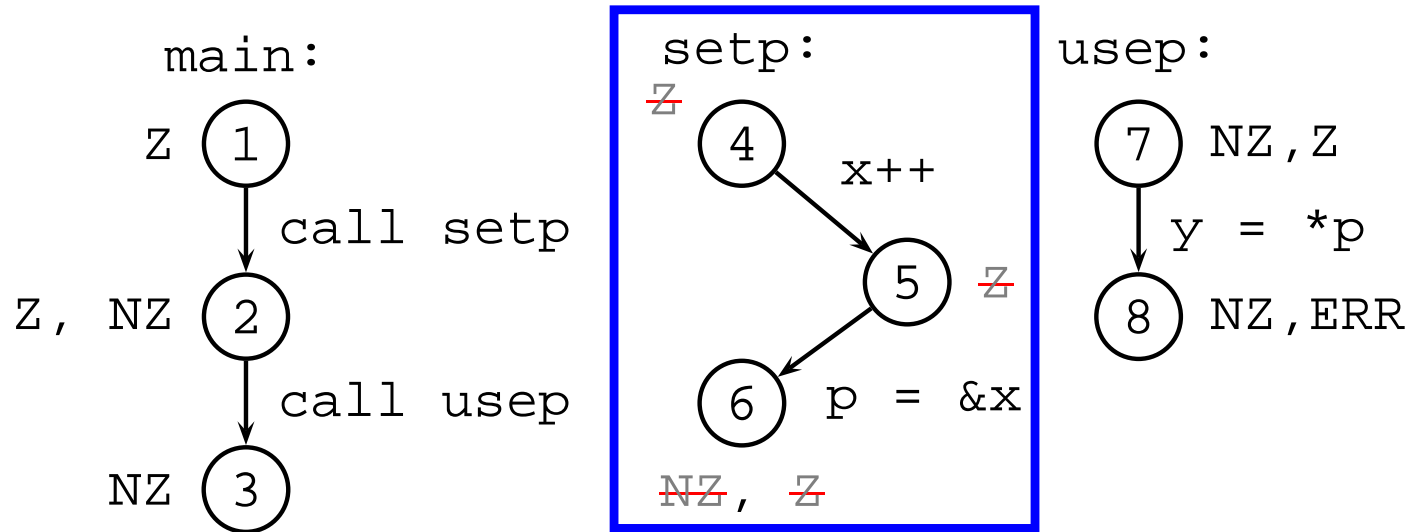
The Backward (Inside-Out) Algorithm

1. Decompose call graph into SCCs.
2. For each SCC C (in reverse topological order):
If a function in C is modified, or
a function summary used in C has changed,

The Backward (Inside-Out) Algorithm

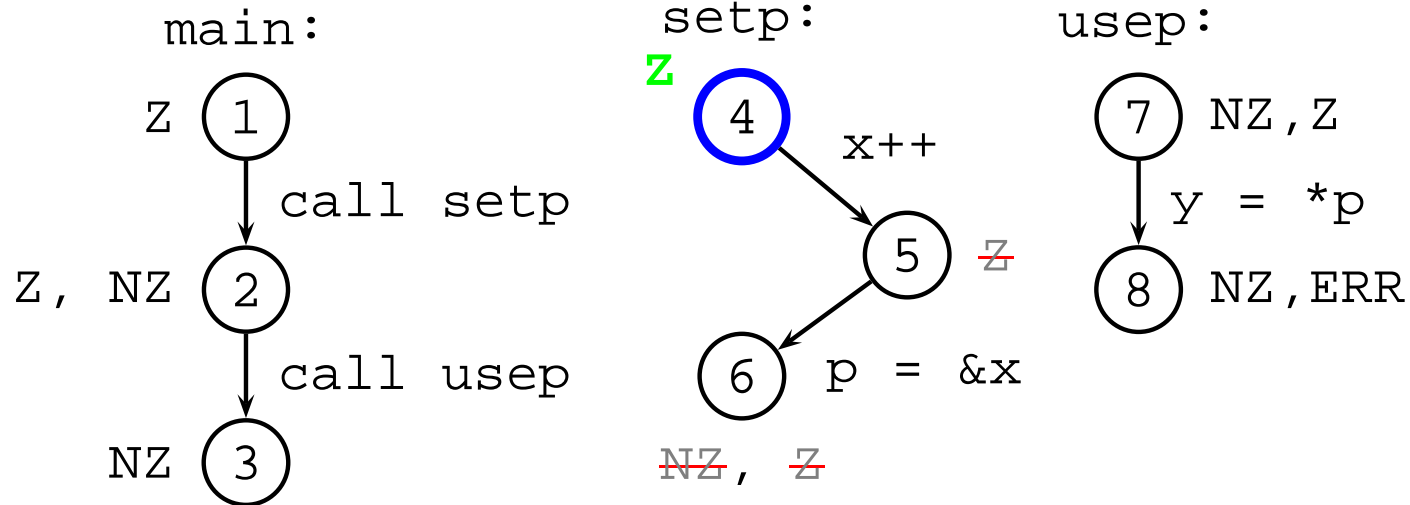
1. Decompose call graph into SCCs.
2. For each SCC C (in reverse topological order):
If a function in C is modified, or
a function summary used in C has changed,
use the forward algorithm on C .

Backward Algorithm (Example)



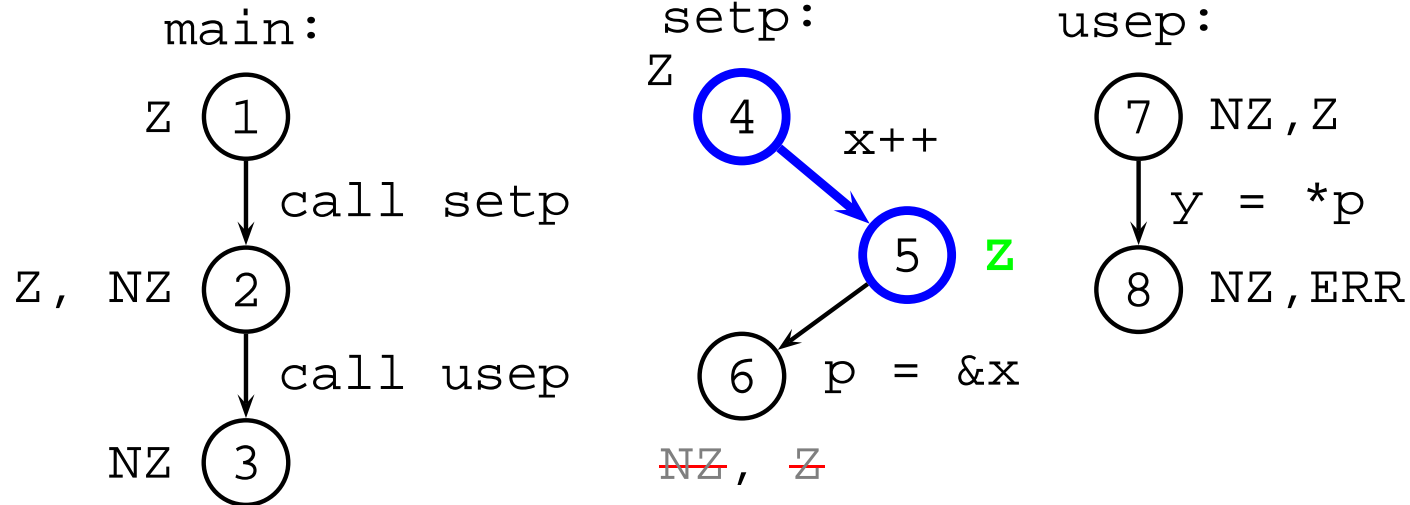
	setp	usep
Summaries:	$\langle z, NZ \rangle$ $\langle z, z \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	(1, z)	(2, NZ) (2, z)

Backward Algorithm (Example)



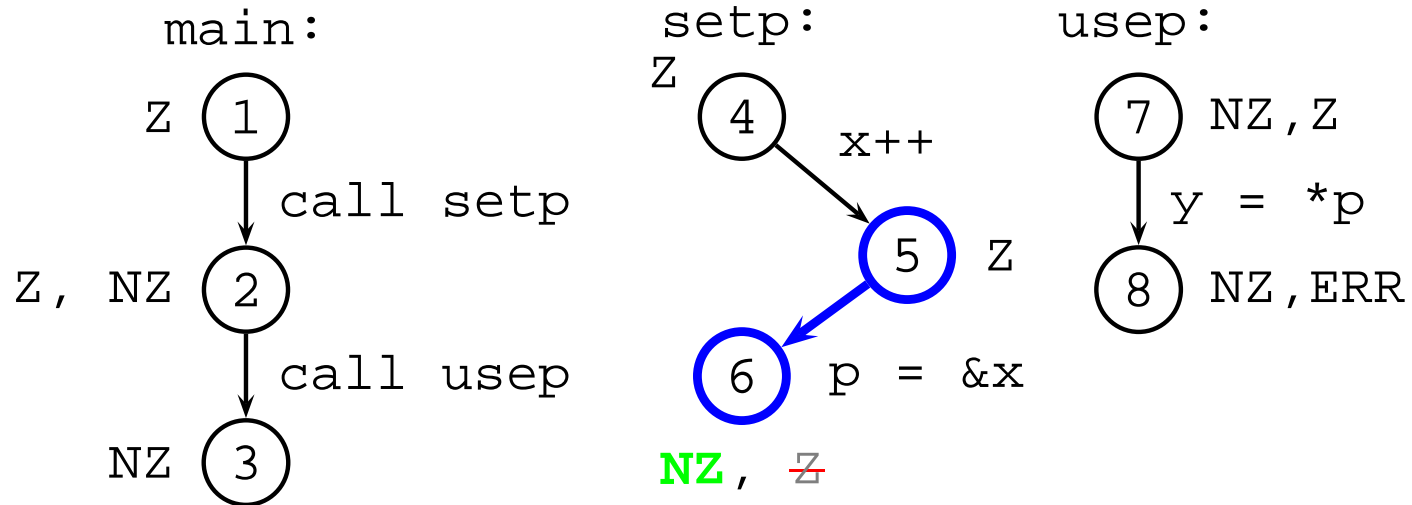
	setp	usep
Summaries:	⟨z, NZ⟩ ⟨z, z⟩	⟨NZ, NZ⟩
Call Sites:	(1, z)	(2, NZ) (2, z)

Backward Algorithm (Example)



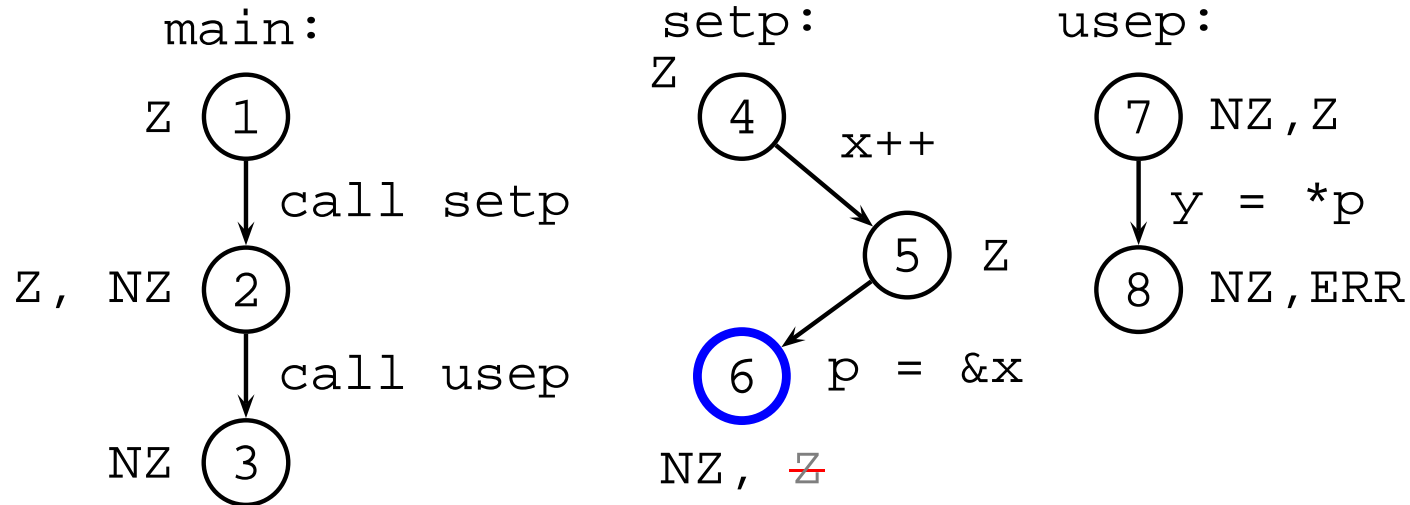
	setp	usep
Summaries:	$\langle z, NZ \rangle$ $\langle z, z \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	(1, z)	(2, NZ) (2, z)

Backward Algorithm (Example)



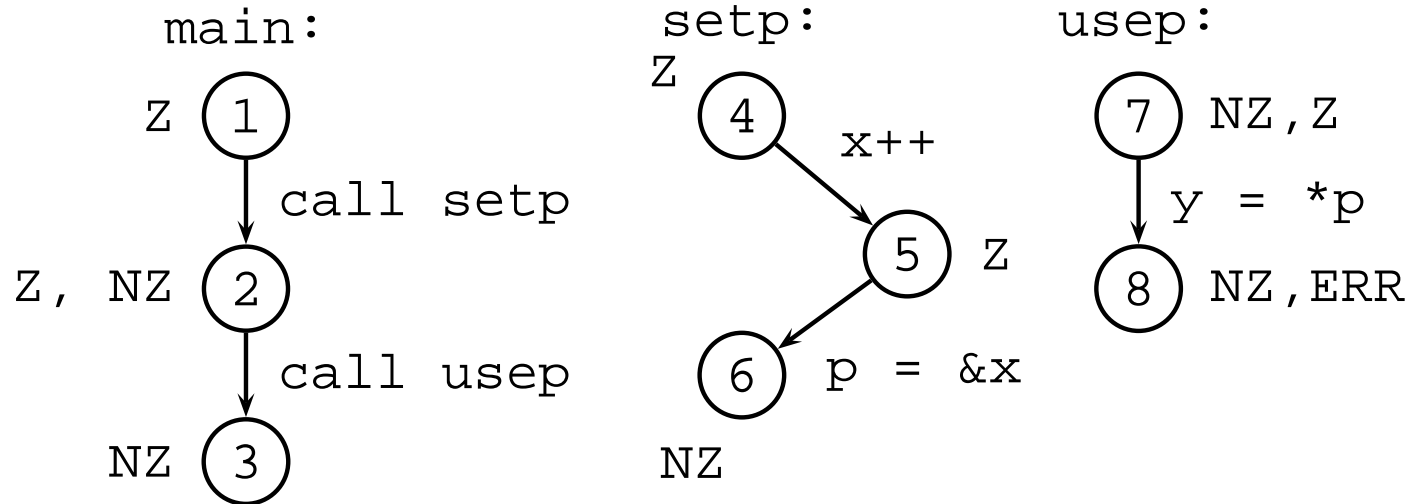
	setp	usep
Summaries:	$\langle Z, NZ \rangle$ $\langle Z, Z \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	(1, Z)	(2, NZ) (2, Z)

Backward Algorithm (Example)



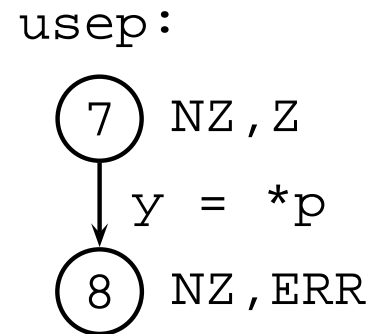
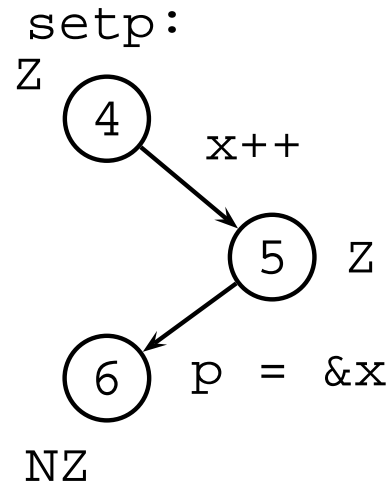
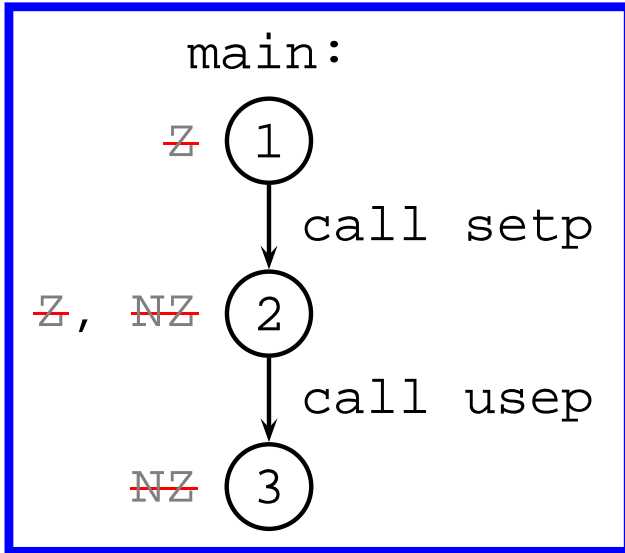
	setp	usep
Summaries:	$\langle z, NZ \rangle$ $\langle z, z \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	(1, z)	(2, NZ) (2, z)

Backward Algorithm (Example)



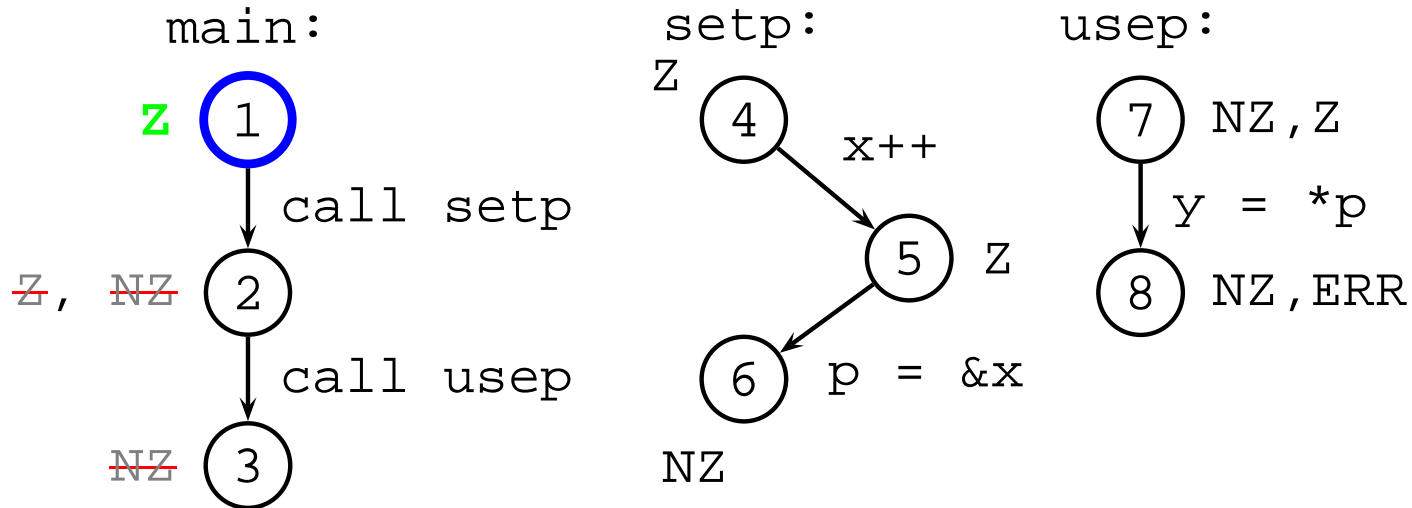
	setp	usep
Summaries:	$\langle Z, NZ \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	$(1, Z)$	$(2, NZ) \quad (2, Z)$

Backward Algorithm (Example)



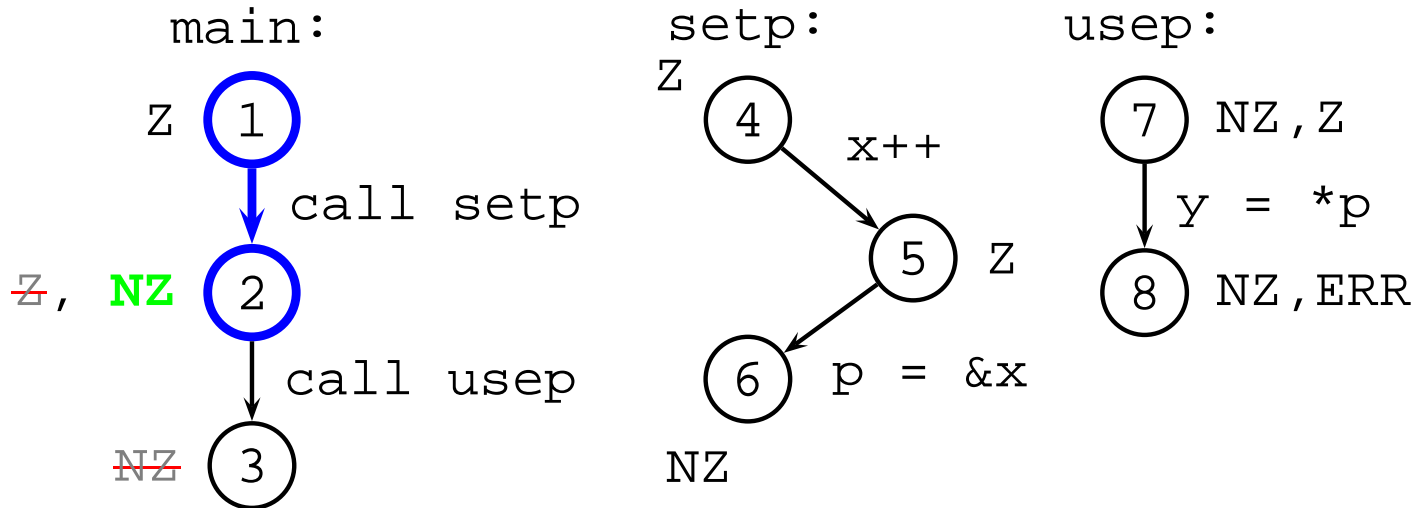
	setp	usep
Summaries:	$\langle Z, NZ \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	(1, Z)	(2, NZ) (2, Z)

Backward Algorithm (Example)



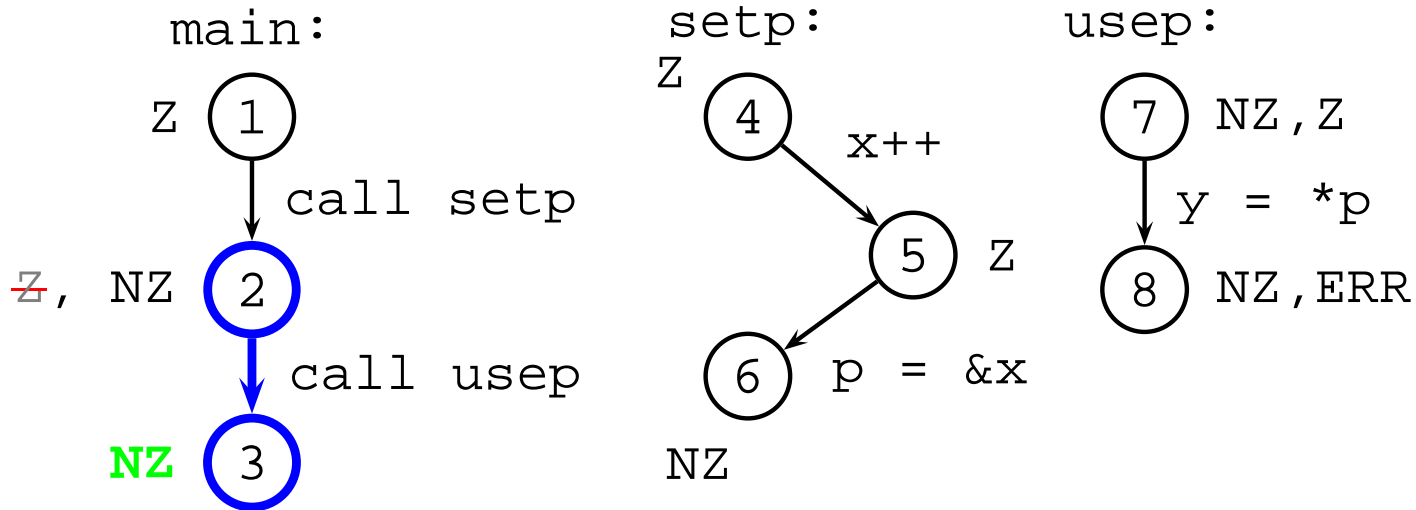
	setp	usep
Summaries:	$\langle Z, NZ \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	(1, Z)	(2, NZ) (2, Z)

Backward Algorithm (Example)



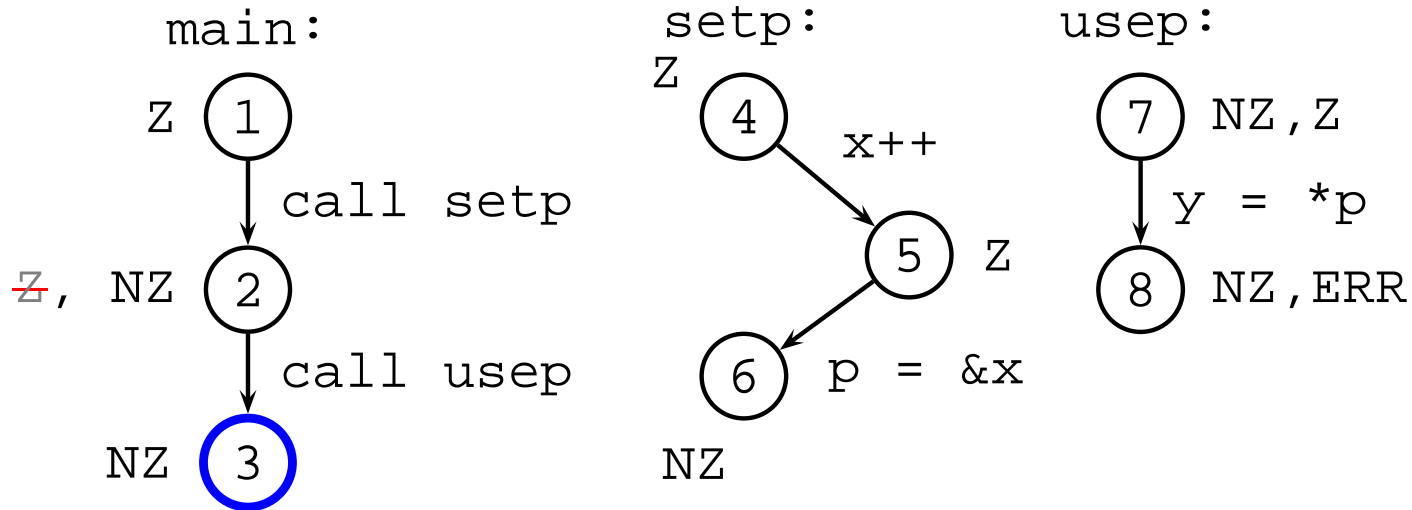
	setp	usep
Summaries:	$\langle z, \text{NZ} \rangle$	$\langle \text{NZ}, \text{NZ} \rangle$
Call Sites:	(1, z)	(2, NZ) (2, z)

Backward Algorithm (Example)



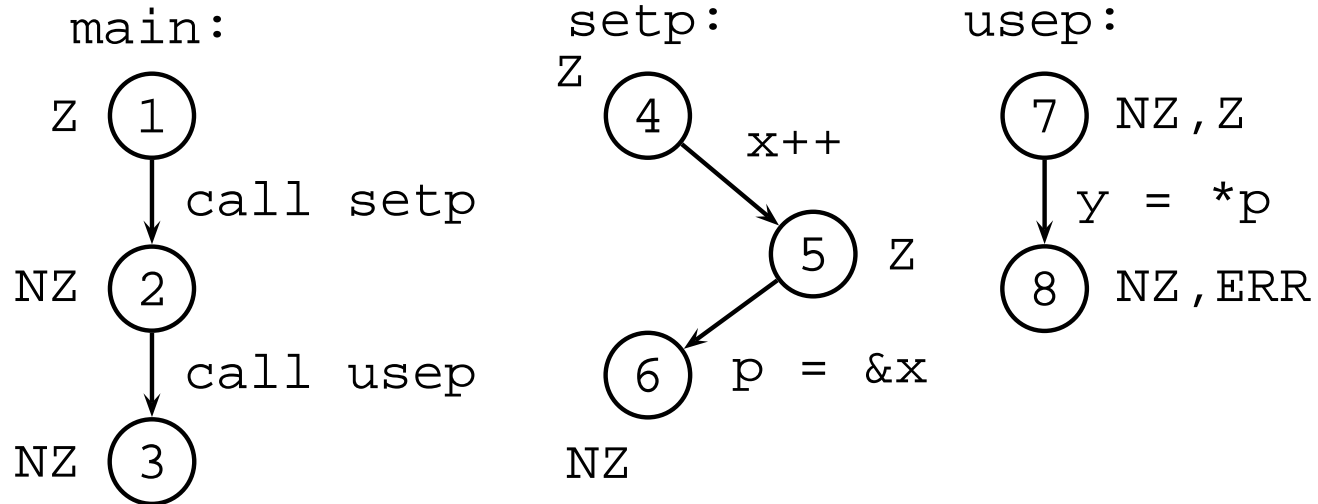
	setp	usep
Summaries:	$\langle Z, NZ \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	(1, Z)	(2, NZ) (2, Z)

Backward Algorithm (Example)



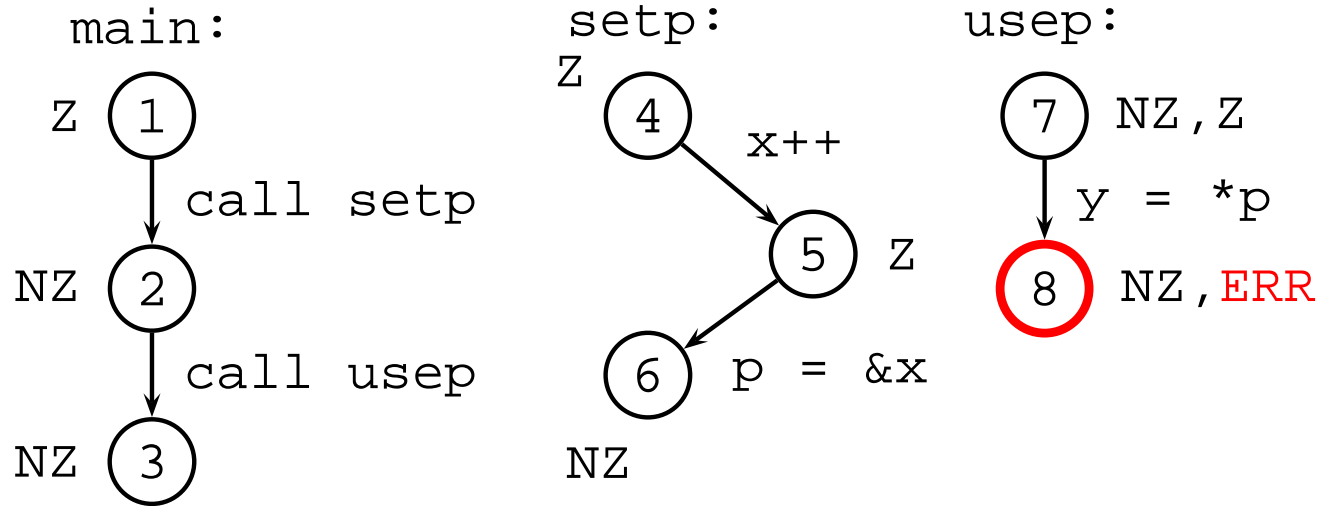
	setp	usep
Summaries:	$\langle Z, NZ \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	$(1, Z)$	$(2, NZ) \quad (2, Z)$

Backward Algorithm (Example)



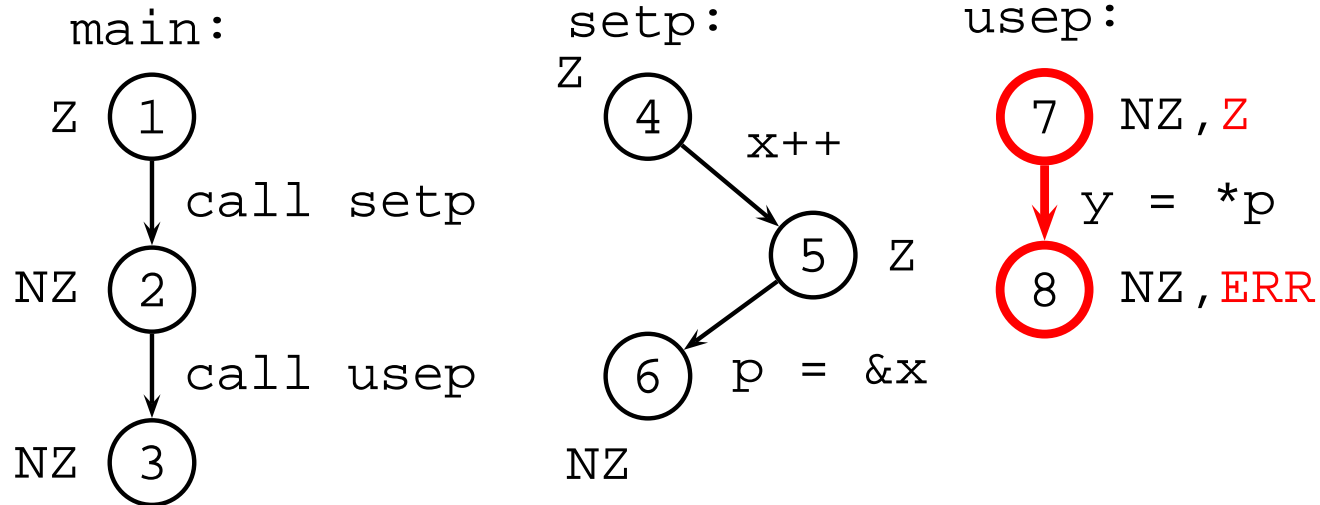
	setp	usep
Summaries:	$\langle Z, NZ \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	$(1, Z)$	$(2, NZ) \quad (2, Z)$

Backward Algorithm (Example)



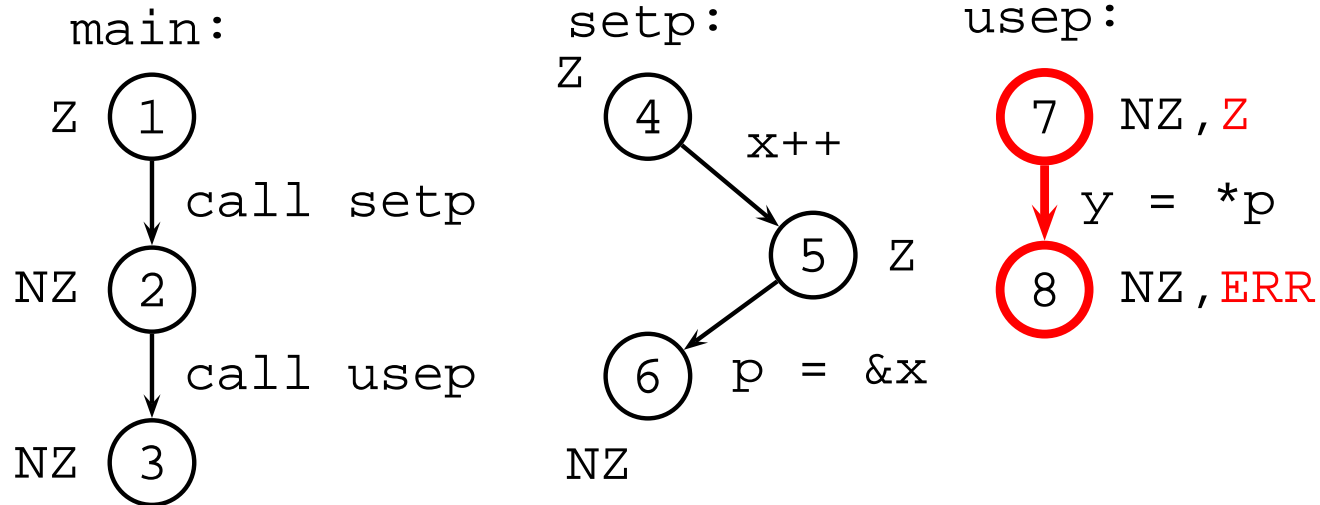
	setp	usep
Summaries:	$\langle Z, NZ \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	(1, Z)	(2, NZ) (2, Z)

Backward Algorithm (Example)



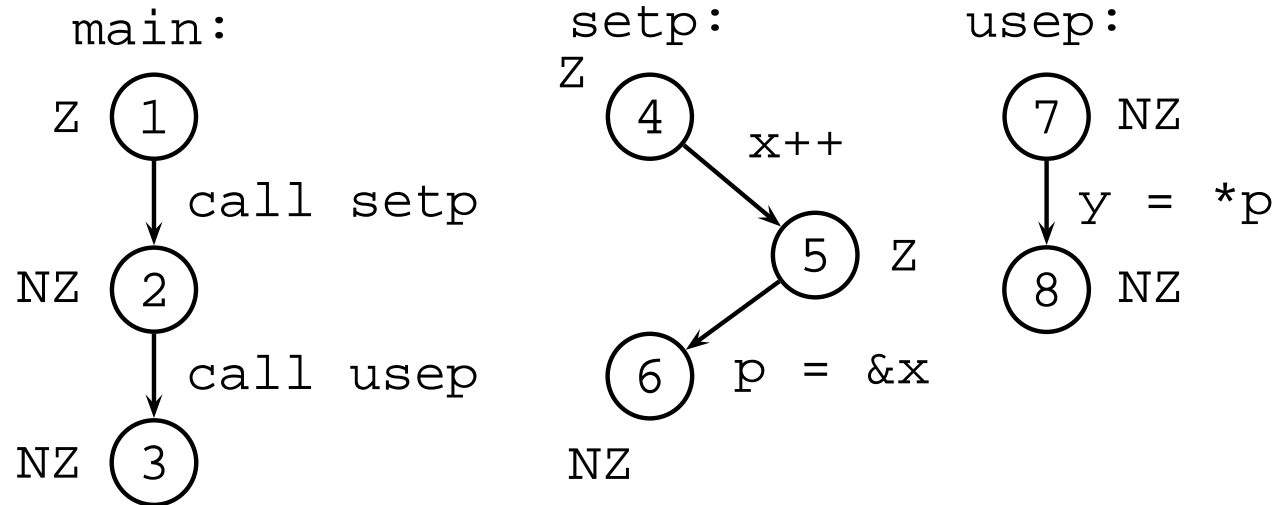
	setp	usep
Summaries:	$\langle Z, NZ \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	$(1, Z)$	$(2, NZ) \quad (2, Z)$

Backward Algorithm (Example)



	setp	usep
Summaries:	$\langle z, NZ \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	$(1, z)$	$(2, NZ)$ $(2, z)$

Backward Algorithm (Example)

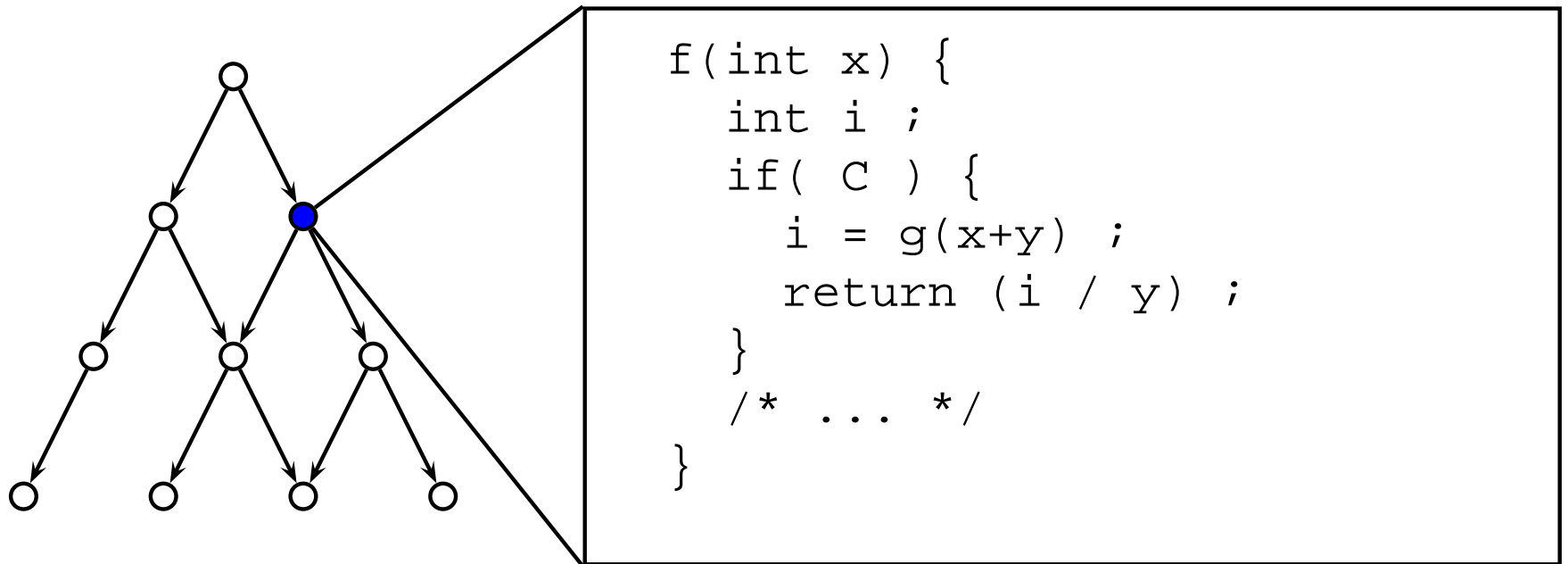


	setp	usep
Summaries:	$\langle Z, NZ \rangle$	$\langle NZ, NZ \rangle$
Call Sites:	$(1, Z)$	$(2, NZ)$

Experimental Design

Simulates the modification of a single function.

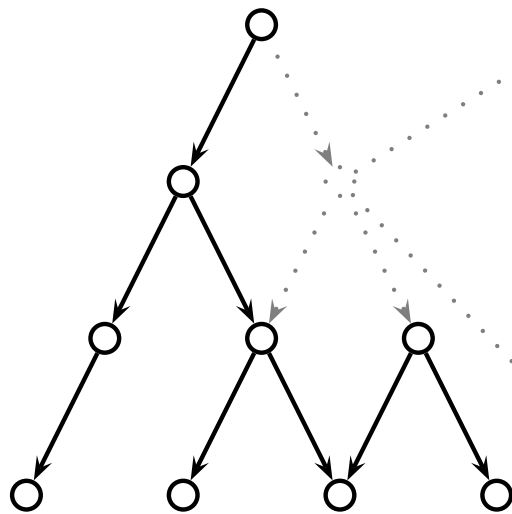
- ◆ For each function f in a program:



Experimental Design

Simulates the modification of a single function.

- ◆ For each function f in a program:
 1. Replace f with an empty stub

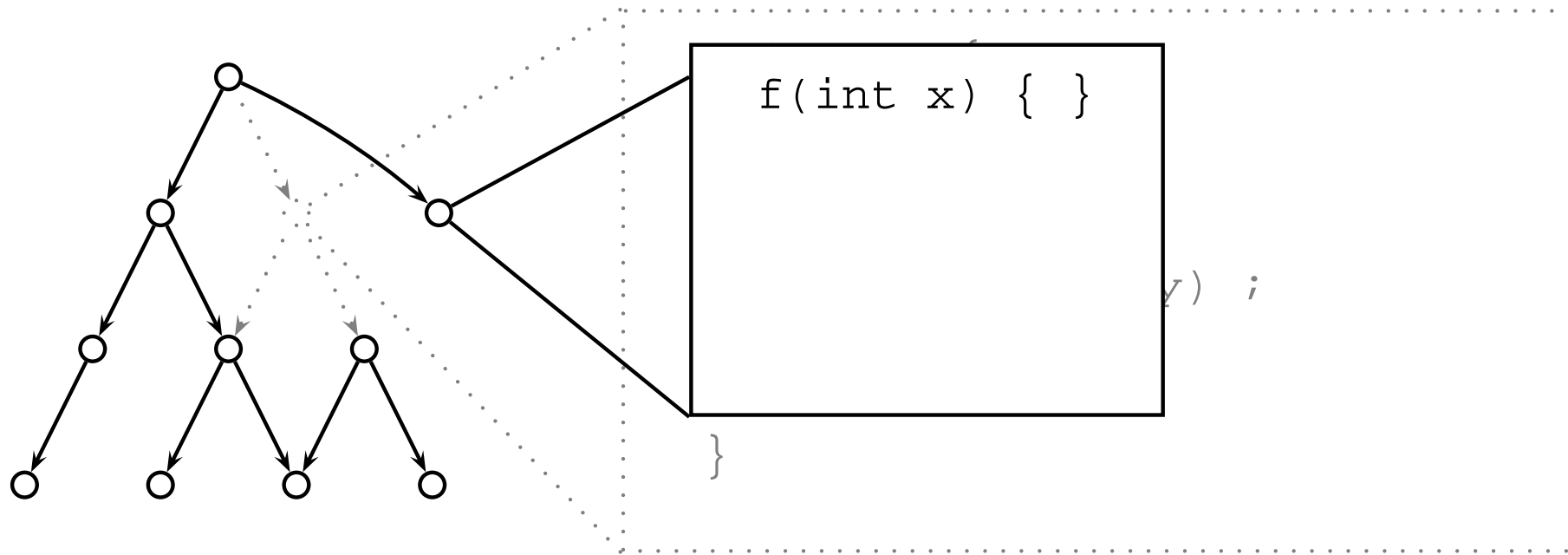


```
f(int x) {  
  int i ;  
  if( C ) {  
    i = g(x+y) ;  
    return (i / y) ;  
  }  
  /* ... */  
}
```

Experimental Design

Simulates the modification of a single function.

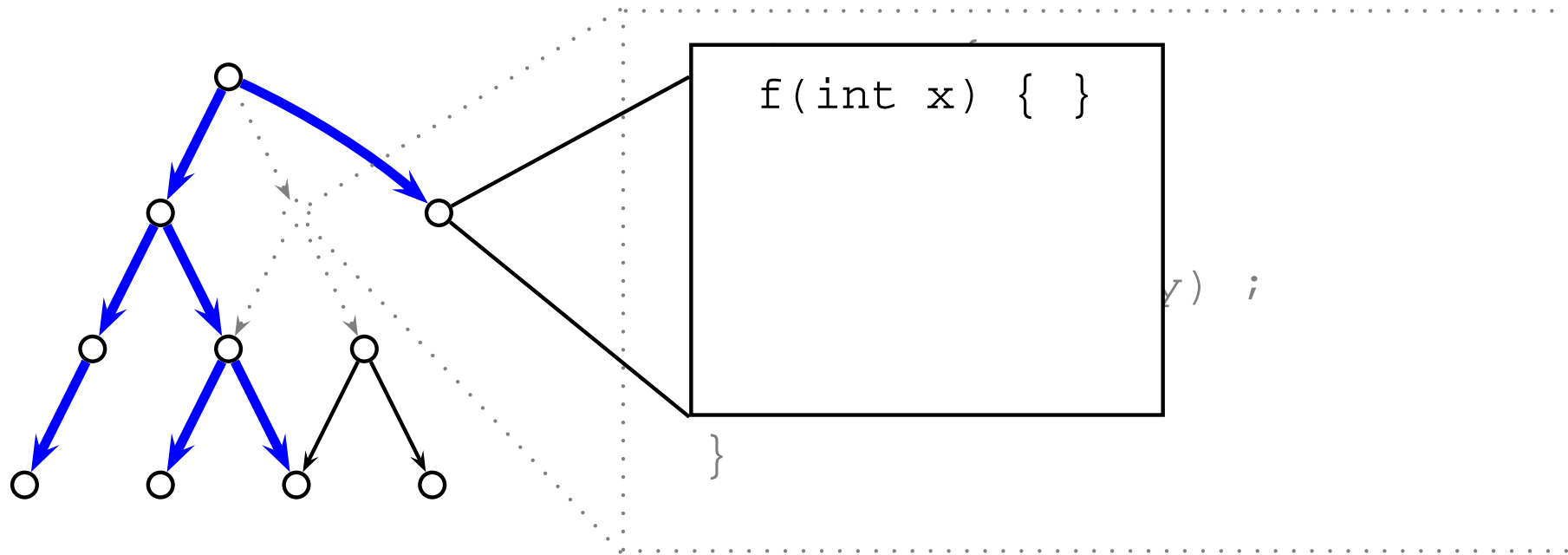
- ◆ For each function f in a program:
 1. Replace f with an empty stub



Experimental Design

Simulates the modification of a single function.

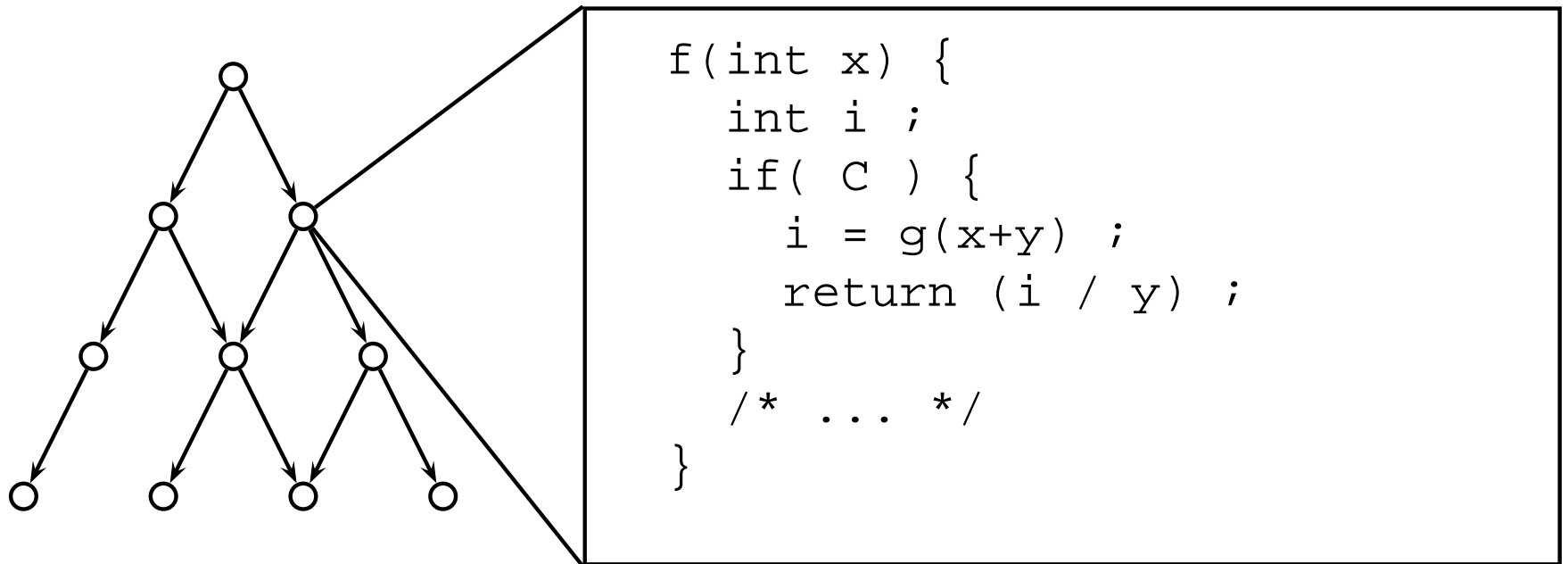
- ◆ For each function f in a program:
 1. Replace f with an empty stub and analyze the program.



Experimental Design

Simulates the modification of a single function.

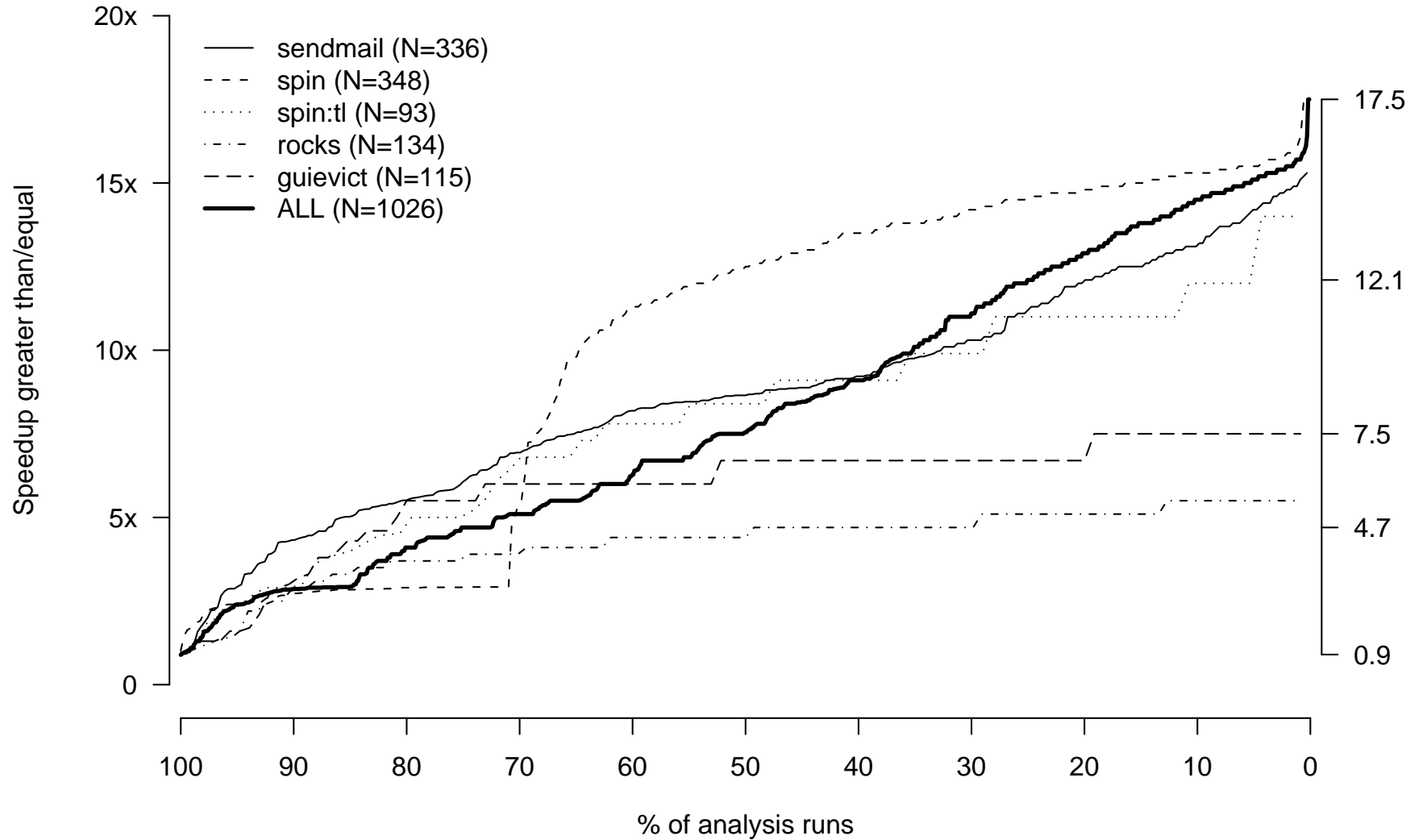
- ◆ For each function f in a program:
 1. Replace f with an empty stub and analyze the program.
 2. Re-insert f and re-analyze.



Experimental Results

	Lines of code	Full analysis time (s)	<u>Average speedup</u>		Incr. data (KB)
			IncrFwd	IncrBack	
sendmail	47,651	33.75	1.6	8.6	73.72
spin	16,540	24.91	1.3	10.1	91.61
spin:tl	2,569	1.09	1.3	8.0	7.01
guievict	4,545	0.60	1.4	5.9	3.54
rocks	4,619	0.66	1.3	4.3	4.36

Distribution of Speedups



Results

- ◆ Forward algorithm speedups: max 1.6, avg. 1.4

Results

- ◆ Forward algorithm speedups: max 1.6, avg. 1.4
- ◆ Backward algorithm speedups: max 17.5, avg. 8.2

Results

- ◆ Forward algorithm speedups: max 1.6, avg. 1.4
- ◆ Backward algorithm speedups: max 17.5, avg. 8.2
- ◆ Libraries (guievict): avg. 11.3

Results

- ◆ Forward algorithm speedups: max 1.6, avg. 1.4
- ◆ Backward algorithm speedups: max 17.5, avg. 8.2
- ◆ Libraries (guievict): avg. 11.3
- ◆ Larger changes (10–50% of functions): avg. 2.5

Results

- ◆ Forward algorithm speedups: max 1.6, avg. 1.4
- ◆ Backward algorithm speedups: max 17.5, avg. 8.2
- ◆ Libraries (guievict): avg. 11.3
- ◆ Larger changes (10–50% of functions): avg. 2.5
- ◆ Incremental data: similar in size to object files

Results

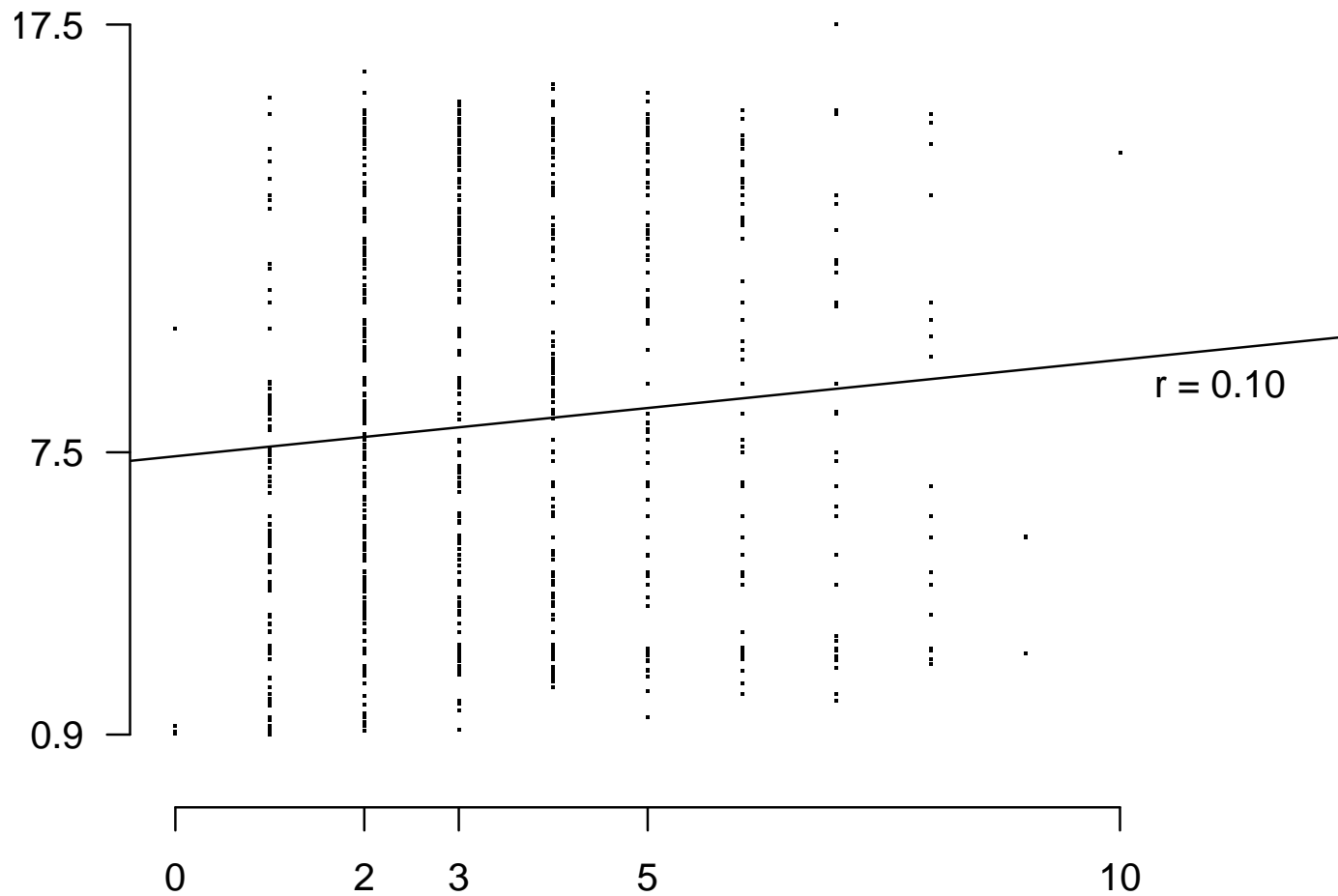
- ◆ Forward algorithm speedups: max 1.6, avg. 1.4
- ◆ Backward algorithm speedups: max 17.5, avg. 8.2
- ◆ Libraries (guievict): avg. 11.3
- ◆ Larger changes (10–50% of functions): avg. 2.5
- ◆ Incremental data: similar in size to object files

Conclusions

Incrementalization provides:

- ◆ significant performance increases
- ◆ no loss of precision
- ◆ a good fit with development practices

Call Depth v. Speedup



Callers v. Speedup

