Static Single-Assignment Form
and
Dataflow Analysis

Roadmap

Last time:

- Optimization overview
    - Soundness and completeness
- Simple optimizations
    - Peephole
    - LICM

This time:

- Data structures (and data) used to determine when it is safe (i.e., sound) to perform an optimizing transformation
    - Dominators
    - SSA form
    - Dataflow analysis

DOMINATOR REVIEW

Dominator terms

Domination (A dominates B):

– to reach block B, you must have gone through block A

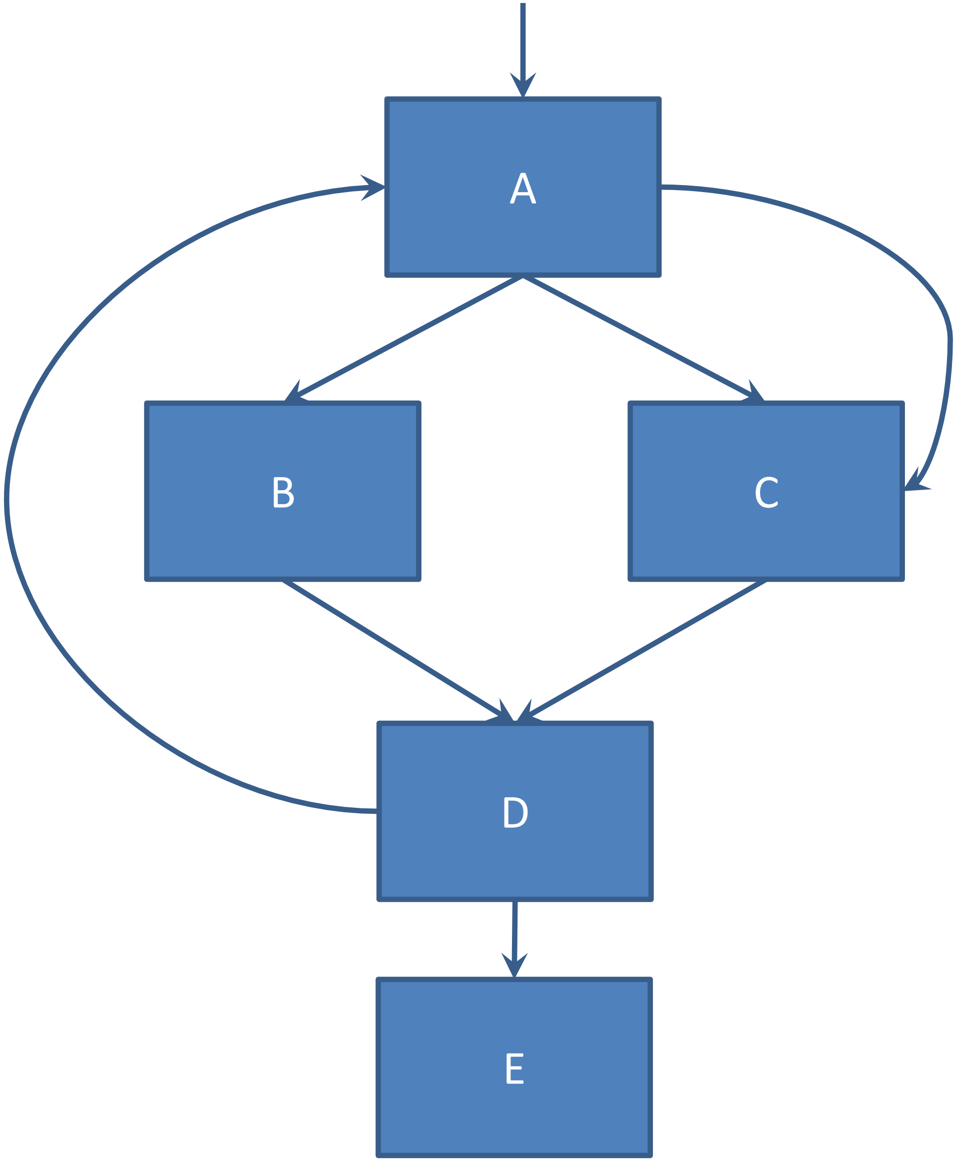
Strict Domination (A strictly dominates B)

– A dominates B and A is not B

Immediate Domination (A immediately dominates B)

– A immediately dominates B if A dominates B and has no intervening dominators

Dominator Example



Dominance Frontier

Definition: For a block X, the set of nodes Y such that X dominates an immediate predecessor of Y but does not strictly dominate Y

STATIC SINGLE ASSIGNMENT FORM (SSA FORM)

Goal of SSA Form

Build an intermediate representation of the program in which each variable is assigned a value in **at most 1 program point**:

✓

```
x = 1
z = 2
y = 3
```

✓

```
x = y
z = y
w = z
```

✗

```
x = 1
x = 2
y = 3
```

✓

```
i = 0;
while( i < 10){
  k = i + 1;
}
```

Statically: There is at most *one* assignment statement that assigns to k
Dynamically: k can be assigned to *multiple* times

Conversion

We make new variables to carry over the effect of the original program

✗	✓
$x = 1$	$x_1 = 1$
$x = x$	$x_2 = x_1$
$y = x$	$y_1 = x_2$

Benefits of SSA Form

There are some obvious advantages to this format for program analysis

– Easy to see the *live range* of a given variable x assigned to in statement s
  - The region from “x = …;” until the last use(s) of x before x is redefined
  - In SSA form, from “x_i = …;” to all uses of x_i, e.g., “… = f(…, x_i, …);”

– Easy to see when an assignment is *useless*
  - We have “x_i = …;” and there are *no uses* of x_i in any expression or assignment RHS
  - “‘x_i = …;’ is a useless assignment”
  - “‘x_i = …;’ is dead code”

In other words, some useful information is pre-computed, or at least easily recoverable [illegible]

Warning 1: Dead code = useless assignments + unreachable code

Optim … Helps

At “if (b < 4)”, b is only reached by “b = 2;” Therefore, the else branch is unreachable (dead), and can be removed

Dead-Code Eli… tion

~~int a = 9;~~
~~int b = 2;~~

~~if (g < 12){~~
    ~~a = 1;~~
~~} else {~~
    ~~if (b < 4){~~
        ~~a = 2;~~
    } ~~else {~~
        ~~a = 3;~~
    ~~}~~
~~}~~
~~b = a;~~
return 2;

$\texttt{int } a_1 = 9;$
$\texttt{int } b_1 = 2;$

$\texttt{if } (g_1 < 12)\{$
$\quad a_2 = 1;$
$\}\texttt{ else }\{$
$\quad \texttt{if } (b_1 < 4)\{$
$\qquad a_3 = 2;$
$\quad \}\texttt{ else }\{$
$\qquad a_4 = 3;$
$\quad \}$
$\quad a_5 = \phi(a_3, a_4);$
$\}$
$a_6 = \phi(a_2, a_5);$
$b_2 = a_6;$
$\texttt{return } 2;$

Optimizations Where SSA Helps

Constant-propagation/constant-folding

```
int a = 30;
int b = 9 - (a / 5);
int c;
c = b * 4;
if (c > 10) {
    c = c - 10;
}
return c * (60 / a);
```

```
int a = 30;
int b = 3;
int c;
c = 12;
if (true) {
    c = 2;
}
return 4;
```

6

true

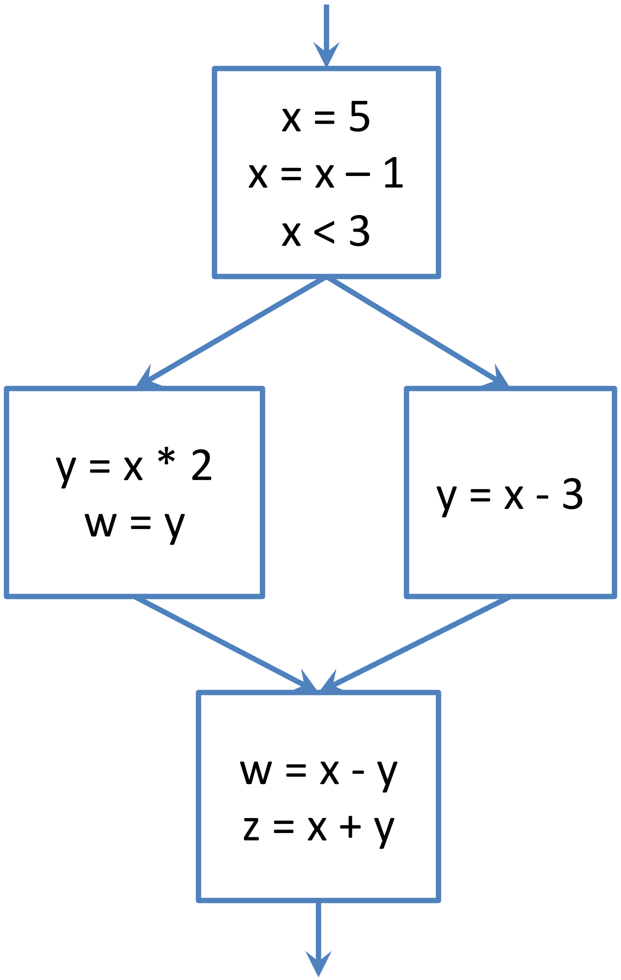
12

2

2

4

What About Conditionals?



$x = 5$

$x = x – 1$

$x < 3$

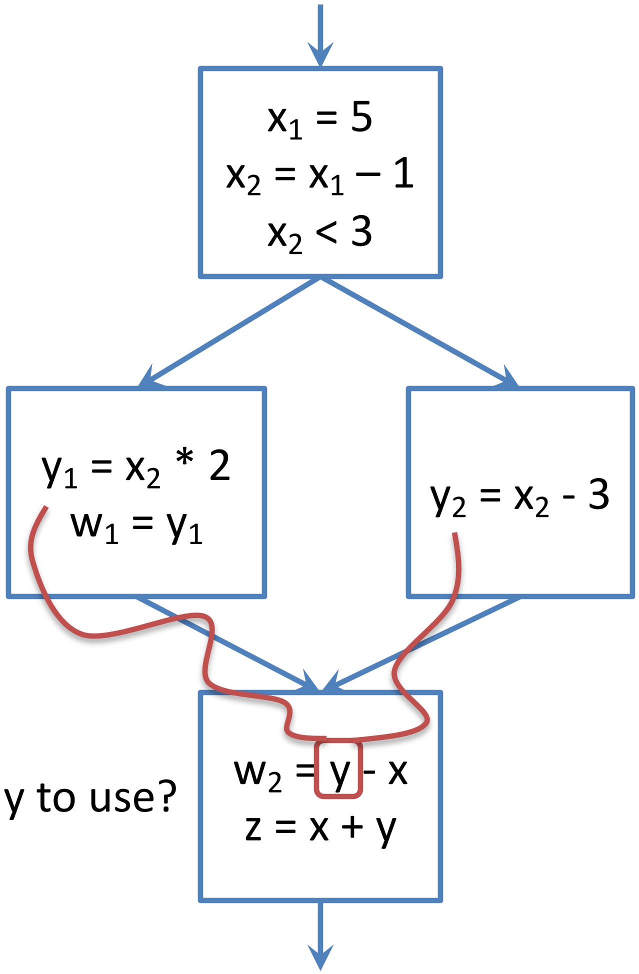
$y = x * 2$

$w = y$

$y = x - 3$

$w = x - y$

$z = x + y$



$x_1 = 5$

$x_2 = x_1 – 1$

$x_2 < 3$

$y_1 = x_2 * 2$

$w_1 = y_1$

$y_2 = x_2 - 3$

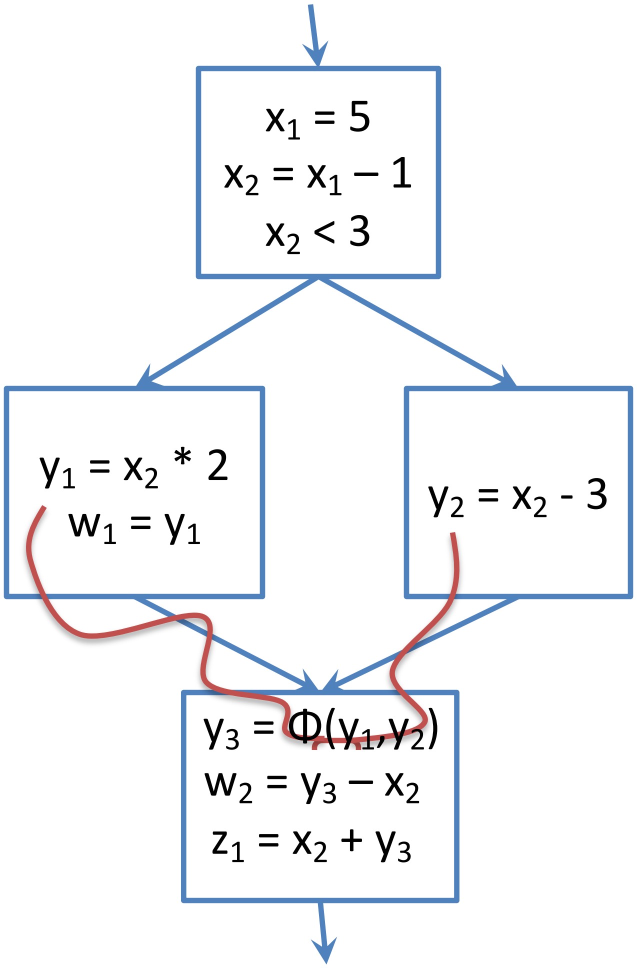
$w_2 = \boxed{y} - x$

$z = x + y$

Which y to use?

Phi Functions (ϕ)

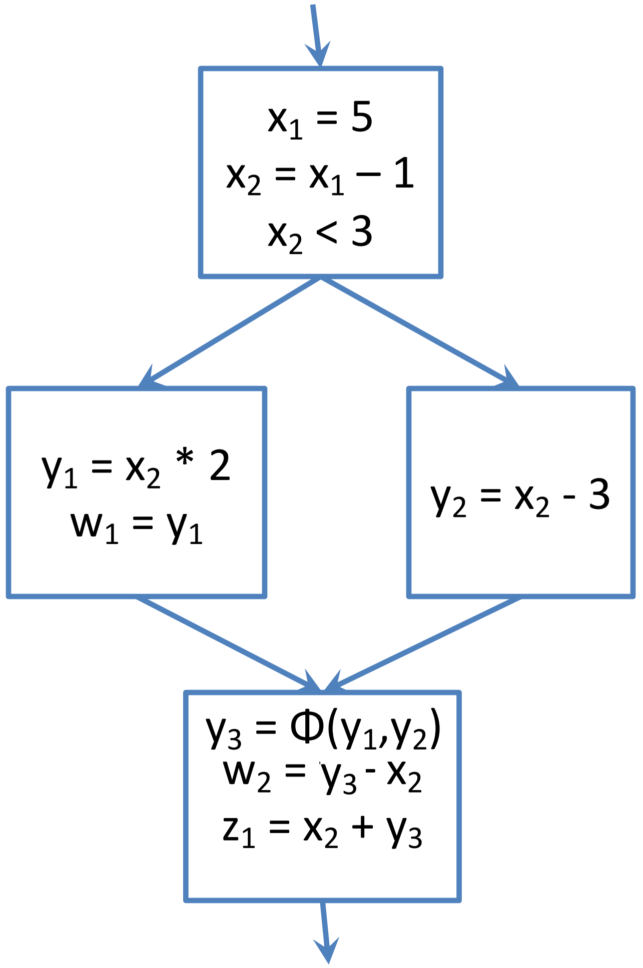
- We introduce a special symbol Φ at such points of confluence
- Φ's arguments are all the instances of variable y that might be the most recently assigned variant of y
- Returns the "correct" one
- Do we need a Φ for x?
  - No!



Computing Phi-Function Placement

Intuitively, we want to figure out cases where there are multiple assignments that can reach a node

To be safe, we can place a Φ function for each assignment at every node in the *dominance frontier*



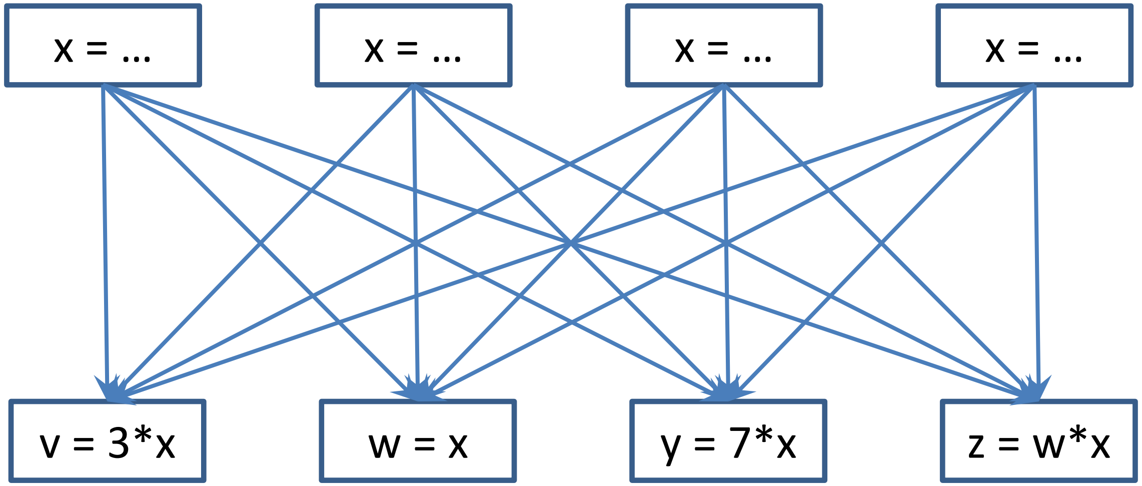
Pruned Phi Functions

This criterion causes a bunch of useless Φ functions to be inserted

- Cases where the result is never used “downstream” (useless)

Pruned SSA is a version where useless Φ nodes are suppressed

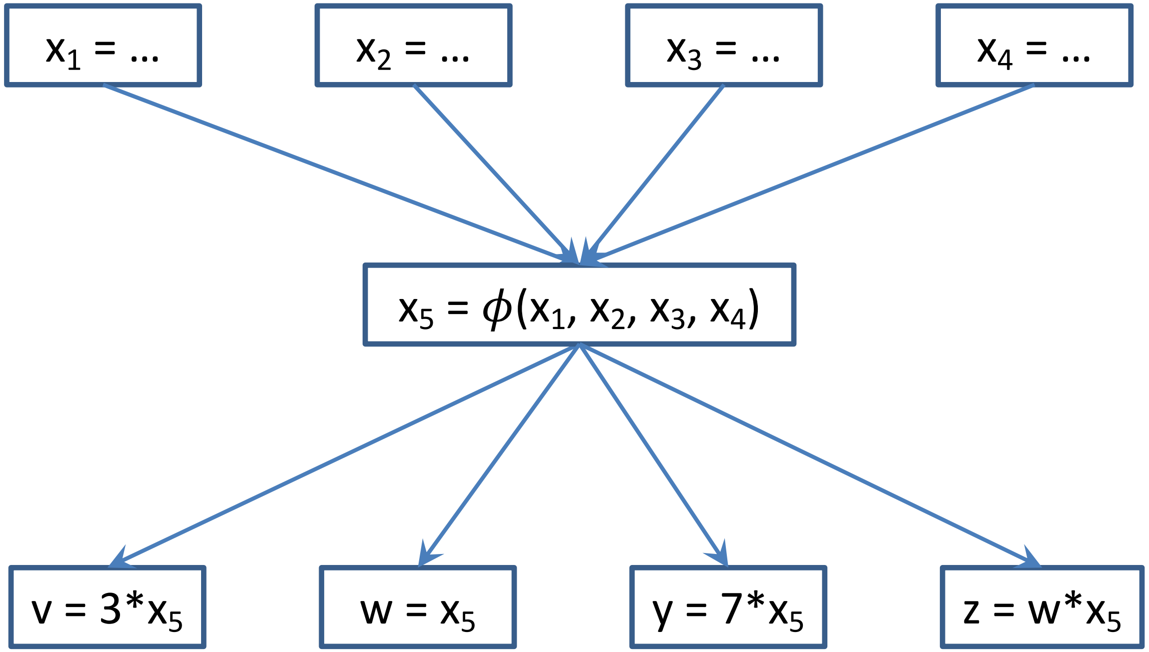
Other Advantages of SSA Form



Flow dependences

4×4 edges

Other Benefits of SSA Form



Multiplicative representation → Additive representation

4×4 edges → 4 + 4 edges

DATAFLOW ANALYSIS

Dataflow-Analysis Example 1

Reaching definitions

Transfer function:
$\lambda S.(S - \{< p_i, x >\}) \cup \{< p2, x >\}$

Before p1: ∅
After p1: {<p1, x>}

p1: x = 1;

. . .

Before p2: {<p1, x>, …}
After p2: {<p2, x>, …}

p2: x = 2;

. . .

Before p3: {<p2, x>, …}
After p3: {<p2, x>, <p3, y>, …}

p3: y = x;

Data: sets of <program-point, variable> pairs

Note: for expository purposes, it is convenient to assume we have a statement-level CFG rather than a basic-block-level CFG.

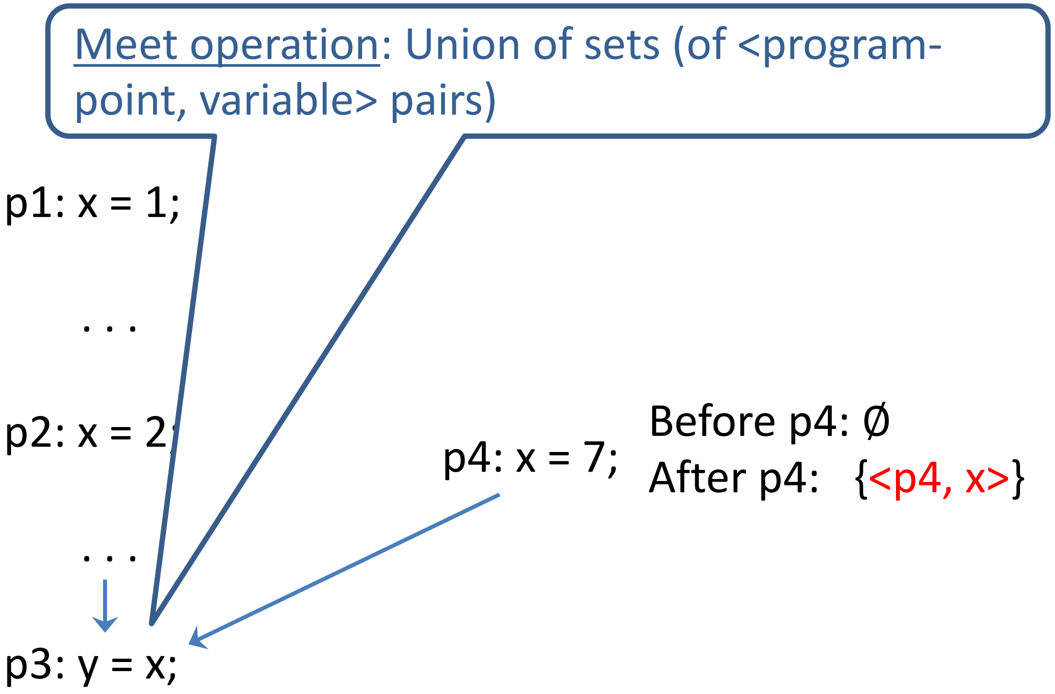
Dataflow-Analysis Example 1

Reaching definitions

Before p1: Ø
After p1: {<p1, x>}

Before p2: {<p1, x>, …}
After p2: {<p2, x>, …}

Before p3: {<p2, x>, <p4,x>, …}
After p3: {<p2, x>, <p3, y>, <p4,x>,…}



Meet operation: Union of sets (of <program-point, variable> pairs)

p1: x = 1;

. . .

p2: x = 2;

. . .

p3: y = x;

p4: x = 7;

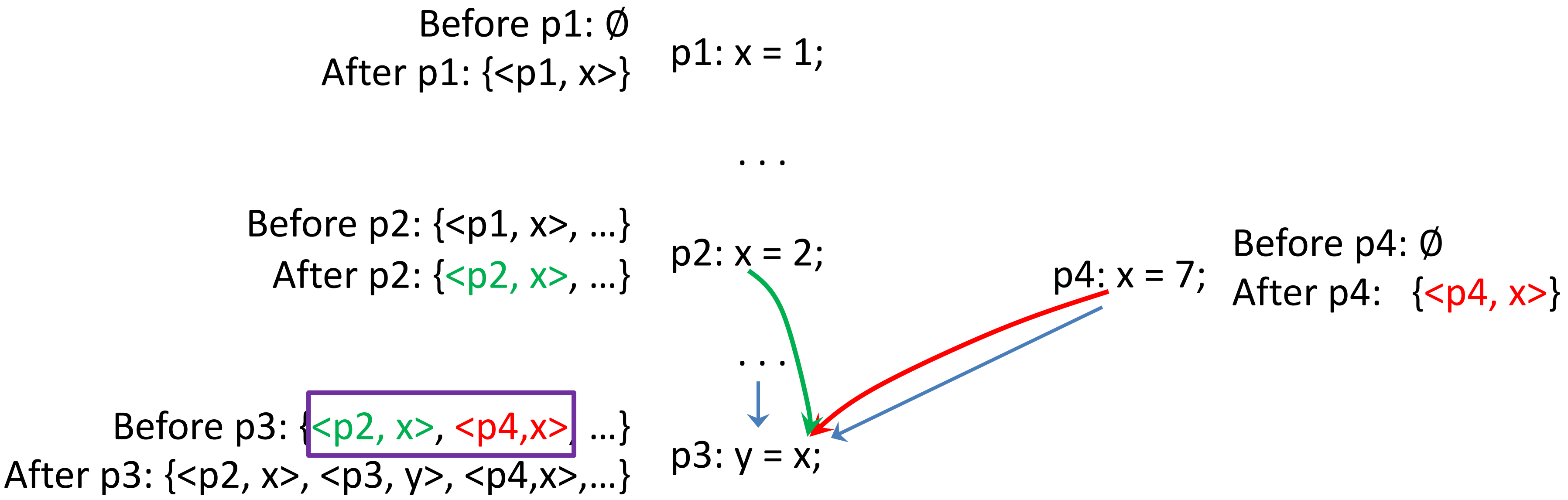
Before p4: Ø
After p4: {<p4, x>}

Note: for expository purposes, it is convenient to assume we have a statement-level CFG rather than a basic-block-level CFG.

Dataflow-Analysis Example 1

Reaching definitions: Why is it useful?

Answers the question “Where could this variable have been defined?”



Before p1: ∅

After p1: {<p1, x>}

p1: x = 1;

. . .

Before p2: {<p1, x>, …}

After p2: {<p2, x>, …}

p2: x = 2;

p4: x = 7;

Before p4: ∅

After p4: {<p4, x>}

. . .

Before p3: {<p2, x>, <p4,x>, …}

After p3: {<p2, x>, <p3, y>, <p4,x>,…}

p3: y = x;

Dataflow-Analysis Example 2

Live Variables

Before p1:	∅
After p1:	{x}
Before p2:	{x}
After p2:	{x,y}
Before p3:	{x,y}
After p3:	∅
Before p4:	∅
After p4:	{x}
Before p5:	{x}
After p5:	{x}
Before p6:	{x}
After p6:	∅

```
p1: x = 1;

    if (...) {
        p2: y = 0;

        p3: z = x + y;
    }

p4: x = 2;

p5: z = 3;

p6: cout << x;
```

Transfer function:
$\lambda S.(S - \{z\}) \cup \{x, y\}$

Data: sets of variables

z is not live after p5, and thus p5 is a useless assignment (= dead code)