Lecture 3: WHILE3ADDR, Control-Flow Graphs and Intro to Data-Flow Anlaysis

17-355/17-665/17-819: Program Analysis

Rohan Padhye

September 4, 2025

* Course materials developed with Jonathan Aldrich and Claire Le Goues

Review: WHILE abstract syntax

S — statements

a — arithmetic expressions (AExp)

x, y — program variables (Vars)

n — number literals

b — boolean expressions (BExp)

$$
\begin{aligned}
S \; ::= \; & x := a \\
| \; & \text{skip} \\
| \; & S_1;\; S_2 \\
| \; & \text{if } b \text{ then } S_1 \text{ else } S_2 \\
| \; & \text{while } b \text{ do } S
\end{aligned}
\qquad
\begin{aligned}
b \; ::= \; & \text{true} \\
| \; & \text{false} \\
| \; & \text{not } b \\
| \; & b_1 \; op_b \; b_2 \\
| \; & a_1 \; op_r \; a_2
\end{aligned}
\qquad
\begin{aligned}
a \; ::= \; & x \\
| \; & n \\
| \; & a_1 \; op_a \; a_2
\end{aligned}
\qquad
\begin{aligned}
op_b \; ::= \; & \text{and} \mid \text{or} \\
op_r \; ::= \; & < \; \mid \; \leqslant \; \mid \; = \\
& \mid \; > \; \mid \; \geqslant \\
op_a \; ::= \; & + \mid - \mid * \mid /
\end{aligned}
$$

WHILE syntax

- Abstract representation that corresponds well to concrete syntax

- Useful for recursive or inductive reasoning

- Sometimes challenging to track how data and control flows in program execution order

- 3-address-code is commonly used by compilers to represent imperative language code.
  - AST -> 3-address transformation is straightforward.

WHILE3ADDR

- w = x * y + z

- if b then S1 else S2

- 1: t = x * y
  2: w = t + z

- 1: if b then goto 4
  2: S2
  3: goto 5
  4: S1
  5: ...

WHILE3ADDR: An Intermediate Representation

Simpler, more uniform than WHILE syntax

Categories:

I ∈ **Instruction**   instructions
x, y ∈ **Var**   variables
n ∈ **Num**   number literals

Syntax:

I ::= x := n | x := y | x := y op z
         | goto n | if x op_r 0 goto n
op_a ::= + | - | * | / | …
op_r ::= < | ≤ | = | > | ≥ | ...
P ∈ **Num** → I

Exercise: Translate *while b do S* to WHILE3ADDR

Categories:

$I \in$ **Instruction**   instructions

$x, y \in$ **Var**   variables

$n \in$ **Num**   number literals

Syntax:

$I ::= x := n \mid x := y \mid x := y\ op\ z$
    $\mid$ goto $n \mid$ if $x\ op_r\ 0$ goto n

$op_a ::= + \mid - \mid * \mid / \mid \ldots$

$op_r ::= < \mid \leq \mid = \mid > \mid \geq \mid \ldots$

$P \in$ **Num** $\rightarrow I$

S3D — Software and Societal Systems Department

Carnegie Mellon University

While3Addr Extensions (more later)

```
I  ::=     x := n | x := y | x := y op z |  goto n | if x op_r 0 goto n
        |  halt
        |  x := f(y)
        |  return x
        |  x := y.m(z)
        |  read x
        |  print x
        |  x := &p
        |  x := *p
        |  *p := x
        |  x := y.f
        |  x.f := y
```

WHILE3ADDR Semantics

- Configuration (state) includes environment + program counter:

$$c \in E \times \mathbb{N}$$

- Evaluation occurs with respect to a global program that maps labels to instructions: $P \in \mathbb{N} \to I$

$$P \vdash \langle E, n \rangle \rightsquigarrow \langle E', n' \rangle$$

S3D Software and Societal Systems Department

Carnegie Mellon University

$$\frac{P(n) = x := m}{P \vdash \langle E, n \rangle \rightsquigarrow \langle E[x \mapsto m], n+1 \rangle} \textit{step-const}$$

$$\frac{P[n] = x := y}{P \vdash \langle E, n \rangle \rightsquigarrow \langle E[x \mapsto E(y)], n+1 \rangle} \textit{step-copy}$$

$$\frac{P(n) = x := y \ op \ z \quad E(y) \ \mathbf{op} \ E(z) = m}{P \vdash \langle E, n \rangle \rightsquigarrow \langle E[x \mapsto m], n+1 \rangle} \textit{step-arith}$$

$$\frac{P(n) = \mathbf{goto} \ m}{P \vdash \langle E, n \rangle \rightsquigarrow \langle E, m \rangle} \textit{step-goto}$$

$$\frac{P(n) = \mathbf{if} \ x \ op_r \ 0 \ \mathbf{goto} \ m \quad E(x) \ \mathbf{op_r} \ 0 = \textit{true}}{P \vdash \langle E, n \rangle \rightsquigarrow \langle E, m \rangle} \textit{step-iftrue}$$

$$\frac{P(n) = \mathbf{if} \ x \ op_r \ 0 \ \mathbf{goto} \ m \quad E(x) \ \mathbf{op_r} \ 0 = \textit{false}}{P \vdash \langle E, n \rangle \rightsquigarrow \langle E, n+1 \rangle} \textit{step-iffalse}$$

Data-Flow Analysis

Computes universal properties about program state at specific program points. (e.g. will *x* be zero at line 7?)

- About program state
  - About data store (e.g. variables, heap memory)
  - Not about control (e.g. termination, performance)
- At program points
  - Statically identifiable (e.g. line 7, or when foo() calls bar())
  - Not dynamically computed (E.g. when x is 12 or when foo() is invoked 12 times)
- Universal
  - Reasons about all possible executions (always/never/maybe)
  - Not about specific program paths (see: symbolic execution, testing)

Abstraction

$$\sigma \in Var \rightarrow L$$

$$\alpha : \mathbb{Z} \rightarrow L$$

Abstraction

Zero Analysis

$$\sigma \in Var \rightarrow L$$

$$L = \{Z, N, \top\}$$

$$\alpha : \mathbb{Z} \rightarrow L$$

$$\alpha_Z(0) = Z$$
$$\alpha_Z(n) = N \text{ where } n \neq 0$$

Flow Functions for Zero Analysis

A flow function maps values from σ to σ $f[\![I]\!]$ -- flow across instruction I (think: "abstract semantics")

$f_Z[\![x := 0]\!](\sigma) \quad = $

$f_Z[\![x := n]\!](\sigma) \quad = $

$f_Z[\![x := y]\!](\sigma) \quad = $

$f_Z[\![x := y\ op\ z]\!](\sigma) \quad = $

$f_Z[\![\mathrm{goto}\ n]\!](\sigma) \quad = $

$f_Z[\![\mathrm{if}\ x = 0\ \mathrm{goto}\ n]\!](\sigma) \quad = $

Software and Societal
Systems Department
(c) J. Aldrich, C. Le Goues, R. Padhye
Carnegie
Mellon
University

Flow Functions for Zero Analysis

A flow function maps values from σ to σ $f[\![I]\!]$ -- flow across instruction I (think: "abstract semantics")

$$f_Z[\![x := 0]\!](\sigma) \qquad = \sigma[x \mapsto Z]$$
$$f_Z[\![x := n]\!](\sigma) \qquad = \sigma[x \mapsto N] \ \textbf{ where } n \neq 0$$

$$f_Z[\![x := y]\!](\sigma) \qquad = \sigma[x \mapsto \sigma(y)]$$

$$f_Z[\![x := y \ op \ z]\!](\sigma) \qquad = \sigma[x \mapsto \top]$$

$$f_Z[\![\text{goto } n]\!](\sigma) \qquad = \sigma$$

$$f_Z[\![\text{if } x = 0 \text{ goto } n]\!](\sigma) \qquad = \sigma$$

Flow Functions for Zero Analysis

Specializing for Precision

$$f_Z[\![x := y - y]\!](\sigma) \quad =$$

$$f_Z[\![x := y + z]\!](\sigma) \quad =$$

S3D Software and Societal
Systems Department

(c) J. Aldrich, C. Le Goues, R. Padhye

Carnegie
Mellon
University

Flow Functions for Zero Analysis

Specializing for Precision

$$f_Z[\![x := y - y]\!](\sigma) \quad = \sigma[x \mapsto Z]$$

$$f_Z[\![x := y + z]\!](\sigma) \quad = \sigma[x \mapsto \sigma(y)] \quad \textbf{where } \sigma(z) = Z$$

Exercise: Define another flow function for some arithmetic instruction and certain conditions where you can also provide a more precise result than ⊤

Flow Functions for Zero Analysis

Specializing for Precision

$$f_Z[\![\text{if } x = 0 \text{ goto } n]\!]_T(\sigma) \quad =$$
$$f_Z[\![\text{if } x = 0 \text{ goto } n]\!]_F(\sigma) \quad =$$

Flow Functions for Zero Analysis

Specializing for Precision

$$f_Z[\![\text{if } x = 0 \text{ goto } n]\!]_T(\sigma) \quad = \sigma[x \mapsto Z]$$
$$f_Z[\![\text{if } x = 0 \text{ goto } n]\!]_F(\sigma) \quad = \sigma[x \mapsto N]$$

Exercise: Define a flow function for a conditional branch testing whether a variable x < 0

S3D Software and Societal
Systems Department

(c) J. Aldrich, C. Le Goues, R. Padhye

Carnegie
Mellon
University

Control-flow Graphs

$1:$ if $x = 0$ goto 4
$2:$ $y := 0$
$3:$ goto 6
$4:$ $y := 1$
$5:$ $x := 1$
$6:$ $z := y$

Nodes = Statements
Edges = (s1, s2) is an edge iff s1 and s2
       can be executed consecutively
       aka "control flow"

1: if $x = 0$ goto 4

2: $y \coloneqq 0$

3: goto 6

4: $y \coloneqq 1$

5: $x \coloneqq 1$

6: $z \coloneqq y$

Control-flow Graphs

$$1: \quad \text{if } x = 0 \text{ goto } 4$$
$$2: \quad y := 0$$
$$3: \quad \text{goto } 6$$
$$4: \quad y := 1$$
$$5: \quad x := 1$$
$$6: \quad z := y$$
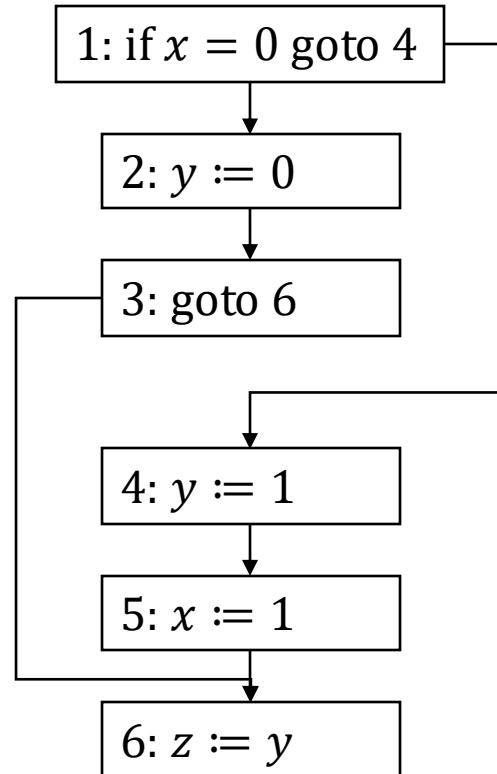
Nodes = Statements
Edges = (s1, s2) is an edge iff s1 and s2
        can be executed consecutively
        aka "control flow"

1: if $x = 0$ goto 4
2: $y \coloneqq 0$
3: goto 6
4: $y \coloneqq 1$
5: $x \coloneqq 1$
6: $z \coloneqq y$

Common properties of CFGs:
- Weakly connected
- Only one entry node
- Only one exit (terminal) node

Example of Zero Analysis: Straightline Code

$$1: \quad x := 0$$
$$2: \quad y := 1$$
$$3: \quad z := y$$
$$4: \quad y := z + x$$
$$5: \quad x := y - z$$

P1

1: $x := 0$

↓ P2

2: $y := 1$

↓ P3

3: $z := y$

↓ P4

4: $y := z + x$

↓ P5

5: $x := y - z$

P6
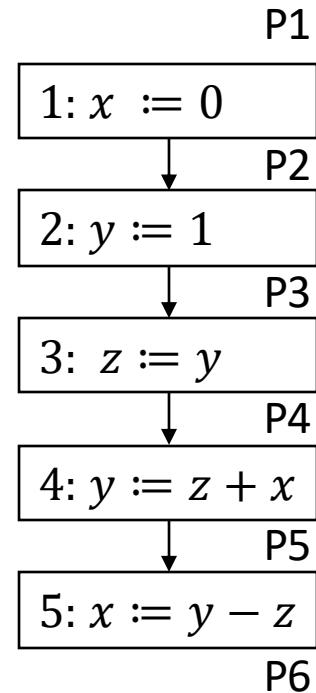
	x	y	z
P1			
P2			
P3			
P4			
P5			
P6			

Example of Zero Analysis: Branching Code

$$1: \quad \text{if } x = 0 \text{ goto } 4$$
$$2: \quad y := 0$$
$$3: \quad \text{goto } 6$$
$$4: \quad y := 1$$
$$5: \quad x := 1$$
$$6: \quad z := y$$
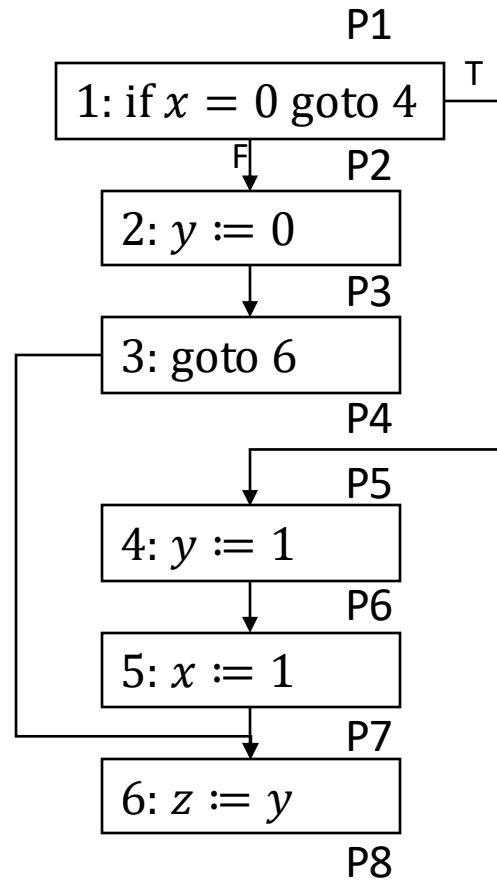
	x	y	z
P1			
P2			
P3			
P4			
P5			
P6			
P7			
P8			

Example of Zero Analysis: Branching Code

$1:$ if $x = 0$ goto 4
$2:$ $y := 0$
$3:$ goto 6
$4:$ $y := 1$
$5:$ $x := 1$
$6:$ $z := y$

P1
| 1: if $x = 0$ goto 4 | T |

F P2
| 2: $y := 0$ |

P3
| 3: goto 6 |

P4

P5
| 4: $y := 1$ |

P6
| 5: $x := 1$ |

P7
| 6: $z := y$ |

P8
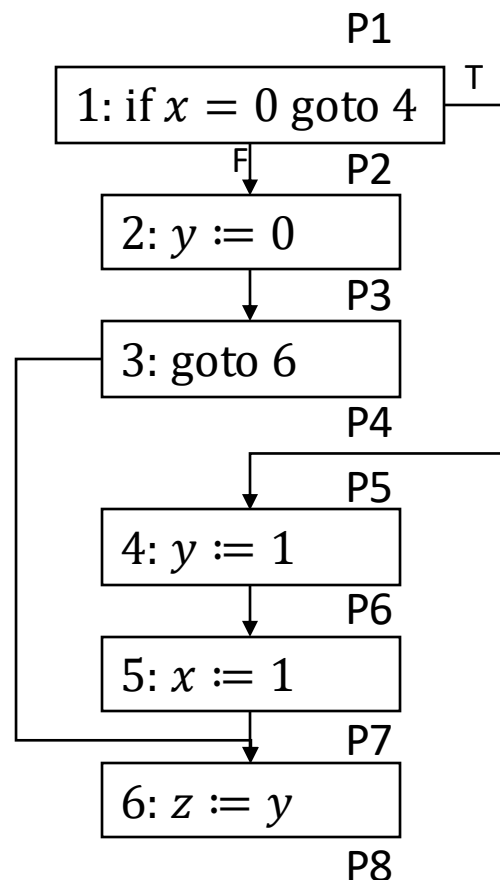
	x	y	z
P1	?	?	?
P2	Z_T, N_F	?	?
P3	N	Z	?
P4	N	Z	?
P5	Z	?	?
P6	Z	N	?
P7	N	N?	?
P8	N??	N??	N??

Example of Zero Analysis: Branching Code

$$1: \quad \text{if } x = 0 \text{ goto } 4$$
$$2: \quad y := 0$$
$$3: \quad \text{goto } 6$$
$$4: \quad y := 1$$
$$5: \quad x := 1$$
$$6: \quad z := y$$

P1

| 1: if $x = 0$ goto 4 | T |

F P2

| 2: $y := 0$ |

P3

| 3: goto 6 |

P4

P5

| 4: $y := 1$ |

P6

| 5: $x := 1$ |

P7

| 6: $z := y$ |

P8

	x	y	z
P1	?	?	?
P2	Z_T, N_F	?	?
P3	N	Z	?
P4	N	Z	?
P5	Z	?	?
P6	Z	N	?
P7	N	$\top$	?
P8	N	$\top$	$\top$
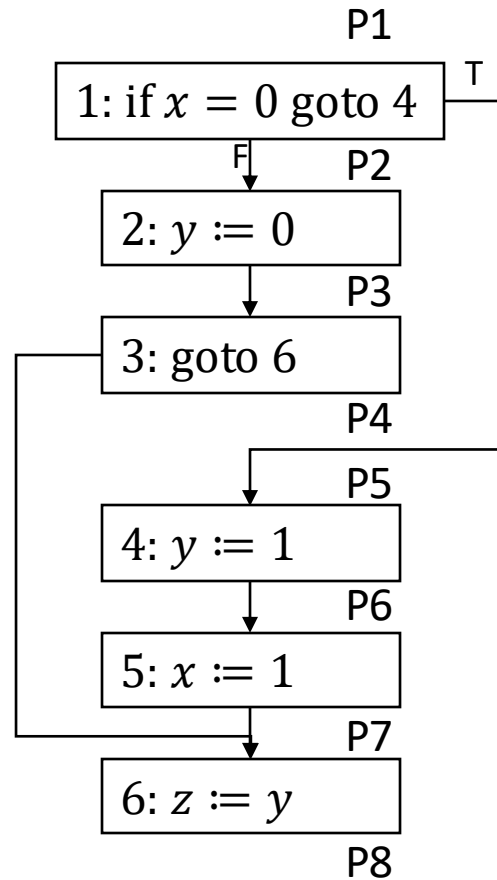
Next Time

- Lattices
- Definition of a Data-Flow Analysis
- Solution of a Data-Flow Analysis
- Kildall's Algorithm