Dataflow Analysis + Intro to SSA

CSE 401 Section 9

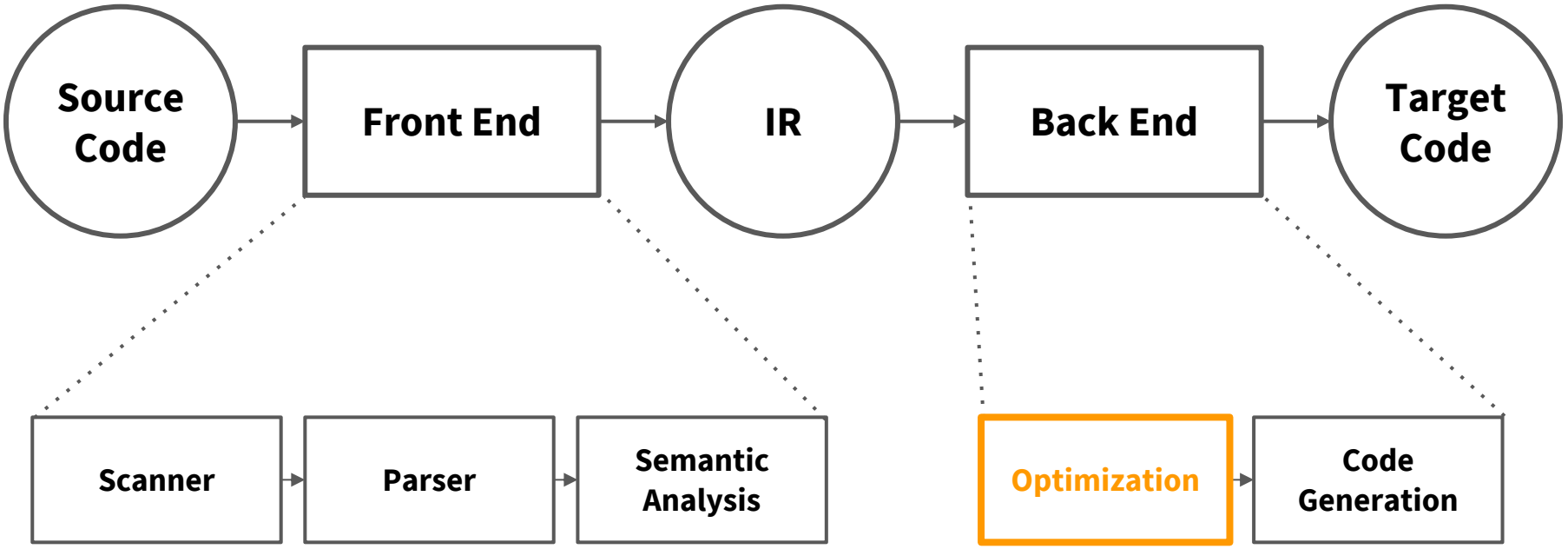
by Anand, Gavin, Yukai

Announcements

- CodeGen due tonight
  - Late days can be used on any – You have 2 free late days for CodeGen!
- 401 report due next Tuesday; M501 project/ report due next Saturday
- HW 4 due next Thursday

24 14:30-15:20 Lecture zoom link *Dataflow (cont.)*	25	26 14:30-15:20 Lecture zoom link *Dataflow (concl.); SSA (start)* ssa: slides	27 WE ARE HERE Section *Dataflow & SSA* 23:00 Project: code generation due	28 14:30-15:20 Lecture zoom link *SSA (concl.); Back end overview; instruction selection*
31 Memorial Day	01 23:00 Project: CSE 401 project reports due **No late submissions accepted.**	02 14:30-15:20 Lecture zoom link *Instruction scheduling; register allocation (no new slides)*	03 Section *hw4 last-minute questions; wrapup* 23:00 hw4 due (dataflow & SSA)	04 14:30-15:20 Lecture zoom link *Garbage collection & wrapup* 23:00 CSE M 501 project due **Saturday** 11 pm **No late submissions accepted** 23:00 CSE M 501 report due **Sunday** 11 pm **No late submissions accepted**

Review of Optimizations



Review of Optimizations

Peephole

Local

Intraprocedural / Global

Interprocedural

Review of Optimizations

Peephole	A few Instructions
Local	
Intraprocedural / Global	
Interprocedural	

Review of Optimizations

Peephole	A few Instructions
Local	A Basic Block
Intraprocedural / Global	
Interprocedural	

Review of Optimizations

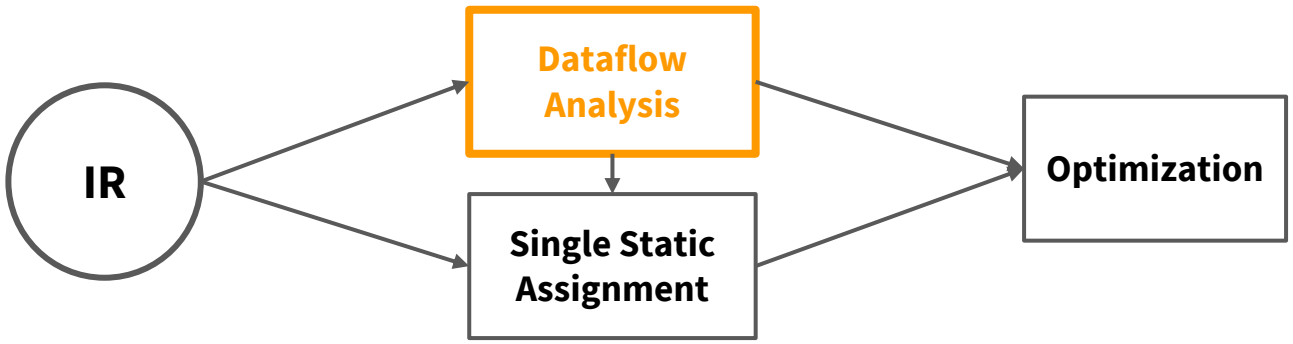
Peephole	A few Instructions
Local	A Basic Block
Intraprocedural / Global	A Function/Method
Interprocedural	

Review of Optimizations

Peephole	A few Instructions
Local	A Basic Block
Intraprocedural / Global	A Function/Method
Interprocedural	A Program

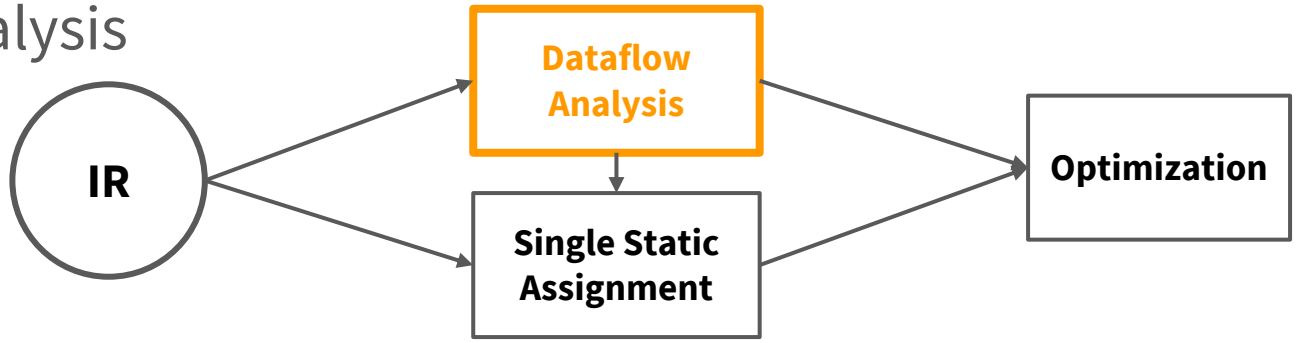
Overview of Dataflow Analysis

- A framework for exposing properties about programs
- Operates using sets of “facts”



Overview of Dataflow Analysis

- A framework for exposing properties about programs
- Operates using sets of “facts”
- Just the initial discovery phase
  - Changes can then be made to optimize based on the analysis



Overview of Dataflow Analysis

- Basic Framework of Set Definitions (for a Basic Block *b*):
  - `IN(`*b*`)`: facts true on entry to *b*
  - `OUT(`*b*`)`: facts true on exit from *b*
  - `GEN(`*b*`)`: facts created (and not killed) in *b*
  - `KILL(`*b*`)`: facts killed in *b*

Reaching Definitions (*A Dataflow Problem*)

“What definitions of each variable might reach this point”

- Could be used for:
  - Constant Propagation
  - Uninitialized Variables

```
int x;

if (y > 0) {
  x = y;
} else {
  x = 0;
}

System.out.println(x);
```

“x=y”, “x=0”

Reaching Definitions (*A Dataflow Problem*)

"What definitions of each variable might reach this point"

- **Be careful**: Does not involve the *value* of the definition
  - The dataflow problem "Available Expressions" is designed for that

```
int x;

if (y > 0) {
  x = y;
} else {
  x = 0;
}

y = -1;
System.out.println(x);
```

still: "x=y", "x=0"

Equations for Reaching Definitions

- $\texttt{IN}(b)$: the definitions reaching upon entering block b
- $\texttt{OUT}(b)$: the definitions reaching upon exiting block b
- $\texttt{GEN}(b)$: the definitions assigned and not killed in block b
- $\texttt{KILL}(b)$: the definitions of variables overwritten in block b

$$\texttt{IN}(b) = \bigcup_{p \in \texttt{pred}(b)} \texttt{OUT}(\texttt{p})$$

$$\texttt{OUT}(b) = \texttt{GEN}(b) \cup (\texttt{IN}(b) - \texttt{KILL}(b))$$

Another *Equivalent* Set of Equations (from Lecture):

- Sets:
  - `DEFOUT(b)`: set of definitions in b that reach the end of b (i.e., not subsequently redefined in b)
  - `SURVIVED(b)`: set of all definitions not obscured by a definition in b
  - `REACHES(b)`: set of definitions that reach b
- Equations:

$$\texttt{REACHES(b)} = \left(\bigcup_{p \in \texttt{preds(b)}} \texttt{DEFOUT(p)}\right) \cup \left(\texttt{REACHES(p)} \cap \texttt{SURVIVED(p)}\right)$$

Problems 1_a and 1_b

```
L0:  a = 0
L1:  b = a + 1
L2:  c = c + b
L3:  a = b * 2
L4:  if a < N goto L1
L5:  return c
```

Block	GEN	KILL	IN (1)	OUT (1)	IN (2)	OUT (2)
L0	L0					
L1	L1					
L2	L2					
L3	L3					
L4						
L5						

```
L0:  a = 0
L1:  b = a + 1
L2:  c = c + b
L3:  a = b * 2
L4:  if a < N goto L1
L5:  return c
```

Block	GEN	KILL	IN (1)	OUT (1)	IN (2)	OUT (2)
L0	L0	L3				
L1	L1					
L2	L2					
L3	L3	L0				
L4						
L5						

```
L0:  a = 0
L1:  b = a + 1
L2:  c = c + b
L3:  a = b * 2
L4:  if a < N goto L1
L5:  return c
```

Block	GEN	KILL	IN (1)	OUT (1)	IN (2)	OUT (2)
L0	L0	L3				
L1	L1		L0			
L2	L2		L0, L1			
L3	L3	L0	L0, L1, L2			
L4			L1, L2, L3			
L5			L1, L2, L3			

```
L0:  a = 0
L1:  b = a + 1
L2:  c = c + b
L3:  a = b * 2
L4:  if a < N goto L1
L5:  return c
```

Block	GEN	KILL	IN (1)	OUT (1)	IN (2)	OUT (2)
L0	L0	L3		L0		
L1	L1		L0	L0, L1		
L2	L2		L0, L1	L0, L1, L2		
L3	L3	L0	L0, L1, L2	L1, L2, L3		
L4			L1, L2, L3	L1, L2, L3		
L5			L1, L2, L3	L1, L2, L3		

```
L0:  a = 0
L1:  b = a + 1
L2:  c = c + b
L3:  a = b * 2
L4:  if a < N goto L1
L5:  return c
```

Block	GEN	KILL	IN (1)	OUT (1)	IN (2)	OUT (2)
L0	L0	L3		L0		L0
L1	L1		L0	L0, L1	L0, L1, L2, L3	L0, L1, L2, L3
L2	L2		L0, L1	L0, L1, L2	L0, L1, L2, L3	L0, L1, L2, L3
L3	L3	L0	L0, L1, L2	L1, L2, L3	L0, L1, L2, L3	L1, L2, L3
L4			L1, L2, L3	L1, L2, L3	L1, L2, L3	L1, L2, L3
L5			L1, L2, L3	L1, L2, L3	L1, L2, L3	L1, L2, L3

```
L0:  a = 0
L1:  b = a + 1
L2:  c = c + b
L3:  a = b * 2
L4:  if a < N goto L1
L5:  return c
```

Convergence!

Block	GEN	KILL	IN (1)	OUT (1)	IN (2)	OUT (2)
L0	L0	L3		L0		L0
L1	L1		L0	L0, L1	L0, L1, L2, L3	L0, L1, L2, L3
L2	L2		L0, L1	L0, L1, L2	L0, L1, L2, L3	L0, L1, L2, L3
L3	L3	L0	L0, L1, L2	L1, L2, L3	L0, L1, L2, L3	L1, L2, L3
L4			L1, L2, L3	L1, L2, L3	L1, L2, L3	L1, L2, L3
L5			L1, L2, L3	L1, L2, L3	L1, L2, L3	L1, L2, L3

```
L0:  a = 0
L1:  b = a + 1
L2:  c = c + b
L3:  a = b * 2
L4:  if a < N goto L1
L5:  return c
```

Is it possible to replace the use of *a* in block L1 with the constant 0?

Block	GEN	KILL	IN (1)	OUT (1)	IN (2)	OUT (2)
L0	L0	L3		L0		L0
L1	L1		L0	L0, L1	L0, L1, L2, L3	L0, L1, L2, L3
L2	L2		L0, L1	L0, L1, L2	L0, L1, L2, L3	L0, L1, L2, L3
L3	L3	L0	L0, L1, L2	L1, L2, L3	L0, L1, L2, L3	L1, L2, L3
L4			L1, L2, L3	L1, L2, L3	L1, L2, L3	L1, L2, L3
L5			L1, L2, L3	L1, L2, L3	L1, L2, L3	L1, L2, L3

```
L0:  a = 0
L1:  b = a + 1
L2:  c = c + b
L3:  a = b * 2
L4:  if a < N goto L1
L5:  return c
```

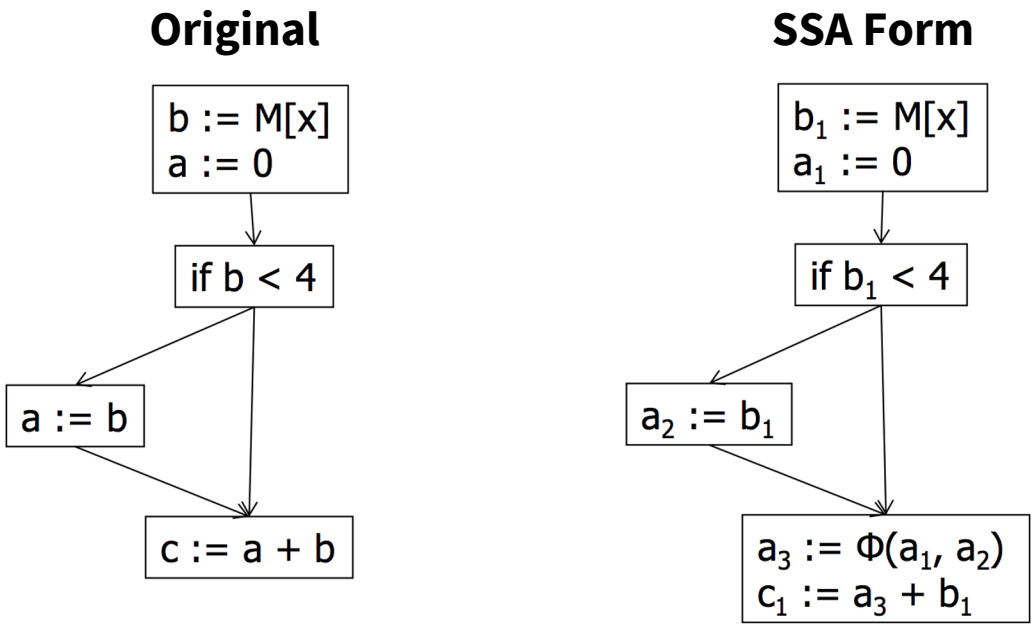
Is it possible to replace the use of *a* in block L1 with the constant 0?

No. To determine this, we would look at the IN set for block L1 -- the fact that the IN set contains two definitions of ‘a’ (L0 and L3) means we cannot perform this constant propagation. In other words, more than one definition of ‘a’ is a reaching definition to block L1, and therefore performing constant propagation would only preserve one possible value of ‘a’ and the generated code would not be equivalent.

Block	GEN	KILL	IN (1)	OUT (1)	IN (2)	OUT (2)
L0	L0	L3		L0		L0
L1	L1		L0	L0, L1	L0, L1, L2, L3	L0, L1, L2, L3
L2	L2		L0, L1	L0, L1, L2	L0, L1, L2, L3	L0, L1, L2, L3
L3	L3	L0	L0, L1, L2	L1, L2, L3	L0, L1, L2, L3	L1, L2, L3
L4			L1, L2, L3	L1, L2, L3	L1, L2, L3	L1, L2, L3
L5			L1, L2, L3	L1, L2, L3	L1, L2, L3	L1, L2, L3

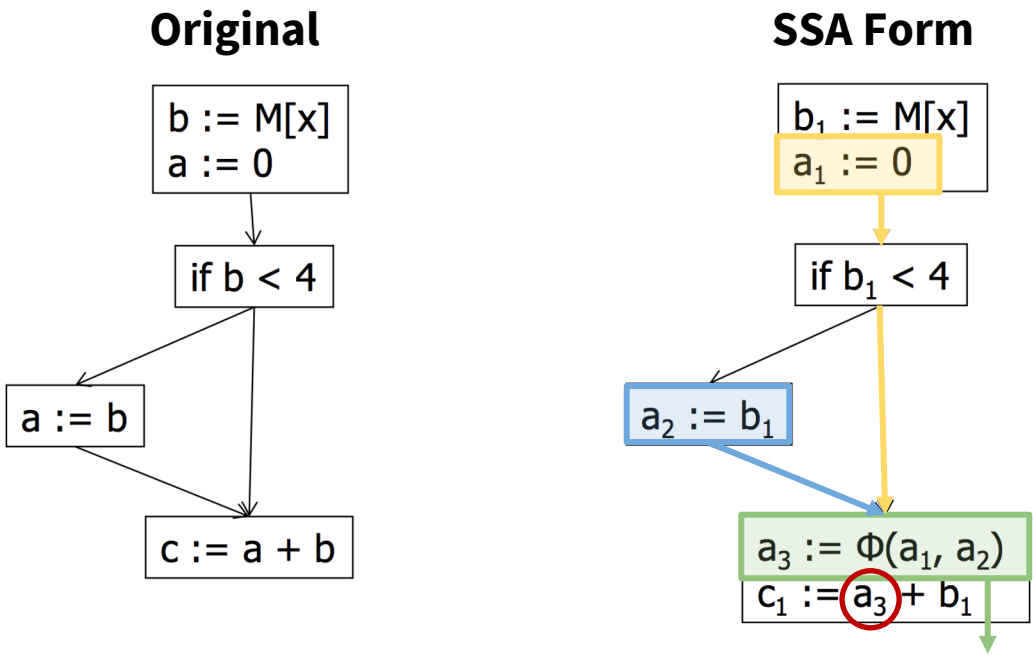
Phi-Functions

- A way to represent multiple possible values for a certain definition
  - Not a “real” instruction – just a form of bookkeeping needed for SSA

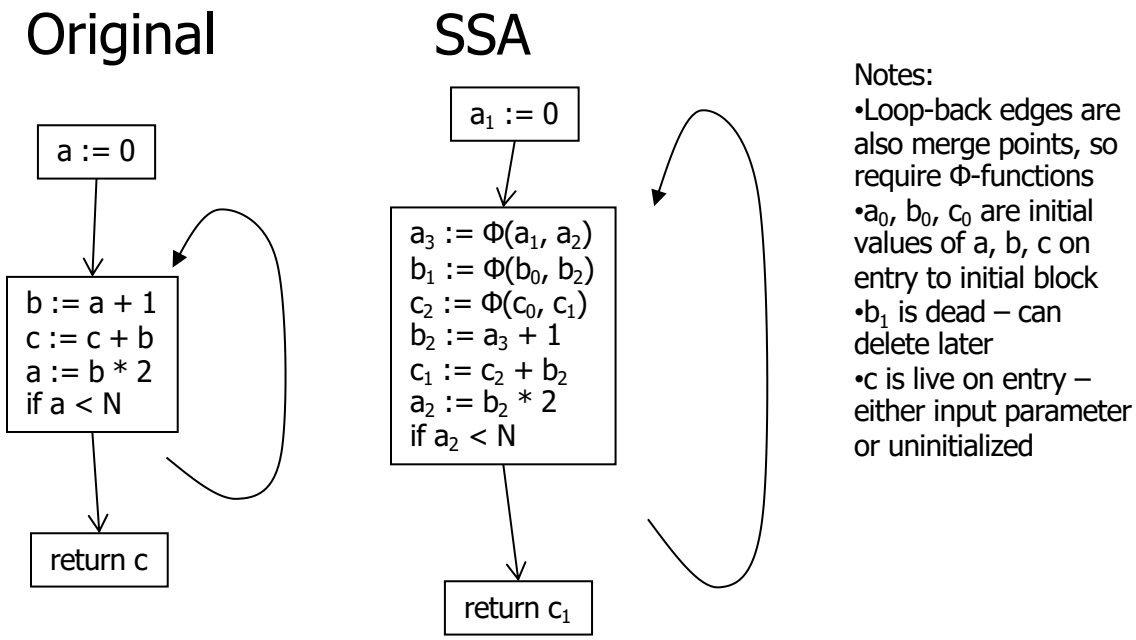


Where to place Phi-Functions?

- Wherever a variable has multiple possible definitions entering a block
  - Inefficient (and unnecessary!) to consider all possible phi-functions at the start of each block



Example With a Loop

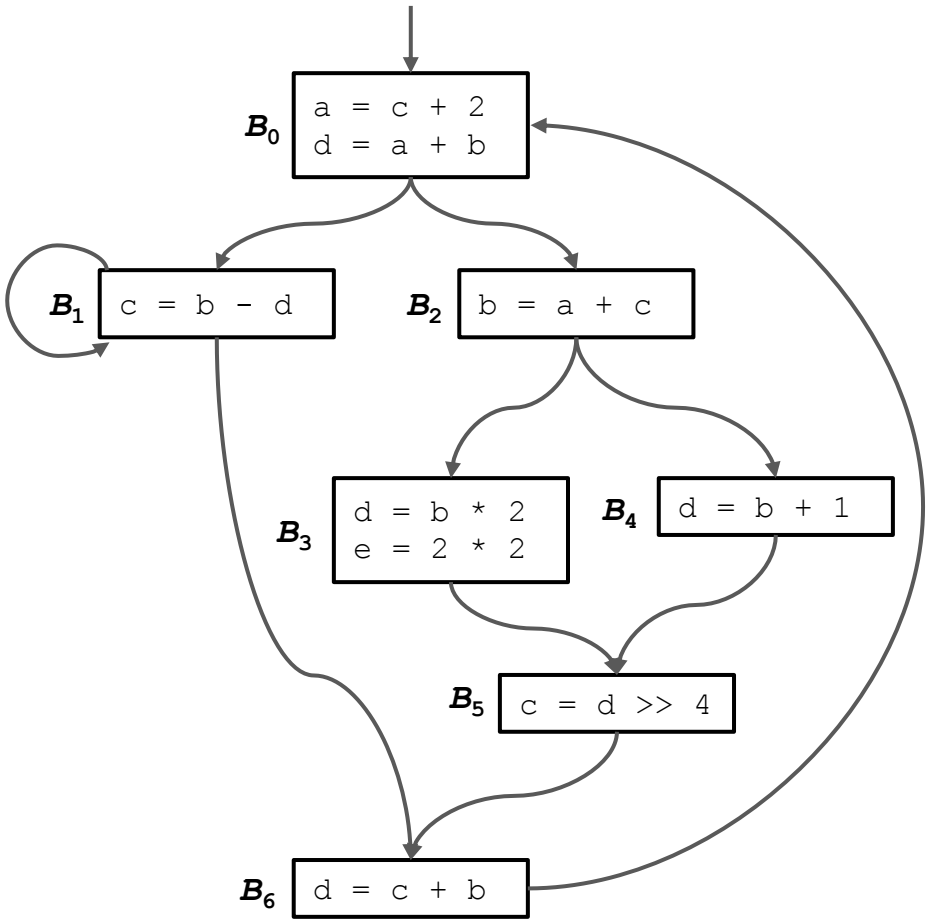


Notes:

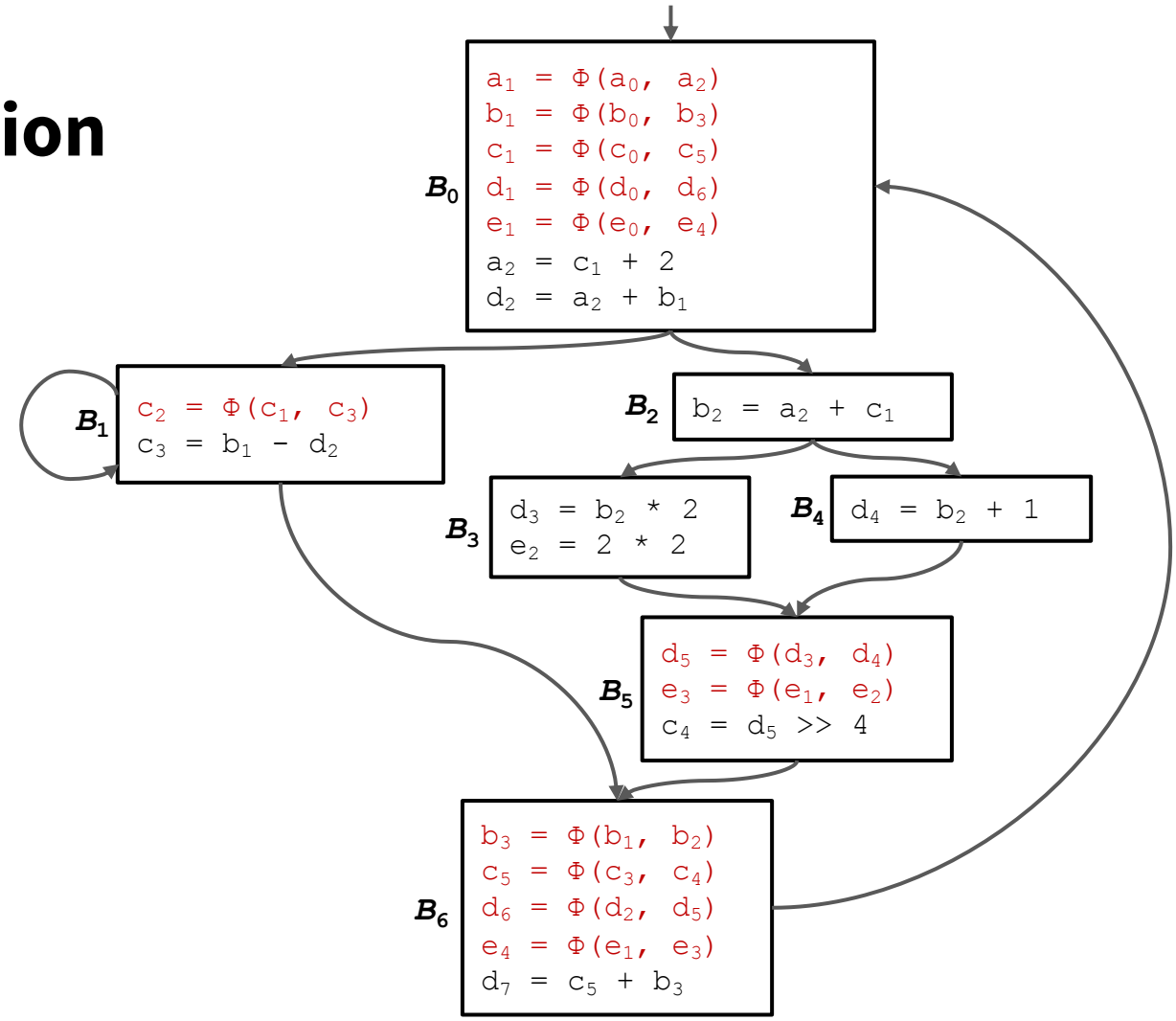
- Loop-back edges are also merge points, so require Φ-functions
- a_0, b_0, c_0 are initial values of a, b, c on entry to initial block
- b_1 is dead – can delete later
- c is live on entry – either input parameter or uninitialized

Problem 2_a

Note: there is a more formal way to decide which phi functions are necessary to include


B0
a = c + 2
d = a + b
B1
c = b - d
B2
b = a + c
B3
d = b * 2
e = 2 * 2
B4
d = b + 1
B5
c = d >> 4
B6
d = c + b

Solution


B0
a1 = Φ(a0, a2)
b1 = Φ(b0, b3)
c1 = Φ(c0, c5)
d1 = Φ(d0, d6)
e1 = Φ(e0, e4)
a2 = c1 + 2
d2 = a2 + b1
B1
c2 = Φ(c1, c3)
c3 = b1 - d2
B2
b2 = a2 + c1
B3
d3 = b2 * 2
e2 = 2 * 2
B4
d4 = b2 + 1
B5
d5 = Φ(d3, d4)
e3 = Φ(e1, e2)
c4 = d5 >> 4
B6
b3 = Φ(b1, b2)
c5 = Φ(c3, c4)
d6 = Φ(d2, d5)
e4 = Φ(e1, e3)
d7 = c5 + b3