Compiler Techniques

Lecture 10: Dataflow Analysis (Backward)

Tianwei Zhang

Outline

- **Overview of Optimization**
- **Dataflow Analysis**
- **Liveness Analysis**
- **Very Busy Expression Analysis**

Outline

- **Overview of Optimization**
- **Dataflow Analysis**
- **Liveness Analysis**
- **Very Busy Expression Analysis**

Overview of Optimization

- Two kinds of optimizations:
  - Platform-independent optimizations
  - Platform-specific optimizations
- In this course:
  - We will only consider some simple platform-independent optimizations
  - Such optimizations are usually implemented on intermediate representations like Jimple

Static Analysis

- Optimizations need to be ***behavior-preserving***
  - The optimized program should run faster/take less space
  - It should behave exactly the same as the original program

- **Static analysis**: an optimizer has to reason statically (*i.e.*, without running the program) about *all* possible behaviors of a program

- From some basic results of computability theory: it is impossible to statically predict precisely the possible behaviors of a program

- An optimizer has to be **conservative**: only apply an optimization if we can be absolutely certain that it is behavior-preserving

Examples of Optimizations

- Examples of optimizations performed by modern compilers:
  - Common subexpression elimination (lecture 11): if an expression has already been evaluated, reuse its previously computed value
  - Register allocation (lecture 12): allocate registers to accelerate the memory access time.
  - Loop invariant code motion (lecture 13): move computation out of a loop body to avoid re-computation
  - Loop fusion (lecture 13): combine two loops into one
  - Reduction in strength (lecture 13): replace slow operations with faster ones to accelerate computation
  - Function inlining (lecture 13): replace a call to a function by its function body to avoid overhead
  - Devirtualisation (lecture 13): turn a virtual method call into a static one
  - Dead code detection: remove code that cannot be executed
  - Bounds check elimination: in Java, if an array index can be shown to be in range, we do not need to check it at run-time
  - …

Intra-Procedural vs. Inter-Procedural

- Intra-procedural optimizaton
  - Only concern the code within a single method or function
  - A *control flow graph* is used to represent potential execution paths within a method/function

- Inter-procedural/whole-program optimizaton
  - Try to optimize several methods or functions at once;
  - A *procedure call graph* is used to represent potential execution paths between the methods/functions of a program
  - Optimizations have the potential to deliver greater performance improvements, but they are more difficult to apply, and it is harder to reason about their safety

Outline of the Following Lectures

- Intra-Procedural Analysis
  - Lecture 10: Data-flow analysis (backward)
    - Liveness analysis & very busy expression analysis
  - Lecture 11: Data-flow analysis (forward)
    - Available expression analysis & reaching definitions analysis
  - Lecture 12: Register allocation
  - Lecture 13: control flow analysis
    - Dominance analysis & loop optimization & static single assignment

- Inter-Procedural Analysis
  - Lecture 13: analysis and optimization based on call graphs

Outline

- Overview of Optimization
- **Dataflow Analysis**
- Liveness Analysis
- Very Busy Expression Analysis

Control Flow Graphs (CFG)

- A representation of all instructions in a method and the possible flow of execution through these instructions
  - Used by many optimizations to reason about the possible behaviors of a method
- Nodes of the CFG: the instructions of the method
  - ENTRY and EXIT nodes: representing the beginning and end of the method execution
- An edge from node n_1 to n_2: if n_2 could be executed immediately after n_1
  - In Jimple, most instructions only have a single successor, except for conditional jumps, which have two (we ignore exceptions)

Control Flow Graph Example

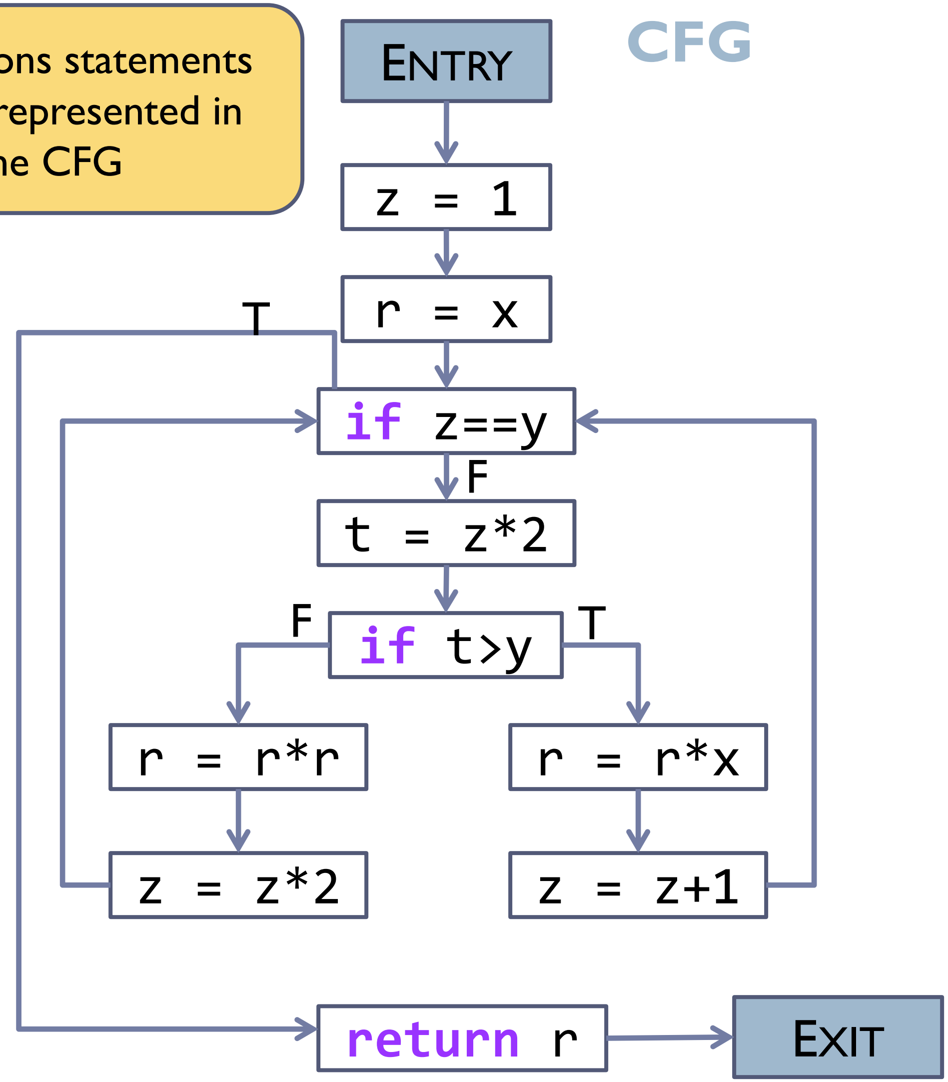
Jimple code

```
 int r, t, x, y, z;
 x:= @parameter0;
 y:= @parameter1;
 z = 1;
 r = x;
l1: if z==y goto l3;
 t = z*2;
 if t>y goto l2;
 r = r*r;
 z = z*2;
 goto l1;
l2:
 r = r*x;
 z = z+1;
 goto l1;
l3: return r;
```

declarations statements are not represented in the CFG

goto is not explicitly represented in the CFG

CFG



Data Flow Analysis

- Gather information about the values computed (and assigned to variables) at different points in the program
- Data flow analysis on the CFG
  - Considering all possible paths from the ENTRY node to a certain node, or from that node to the EXIT node
- Four possible applications:
    - Liveness analysis (backward-may)
    - Very busy expression analysis (backward-must)
    - Available expression analysis (forward-must)
    - Reaching definitions analysis (forward-may)

Outline

- Overview of Optimization
- Dataflow Analysis
- **Liveness Analysis**
- Very Busy Expression Analysis

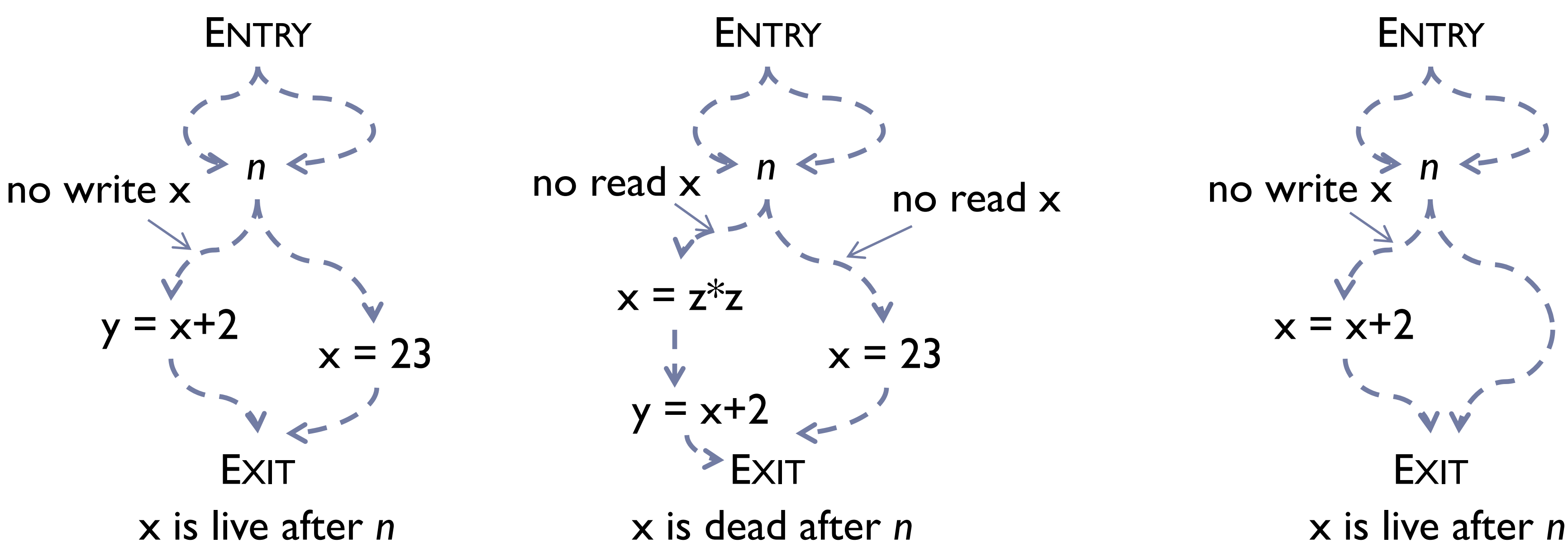
Liveness

- Liveness analysis can be used for register allocation (Lecture 12)
- A local variable is *live* at a program point if its value may be <u>read</u> before it is <u>reassigned</u>; otherwise, it is *dead*
- *Live range* of a variable: the set of statements within which the variable is live

```
    x = 0                   // x
    y = 1                   // x, y
l0:z = 2                    // x, y, z
    a = x                   // a, x, y, z
l1:if z == y goto l3        // a, x, y, z
    b = z * 2               // a, b, x, y, z
    if b > y goto l2        // a, x, y, z
    a = a * a               // a, x, y, z
    z = z * 2               // a, x, y, z
    goto l1
l2:a = y * x                // a, x, y, z
    z = z + 1               // a, x, y, z
    goto l1
l3:
```

Liveness analysis based on CFG

- A variable *x* is live *after* a CFG node *n* if:
  - There exists a path *p* from *n* to EXIT.
  - There exists a node *r* on *p* that reads *x*
  - *r* is not preceded on *p* by any node that writes *x*



Liveness analysis

- A variable can be live *before* or *after* a node *n*.

- Flow sets:
  - $in_L(n)$: the set of variables that are live before *n*
  - $out_L(n)$: the set of variables that are live after *n*

- Goal of liveness analysis: compute $in_L(n)$ and $out_L(n)$ for every CFG node *n*

Transfer Functions

- If we already know $\text{out}_L(n)$, it is easy to compute $\text{in}_L(n)$:
  - A variable *x* is live before *n* if

    (1) either *n* reads x,

    (2) or x is live after *n* and *n* does not write *x*
- More denotations:
  - $\text{use}(n)$: the set of (local) variables read by a node *n*
  - $\text{def}(n)$: the set of variables written by a node *n*
- **Transfer function** for $\text{in}_L(n)$:

$$\text{in}_L(n) = \text{out}_L(n) \setminus \text{def}(n) \cup \text{use}(n)$$

- This is called **backward flow analysis** since we compute $\text{in}(n)$ from $\text{out}(n)$. We also have **forward flow analysis** discussed in the next Lecture, which computes $\text{out}(n)$ from $\text{in}(n)$.

Transfer Functions

- How do we get $out_L(n)$? There are two cases:
  - Node n is the Exit node. Then no variable is live at the end of a method:

$$out_L(Exit) = \varnothing$$

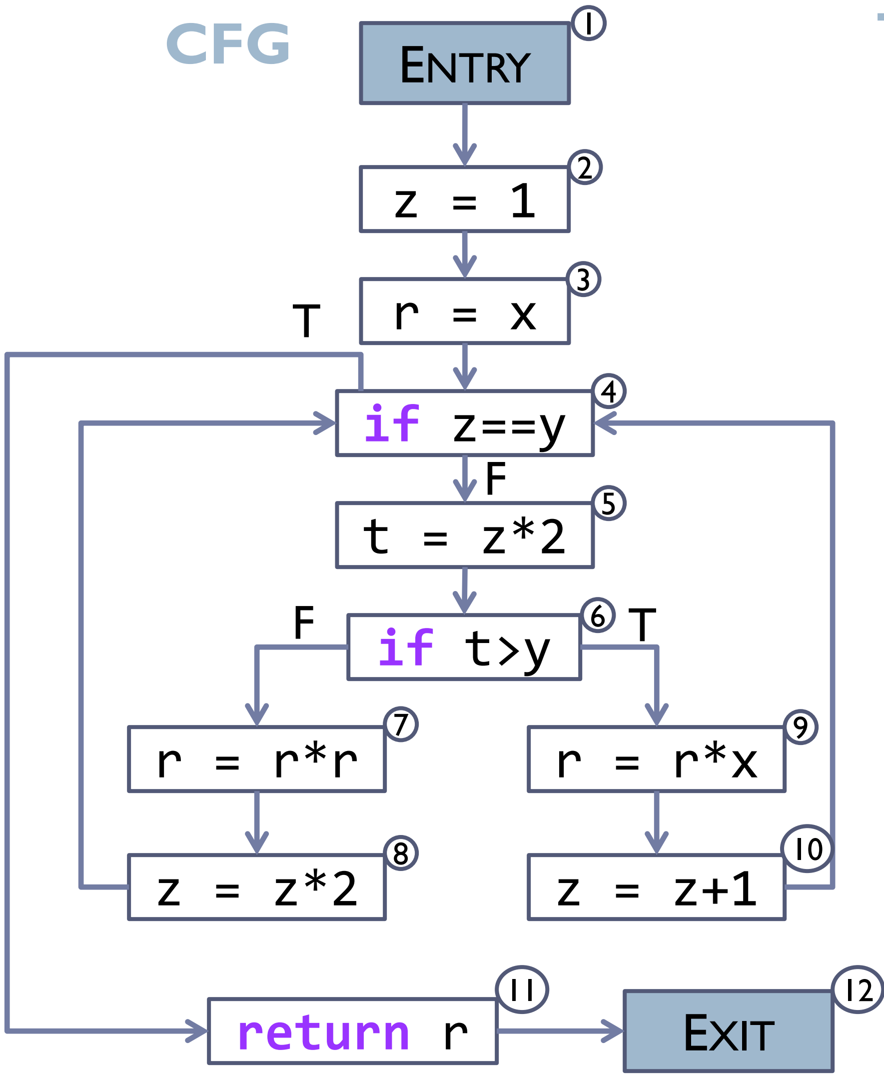
  - Node *n* has at least one successor node.
    - $succ(n)$: the set of successor nodes of *n*.
    - <u>A variable x is live after *n* if it is live before any successor of n</u>

$$out_L(n) = \cup \{ in_L(m) \mid m \in succ(n) \}$$

- This is called **May analysis,** as union is used to combine results from successor nodes. We also have **Must analysis** where intersection is used

Transfer Functions Example

CFG



Transfer Functions

$in_L(n)$ = $out_L(n)$ \ $def(n) \cup use(n)$

$in_L(1) = out_L(1) \setminus \varnothing \cup \varnothing = out_L(1)$

$in_L(2) = out_L(2) \setminus \{z\} \cup \varnothing = out_L(2) \setminus \{z\}$

$in_L(3) = out_L(3) \setminus \{r\} \cup \{x\}$

$in_L(4) = out_L(4) \setminus \varnothing \cup \{y, z\} = out_L(4) \cup \{y, z\}$

$in_L(5) = out_L(5) \setminus \{t\} \cup \{z\}$

$in_L(6) = out_L(6) \setminus \varnothing \cup \{t, y\} = out_L(6) \cup \{t, y\}$

$in_L(7) = out_L(7) \setminus \{r\} \cup \{r\} = out_L(7) \cup \{r\}$

$in_L(8) = out_L(8) \setminus \{z\} \cup \{z\} = out_L(8) \cup \{z\}$

$in_L(9) = out_L(9) \setminus \{r\} \cup \{r, x\} = out_L(9) \cup \{r, x\}$

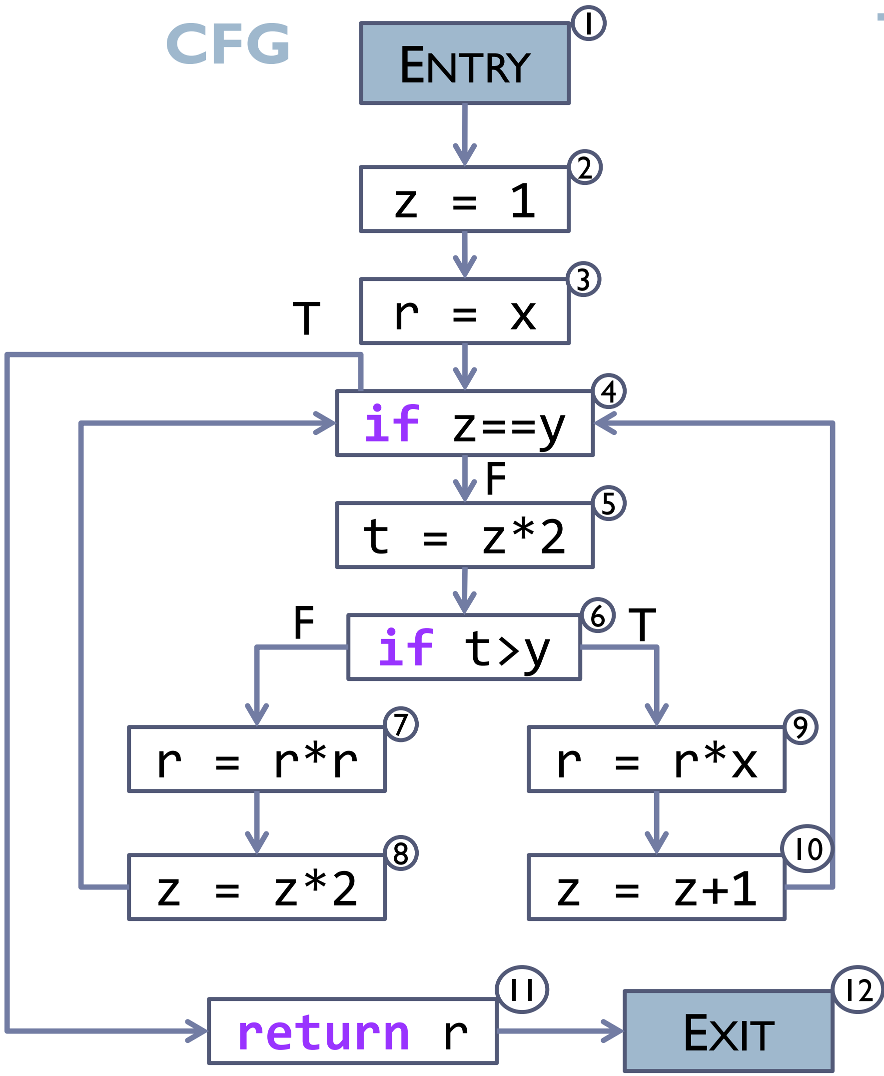
$in_L(10) = out_L(10) \setminus \{z\} \cup \{z\} = out_L(10) \cup \{z\}$

$in_L(11) = out_L(11) \setminus \varnothing \cup \{r\} = out_L(11) \cup \{r\}$

$in_L(12) = out_L(12) \setminus \varnothing \cup \varnothing = out_L(12)$

Transfer Functions Example

CFG



Transfer Functions

$\text{out}_L(n) = \bigcup \{ \text{in}_L(m) \mid m \in \text{succ}(n) \}$

$\text{out}_L(1) = \text{in}_L(2)$

$\text{out}_L(2) = \text{in}_L(3)$

$\text{out}_L(3) = \text{in}_L(4)$

$\text{out}_L(4) = \text{in}_L(11) \cup \text{in}_L(5)$

$\text{out}_L(5) = \text{in}_L(6)$

$\text{out}_L(6) = \text{in}_L(7) \cup \text{in}_L(9)$

$\text{out}_L(7) = \text{in}_L(8)$

$\text{out}_L(8) = \text{in}_L(4)$

$\text{out}_L(9) = \text{in}_L(10)$

$\text{out}_L(10) = \text{in}_L(4)$

$\text{out}_L(11) = \text{in}_L(12)$

$\text{out}_L(12) = \varnothing$

Solving Transfer Functions

- The system of equations for the transfer functions cannot be solved directly: the definitions are circular!

$$
\begin{aligned}
\mathbf{in_L(4)} &= out_L(4) \cup \{ y, z \} \\
&= in_L(11) \cup in_L(5) \cup \{ y, z \} \\
&= in_L(11) \cup out_L(5) \setminus \{ t \} \cup \{ y, z \} \\
&= in_L(11) \cup in_L(6) \setminus \{ t \} \cup \{ y, z \} \\
&= in_L(11) \cup (out_L(6) \cup \{ t, y \}) \setminus \{ t \} \cup \{ y, z \} \\
&= in_L(11) \cup (in_L(7) \cup in_L(9) \cup \{ t, y \}) \setminus \{ t \} \cup \{ y, z \} \\
&= in_L(11) \cup (out_L(7) \cup in_L(9) \cup \{ r, t, y \}) \setminus \{ t \} \cup \{ y, z \} \\
&= in_L(11) \cup (in_L(8) \cup in_L(9) \cup \{ r, t, y \}) \setminus \{ t \} \cup \{ y, z \} \\
&= in_L(11) \cup (out_L(8) \cup in_L(9) \cup \{ r, t, y, z \}) \setminus \{ t \} \cup \{ y, z \} \\
&= in_L(11) \cup (\mathbf{in_L(4)} \cup in_L(9) \cup \{ r, t, y, z \}) \setminus \{ t \} \cup \{ y, z \} \\
&= \ldots
\end{aligned}
$$

Method 1: Iterative Solution

- However, the equation system can be solved by iteration:
  1. Initialization: set $in_L(n) = out_L(n) = \emptyset$ for all nodes n
  2. For each iteration: recompute $out_L(n)$ for every node n based on the values we have computed in the previous iteration and then recompute $in_L(n)$ from $out_L(n)$
  3. Finished: when all values do not change any further

- This method can be used for other data flow analyses as well

Simplifying Transfer Equations

- To make it easier to solve the equations, we substitute away the $out_L(n)$ equations, so we only have to solve for $in_L(n)$

$$
\begin{aligned}
in_L(1) &= out_L(1)\\
in_L(2) &= out_L(2) \setminus \{z\}\\
in_L(3) &= out_L(3)\setminus\{r\} \cup \{x\}\\
in_L(4) &= out_L(4) \cup \{y, z\}\\
in_L(5) &= out_L(5)\setminus\{t\} \cup \{z\}\\
in_L(6) &= out_L(6) \cup \{t, y\}\\
in_L(7) &= out_L(7) \cup \{r\}\\
in_L(8) &= out_L(8) \cup \{z\}\\
in_L(9) &= out_L(9) \cup \{r, x\}\\
in_L(10) &= out_L(10) \cup \{z\}\\
in_L(11) &= out_L(11) \cup \{r\}\\
in_L(12) &= out_L(12)
\end{aligned}
$$

$$
\begin{aligned}
out_L(1) &= in_L(2)\\
out_L(2) &= in_L(3)\\
out_L(3) &= in_L(4)\\
out_L(4) &= in_L(11) \cup in_L(5)\\
out_L(5) &= in_L(6)\\
out_L(6) &= in_L(7) \cup in_L(9)\\
out_L(7) &= in_L(8)\\
out_L(8) &= in_L(4)\\
out_L(9) &= in_L(10)\\
out_L(10) &= in_L(4)\\
out_L(11) &= in_L(12)\\
out_L(12) &= \varnothing
\end{aligned}
$$

$\Rightarrow$

$$
\begin{aligned}
in_L(1) &= in_L(2)\\
in_L(2) &= in_L(3) \setminus \{z\}\\
in_L(3) &= in_L(4) \setminus \{r\} \cup \{x\}\\
in_L(4) &= in_L(11) \cup in_L(5) \cup \{y, z\}\\
in_L(5) &= in_L(6) \setminus \{t\} \cup \{z\}\\
in_L(6) &= in_L(7) \cup in_L(9) \cup \{t,y\}\\
in_L(7) &= in_L(8) \cup \{r\}\\
in_L(8) &= in_L(4) \cup \{z\}\\
in_L(9) &= in_L(10) \cup \{r, x\}\\
in_L(10) &= in_L(4) \cup \{z\}\\
in_L(11) &= in_L(12) \cup \{r\}\\
in_L(12) &= \varnothing
\end{aligned}
$$

Iterative Solution Example

$in_L(1) = in_L(2)$

$in_L(2) = in_L(3) \setminus \{z\}$

$in_L(3) = in_L(4) \setminus \{r\} \cup \{x\}$

$in_L(4) = in_L(11) \cup in_L(5) \cup \{y, z\}$

$in_L(5) = in_L(6) \setminus \{t\} \cup \{z\}$

$in_L(6) = in_L(7) \cup in_L(9) \cup \{t, y\}$

$in_L(7) = in_L(8) \cup \{r\}$

$in_L(8) = in_L(4) \cup \{z\}$

$in_L(9) = in_L(10) \cup \{r, x\}$

$in_L(10) = in_L(4) \cup \{z\}$

$in_L(11) = in_L(12) \cup \{r\}$

$in_L(12) = \varnothing$

	0	1	2	3	4	5	6
$in_L(1)$	$\varnothing$						
$in_L(2)$	$\varnothing$						
$in_L(3)$	$\varnothing$						
$in_L(4)$	$\varnothing$						
$in_L(5)$	$\varnothing$						
$in_L(6)$	$\varnothing$						
$in_L(7)$	$\varnothing$						
$in_L(8)$	$\varnothing$						
$in_L(9)$	$\varnothing$						
$in_L(10)$	$\varnothing$						
$in_L(11)$	$\varnothing$						
$in_L(12)$	$\varnothing$						

Iterative Solution Example

$in_L(1) = in_L(2)$

$in_L(2) = in_L(3) \setminus \{z\}$

$in_L(3) = in_L(4) \setminus \{r\} \cup \{x\}$

$in_L(4) = in_L(11) \cup in_L(5) \cup \{y, z\}$

$in_L(5) = in_L(6) \setminus \{t\} \cup \{z\}$

$in_L(6) = in_L(7) \cup in_L(9) \cup \{t, y\}$

$in_L(7) = in_L(8) \cup \{r\}$

$in_L(8) = in_L(4) \cup \{z\}$

$in_L(9) = in_L(10) \cup \{r, x\}$

$in_L(10) = in_L(4) \cup \{z\}$

$in_L(11) = in_L(12) \cup \{r\}$

$in_L(12) = \varnothing$

	0	1	2	3	4	5	6
$in_L(1)$	$\varnothing$	$\varnothing$					
$in_L(2)$	$\varnothing$	$\varnothing$					
$in_L(3)$	$\varnothing$	x					
$in_L(4)$	$\varnothing$	y,z					
$in_L(5)$	$\varnothing$	z					
$in_L(6)$	$\varnothing$	t,y					
$in_L(7)$	$\varnothing$	r					
$in_L(8)$	$\varnothing$	z					
$in_L(9)$	$\varnothing$	r,x					
$in_L(10)$	$\varnothing$	z					
$in_L(11)$	$\varnothing$	r					
$in_L(12)$	$\varnothing$	$\varnothing$					

Iterative Solution Example

$in_L(1) = in_L(2)$

$in_L(2) = in_L(3) \setminus \{z\}$

$in_L(3) = in_L(4) \setminus \{r\} \cup \{x\}$

$in_L(4) = in_L(11) \cup in_L(5) \cup \{y, z\}$

$in_L(5) = in_L(6) \setminus \{t\} \cup \{z\}$

$in_L(6) = in_L(7) \cup in_L(9) \cup \{t,y\}$

$in_L(7) = in_L(8) \cup \{r\}$

$in_L(8) = in_L(4) \cup \{z\}$

$in_L(9) = in_L(10) \cup \{r, x\}$

$in_L(10) = in_L(4) \cup \{z\}$

$in_L(11) = in_L(12) \cup \{r\}$

$in_L(12) = \emptyset$

	0	1	2	3	4	5	6
$in_L(1)$	$\emptyset$	$\emptyset$	$\emptyset$				
$in_L(2)$	$\emptyset$	$\emptyset$	x				
$in_L(3)$	$\emptyset$	x	x,y,z				
$in_L(4)$	$\emptyset$	y,z	r,y,z				
$in_L(5)$	$\emptyset$	z	y,z				
$in_L(6)$	$\emptyset$	t,y	r,t,x,y				
$in_L(7)$	$\emptyset$	r	r,z				
$in_L(8)$	$\emptyset$	z	y,z				
$in_L(9)$	$\emptyset$	r,x	r,x,z				
$in_L(10)$	$\emptyset$	z	y,z				
$in_L(11)$	$\emptyset$	r	r				
$in_L(12)$	$\emptyset$	$\emptyset$	$\emptyset$				

Iterative Solution Example

$in_L(1) = in_L(2)$

$in_L(2) = in_L(3) \setminus \{z\}$

$in_L(3) = in_L(4) \setminus \{r\} \cup \{x\}$

$in_L(4) = in_L(11) \cup in_L(5) \cup \{y, z\}$

$in_L(5) = in_L(6) \setminus \{t\} \cup \{z\}$

$in_L(6) = in_L(7) \cup in_L(9) \cup \{t,y\}$

$in_L(7) = in_L(8) \cup \{r\}$

$in_L(8) = in_L(4) \cup \{z\}$

$in_L(9) = in_L(10) \cup \{r, x\}$

$in_L(10) = in_L(4) \cup \{z\}$

$in_L(11) = in_L(12) \cup \{r\}$

$in_L(12) = \varnothing$

	0	1	2	3	4	5	6
$in_L(1)$	$\varnothing$	$\varnothing$	$\varnothing$	x			
$in_L(2)$	$\varnothing$	$\varnothing$	x	x,y			
$in_L(3)$	$\varnothing$	x	x,y,z	x,y,z			
$in_L(4)$	$\varnothing$	y,z	r,y,z	r,y,z			
$in_L(5)$	$\varnothing$	z	y,z	r,x,y,z			
$in_L(6)$	$\varnothing$	t,y	r,t,x,y	r,t,x,y,z			
$in_L(7)$	$\varnothing$	r	r,z	r,y,z			
$in_L(8)$	$\varnothing$	z	y,z	r,y,z			
$in_L(9)$	$\varnothing$	r,x	r,x,z	r,x,y,z			
$in_L(10)$	$\varnothing$	z	y,z	r,y,z			
$in_L(11)$	$\varnothing$	r	r	r			
$in_L(12)$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$			

Iterative Solution Example

$in_L(1) = in_L(2)$

$in_L(2) = in_L(3) \setminus \{z\}$

$in_L(3) = in_L(4) \setminus \{r\} \cup \{x\}$

$in_L(4) = in_L(11) \cup in_L(5) \cup \{y, z\}$

$in_L(5) = in_L(6) \setminus \{t\} \cup \{z\}$

$in_L(6) = in_L(7) \cup in_L(9) \cup \{t, y\}$

$in_L(7) = in_L(8) \cup \{r\}$

$in_L(8) = in_L(4) \cup \{z\}$

$in_L(9) = in_L(10) \cup \{r, x\}$

$in_L(10) = in_L(4) \cup \{z\}$

$in_L(11) = in_L(12) \cup \{r\}$

$in_L(12) = \emptyset$

	0	1	2	3	4	5	6
$in_L(1)$	$\emptyset$	$\emptyset$	$\emptyset$	x	x,y		
$in_L(2)$	$\emptyset$	$\emptyset$	x	x,y	x,y		
$in_L(3)$	$\emptyset$	x	x,y,z	x,y,z	x,y,z		
$in_L(4)$	$\emptyset$	y,z	r,y,z	r,y,z	r,x,y,z		
$in_L(5)$	$\emptyset$	z	y,z	r,x,y,z	r,x,y,z		
$in_L(6)$	$\emptyset$	t,y	r,t,x,y	r,t,x,y,z	r,t,x,y,z		
$in_L(7)$	$\emptyset$	r	r,z	r,y,z	r,y,z		
$in_L(8)$	$\emptyset$	z	y,z	r,y,z	r,y,z		
$in_L(9)$	$\emptyset$	r,x	r,x,z	r,x,y,z	r,x,y,z		
$in_L(10)$	$\emptyset$	z	y,z	r,y,z	r,y,z		
$in_L(11)$	$\emptyset$	r	r	r	r		
$in_L(12)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$		

Iterative Solution Example

$in_L(1) = in_L(2)$

$in_L(2) = in_L(3) \setminus \{z\}$

$in_L(3) = in_L(4) \setminus \{r\} \cup \{x\}$

$in_L(4) = in_L(11) \cup in_L(5) \cup \{y, z\}$

$in_L(5) = in_L(6) \setminus \{t\} \cup \{z\}$

$in_L(6) = in_L(7) \cup in_L(9) \cup \{t, y\}$

$in_L(7) = in_L(8) \cup \{r\}$

$in_L(8) = in_L(4) \cup \{z\}$

$in_L(9) = in_L(10) \cup \{r, x\}$

$in_L(10) = in_L(4) \cup \{z\}$

$in_L(11) = in_L(12) \cup \{r\}$

$in_L(12) = \emptyset$

	0	1	2	3	4	5	6
$in_L(1)$	∅	∅	∅	x	x,y	x,y	
$in_L(2)$	∅	∅	x	x,y	x,y	x,y	
$in_L(3)$	∅	x	x,y,z	x,y,z	x,y,z	x,y,z	
$in_L(4)$	∅	y,z	r,y,z	r,y,z	r,x,y,z	r,x,y,z	
$in_L(5)$	∅	z	y,z	r,x,y,z	r,x,y,z	r,x,y,z	
$in_L(6)$	∅	t,y	r,t,x,y	r,t,x,y,z	r,t,x,y,z	r,t,x,y,z	
$in_L(7)$	∅	r	r,z	r,y,z	r,y,z	r,y,z	
$in_L(8)$	∅	z	y,z	r,y,z	r,y,z	r,x,y,z	
$in_L(9)$	∅	r,x	r,x,z	r,x,y,z	r,x,y,z	r,x,y,z	
$in_L(10)$	∅	z	y,z	r,y,z	r,y,z	r,x,y,z	
$in_L(11)$	∅	r	r	r	r	r	
$in_L(12)$	∅	∅	∅	∅	∅	∅	

Iterative Solution Example

No further changes after iteration 6

$in_L(1) = in_L(2)$

$in_L(2) = in_L(3) \setminus \{z\}$

$in_L(3) = in_L(4) \setminus \{r\} \cup \{x\}$

$in_L(4) = in_L(11) \cup in_L(5) \cup \{y, z\}$

$in_L(5) = in_L(6) \setminus \{t\} \cup \{z\}$

$in_L(6) = in_L(7) \cup in_L(9) \cup \{t, y\}$

$in_L(7) = in_L(8) \cup \{r\}$

$in_L(8) = in_L(4) \cup \{z\}$

$in_L(9) = in_L(10) \cup \{r, x\}$

$in_L(10) = in_L(4) \cup \{z\}$

$in_L(11) = in_L(12) \cup \{r\}$

$in_L(12) = \varnothing$

	0	1	2	3	4	5	6
$in_L(1)$	∅	∅	∅	x	x,y	x,y	x,y
$in_L(2)$	∅	∅	x	x,y	x,y	x,y	x,y
$in_L(3)$	∅	x	x,y,z	x,y,z	x,y,z	x,y,z	x,y,z
$in_L(4)$	∅	y,z	r,y,z	r,y,z	r,x,y,z	r,x,y,z	r,x,y,z
$in_L(5)$	∅	z	y,z	r,x,y,z	r,x,y,z	r,x,y,z	r,x,y,z
$in_L(6)$	∅	t,y	r,t,x,y	r,t,x,y,z	r,t,x,y,z	r,t,x,y,z	r,t,x,y,z
$in_L(7)$	∅	r	r,z	r,y,z	r,y,z	r,y,z	r,x,y,z
$in_L(8)$	∅	z	y,z	r,y,z	r,y,z	r,x,y,z	r,x,y,z
$in_L(9)$	∅	r,x	r,x,z	r,x,y,z	r,x,y,z	r,x,y,z	r,x,y,z
$in_L(10)$	∅	z	y,z	r,y,z	r,y,z	r,x,y,z	r,x,y,z
$in_L(11)$	∅	r	r	r	r	r	r
$in_L(12)$	∅	∅	∅	∅	∅	∅	∅

Termination of Iterative Solution

- Iterative solution always terminates
  - Each step of the iteration can only grow a set or leave unchanged
  - Finite number of elements in each set, so finite number of times can change
  - Each iteration either has a change or stops
  - Must terminate

- Finite descending chain property

Method 2: Worklist Algorithm (Backward)

- $in_L(n)$ can only change if $out_L(n)$ changes, and $out_L(n)$ changes only if $in_L(m)$ changes for some successor *m* of *n*
- We can avoid unnecessary recomputation by keeping a *worklist* of nodes for which $out_L(n)$ has changed:

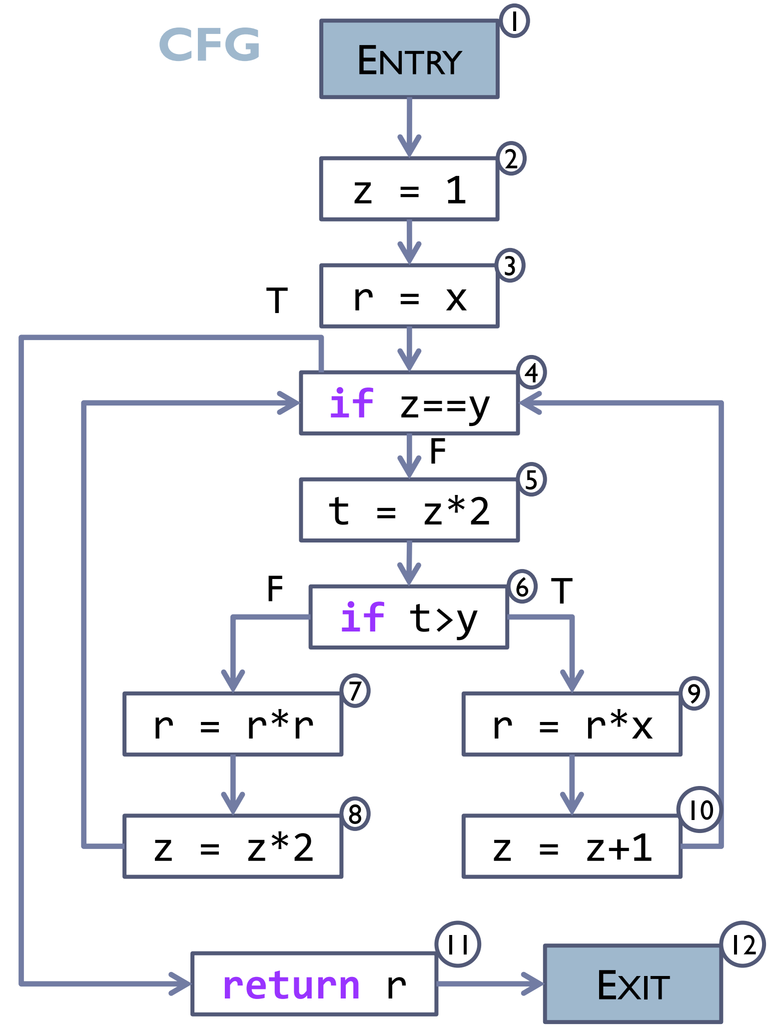
```
worklist = [ all nodes ]
  while worklist != empty do
    m = removeFirst(worklist)
    recompute in_L(m)
    if in_L(m) has changed then
      for each predecessor n of m
        compute out_L(n)
        if out_L(n) has changed then
          put n into worklist (if not already in worklist)
```

Worklist Algorithm Example (Backward)

- **Liveness analysis using the worklist algorithm.**
  - We initialize the worklist to contain all nodes in the CFG to ensure that each node is evaluated at least once
  - These nodes can be in an arbitrary order. For backward flow analysis, we sort these nodes as a reversed order in the worklist, to reduce number of iterations. For forward flow analysis, it is recommended to sort the nodes in the normal order.
  - For all nodes, $out_L(n)$ and $in_L(n)$ are initialized to $\emptyset$

Example

Worklist	**$in_L(m)$ &**	**$out_L(n)$**
12, …, 1	$in_L(12) = \varnothing$	$out_L(11) = \varnothing$
11, …, 1	$in_L(11) = \{r\}$	$out_L(4) = \{r\}$
10, …, 1	$in_L(10) = \{z\}$	$out_L(9) = \{z\}$
9, …, 1	$in_L(9) = \{r, x, z\}$	$out_L(6) = \{r, x, z\}$
8, …, 1	$in_L(8) = \{z\}$	$out_L(7) = \{z\}$
7, …, 1	$in_L(7) = \{r, z\}$	$out_L(6) = \{r, x, z\}$
6, …, 1	$in_L(6) = \{r, t, x, y, z\}$	$out_L(5) = \{r, t, x, y, z\}$
5, …, 1	$in_L(5) = \{r, x, y, z\}$	$out_L(4) = \{r, x, y, z\}$
4, …, 1	$in_L(4) = \{r, x, y, z\}$	$out_L(3) = \{r, x, y, z\}$
		$out_L(8) = \{r, x, y, z\}$
		$out_L(10) = \{r, x, y, z\}$
3, …, 1, 8, 10	$in_L(3) = \{x, y, z\}$	$out_L(2) = \{x, y, z\}$
2, 1, 8, 10	$in_L(2) = \{x, y\}$	$out_L(1) = \{x, y\}$
1, 8, 10	$in_L(1) = \{x, y\}$	
8, 10	$in_L(8) = \{r, x, y, z\}$	$out_L(7) = \{r, x, y, z\}$
10, 7	$in_L(10) = \{r, x, y, z\}$	$out_L(9) = \{r, x, y, z\}$
7, 9	$in_L(7) = \{r, x, y, z\}$	$out_L(6) = \{r, x, y, z\}$
9, 6	$in_L(9) = \{r, x, y, z\}$	$out_L(6) = \{r, x, y, z\}$
6	$in_L(6) = \{r, t, x, y, z\}$	$out_L(5) = \{r, t, x, y, z\}$



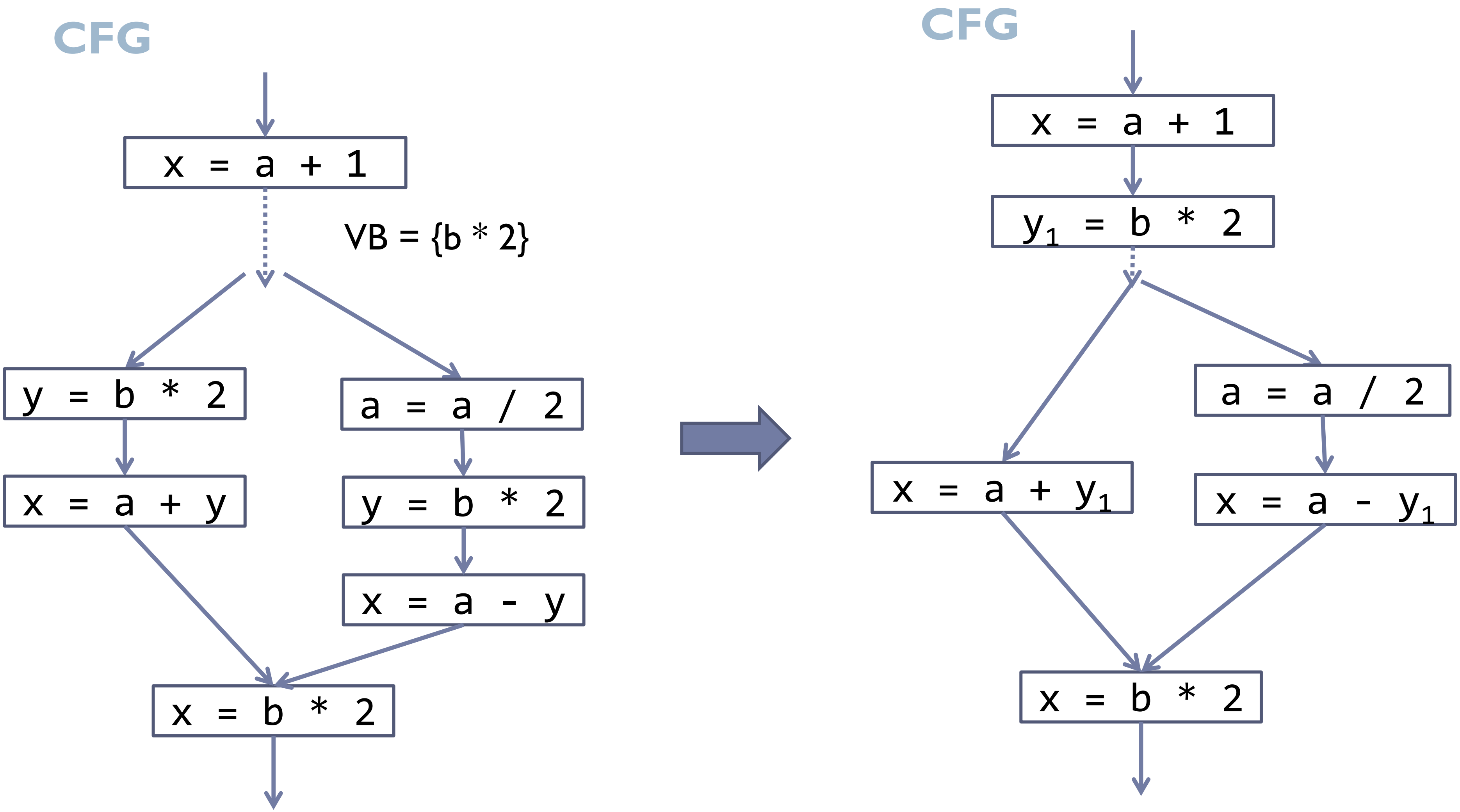
Outline

- Overview of Optimization
- Dataflow Analysis
- Liveness Analysis
- **Very Busy Expression Analysis**

Very Busy Expression

- An expression e is very busy at a point *p* if
  - Along every path starting from *p* there is an expression e before redefinitions of any variables in it.
- Code hoisting:
  - If an expression e is very busy at program point *p*, we can hoist the computation of e to point *p* and store the result in a variable.
  - Subsequent computation of e can simply be replaced by the variable.
  - Potential benefit:
    - The size of the resulting program is smaller
    - The running time of the program is smaller because we avoid redoing work.

Very Busy Expressions Example



Very Busy Expressions Analysis

- Determine which expressions are very busy at each program point

- An expression can be very busy *before* or *after* a node *n*.

- Flow sets:
  - $in_{VB}(n)$: the set of expressions that are very busy before *n*
  - $out_{VB}(n)$: the set of expressions that are very busy after *n*;

- Goal of very busy expression analysis: compute $in_{VB}(n)$ and $out_{VB}(n)$ for every CFG node *n*

Transfer Functions

- If we already know $out_{VB}(n)$, it is easy to compute $in_{VB}(n)$:
  - An expression e is very busy before *n* if

    (1) either *n* computes e,

    (2) or e is very busy after *n* and *n* does not write to any variable in e
- More denotations:
  - vars(e): the set of (local) variables in an expression e
  - comp(n): the set of expressions computed by n
  - def(*n*): the set of variables node *n* writes to
- **Transfer function** for $in_L(n)$:

$$in_{VB}(n) = out_{VB}(n) \setminus \{e \mid vars(e) \cap def(n) \neq \emptyset\} \cup comp(n)$$

- This is also backward flow analysis.

Transfer Functions

- How do we get $out_{VB}(n)$? There are two cases:
  - Node n is the Exit node. Then no expression is very busy at the end of a method:

$$out_{VB}(\text{Exit}) = \varnothing$$

  - Node n has at least one successor node.
    - $succ(n)$: the set of successor nodes of n.
    - <u>An expression e is very busy after n if it is very busy before all the successors of n</u>

$$out_{VB}(n) = \cap \{ \, in_{VB}(m) \mid m \in succ(n) \, \}$$

- This is **must analysis,** as intersection is used to combine results from successor nodes.

Solving Transfer Functions

- The transfer functions can be solved in the same way as liveness analysis, using iterative or worklist algorithms.

- Initialization:
  - $out_{VB}(\text{EXIT}) = in_{VB}(\text{EXIT}) = \emptyset$
  - $out_{VB}(n) = in_{VB}(n) = U$ for all nodes n except EXIT, since we use intersection to compute $out_{VB}(n)$, *where* U is the set of *all* expressions in the method