Dataflow Analysis

October 28 2022

Partial Order

Properties for meet operator

1. *idempotent*: x $\wedge$ x = x
2. *commutative*: x $\wedge$ y = y $\wedge$ x
3. *associative*: x $\wedge (y \wedge z) = (x \wedge y) \wedge z$

Partial order: $x \leq y$ *iff* $x \wedge y = x$

Properties for partial order

1. *reflexive*: $x \leq x$
2. *anti-symmetry*: $x \leq y$ *and* $y \leq x$ *then* $x = y$
3. *transitive*: $x \leq y$ *and* $y \leq z$ *then* $x \leq z$

Prove the transitivity property of the partial order.

Solution:

Given Facts:

$$x \vdash y \tag{0.1}$$

$$y \vdash z \tag{0.2}$$

WTS:

$$x \vdash z \tag{0.3}$$

By the definition of partial order:

$$x \vdash y = x \wedge y = x \tag{0.4}$$

Similarly

$$y \vdash z = y \wedge z = y \tag{0.5}$$

Then

$$x \wedge z = (x \wedge y) \wedge z \tag{0.6}$$

By associativity of meet order

$$(x \wedge y) \wedge z = x \wedge (y \wedge z) \tag{0.7}$$

By (0.5)

$$x \wedge (y \wedge z) = x \wedge y \tag{0.8}$$

By (0.4)

$$x \wedge y = x \tag{0.9}$$

Put all together

$$x \wedge z = x \tag{0.10}$$

QED

Monotonicity

1.What is the point of monotonicity?

Solution:

The monotonicity guarantees that the result after applying the transfer function will never increase. This approves that the algorithm can eventually converge.

2.Is the result optimal? (Note: This question is not covered in this week's material)

Solution:

If the framework is monotone, it yields to a fix-point solution.

A stronger version is the distributive framework, the distributive framework yields to a maximum fix-point solution.

$FP \leq MXP \leq MOP \leq \textit{Ideal Solution}$

3.Prove the equivalence of equation (0.11) and (0.12)

$$x \leq y \textit{ implies } f(x) \leq f(y) \tag{0.11}$$

$$f(x \wedge y) \leq f(x) \wedge f(y) \tag{0.12}$$

Note: The greatest lower bound for x, and y can only be $x \wedge y$

Solution:

From 0.11 to 0.12

Given facts:

$$x \vdash y \ \mathit{implies} \ f(x) \vdash f(y) \tag{0.13}$$

WTS:

$$f(x \wedge y) \vdash f(x) \wedge f(x) \tag{0.14}$$

According to the definition of partial order

$$x \vdash y \ \mathit{implies} \ x \wedge y = x \tag{0.15}$$

Since x ∧ y is the glb(x, y)

$$x \wedge y \vdash x \tag{0.16}$$

$$x \wedge y \vdash y \tag{0.17}$$

Based on 0.13:

$$f(x \wedge y) \vdash f(x) \tag{0.18}$$

$$f(x \wedge y) \vdash f(y) \tag{0.19}$$

Since f(x) ∧ f(y) is the glb(f(x), f(y))

$$f(x) \wedge f(y) \vdash f(x) \tag{0.20}$$

$$f(x) \wedge f(y) \vdash f(y) \tag{0.21}$$

Put all together

$$f(x \wedge y) \vdash f(x) \wedge f(y) \tag{0.22}$$

QED

Solution:

From 0.12 to 0.11

Given Facts:

$$f(x \wedge y) \vdash f(x) \wedge f(y) \tag{0.23}$$

$$x \vdash y \tag{0.24}$$

WTS:

$$f(x) \vdash f(y) \tag{0.25}$$

According to the definition of partial order

$$x \vdash y = x \wedge y = x \tag{0.26}$$

Substitute it to 0.12

$$f(x \wedge y) \vdash f(x) \wedge f(y) \tag{0.27}$$

$$f(x) \vdash f(x) \wedge f(y) \tag{0.28}$$

Since f(x) $\wedge$ f(y) is the glb(f(x), f(y))

$$f(x) \wedge f(y) \vdash f(y) \tag{0.29}$$

Put all together

$$f(x) \vdash f(x) \wedge f(y) \vdash f(y) \tag{0.30}$$

QED

Dataflow Analysis

A uniform framework for computing properties of basic blocks useful for optimization. A type of **static analysis**

Checklist

- Domain: Inputs for the analysis
- Direction:
  - Forward Flow: What can **happen** before a given point
  - Backward Flow: What can happen **after** the given point.
- May / Must:
  - May Analysis: What property holds on **some path**
    - Meet Operator: $\cup$
    - Initial value: *min*
  - Must Analysis: What property holds on **all paths**
    - Meet Operator: $\cap$
    - Initial value: *max*
- Transfer Function: How the value changes among the statement
  - $IN(b)$: What properties hold **on entry to** basic block b
  - $OUT(b)$: What properties hold **on exit from** basic block b
  - $Gen(b)$: The properties that are **generated in** basic block b
  - $Kill(b)$: The properties that are **invalidated in** basic block b

Reaching Definition

Problem Definition

- A definition d reaches a point p if there **is a path** from the point immediately following d to p, such that **d is not killed along that path**
- $Gen(B)$: Definitions made in B
- $Kill(B)$: Definitions invalidate by B

Summary

Domain	Definitions
Direction	Forward
Transfer Function	$Gen \cup (x - Kill)$
Meet Operator($\wedge$)	$\cup$
IN Equation	$IN[B] = \wedge_{P \in Predecessor} OUT[P]$
OUT Equation	$OUT[B] = f(IN[B])$
Initial Condition	$OUT[Entry] = \emptyset$
Boundary Condition	$OUT[All - ExceptEntry] = \emptyset$

Table 1: Summary for Reaching Definition

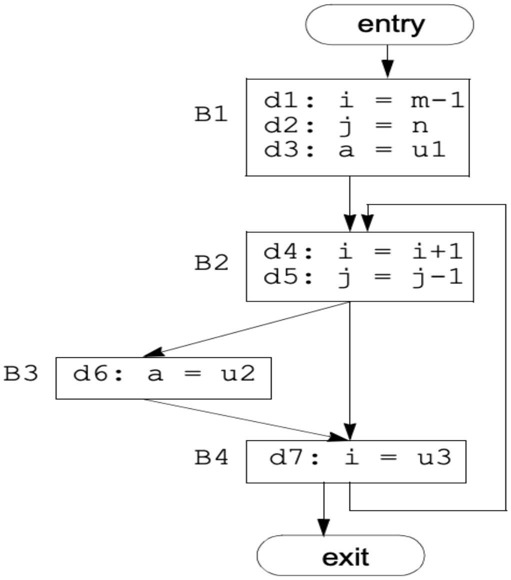
Usage

- Determine possible usage of uninitialized value

```
// dummy definition added by analyzer
x = dummy
z = dummy

// source code starts
x = 1
x = x + 1
y = z // z = dummy reaches here a possible use of undefined
x = x * 2
print(x)
```

Example


entry
B1
d1: i = m-1
d2: j = n
d3: a = u1
B2
d4: i = i+1
d5: j = j-1
B3
d6: a = u2
B4
d7: i = u3
exit

Solution:

There are 3 iterations performed. I did not write the iteration3 since it is the same as the iteration2.

Block	IN	Gen	Kill	OUT
Entry				$\emptyset$
B1				$\emptyset$
B2				$\emptyset$
B3				$\emptyset$
B4				$\emptyset$
Exit				$\emptyset$

Table 2: Iteration 0

Block	IN	Gen	Kill	OUT
B1	$OUT[Entry]$			
	$\emptyset$	$\{d1\}$	$\{d4, d7\}$	
		$\{d2\}$	$\{d5\}$	
		$\{d3\}$	$\{d6\}$	
		$\{d1, d2, d3\}$	$\{d4, d5, d6, d7\}$	$\{d1, d2, d3\}$
B2	$\wedge$ $OUT[B1]$, $OUT[B4]$			
	$\{d1, d2, d3\}$ $\cup\emptyset$			
		$\{d4\}$	$\{d1, d7\}$	
		$\{d5\}$	$\{d2\}$	
		$\{d4, d5\}$	$\{d1, d2, d7\}$	$\{d3, d4, d5\}$
B3	$OUT[B2]$			
	$\{d3, d4, d5\}$			
		$\{d6\}$	$\{d3\}$	
		$\{d6\}$	$\{d3\}$	$\{d4, d5, d6\}$
B4	$\wedge OUT[B2], OUT[B3]$			
	$\{d3, d4, d5\} \cup \{d4, d5, d6\}$			
		$\{d7\}$	$\{d1, d4\}$	
		$\{d7\}$	$\{d1, d4\}$	$\{d3, d5, d6, d7\}$

Table 3: Iteration 1

Block	IN	Gen	Kill	OUT
B1	$OUT[Entry]$			
	$\emptyset$	$\{d1\}$	$\{d4, d7\}$	
		$\{d2\}$	$\{d5\}$	
		$\{d3\}$	$\{d6\}$	
		$\{d1, d2, d3\}$	$\{d4, d5, d6, d7\}$	$\{d1, d2, d3\}$
B2	$\wedge$ $OUT[B1]$, $OUT[B4]$			
	$\{d1, d2, d3\}$ $\cup\{d3, d5, d6, d7\}$			
		$\{d4\}$	$\{d1, d7\}$	
		$\{d5\}$	$\{d2\}$	
		$\{d4, d5\}$	$\{d1, d2, d7\}$	$\{d3, d4, d5, d6\}$
B3	$OUT[B2]$			
	$\{d3, d4, d5, d6\}$			
		$\{d6\}$	$\{d3\}$	
		$\{d6\}$	$\{d3\}$	$\{d4, d5, d6\}$
B4	$\wedge OUT[B2], OUT[B3]$			
	$\{d3, d4, d5, d6\} \cup \{d4, d5, d6\}$			
		$\{d7\}$	$\{d1, d4\}$	
		$\{d7\}$	$\{d1, d4\}$	{d3 d5, d6, d7}

Table 4: Iteration 2

Live Variable

Problem Definition

- A variable v is live at point p if the **value** of v is used **along some path starting at p**
- $Kill/Def(B)$: Variables defined in B
- $Gen/Use(B)$: Variables whose values may be used in B

Usage

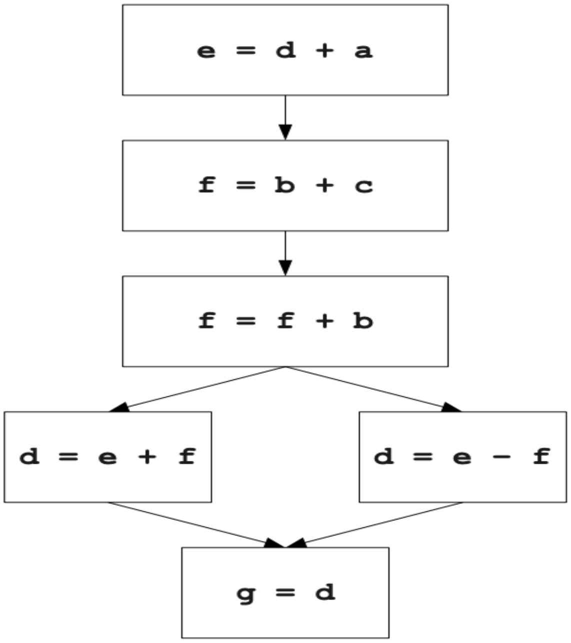
- Build inference graph in register allocation

Summary

Domain	Variable
Direction	Backward
Transfer Function	$Gen \cup (x - Kill)$
Meet Operator($\wedge$)	$\cup$
IN Equation	$IN[B] = f(OUT[B])$
OUT Equation	$OUT[B] = \wedge_{S \in Successor} IN[S]$
Initial Condition	$IN[EXIT] = \emptyset$
Boundary Condition	$IN[ALL - EXIT] = \emptyset$

Table 5: Summary for Live Variable

Example



Solution:

There are 2 iterations performed. I omitted the iteration 2 since it is the same as the iteration 2.

Block	OUT	Gen	Kill	IN
Exit				∅
B6				∅
B5				∅
B4				∅
B3				∅
B2				∅
B1				∅
Entry				∅

Table 6: Iteration 0

Block	OUT	Gen/Use	Kill/Def	IN
B6	$IN[EXIT]$			
	$\emptyset$			
		$\{d\}$	$\{g\}$	
		$\{d\}$	$\{g\}$	$\{d\}$
B5	$IN[B6]$			
	$\{d\}$			
		$\{e, f\}$	$\{d\}$	
		$\{e, f\}$	$\{d\}$	$\{e, f\}$
B4	$IN[B6]$			
	$\{d\}$			
		$\{e, f\}$	$\{d\}$	
		$\{e, f\}$	$\{d\}$	$\{e, f\}$
B3	$\wedge IN[B3], IN[B4]$			
	$\{e, f\} \cup \{e, f\}$			
		$\{f, b\}$	$\{f\}$	
		$\{f, b\}$	$\{f\}$	$\{e, f, b\}$
B2	$IN[B2]$			
	$\{e, f, b\}$			
		$\{b, c\}$	$\{f\}$	
		$\{b, c\}$	$\{f\}$	$\{b, c, e\}$
B1	$IN[B1]$			
	$\{b, c, e\}$			
		$\{d, a\}$	$\{e\}$	
		$\{d, a\}$	$\{e\}$	$\{a, b, c, d\}$

Table 7: Iteration 1

Usage

The result of liveness can be used to construct the inference graph. Live variables sit in the entry for every basic block cannot fit into the same register.

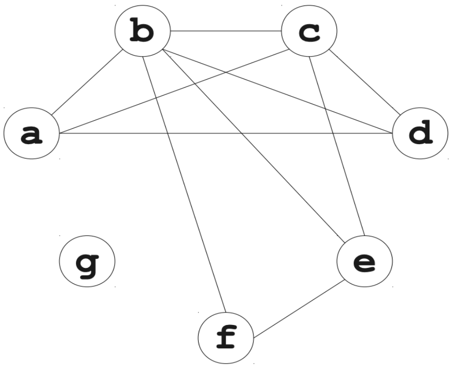


Figure 1: Inference Graph for previous question

Apply graph-coloring algorithm for assigning a color(register) to each node.

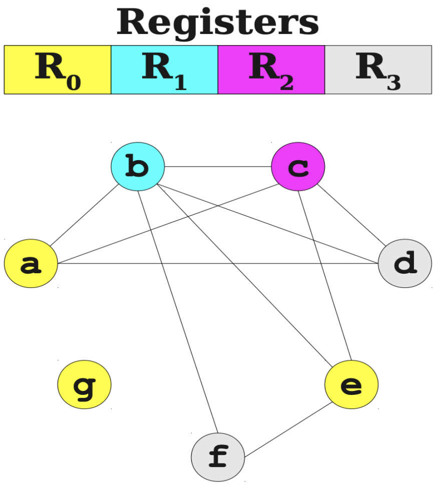


Figure 2: Resulting Graph

Note: Graph-coloring problem is a NP-Complete problem. The greedy algorithm is an approximated result.

Available Expression

Problem Definition

- An expression is available at a point p if **every path leading to p** contains a definition of the expression which is not subsequently killed
- $Gen(B)$: Expression generated in B
- $Kill(B)$: Expression x $\oplus$ y is killed if it assigns x or y and does not subsequently recompute x $\oplus$ y

Usage

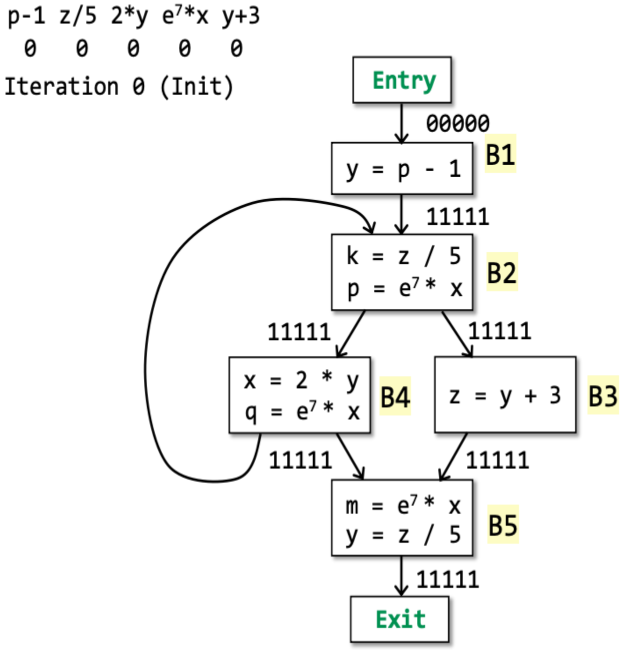
- Determine sub-expression for global common sub-expression elimination

Summary

Domain	Expression
Direction	Forward
Transfer Function	$Gen \cup (X - Kill)$
Meet Operator($\wedge$)	$\cap$
IN Equation	$IN[B] = \wedge_{P \in Predecessor} OUT[P]$
OUT Equation	$OUT[B] = f(IN[B])$
Initial Condition	$OUT[Entry] = \emptyset$
Boundary Condition	$OUT[All - Entry] = U$

Table 8: Summary for Available Expression

Example



p-1 z/5 2*y e7*x y+3
0 0 0 0 0
Iteration 0 (Init)
Entry
00000
y = p - 1
B1
11111
k = z / 5
p = e7 * x
B2
11111
11111
x = 2 * y
q = e7 * x
B4
z = y + 3
B3
11111
11111
m = e7 * x
y = z / 5
B5
11111
Exit

Solution: Usually, the U can be replaced with a bit vector of 1s. As well as, the $\emptyset$ can be replaced with a bit vector of 0s.

Block	IN	Gen	Kill	OUT
Entry				00000
B1				11111
B2				11111
B3				11111
B4				11111
B5				11111
Exit				11111

Table 9: Iteration 0

Block	IN	Gen	Kill	OUT
B1	$OUT[Entry]$			
	00000			
		$\{p-1\}$	$\{2*y, y+3\}$	
		$\{p-1\}$	$\{2*y, y+3\}$	10000
B2	$\wedge OUT[B4], OUT[B1]$			
	$11111 \wedge 10000$			
	10000			
		$\{z/5\}$		
		$\{e^7*x\}$	$\{p-1\}$	
		$\{z/5, e^7*x\}$	$\{p-1\}$	01010
B3	$OUT[B2]$			
	01010			
		$\{y+3\}$	$\{z/5\}$	
		$\{y+3\}$	$\{z/5\}$	00011
B4	$OUT[B2]$			
	01010			
		$\{2*y\}$	$\{e^7*x\}$	
		$\{e^7*x\}$		
		$\{2*y, e^7*x\}$	$\{e^7*x\}$	01110
B5	$\wedge OUT[B3], OUT[B4]$			
	$00011 \wedge 01110$			
	00010			
		$\{e^7*x\}$		
		$\{z/5\}$	$\{2*y, y+3\}$	
		$\{e^7*x, z/5\}$	$\{2*y, y+3\}$	01010

Table 10: Iteration 1

Block	IN	Gen	Kill	OUT
B1	$OUT[Entry]$			
	00000			
		$\{p-1\}$	$\{2*y, y+3\}$	
		$\{p-1\}$	$\{2*y, y+3\}$	10000
B2	$\wedge OUT[B4], OUT[B1]$			
	$01110 \wedge 10000$			
	00000			
		$\{z/5\}$		
		$\{e^7*x\}$	$\{p-1\}$	
		$\{z/5, e^7*x\}$	$\{p-1\}$	01010
B3	$OUT[B2]$			
	01010			
		$\{y+3\}$	$\{z/5\}$	
		$\{y+3\}$	$\{z/5\}$	00011
B4	$OUT[B2]$			
	01010			
		$\{2*y\}$	$\{e^7*x\}$	
		$\{e^7*x\}$		
		$\{2*y, e^7*x\}$	$\{e^7*x\}$	01110
B5	$\wedge OUT[B3], OUT[B4]$			
	$00011 \wedge 01110$			
	00010			
		$\{e^7*x\}$		
		$\{z/5\}$	$\{2*y, y+3\}$	
		$\{e^7*x, z/5\}$	$\{2*y, y+3\}$	01010

Table 11: Iteration 2

Busy/Anticipated Expression

Problem Definition

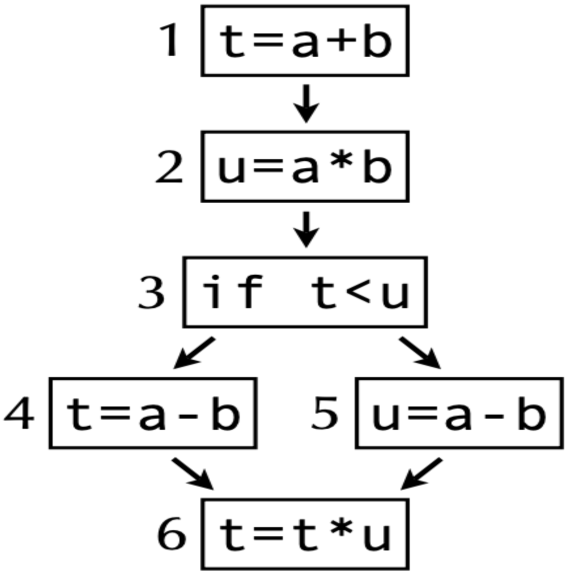
- An expression e is very busy at point p **if not matter what path is taken from p**, the expression e will be evaluated before any of its operands are redefined

Summary

Domain	Expression
Direction	Backward
Transfer Function	$Gen \cup (X - Kill)$
Meet Operator($\wedge$)	$\cap$
IN Equation	$IN[B] = f(OUT[B])$
OUT Equation	$OUT[B] = \wedge_{S \in Successor} IN[S]$
Initial Condition	$IN[EXIT] = \emptyset$
Boundary Condition	$IN[ALL - EXIT] = U$

Table 12: Summary for Busy Expression

Example



Solution:
There are two iterations, I omitted the iteration2 since it is the same as the iteration1.

Block	OUT	Gen	Kill	IN
Exit				$\emptyset$
B6				U
B5				U
B4				U
B3				U
B2				U
B1				U
Entry				U

Table 13: Iteration 0

Block	OUT	Gen	Kill	IN
B6	$IN[EXIT]$ $\emptyset$	$\{t*u\}$ $\{t*u\}$	$\{t*u, t<u\}$ $\{t*u, t<u\}$	$\{t*u\}$
B5	$IN[B6]$ $\{t*u\}$	$\{a-b\}$ $\{a-b\}$	$\{t*u, t<u\}$ $\{t*u, t<u\}$	$\{a-b\}$
B4	$IN[B6]$ $\{t*u\}$	$\{a-b\}$ $\{a-b\}$	$\{t*u, t<u\}$ $\{t*u, t<u\}$	$\{a-b\}$
B3	$\wedge IN[B4], IN[B3]$ $\{a-b\} \cap \{a-b\}$ $\{a-b\}$	$\{t<u\}$ $\{t<u\}$		$\{t<u, a-b\}$
B2	$IN[B3]$ $\{t<u, a-b\}$	$\{a*b\}$ $\{a*b\}$	$\{t<u, t*u\}$ $\{t<u, t*u\}$	$\{a*b, a-b\}$
B1	$IN[B2]$ $\{a*b, a-b\}$	$\{a+b\}$ $\{a+b\}$	$\{t<u, t*u\}$ $\{t<u, t*u\}$	$\{a+b, a-b, a*b\}$

Table 14: Iteration 1

Usage

From the previous example, we know that at the basic block 1, all expressions that involve a and b are busy in the future. We hoist those expressions to the basic block1 with new definitions and replace the usage of them with the new definitions. (This is similar to code refactoring)

Note: Typically, to perform the hoisting, you need to have additional dataflow analysis passes along with the Anticipated Expression. The given example is just a simple case where we can hoist all of them to the first basic block.

In reality, there are more constraints for code hoisting due to the safety concern for optimization. A typical usage for Anticipated expression is on LICM (Loop Invariant Code Motion). This will be covered later in the class.