COM SCI 132 Week 5

Aidan Jan

May 6, 2024

Reducing Variables

A program can have many variables, but a modern computer only has 16 registers. We must find some way to reduce the number of variables!

- Liveness Analysis → Interference Graph

Consider the example:

```
a = 1
b = 10
c = 9 + a
d = a + c
e = c + d
f = b + 8
c = f + e
f = e + c
b = c + 5
return b + f
```

Notice that we have six variables. How do we compress this?

```
                  |                  |  a  b  c  d  e  f
                  |  a = 1           |  *
               a  |                  |  |
                  |  b = 10          |  |  *
         c  b  a  |                  |  |  |
                  |  c = 9 + a       |  |  |  *
         c  b  a  |                  |  |  |  |
                  |  d = a + c       |  0  |  |  *
      d  c  b     |                  |     |  |  |
                  |  e = c + d       |     |  0  0  *
   e        b     |                  |     |        |
                  |  f = b + 8       |     0        |  *
f  e              |                  |              |  |
                  |  c = f + e       |        *     |  0
   e     c        |                  |        |     |
                  |  f = e + c       |        |     0  *
f        c        |                  |        |        |
                  |  b = c + 5       |     *  0        |
f           b     |                  |     |           |
                  |  return b + f    |     0           0
```

Above is a **liveness graph**.

- The middle section shows the code.
- The right section shows the "liveness" of each variable.
  - A variable becomes "alive" (denoted with `*`) when it is assigned a value.
  - A variable stays alive until the last time it is used before its next assignment or the end of the program.
  - A variable "dies" or can be deallocated after the last time it is used. (denoted with `O`).
- The left section shows which variables are alive between each statement.
  - Note that these are written in between each line of code instead of on the same line.
  - The number of registers required is the maximum number of variables on a line plus 1. (In this case, 3 variables + 1 = 4 registers)
  - This rule ONLY works when the code is straight executing (e.g., no branches or loops).

Liveness with loops - Context-free Graph

Consider the following:

```
    c = 0
    a = 0
L1: b = a + 1
    c = c + 2
    a = b * 2
    if (a < 10) goto L1
    return c
```

			init		iter 1		iter 2		iter 3		iter 4		iter 5	
statement	def	use	in	out	in	out	in	out	in	out	in	out	in	out
`a = 0`	a	-	-	-	-	a	-	ac	c	ac	c	ac		
`L1: b = a + 1`	b	a	-	-	a	c	ac	bc	ac	bc	ac	bc		
`c = c + 2`	c	c	-	-	c	b	bc	b	bc	bc	bc	bc		
`a = b * 2`	a	b	-	-	b	a	b	ac	bc	ac	bc	ac		
`if (a < 10) goto L1`	-	a	-	-	a	ac	ac	ac	ac	ac	ac	ac		
`return c`	-	c	-	-	c	-	c	-	c	-	c	-		

- `in` and `out` refer to which variables are live going into the statement and which variables are live coming out of the statement.
- We can stop after iter 4 because iter 4 matches iter 3, signifying that it will no longer change.

Liveness Equations:

$$\text{in}[n] = \text{use}[n] \cup (\text{out}[n] - \text{def}[n])$$
$$\text{out}[n] = \bigcup_{s \in \text{succ}[n]} \text{in}[s]$$

Number of Iterations Time Complexity:

- The number of iterations is $O(n^2)$.
- In the worst case scenario, every box is filled with every variable at the end of all the iterations. We have $2n$ boxes to fill in: n "in" boxes and n "out" boxes.
- In the worst case scenario, each iteration only adds one variable to one box. There are n variables.

- Therefore, the worst case scenario would be of order $2n \cdot n = n^2$.

Full Program Time Complexity:

- When there are n variables...
- $O(n^2)$ iterations
- $O(n)$ set unions per iteration
- $O(n)$ time to do a set union
- Total $= O(n)^4$ (polynomial time!)

Note that this is not the most efficient algorithm - this is a sloppy way to do the algorithm, in other words, the naive way. With more involved analyzing, this algorithm can be reduced to $O(n^2)$.