CSC2/455 Software Analysis and Improvement
Live variable analysis

Sreepathi Pai

URCS

January 30, 2019

Outline

Outline

What we know how to do so far

- Basic blocks to Control Flow Graphs
  - Section 5.4.3 in Cooper and Turczon
- Local optimizations (Value Numbering)

Outline

Definitions

A variable `x` is live at a definition (i.e. a write) if there is a subsequent read of the variable along a program path from that definition.
Can we identify live variables just by analyzing basic blocks (i.e. local analysis)?

Motivation(?)

```
void test() {
    int i;

    // begin delay
    for(i = 0; i < 10000; i++);
    // end delay
}
```

GCC without optimization

```
test():
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-4], 0
 .L3:
        cmp     DWORD PTR [rbp-4], 9999
        jg      .L4
        add     DWORD PTR [rbp-4], 1
        jmp     .L3
 .L4:
        nop
        pop     rbp
        ret
```

Code courtesy Compiler Explorer at `godbolt.org` (this is more convenient than `gcc -Wa` if you have the Internet)

GCC with -O3

```
test():
        ret
```

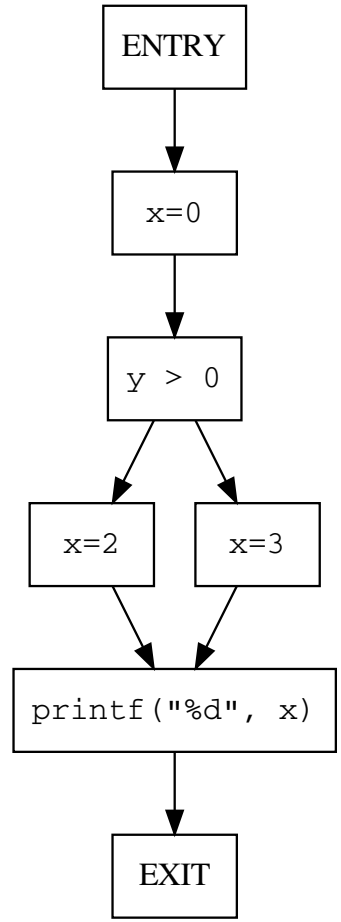
Code courtesy Compiler Explorer at `godbolt.org`

Example 1

```
x = 0;

if(y > 0)
   x = 2;
else
   x = 3;

printf("%d\n", x);
```

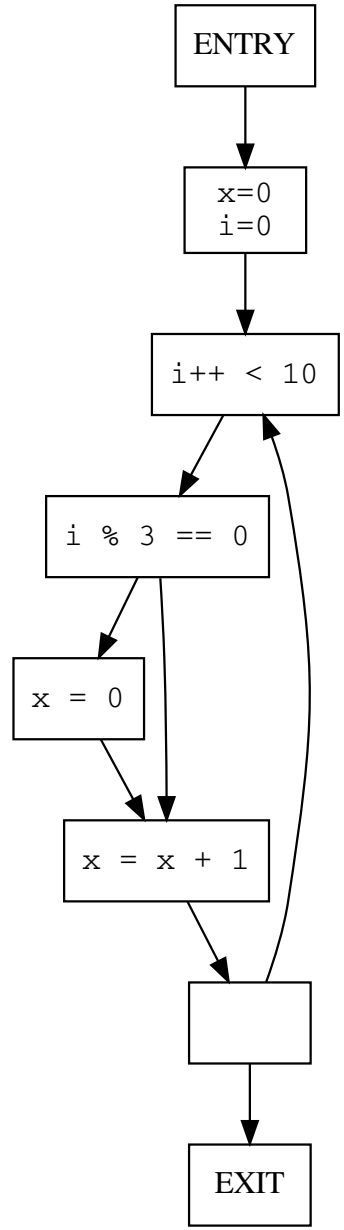


Example 2

```
x = 0
i = 0
while(i++ < 10) {
   if(i % 3 == 0) {
      x = 0;
   }

   x = x + 1;
}
```

- Which definition(s) of x are live at entry of loop?
- Which definition(s) of x are live in the loop?
- Which definition(s) of x are live at exit of loop?



Relook at the Motivating Example

```
    i = 1;
L1:
    if(!(i < 10000)) goto L2;
    i++;
    goto L1;

L2:
    return
```

Where in the CFG must `i` be "dead" to eliminate it?

Some Definitions

For a basic block s:

- $\mathrm{DEF}(s)$ (for "defined") is the set of variables written by s
- $\mathrm{USE}(s)$ is the set of variables read by s

For the blocks below, what are the DEF and USE sets?

```
BB_eg_0:
   i = i + 1

BB_eg_1:
   a = x;
   x = 3;

BB2_eg_2:
   t = a;
```

In general, which variables are kept "alive" and which are "killed" by a basic block in terms of its DEF and USE sets?

More definitions

- Upwardly exposed $\mathrm{UE}(s)$ are those variables that are read *before* they are written to in basic block s.
- A variable is alive in a basic block if:
  - ?
- A variable is killed in a basic block if:
  - ?

Finding out if a definition is live

- Start from the definition, and see if it "reaches" a use
  - i.e., it is not killed on *some* path (in the CFG) to a use
- Start from a use, and walk backwards to see if you can reach the definition
  - Repeat for all uses

When does enumerating paths work in practice?

A More Practical Way: Iterative Data flow Analysis

- Setup data flow equations
- Solve these data flow equations

Outline

Equations for Live Variables

- Remember that each basic block has "incoming" edges (from predecessors) and "outgoing" edges (to successors).
- When is an "incoming" variable live "into" a block?
  - it is in UE for that block
- When is a variable live out of a block?
  - it is live into a successor block
  - first try: $\mathrm{LIVEOUT}(n) = \cup_m \mathrm{UE}(m)$ (where $m \in succ(n)$)
  - what are we missing?

Complete equation

$$\mathrm{LIVEOUT}(n) = \cup_{m \in succ(n)} \mathrm{UE}(m) \cup (\mathrm{LIVEOUT}(m) \cap \overline{\mathrm{DEF}(m)})$$

The variables live out of block n must:

- be upwardly exposed in a successor m
- OR be live out of m and NOT be killed by m (i.e. $\overline{\mathrm{DEF}(m)}$)
  - note this is equivalent to $(\mathrm{LIVEOUT}(m) - \mathrm{DEF}(m))$

How would you begin solving this equation?

Iterative Data flow Analysis Algorithm for Live Variables

- Live variables is a *backwards dataflow analysis*
  - 'Facts" flow from a node's successors to it
- Initialization:
  - Compute $\mathrm{DEF}(s)$ for all blocks s
  - Compute $\mathrm{UE}(s)$ for all blocks s
  - Set $\mathrm{LIVEOUT}(s) = \emptyset$ for all s
- Repeat (iterate):
  - Compute $\mathrm{LIVEOUT}(s)$ for all blocks s
  - Until no set $\mathrm{LIVEOUT}(s)$ changes – i.e. until a fixpoint is reached.

Termination and correctness

- Termination
  - Note $\mathrm{LIVEOUT}(s)$ only increases in size or remains the same in every iteration
  - What is the maximum size of $\mathrm{LIVEOUT}(s)$?
- Correctness
  - When the algorithm terminates, all computed $\mathrm{LIVEOUT}(s)$ sets satisfy the equation
  - and hence meet the definition of "liveness".

Quality of solution?

How does the iterative solution differ from the path-based solution?

Outline

References

- Chapter 8 of Cooper and Turczon
  - Section 8.6.1
- Also recommended:
  - Aho, Lam, Sethi and Ullman, Chapter 9, Section 9.2.5