COSE312: Compilers

Lecture 20 — Data-Flow Analysis (2)

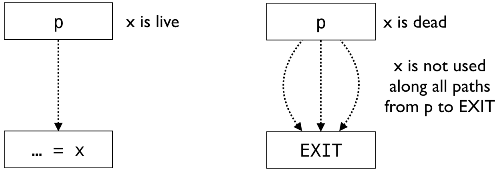
Hakjoo Oh
2017 Spring

Final Exam

- 6/19 (Mon), 15:30–16:45 (in class)
- Do not be late.
- Coverage: semantic analysis, IR, Optimization

Liveness Analysis

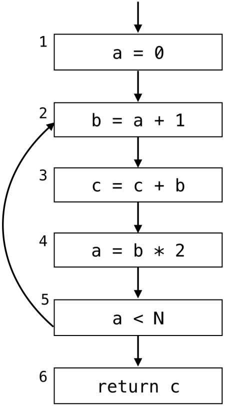
- A variable is *live* at program point p if its value could be used in the future (along some path starting at p).



- Liveness analysis aims to compute the set of live variables for each basic block of the program.

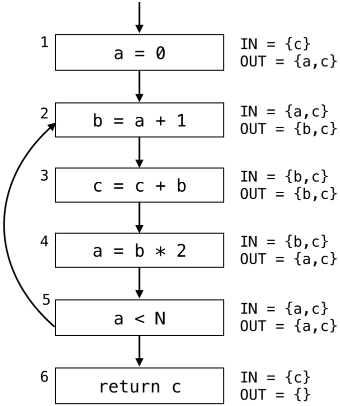
Example: Liveness of Variables

We analyze liveness from the future to the past.



- The live range of b: $\{2 \rightarrow 3, 3 \rightarrow 4\}$
- The live range of a: $\{1 \rightarrow 2, 4 \rightarrow 5 \rightarrow 2\}$ (not from $2 \rightarrow 3 \rightarrow 4$)
- The live range of c: the entire code

Example: Liveness of Variables



Applications

- Deadcode elimination
  - Problem: Eliminate assignments whose computed values never get used.
  - Solution: How?
- Uninitialized variable detection
  - Problem: Detect uninitialized use of variables
  - Solution: How?
- Register allocation
  - Problem: Rewrite the intermediate code to use no more temporaries than there are machine registers
  - Example:

```
a := c + d          r1 := r2 + r3
e := a + b          r1 := r1 + r4
f := e - 1          r1 := r1 - 1
```

  - Solution: How?

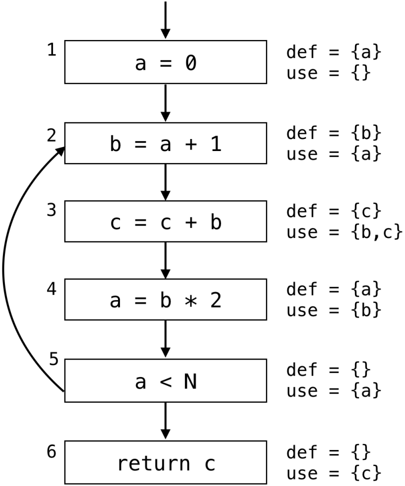
Liveness Analysis

The goal is to compute

$$\begin{array}{rcl} \textbf{in} & : & Block \rightarrow 2^{Var} \\ \textbf{out} & : & Block \rightarrow 2^{Var} \end{array}$$

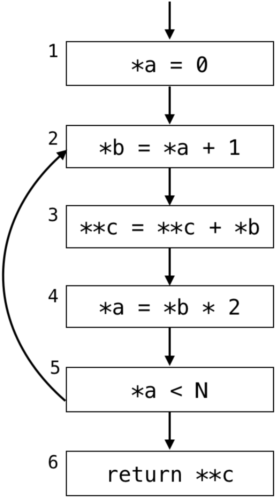
1. Compute def/use sets.
2. Derive transfer functions for each basic block in terms of def/use sets.
3. Derive the set of data-flow equations.
4. Solve the equation by the iterative fixed point algorithm.

Def/Use Sets



1
a = 0
def = {a}
use = {}
2
b = a + 1
def = {b}
use = {a}
3
c = c + b
def = {c}
use = {b,c}
4
a = b * 2
def = {a}
use = {b}
5
a < N
def = {}
use = {a}
6
return c
def = {}
use = {c}

cf) Def/Use sets are only dynamically computable



Data-Flow Equations

Intuitions:

1. If a variable is in **use**(B), then it is live on entry to block B.
2. If a variable is live at the end of block B, and not in **def**(B), then the variable is also live on entry to B.
3. If a variable is live on enty to block B, then it is live at the end of predecessors of B.

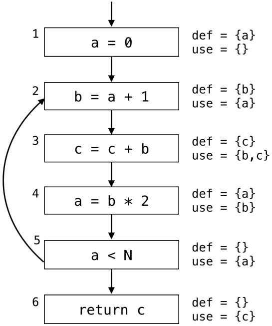
Equations:

$$\begin{aligned}\mathbf{in}(B) &= \mathbf{use}(B) \cup (\mathbf{out}(B) - \mathbf{def}(B))\\ \mathbf{out}(B) &= \bigcup_{B \hookrightarrow S} \mathbf{in}(S)\end{aligned}$$

Fixed Point Computation

For all $i, \mathbf{in}(B_i) = \mathbf{out}(B_i) = \emptyset$
while (changes to any **in** and **out** occur) {
  For all i, update
    $\mathbf{in}(B_i) = \mathbf{use}(B) \cup (\mathbf{out}(B) - \mathbf{def}(B))$
    $\mathbf{out}(B_i) = \bigcup_{B \hookrightarrow S} \mathbf{in}(S)$
}

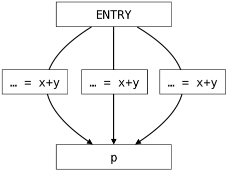
Example



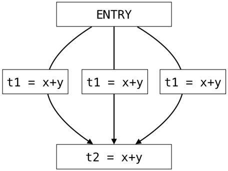
			$1st$		$2nd$		$3rd$	
	use	**def**	**out**	**in**	**out**	**in**	**out**	**in**
6	$\{c\}$	$\emptyset$	$\emptyset$	$\{c\}$	$\emptyset$	$\{c\}$	$\emptyset$	$\{c\}$
5	$\{a\}$	$\emptyset$	$\{c\}$	$\{a, c\}$	$\{a, c\}$	$\{a, c\}$	$\{a, c\}$	$\{a, c\}$
4	$\{b\}$	$\{a\}$	$\{a, c\}$	$\{b, c\}$	$\{a, c\}$	$\{b, c\}$	$\{a, c\}$	$\{b, c\}$
3	$\{b, c\}$	$\{c\}$	$\{b, c\}$	$\{b, c\}$	$\{b, c\}$	$\{b, c\}$	$\{b, c\}$	$\{b, c\}$
2	$\{a\}$	$\{b\}$	$\{b, c\}$	$\{a, c\}$	$\{b, c\}$	$\{a, c\}$	$\{b, c\}$	$\{a, c\}$
1	$\emptyset$	$\{a\}$	$\{a, c\}$	$\{c\}$	$\{a, c\}$	$\{c\}$	$\{a, c\}$	$\{c\}$

Available Expressions Analysis

- An expression $x + y$ is *available* at a point p if every path from the entry node to p evaluates $x + y$, and after the last such evaluation prior to reaching p, there are no subsequent assignments to x or y.



- Application: common subexpression elimination (i.e., given a program that computes e more than once, eliminate one of the duplicate computations)



Available Expressions Analysis

The goal is to compute

$$\begin{array}{rcl} \textbf{in} & : & Block \rightarrow 2^{Expr} \\ \textbf{out} & : & Block \rightarrow 2^{Expr} \end{array}$$

1. Derive the set of data-flow equations.
2. Solve the equation by the iterative fixed point algorithm.

Gen/Kill Sets

- **gen**(B): the set of expressions evaluated and not subsequently killed
- **kill**(B): the set of expressions whose variables can be killed

Exercise:

- What expressions are generated and killed by each of statements?

Statement s	**gen**(s)	**kill**(s)
$x = y + z$		
$x = \texttt{alloc}(n)$		
$x = y[i]$		
$x[i] = y$		

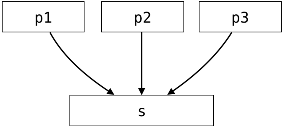
- What expressions are generated and killed by the block?

$$\begin{array}{|l|} \hline a = b + c \\ b = a - d \\ c = b + c \\ d = a - d \\ \hline \end{array}$$

1. Set up a set of data-flow equations

Intuitions:

1. At the entry, no expressions are available.
2. An expression is available at the entry of a block only if it is available at the end of *all* its predecessors.



Equations:

$$\mathbf{in}(ENTRY) = \emptyset$$

$$\mathbf{out}(B) = \mathbf{gen}(B) \cup (\mathbf{in}(B) - \mathbf{kill}(B))$$

$$\mathbf{in}(B) = \bigcap_{P \to B} \mathbf{out}(B)$$

2. Solve the equations

- Trivial solution: $\mathbf{in}(B_i) = \mathbf{out}(B_i) = \emptyset$.
- Need to find the greatest solution (i.e., greatest fixed point) of the equation.

$\mathbf{in}(\mathit{ENTRY}) = \emptyset$
For other $B_i, \mathbf{in}(B_i) = \mathbf{out}(B_i) = \mathit{Expr}$
while (changes to any **in** and **out** occur) {
  For all i, update
    $\mathbf{in}(B_i) = \bigcap_{P \hookrightarrow B_i} \mathbf{out}(P)$
    $\mathbf{out}(B_i) = \mathbf{gen}(B_i) \cup (\mathbf{in}(B_i) - \mathbf{kill}(B_i))$
}

Summary: Data-flow Analysis

	union	intersection
forward	reaching definitions	available expressions
backward	liveness	