CSCI 742 - Compiler Construction

Lecture 33

Live Variable Analysis

Instructor: Hossein Hojjat

April 16, 2018

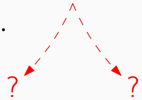
Recap: Control Flow Graphs

- **Control Flow Graph (CFG):** graph representation of computation and control flow in the program
- Framework to statically analyze program control-flow
- Next: use CFG to statically extract information about program
- Reason at compile-time about run-time values of variables in all program executions
- Data-flow analysis: gather information about the possible set of values of variables at various points in a program

Liveness

- Liveness is a data-flow property of variables:
  "Is the value of this variable needed?"
- Optimization: eliminate assignments to dead variables
  (i.e. variables that are never used after definitions)

```
int f(int x, int y) {
  int z = x + y;
...
```



- Live variable analysis is **undecidable** in general
- We compute a syntactic and conservative approximation of liveness
  - Like many other data-flow analysis techniques

```
int f(int x, int y) {
  int z = x + y;
  if (tricky-calculation) x = z;
  return x;
}
```

Live Variable Analysis: Backward

- Liveness is naturally computed using **backward** data-flow analysis
- Usage information from future statements must be propagated backward through the program to discover which variables are live

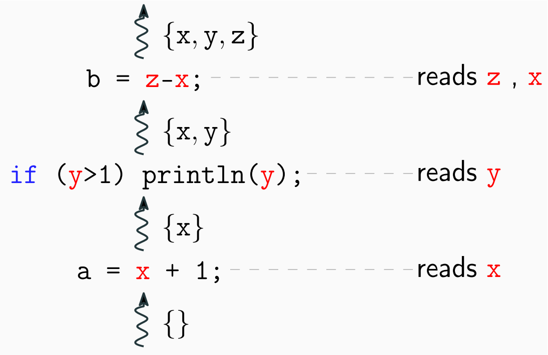
```
int f(int x, int y) {
   int z = x + y;
...

int t=z-2;    println(z);
         if(z!=2) ...;
```

Live Variable Analysis

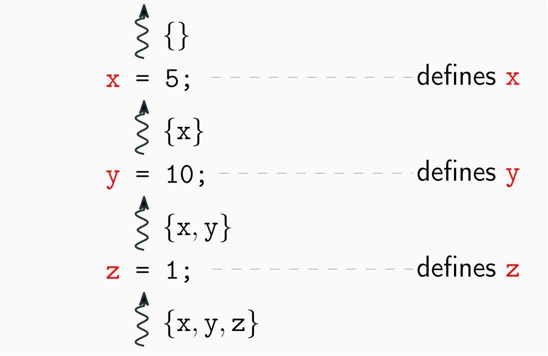
- Variable liveness flows backward through the program
- Each statement has an effect on liveness information as it flows past

- A statement makes a variable **live** when it reads it



Live Variable Analysis

- Variable liveness flows backward through the program
- Each statement has an effect on liveness information as it flows past
- A statement makes a variable **dead** when it defines (assigns to) it



Live Variable Analysis

As liveness flows backwards past an statement, we modify liveness information:

- Add any variables which it reads (they become live)
- Remove any variables which it defines (they become dead)

$\{x, y\}$

use(`println(x)`) = { x }

$\{y\}$

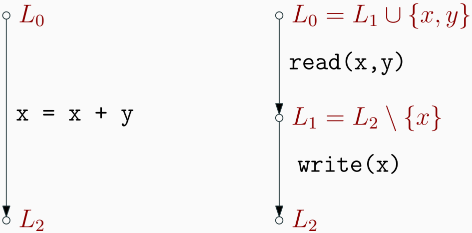
$\{y\}$

def(`x = 3`) = { x }

$\{x, y\}$

Live Variable Analysis

- If a statement both references and defines variables, remove the defined variables before adding the read ones
- L_0 Initial set of live variables



$$L_0 = \left(L_2 \setminus \{x\}\right) \ \cup \ \{x, y\}$$

In general:

- *in*(S): set of live variables immediately before statement S
- *out*(S): set of live variables immediately after statement S

$$\textit{in}(S) = \left(\textit{out}(S) \setminus \textit{def}(S)\right) \ \cup \ \textit{use}(S)$$

Straight-Line Code

- In straight-line code each node has a unique successor
- Variables live at the exit of a node are exactly those variables live at the entry of its successor

i_1:

$out(i_1) = \{ x , y , z \}$

$in(i_2) = \{ x , y , z \}$

```
i2:  x = x+y
```

$out(i_2) = \{ x , z \}$

$in(i_3) = \{ x , z \}$

```
i3:  println(x)
```

$out(i_3) = \{ z \}$

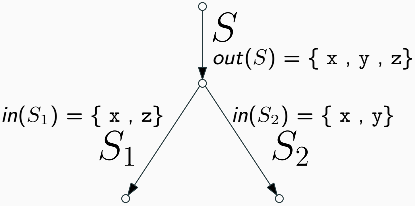
$in(i_4) = \{ z \}$

i_4:

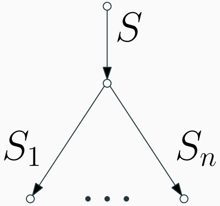
Multiple Successors

- In general each node has an arbitrary number of successors
- Variables live at the exit of a node are exactly those variables live at the entry of all its successors

Example:



General:



$$\mathit{out}(S) = \bigcup_{S_i \in \mathbf{succ}(S)} \mathit{in}(S_i)$$

Data-flow Equations

- Start with CFG and derive a system of constraints between live variable sets

$$
\begin{aligned}
\mathit{in}(S) &= \big(\mathit{out}(S) \setminus \mathit{def}(S)\big) \ \cup \ \mathit{use}(S) \\
\mathit{out}(S) &= \bigcup_{S_i \in \mathit{succ}(S)} \mathit{in}(S_i)
\end{aligned}
$$

Solve constraints:

- Start with empty sets of live variables
- Iteratively apply constraints
- Stop when we reach a fixed point

Constraint Solving Algorithm

for all statements S **do**

$\quad$ $\mathit{in}(S) = \mathit{out}(S) = \emptyset$

repeat

$\quad$ **select** a statement S such that

$\qquad$ $\mathit{in}(S) \neq (\mathit{out}(S) \setminus \mathit{def}(S)) \cup \mathit{use}(S)$

$\quad$ or (respectively)

$\qquad$ $\mathit{out}(S) \neq \bigcup_{S_i \in \mathit{succ}(S)} \mathit{in}(S_i)$

$\quad$ update $\mathit{in}(S)$ (or $\mathit{out}(S)$) accordingly

until no such change is possible

Exercise

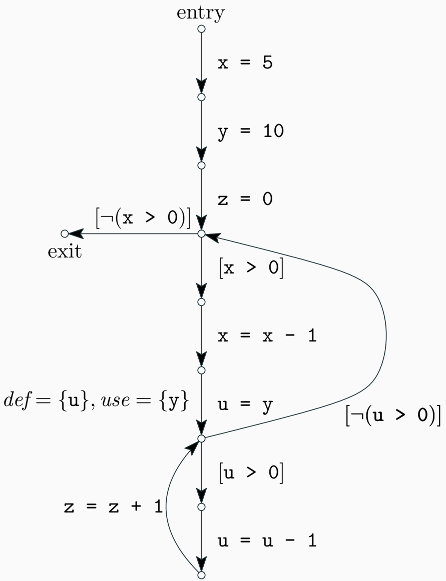
- Compute the set of live variables at each point of the program

```
x = 5;
y = 10;
z = 0;
while (x > 0) {
  x = x - 1;
  u = y;
  while (u > 0) {
    u = u - 1;
    z = z + 1;
  }
}
```

Exercise

- Compute the set of live variables at each point of the program

```
x = 5;
y = 10;
z = 0;
while (x > 0) {
  x = x - 1;
  u = y;
  while (u > 0) {
    u = u - 1;
    z = z + 1;
  }
}
```



$$in(S) = \big(out(S) \setminus def(S)\big) \cup use(S)$$

$$out(S) = \bigcup_{S_i \in succ(S)} in(S_i)$$