CSCI 742 - Compiler Construction

Lecture 35
Live Variable Analysis
Instructor: Hossein Hojjat

April 26, 2017
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)

```c
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

```c
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

```java
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

```
if (y > 1) println(y);
    { x, y }
b = z - x;
    { x, y, z }
a = x + 1;
    { x }
{ }
```

- **reads** `x`
- **reads** `y`
- **reads** `z, x`
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

```plaintext
{ }
{x}
{x}
{x, y}
{x, y, z}

x = 5;  ---- defines x
y = 10;  ---- defines y
z = 1;   ---- defines z
```
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)

\[
\begin{align*}
\text{use}(\text{println}(x)) &= \{ x \} \\
\text{def}(x = 3) &= \{ x \}
\end{align*}
\]
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

\[ L_0 = (L_2 \setminus \{x\}) \cup \{x, y\} \]

In general:

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

\[ in(S) = \left( out(S) \setminus def(S) \right) \cup use(S) \]
• 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
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:**

\[
\begin{align*}
\text{out}(S) &= \{ x, y, z \} \\
\text{in}(S_1) &= \{ x, z \} \\
\text{in}(S_2) &= \{ x, y \}
\end{align*}
\]

**General:**

\[
\text{out}(S) = \bigcup_{S_i \in \text{succ}(S)} \text{in}(S_i)
\]
Data-flow Equations

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

\[
\begin{align*}
    in(S) &= (out(S) \setminus \text{def}(S)) \cup \text{use}(S) \\
    out(S) &= \bigcup_{S_i \in \text{succ}(S)} in(S_i)
\end{align*}
\]

Solve constraints:

- Start with empty sets of live variables
- Iteratively apply constraints
- Stop when we reach a fixed point
for all statements $S$ do
  $in(S) = out(S) = \emptyset$

repeat
  select a statement $S$ such that
  
  $in(S) \neq (out(S) \setminus \text{def}(S)) \cup \text{use}(S)$
  or (respectively)
  
  $out(S) \neq \bigcup_{S_i \in \text{succ}(S)} \text{in}(S_i)$
  update $in(S)$ (or $out(S)$) accordingly

until no such change is possible
Exercise

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

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