Compiler Construction

Chapter 7 – Liveness Analysis

Prof. Dr. Jörg Kreiker
joerg.kreiker@informatik.hs-fulda.de

Fachbereich Angewandte Informatik
Hochschule Fulda – University of Applied Sciences

Agenda

1. Introduction
2. Live and dead variables
3. Computation of liveness information
4. Example

Liveness Example

A liveness analysis computes, for each program point, which variables/registers are in use (live).

Example

```
1:      x = y + 2;
2:      y = 5;
3:      x = y + 3;
```

- The value of `x` at program points 1 and 2 is not used before it is overwritten.
- We say `x` is dead at program points 1 and 2
- Liveness Analysis is useful for
  - Assignments to dead variables can be removed
  - Such assignments may be the result of a number of transformations

Programming Language

In principle, liveness analysis is performed on (register) machine instructions. We use the following programming language as an abstraction.

• $x, y, \ldots$	Register
• $R = e;$	Assignments
• $R = M[e];$	Read from memory
• $M[e_1] = e_2;$	Store to memory
• if (e) s_1 else s_2	Conditional jump
• goto L;	Jump or loop
• stop	end of function/program

- Consider statements as edge labels of a graph
- $Pos(e)$ is the true edge of a conditional
- $False(e)$ is the false edge of a conditional

Definition: live

- Let $\pi = s_1; \dots s_n$ be a sequence of statements
- Variable x is live at program point s_1, if all variables X are live after s_n and
    - $x \in X$ and not defined in π or
    - π can be de-composed into $\pi = \pi_1 k \pi_2$ such that
        - k is a use of x and
        - x is not defined in π_1

Definitions and Uses

Statement	Use	Definition
;	$\emptyset$	$\emptyset$
$Pos(e)$	$Vars(e)$	$\emptyset$
$Neg(e)$	$Vars(e)$	$\emptyset$
$x = e$	$Vars(e)$	$\{x\}$
$x = M[e]$	$Vars(e)$	$\{x\}$
$M[e_1] = e_2$	$Vars(e_1) \cup Vars(e_2)$	$\emptyset$

- $Vars(e)$ is the set of variables occurring in an expression
- Example: $Vars((x + y) * (x + 1)) = \{x, y\}$

Definition: dead

- A variable x that is not live at program point s along path π relative to X is called dead at s relative to X.

- Replace each statement s of edge $k = (u, s, v)$ by a function $[\![k]\!]$ that maps the live variables at v to the live variables at u
- Let $\mathcal{L}$ be sets of variables
- The meaning of a path $\pi = k_1 \ldots k_r$ is then the function composition

$$[\![\pi]\!] = [\![k_1]\!] \circ \ldots \circ [\![k_r]\!]$$

$$\begin{array}{lclcl}
[\![;]\!]\ L & = & L & & \\
[\![Pos(e)]\!]\ L & = & [\![Pos(e)]\!]\ L & = & L \cup Vars(e) \\
[\![x = e]\!]\ L & = & (L \setminus \{x\}) \cup Vars(E) & & \\
[\![x = M[e]]\!]\ L & = & (L \setminus \{x\}) \cup Vars(E) & & \\
[\![M[e_1] = e_2]\!]\ L & = & L \cup Vars(e_1) \cup Vars(e_2) & &
\end{array}$$

Live Variables of a Program

Solve the following equation system

$$\begin{array}{lll} L[stop] & \supseteq & X \\ L[u] & \supseteq & [\![k]\!](L[v]) \qquad \text{for edges } k = (u, \_, v) \end{array}$$

- Equations propagate sets from right to left

$\Rightarrow$ Information flows backwards

- Since there are only finitely many variables, a fixed point always exists
- The live variables at stop are supposed to be X