# Compiler Construction 2016/2017
# Liveness Analysis

Peter Thiemann

December 19, 2016

# Outline

1. Liveness Analysis

# Liveness Analysis

## IR after instruction selection

- abstract assembly code
- operates on unbounded number of temporaries

## Next goal

- register allocation

# Register allocation

- instruction operands in registers
- bounded number of registers $\Rightarrow$ limited resource
- questions to be addressed
  - how many registers are needed at every program point?
  - what to do if fewer registers are available than needed?
- optimal allocation is NP-complete

# How many registers are needed?

## Concept: Live range

The live range of a temporary spans all instructions that may be executed between its definition and one of its uses.

## Concept: Liveness

A temporary is live at some instruction if its value may be used in the future.

## Answers

- At any given instruction, all live temporaries may be needed.
- Temporaries that are not needed at the same time may share a register.

# What if fewer registers are available than needed?

## Concept: Spill

Spilling a temporary means

- allocate it in a stack frame
- insert store instruction right after its definition
- insert load instruction before every use

## Consequences of spilling

- shortens the live range of a temporary
- increases the size of a stack frame
- accessing the temporary becomes more expensive

# Roadmap

1. control-flow graph
2. liveness analysis
3. interference graph

# Control Flow Graph (CFG)

Graphical representation of control flow in a program

## CFG of a program

- Nodes: entry, exit, and each occurrence of a statement in program
- Edges: an edge from $n$ to $n'$ represents a potential control transfer from (the end of ) $n$ to (the beginning of) $n'$

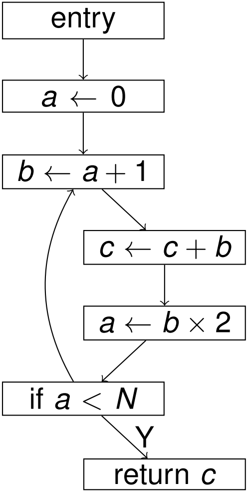
Terminology

Out-edges from $n$ lead to successor nodes, **succ**[$n$]

In-edges to $n$ come from predecessor nodes, **pred**[$n$]

# Example CFG

$$
\begin{array}{ll}
 & a \leftarrow 0 \\
L_1: & b \leftarrow a + 1 \\
 & c \leftarrow c + b \\
 & a \leftarrow b \times 2 \\
 & \text{if } a < N \text{ goto } L_1 \\
 & \text{return } c
\end{array}
$$



# Definitions and uses

Consider a CFG

- A variable $v$ gets defined by node $n$,
  if the statement at $n$ assigns to $v$.
- A variable $v$ gets used by node $n$,
  if $v$ occurs in an expression at $n$, i.e., it reads from $v$.
- **def**$[n]$ set of variables defined by $n$
- **use**$[n]$ set of variables used by $n$
- **def**$[n]$ and **use**$[n]$ are fixed by program/CFG

# Example def-use

| | | def[$n$] | use[$n$] |
|---|---|---|---|
| | $a \leftarrow 0$ | $\{a\}$ | $\emptyset$ |
| $L_1$ : | $b \leftarrow a + 1$ | $\{b\}$ | $\{a\}$ |
| | $c \leftarrow c + b$ | $\{c\}$ | $\{c, b\}$ |
| | $a \leftarrow b \times 2$ | $\{a\}$ | $\{b\}$ |
| | if $a < N$ goto $L_1$ | $\emptyset$ | $\{a\}$ |
| | return $c$ | $\emptyset$ | $\{c\}$ |

# Liveness

## Definition

Variable $v$ is live on edge $e$ if there is an execution path from $e$ to a use of $v$ that does not pass through any definition of $v$.

## Liveness Analysis

A data flow analysis that computes the variables that may be live at each edge of a control flow graph.

## Definition for analysis

Variable $v$ is live on edge $e$ if there is a directed path from $e$ to a use of $v$ that does not pass through any definition of $v$.

# More on liveness

## Liveness at node $n$

- $v$ is live-in at $n$ if $v$ is live on any in-edge of $n$
  **in**$[n]$ variables live-in at $n$
- $v$ is live-out at $n$ if $v$ is live on any out-edge of $n$
  **out**$[n]$ variables live-out at $n$

# Liveness analysis

## Computation rules for liveness

1. $v \in \mathbf{use}[n]$ implies $v$ live-in at $n$
2. $v$ live-in at $n$ implies $v$ live-out at all $m \in \mathbf{pred}[n]$
3. $v$ live-out at $n$ and $v \notin \mathbf{def}[n]$ implies $v$ live-in at $n$

$\Rightarrow$ liveness information is propagated <u>backwards</u>

# Liveness analysis

## Computation rules for liveness

1. $v \in \textbf{use}[n]$ implies $v$ live-in at $n$
2. $v$ live-in at $n$ implies $v$ live-out at all $m \in \textbf{pred}[n]$
3. $v$ live-out at $n$ and $v \notin \textbf{def}[n]$ implies $v$ live-in at $n$

$\Rightarrow$ liveness information is propagated backwards

## Inequations from computation rules

$$\textbf{in}[n] \supseteq \underbrace{\textbf{use}[n]}_{\text{rule 1}} \cup \underbrace{(\textbf{out}[n] \setminus \textbf{def}[n])}_{\text{rule 3}}$$

$$\textbf{out}[n] \supseteq \underbrace{\bigcup_{m \in \textbf{succ}[n]} \textbf{in}[m]}_{\text{rule 2}}$$

# Liveness analysis

- Each solution of the inequations is valid liveness information
- Wanted: least solution that does not contain spurious information
- computed by fixpoint iteration
  - treat inequations (from right to left) as functions
  - update the left-hand **in**[$n$] and **out**[$n$] until no further change happens
- result is a fixpoint because afterwards

$$\mathbf{in}[n] = \mathbf{use}[n] \cup (\mathbf{out}[n] \setminus \mathbf{def}[n])$$
$$\mathbf{out}[n] = \bigcup_{m \in \mathbf{succ}[n]} \mathbf{in}[m]$$

# Algorithm: liveness analysis

**for all** node $n$ **do**
    $\mathbf{in}^0[n] \leftarrow \emptyset$
    $\mathbf{out}^0[n] \leftarrow \emptyset$
**end for**
$i = 0$
**repeat**
    $i \leftarrow i + 1$
    **for all** node $n$ **do**
        $\mathbf{in}^i[n] \leftarrow \mathbf{use}[n] \cup (\mathbf{out}^{i-1}[n] \setminus \mathbf{def}[n])$
        $\mathbf{out}^i[n] \leftarrow \bigcup_{s \in \mathbf{succ}[n]} \mathbf{in}^{i-1}[s]$
    **end for**
**until** $\forall n, \mathbf{in}^i[n] = \mathbf{in}^{i-1}[n] \wedge \mathbf{out}^i[n] = \mathbf{out}^{i-1}[n]$

# Notes on the algorithm

- Each loop iteration increases **in**$[n]$ and/or **out**$[n]$
- Liveness flows backwards along control-flow arcs
- The inner loop should visit nodes in reverse flow order as much as possible
- Speedup: compress nodes to basic blocks

# Correctness

## Monotone

$$\mathbf{in}^{i+1}[n] \supseteq \mathbf{in}^{i}[n] \qquad \mathbf{out}^{i+1}[n] \supseteq \mathbf{out}^{i}[n]$$

## Bounded

$$\mathbf{in}^{i}[n] \subseteq \mathbf{use}[n] \cup (\mathbf{out}^{i}[n] \setminus \mathbf{def}[n])$$

$$\mathbf{out}^{i}[n] \subseteq \bigcup_{s \in \mathbf{succ}[n]} \mathbf{in}^{i}[s]$$

# Example analysis, 1st iteration

| | | $\mathbf{def}[n]$ | $\mathbf{use}[n]$ | $\mathbf{in}^1[n]$ | $\mathbf{out}^1[n]$ | $\mathbf{in}^2[n]$ | $\mathbf{out}^2[n]$ |
|---|---|---|---|---|---|---|---|
| | $a \leftarrow 0$ | $\{a\}$ | $\emptyset$ | $\{c\}$ | $\{c, a\}$ | | |
| $L_1$ : | $b \leftarrow a + 1$ | $\{b\}$ | $\{a\}$ | $\{c, a\}$ | $\{c, b\}$ | | |
| | $c \leftarrow c + b$ | $\{c\}$ | $\{c, b\}$ | $\{c, b\}$ | $\{c, b\}$ | | |
| | $a \leftarrow b \times 2$ | $\{a\}$ | $\{b\}$ | $\{c, b\}$ | $\{c, a\}$ | | |
| | if $a < N$ goto $L_1$ | $\emptyset$ | $\{a\}$ | $\{c, a\}$ | $\{c\}$ | | |
| | return $c$ | $\emptyset$ | $\{c\}$ | $\{c\}$ | $\emptyset$ | | |

# Example analysis, 2nd iteration

| | | def[$n$] | use[$n$] | in$^1$[$n$] | out$^1$[$n$] | in$^2$[$n$] | out$^2$[$n$] |
|---|---|---|---|---|---|---|---|
| | $a \leftarrow 0$ | $\{a\}$ | $\emptyset$ | $\{c\}$ | $\{c, a\}$ | $\{c\}$ | $\{c, a\}$ |
| $L_1$ : | $b \leftarrow a + 1$ | $\{b\}$ | $\{a\}$ | $\{c, a\}$ | $\{c, b\}$ | $\{c, a\}$ | $\{c, b\}$ |
| | $c \leftarrow c + b$ | $\{c\}$ | $\{c, b\}$ | $\{c, b\}$ | $\{c, b\}$ | $\{c, b\}$ | $\{c, b\}$ |
| | $a \leftarrow b \times 2$ | $\{a\}$ | $\{b\}$ | $\{c, b\}$ | $\{c, a\}$ | $\{c, b\}$ | $\{c, a\}$ |
| | if $a < N$ goto $L_1$ | $\emptyset$ | $\{a\}$ | $\{c, a\}$ | $\{c\}$ | $\{c, a\}$ | $\{c, a\}$ |
| | return $c$ | $\emptyset$ | $\{c\}$ | $\{c\}$ | $\emptyset$ | $\{c\}$ | $\emptyset$ |

# Example analysis, 2nd iteration

| | | def[$n$] | use[$n$] | in$^1$[$n$] | out$^1$[$n$] | in$^2$[$n$] | out$^2$[$n$] |
|---|---|---|---|---|---|---|---|
| | $a \leftarrow 0$ | $\{a\}$ | $\emptyset$ | $\{c\}$ | $\{c, a\}$ | $\{c\}$ | $\{c, a\}$ |
| $L_1$ : | $b \leftarrow a + 1$ | $\{b\}$ | $\{a\}$ | $\{c, a\}$ | $\{c, b\}$ | $\{c, a\}$ | $\{c, b\}$ |
| | $c \leftarrow c + b$ | $\{c\}$ | $\{c, b\}$ | $\{c, b\}$ | $\{c, b\}$ | $\{c, b\}$ | $\{c, b\}$ |
| | $a \leftarrow b \times 2$ | $\{a\}$ | $\{b\}$ | $\{c, b\}$ | $\{c, a\}$ | $\{c, b\}$ | $\{c, a\}$ |
| | if $a < N$ goto $L_1$ | $\emptyset$ | $\{a\}$ | $\{c, a\}$ | $\{c\}$ | $\{c, a\}$ | $\{c, a\}$ |
| | return $c$ | $\emptyset$ | $\{c\}$ | $\{c\}$ | $\emptyset$ | $\{c\}$ | $\emptyset$ |

## Fixpoint reached

- maximum number of live variables $= 2$
- 2 registers sufficient

# Complexity of the algorithm

## For input program of size $N$

- $\leq N$ nodes in CFG
  $\Rightarrow \leq N$ variables
  $\Rightarrow \leq N$ elements per $\mathbf{in}[n]$ and $\mathbf{out}[n]$
  $\Rightarrow O(N)$ time per set operation
- for-loop performs constant number of set operations per node
  $\Rightarrow O(N^2)$ time for the loop
- the repeat loop cannot decrease any set
  sizes of all in and out sets $\leq 2N^2$
  $\Rightarrow$ repeat loop terminates after $\leq 2N^2$ iterations

$\Rightarrow$ overall worst-case complexity $O(N^4)$

- in practice only few iterations when ordering is observed

# Least fixpoints

- Technically, the algorithm computes the least fixpoint / least solution of the inequations
- Any fixpoint/solution is a conservative approximation that tacitly assumes further uses of variables
- The least fixpoint only considers manifest uses in the CFG
- It is always safe to assume a variable is live
- It is unsafe to assume a variable is dead

# Interference

Suppose that $\mathbf{in}[n]$ and $\mathbf{out}[n]$ solve the liveness inequations.

## Interference graph

The interference graph is an undirected graph with

- nodes the variables of the CFG
- an edge $\{v, v'\}$ if exists node $n$ in the CFG such that $\{v, v'\} \subseteq \mathbf{in}[n]$

# Interference

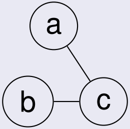
Suppose that **in**$[n]$ and **out**$[n]$ solve the liveness inequations.

## Interference graph

The interference graph is an undirected graph with

- nodes the variables of the CFG
- an edge $\{v, v'\}$ if exists node $n$ in the CFG such that $\{v, v'\} \subseteq \mathbf{in}[n]$

## Interference graph for example



# Approach to register allocation

- Find a coloring of the interference graph with *n* colors where *n* is the number of available registers
- Difficulties
  - include spilling
  - efficiency

## 2-colored interference graph for example