CS42/413

Introduction to Compilers
Radu Rugina

Lecture 23: Live Variable Analysis
26 Mar 06

Control Flow Graphs

• Control Flow Graph (CFG) = graph representation of computation and control flow in the program
  – framework to statically analyze program control-flow

• In a CFG:
  – Nodes are basic blocks; they represent computation
  – Edges characterize control flow between basic blocks

• Can build the CFG representation either from the high IR or from the low IR

From Three-Address Code to CFG

\[
\begin{align*}
\text{label L1} \\
fjump \text{ c L2} \\
x &= y + 1; \\
y &= 2 * z; \\
fjump \text{ d L3} \\
x &= y * z; \\
\text{label L3} \\
z &= 1; \\
jump \text{ L1} \\
z &= x;
\end{align*}
\]

Basic Blocks

\[
\begin{align*}
\text{if (c)} \\
x &= y + 1; \\
y &= 2 * z; \\
\text{if (d)} \\
x &= y * z; \\
\text{label L3} \\
z &= 1; \\
jump \text{ L1} \\
z &= x;
\end{align*}
\]

Using CFGs

• **Next**: use CFG representation to statically extract information about the program
  – Reason at compile-time
  – About the run-time values of variables and expressions in all program executions

• Extracted information example: live variables

• **Idea**:
  – Define program points in the CFG
  – Reason statically about how the information flows between these program points

Program Points

• **Two program points** for each instruction:
  – There is a program point before each instruction
  – There is a program point after each instruction

• In a basic block:
  – Program point after an instruction = program point before the successor instruction
### Program Points: Example

- Multiple successor CFG nodes:
  - Control will flow to one of the successor program points
  - It is not statically known which one
- Similar situation when there are multiple predecessor CFG nodes
- How does information propagate between program points?

![CFG Diagram]

### Flow of Extracted Information

- Question 1: how does information flow between the program points before and after an instruction?
- Question 2: how does information flow between different CFG nodes?
- ... in other words:
  1. Q1: what is the effect of computation?
  2. Q2: what is the effect of control flow?

![CFG Diagram]

### Using CFGs

- To extract information: reason about how it propagates between program points
- Rest of this lecture: how to use CFGs to compute information at each program point for:
  - Live variable analysis, which computes live variables are available at each program point
  - Copy propagation analysis, which computes the variable copies available at each program point

### Live Variable Analysis

- Using the CFG structure to perform live variable analysis
  - A variable is live if its value might be needed later
  - Goal: compute all live variables at each program point
- For each CFG node n, consider:
  - in[n] = live variables at program point before n
  - out[n] = live variables at program point after n
- CFG node can be either:
  - An instruction i
  - A basic block B

### How to Compute Liveness?

- Answer question 1: for each CFG node n, what is the relation between in[n] and out[n]?
- Answer question 2: for each CFG node n with successors n₁, ..., nₙ, what is the relation between out[n] and in[n₁], ..., in[nₙ]?

### Part 1: Analyze Instructions

- Question: what is the relation between in[n] and out[n] sets of live variables before and after a node?
  - in[n] = ?
  - out[n] = ?

- Examples:
  - in[n] = y
  - in[n] = x + y
  - x = x + y
  - out[n] = (x, t)

- ... is there a general rule?
Analyze Instructions

- **Yes:** knowing variables live after \( n \), we can compute variables live before \( n \):
  - All variables live after \( n \) are also live before \( n \), unless \( n \) defines (writes) them
  - All variables that \( n \) uses (reads) are also live before instruction \( n \)
- **Mathematically:**
  \[
  \text{in}[n] = (\text{out}[n] - \text{def}[n]) \cup \text{use}[n]
  \]
  where:
  - \( \text{def}[n] \) = variables defined (written) by node \( n \)
  - \( \text{use}[n] \) = variables used (read) by node \( n \)

\[\text{CS 412-613 Spring 2006 Introduction to Compilers}\]

Computing Use/Def

- Compute \( \text{use}[n] \) and \( \text{def}[n] \) for each instruction:
  - if \( n \) is \( x = y \) OP \( z \) : \( \text{use}[n] = \{ y, z \} \), \( \text{def}[n] = \{ x \} \)
  - if \( n \) is \( x = y \) OP \( y \) : \( \text{use}[n] = \{ y \} \), \( \text{def}[n] = \{ x \} \)
  - if \( n \) is \( x = y \) : \( \text{use}[n] = \{ y \} \), \( \text{def}[n] = \{ x \} \)
  - if \( n \) is if \( (a) \) : \( \text{use}[n] = \{ x \} \), \( \text{def}[n] = \{ \} \)
  - if \( n \) is return \( x \) : \( \text{use}[n] = \{ x \} \), \( \text{def}[n] = \{ \} \)
  - if \( n \) is \( x = f(y_1, ..., y_k) \) : \( \text{use}[n] = \{ y_1, ..., y_k \} \), \( \text{def}[n] = \{ x \} \)

(For now, ignore load and store instructions)

\[\text{CS 412-613 Spring 2006 Introduction to Compilers}\]

Example

- Example: three consecutive instructions
  I1, I2, I3:
  \[
  \begin{align*}
  \text{Live1} &= \text{in}[I1] \\
  \text{Live2} &= \text{out}[I1] = \text{in}[I2] \\
  \text{Live3} &= \text{out}[I2] = \text{in}[I3] \\
  \text{Live4} &= \text{out}[I3]
  \end{align*}
  \]
  \[
  \begin{align*}
  \text{I1} &= x = y + 1 \\
  \text{I2} &= y = 2z \\
  \text{I3} &= \text{if (d)}
  \end{align*}
  \]
- Relation between Live sets:
  \[
  \begin{align*}
  \text{Live1} &= (\text{Live2}-\{x\}) \cup \{y\} \\
  \text{Live2} &= (\text{Live3}-\{x\}) \cup \{z\} \\
  \text{Live3} &= (\text{Live4}-\{\} ) \cup \{d\}
  \end{align*}
  \]

\[\text{CS 412-613 Spring 2006 Introduction to Compilers}\]

Backward Flow

- **Relation:**
  \[
  \text{in}[I1] = (\text{out}[I1] - \text{def}[I1]) \cup \text{use}[I1]
  \]
- Can compute \( \text{in}[I1] \) if we know \( \text{out}[I1] \)
- The information flows backward

\[\text{CS 412-613 Spring 2006 Introduction to Compilers}\]

Part 2: Analyze Control Flow

- **Question:** for each node \( n \) with successor blocks \( n_1, ..., n_s \), what is the relation between \( \text{out}[n] \) and \( \text{in}[n_1], ..., \text{in}[n_s] \)?

- **Examples:**

- **What is the general rule?**

\[\text{CS 412-613 Spring 2006 Introduction to Compilers}\]

Analyze Control Flow

- **Rule:** A variables is live at a program point if it is live at one of the successor points
- **Characterizes all possible program executions**
  - Mathematically:
  \[
  \text{out}[n] = \bigcup_{n' \in \text{succ}(n)} \text{in}[n']
  \]
  - Again, information flows backward: from successors \( n' \) of \( n \), to \( n \) itself

\[\text{CS 412-613 Spring 2006 Introduction to Compilers}\]
Constraint System

- Put parts together: start with CFG and derive a system of constraints between live variable sets:

\[
\begin{align*}
\text{in}[n] &= (\text{out}[n] - \text{def}[n]) \cup \text{use}[n] \quad \text{for each node } n \\
\text{out}[n] &= \bigcup_{n' \in \text{succ}(n)} \text{in}[n']
\end{align*}
\]

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

Example: Live Variable Analysis

```
def = \emptyset, \text{use} = (c)
def = (x), \text{use} = (y)
def = (y), \text{use} = (z)
def = (d), \text{use} = (d)
def = (x), \text{use} = (y, z)
def = (x), \text{use} = (y, z)
def = (x), \text{use} = (x)
```

```
\text{if (c)}
\text{z} = y + 1
\text{y} = 2 + z
\text{if (d)}
\text{z} = 1
\text{z} = x
```

Copy Propagation

- Goal: determine copies available at each program point
- Information: set of copies <x=y> at each point
- For each CFG node:
  - in[n] = copies available at program point before n
  - out[n] = copies available at program point after n

Copy Propagation Analysis

- Knowing \text{in}[n], we can compute \text{out}[n] by:
  - Remove from \text{in}[n] all copies <u=v> if variable u or v is written by n
  - Keep all other copies from \text{in}[n]
  - if n is of the form x=y, add <x=y> to \text{out}[n]

- A copy is available at point before n if it is available at the end of all predecessor program points

Example: Copy Propagation

- What are the available copies at the end of the program?

```
x=y
\text{if (c)}
\text{z} = x
\text{y} = 2 + z
\text{if (d)}
\text{t} = 1
\text{u} = z + 1
\text{z} = t
```

```
x = y
\text{if (c)}
\text{z} = x
\text{y} = 2 + z
\text{if (d)}
\text{t} = 1
\text{u} = z + 1
\text{z} = t
```
Summary

• Extracting information about live variables, available copies:
  – Define the required information
  – Build constraints for instructions/control flow
  – Solve constraints to get needed information

• …is there a general framework?
  – Yes: dataflow analysis!