CS412/413

Introduction to Compilers
Radu Rugina

Lecture 19: Liveness and Copy Propagation
8 Mar 02

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

  - 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

Build CFG from High IR

while (c) {
  x = y + 1;
  y = z * z;
  if (d) x = y + z;
  z = 1;
}

Build CFG from Low IR

label L1
  fjump C2
  x = y + 1;
  y = z * z;
  fjump d3
  x = y + z;
  label L3
  Z = 1;
  jump L1
  label L2
  Z = x;

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

  Point before
  x = y + 1
  Point after

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

- Multiple successor blocks means that point at the end of a block has multiple successor program points.
- Depending on the execution, control flows from a program point to one of its successors.
- Also multiple predecessors.
- How does information propagate between program points?

\[
\begin{align*}
&x = y + z \\
&y = 2^z \\
&\text{if (d)} \\
&z = 1
\end{align*}
\]

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 successor and predecessor basic blocks.
- In other words:
  - Q1: what is the effect of instructions?
  - Q2: what is the effect of control flow?

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 live at each program point.
  - Copy propagation analysis, which computes the variable copies available at each program point.

Live Variable Analysis

- Computes live variables at each program point:
  - In, variables holding values which may be used later (in some execution of the program).
  - For an instruction I, consider:
    - \( \text{In}[I] \) = live variables at program point before I.
    - \( \text{Out}[I] \) = live variables at program point after I.
  - For a basic block B, consider:
    - \( \text{In}[B] \) = live variables at beginning of B.
    - \( \text{Out}[B] \) = live variables at end of B.
  - If I = first instruction in B, then \( \text{In}[B] = \text{In}[I] \).
  - If I = last instruction in B, then \( \text{Out}[B] = \text{Out}[I] \).

How to Compute Liveness?

- Answer question 1: For each instruction I, what is the relation between \( \text{In}[I] \) and \( \text{Out}[I] \)?
  - \( \text{In}[I] \)
  - \( \text{Out}[I] \)

- Answer question 2: For each basic block B with successor blocks \( B_1, ..., B_n \), what is the relation between \( \text{Out}[B] \) and \( \text{In}[B_1], ..., \text{In}[B_n] \)?
Analyze Instructions

- **Yes:** knowing variables live after I, can compute variables live before I: 
  - All variables live after I are also live before I, unless I defines (writes) them 
  - All variables that I uses (reads) are also live before instruction I
- **Mathematically:**
  \[ \text{in}[I] = (\text{out}[I] - \text{def}[I]) \cup \text{use}[I] \]

where:
- \( \text{def}[I] \): variables defined (written) by instruction I
- \( \text{use}[I] \): variables used (read) by instruction I

Computing Use/Def

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

(For now, ignore load and store instructions)

Example

- Example: block B with three instructions 11, 12, 13:
- Relation between live sets:
  - Live1 = \( \text{Live2}(x) \cup \{ y \} \)
  - Live2 = \( \text{Live3}(y) \cup \{ z \} \)
  - Live3 = \( \text{Live4}(x) \cup \{ y \} \)

Backward Flow

- Relation:
  \[ \text{in}[I] = (\text{out}[I] - \text{def}[I]) \cup \text{use}[I] \]

- The information flows backward!
- Instructions: can compute \( \text{in}[I] \) if we know \( \text{out}[I] \)
- Basic blocks: information about live variables flows from \( \text{out}[B] \) to \( \text{in}[B] \)

Part 2: Analyze Control Flow

- **Question:** for each basic block B with successor blocks \( B_1, ..., B_n \) what is the relation between \( \text{out}[B] \) and \( \text{in}[B_1], ..., \text{in}[B_n] \)?
- **Examples:**
  - Block B
  - \( \text{in}[B] = \emptyset \)
  - \( \text{out}[B] = \emptyset \)

- What is the general rule?

Analyze Control Flow

- **Rule:** A variables is live at end of block B if it is live at the beginning of one successor block
- Characterizes all possible program executions

- Mathematically:
  \[ \text{out}[B] = \bigcup_{B' \in \text{succ}(B)} \text{in}[B'] \]

- Again, information flows backward: from successors \( B' \) of B to basic block B
### Constraint System

- **Put parts together**: start with CFG and derive a system of constraints between live variable sets:
  
  \[
  \begin{align*}
  \text{in}[I] &= (\text{out}[I] - \text{def}[I]) \cup \text{use}[I] \\
  \text{out}[B] &= \bigcup_{b_1, b_2} \text{in}[B]
  \end{align*}
  \]
  
  for each instruction $I$ and basic block $B$.

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

### Constraint Solving Algorithm

For all instructions $\text{in}[I] = \text{out}[I] = \emptyset$

Repeat

For each instruction $I$

- $\text{in}[I] = (\text{out}[I] - \text{def}[I]) \cup \text{use}[I]$

For each basic block $B$

- $\text{out}[B] = \bigcup_{b_1, b_2} \text{in}[B]$

Until no change in live sets

### Example

```
def = (), use = (c)  

| def | use |  |  |  |
|-----|-----|  |  |  |
| (x) | (y) | x = y+1 | y = 2*z | if (d) |
| (x) | (z) | x = y*z |  | |
| (x) | (c) | z = 1 |  | |
| (x) | (x) | z = x |  | |
```

### Copy Propagation

- **Goal**: determine copies available at each program point
- **Information**: set of copies $<x=y>$ at each point

For each instruction $I$

- $\text{in}[I] = \text{copies available at program point before } I$
- $\text{out}[I] = \text{copies available at program point after } I$

For each basic block $B$

- $\text{in}[B] = \text{copies available at beginning of } B$
- $\text{out}[B] = \text{copies available at end of } B$

If $I$ is first instruction in $B$, then $\text{in}[B] = \text{in}[I]$.
If $I$ is last instruction in $B$, then $\text{out}[B] = \text{out}[I]$.

### Same Methodology

1. **Express flow of information** (i.e. available copies):
   - For points before and after each instruction $\text{in}[I], \text{out}[I]$
   - For points at exit and entry of basic blocks $\text{in}[B], \text{out}[B]$

2. **Build constraint system** using the relations between available copies

3. **Solve constraints** to determine available copies at each point in the program

### Analyze Instructions

- Knowing $\text{in}[I]$, can compute $\text{out}[I]$: $\text{out}[I] = (\text{in}[I] - \text{kill}[I]) \cup \text{gen}[I]$.

- **Mathematically**:
  
  \[
  \text{out}[I] = (\text{in}[I] - \text{kill}[I]) \cup \text{gen}[I]
  \]

  where:
  
  - $\text{kill}[I] = \text{copies "killed" by instruction } I$
  - $\text{gen}[I] = \text{copies generated by instruction } I$

CS 412412 Spring 2002 Introduction to Compilers
Computing Kill/Gen
- Compute kill[I] and gen[I] for each instruction I:
  - If 
x = y
  OP:
  z:
  gen[I] = \{
  kill[I] = (u \rightarrow u \text{ or } v \text{ if } b)
    \}
  - If 
x = OP y
  gen[I] = \{
  kill[I] = (u \rightarrow u \text{ or } v \text{ if } b)
    \}
  - If 
x = \text{addr } y
  gen[I] = \{
  kill[I] = (u \rightarrow u \text{ or } v \text{ if } b)
    \}
  - If 
x = f(y, \ldots, y_n)
  gen[I] = \{
  kill[I] = (u \rightarrow u \text{ or } v \text{ if } b)
    \}
  (again, ignore load and store instructions)

Forward Flow
- Relation:
  - \text{out}[I] = (\text{in}[I] - \text{kill[I]}) \cup \text{gen[I]}
  - The information flows forward!
- Instructions: can compute \text{out}[I] if we know \text{in}[I]
- Basic blocks: information about available copies flows from \text{in}[B] to \text{out}[B]

Analyze Control Flow
- Rule: A copy is available at end of block B if it is live at the beginning of all predecessor blocks.
- Characterizes all possible program executions
  - Mathematically:
    - \text{in}[B] = \bigcap_{B' \text{ pred(B)}} \text{out}[B']
  - Information flows forward: from predecessors \text{B'} of \text{B} to basic block \text{B}

Constraint System
- Build constraints: start with CFG and derive a system of constraints between sets of available copies:
  - \text{out}[I] = (\text{in}[I] - \text{kill[I]}) \cup \text{gen[I]}
  - \text{in}[B] = \bigcap_{B' \text{ pred(B)}} \text{out}[B']
  for each instruction I
  for each basic block B
- Solve constraints:
  - Start with empty sets of available copies
  - Iteratively apply constraints
  - Stop when we reach a fixed point

Example
- What are the available copies at the end of the program?
  - \text{x=y}
  - \text{z=t}
  - \text{x=z}

Summary
- Extracting information about live variables and available copies is similar:
  - Define the required information
  - Define information before/after instructions
  - Define information at entry/exit of blocks
  - Build constraints for instructions/control flow
  - Solve constraints to get needed information
- Is there a general framework?
  - Yes: dataflow analysis