CS443: Compiler Construction

Lecture 14: Dataflow Analysis
Stefan Muller
Based on material by Steve Zdancewic
Dataflow algorithm can be used for more than just liveness analysis

- Reaching definitions analysis
- Available expressions analysis
- Alias Analysis
- Constant Propagation
Generalized dataflow analysis: produce a set of “facts” in and out of each node

• Every statement (node):
  • Produces (generates) some set of facts
  • Eliminates (kills) some set of facts

• Constraints at each node computed from other nodes based on constraints (somewhat) specific to the analysis
Dataflow analysis in 4 steps

1. Define facts, gen, kill
2. Define constraints
3. Convert constraints to equations
   • Sets should increase or decrease monotonically
4. Initialize facts for each node
   • Initial value should be consistent with whether sets are increasing or decreasing
Liveness analysis as a dataflow analysis (Steps 1-2)

• Facts: Live variables
  • $\text{gen}[n] = \text{use}[n]$
  • $\text{kill}[n] = \text{def}[n]$

• Constraints:
  • $\text{in}[n] \supseteq \text{gen}[n]$
  • $\text{out}[n] \supseteq \text{in}[n']$ if $n' \in \text{succ}[n]$
  • $\text{in}[n] \supseteq \text{out}[n] / \text{kill}[n]$
Liveness analysis as a dataflow analysis (Steps 3-4)

• Equations:
  • \( \text{out}[n] := \bigcup_{n' \in \text{succ}[n]} \text{in}[n'] \)
  • \( \text{in}[n] := \text{gen}[n] \cup (\text{out}[n] / \text{kill}[n]) \)

• Initial values:
  • \( \text{out}[n] := \emptyset \)
  • \( \text{in}[n] := \emptyset \)
Dataflow algorithm can be used for more than just liveness analysis

- Reaching definitions analysis
- Available expressions analysis
- Alias Analysis
- Constant Propagation
Recall from last time: a variable might be live for a long time, but with different definitions.

\[
v = a \\
v = b \\
v = c \\
\text{return } v
\]
Reaching definitions:
What *definitions* of a var might reach a node?

```
b = a + 2
```

```
c = b * b
```

```
b = c + 1
```

```
return b * a
```

out[1]: {1}
in[2]:   {1}

out[2]: {1,2}
in[3]:   {1,2}

out[3]: {2,3}
in[4]:   {2,3}

2 still reaches even though c not live
Reaching definitions:
What *definitions* of a var might reach a node?

```c
if (a > 2)
c = b * b
out[2]: {2}
in[2]: {}

b = c + 1
out[3]: {3}
in[3]: {}

return b * a
in[4]: {2,3}
out[4]: {}
```
Reaching definitions as a dataflow analysis (Step 1)

• Facts: set of nodes whose definition of a variable reaches n
• Let $\text{defs}[a]$ be the set of nodes that define the variable $a$

<table>
<thead>
<tr>
<th>n</th>
<th>gen[n]</th>
<th>kill[n]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a = b \text{ op } c$</td>
<td>{n}</td>
<td>$\text{defs}[a] - {n}$</td>
</tr>
<tr>
<td>$a = \text{ load } b$</td>
<td>{n}</td>
<td>$\text{defs}[a] - {n}$</td>
</tr>
<tr>
<td>store $b$, $a$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$a = f(b_1,\ldots,b_n)$</td>
<td>{n}</td>
<td>$\text{defs}[a] - {n}$</td>
</tr>
<tr>
<td>$f(b_1,\ldots,b_n)$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>br L</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>br a L1 L2</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>return $a$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>
Reaching definitions as a dataflow analysis (Step 2)

• $\text{out}[n] \supseteq \text{gen}[n]$

• $\text{in}[n] \supseteq \text{out}[n']$ if $n'$ is in $\text{pred}[n]$

• $\text{out}[n] \cup \text{kill}[n] \supseteq \text{in}[n]$
  
  • Equivalently: $\text{out}[n] \supseteq \text{in}[n] / \text{kill}[n]$
Reaching definitions as a dataflow analysis (Steps 3-4)

• \( \text{in}[n] := \bigcup_{n' \in \text{pred}[n]} \text{out}[n'] \)

• \( \text{out}[n] := \text{gen}[n] \cup (\text{in}[n] / \text{kill}[n]) \)

• Algorithm: initialize \( \text{in}[n] \) and \( \text{out}[n] \) to \( \emptyset \)
Dataflow algorithm can be used for more than just liveness analysis

• Reaching definitions analysis
• Available expressions analysis
• Alias Analysis
• Constant Propagation
When is this optimization safe?

• $a = x + 1$  $\Rightarrow$  $a = x + 1$

  ...  $\Rightarrow$  ...

• $b = x + 1$  $\Rightarrow$  $b = a$

• Available expressions: nodes whose definitions are “available”
Available $\neq$ Live

\[
\begin{align*}
    a &= x + 1 \\
    c &= a \\
    b &= x + 1 \\
    d &= b \times 2 \\
    \text{return } d - c
\end{align*}
\]
### Available expressions as a dataflow analysis (Step 1)

<table>
<thead>
<tr>
<th>n:</th>
<th>gen[{n}]</th>
<th>kill[{n}]</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = b op c</td>
<td>{n}</td>
<td>uses[a]</td>
</tr>
<tr>
<td>a = load b</td>
<td>{n}</td>
<td>uses[a]</td>
</tr>
<tr>
<td>store b, a</td>
<td>\ø</td>
<td>uses[*x] (for all x that may equal a)</td>
</tr>
<tr>
<td>br L</td>
<td>\ø</td>
<td>\ø</td>
</tr>
<tr>
<td>br a L1 L2</td>
<td>\ø</td>
<td>\ø</td>
</tr>
<tr>
<td>a = f(b₁,...,bₙ)</td>
<td>\ø</td>
<td>uses[a]U uses[*x] (for all x)</td>
</tr>
<tr>
<td>f(b₁,...,bₙ)</td>
<td>\ø</td>
<td>uses[*x] (for all x)</td>
</tr>
<tr>
<td>return a</td>
<td>\ø</td>
<td>\ø</td>
</tr>
</tbody>
</table>
Available expressions as a dataflow analysis (Steps 2-3)

• $\text{out}[n] \supseteq \text{gen}[n]$

• $\text{in}[n] \subseteq \text{out}[n']$ if $n'$ is in $\text{pred}[n]$

• $\text{out}[n] \cup \text{kill}[n] \supseteq \text{in}[n]$
  • Equivalently: $\text{out}[n] \supseteq \text{in}[n] / \text{kill}[n]$

• $\text{in}[n] := \bigcap_{n' \in \text{pred}[n]} \text{out}[n']$

• $\text{out}[n] := \text{gen}[n] \cup (\text{in}[n] / \text{kill}[n])$
Available expressions as a dataflow analysis (Steps 3-4)

- \( \text{in}[n] := \bigcap_{n' \in \text{pred}[n]} \text{out}[n'] \)
- \( \text{out}[n] := \text{gen}[n] \cup (\text{in}[n] / \text{kill}[n]) \)

- Initialize \( \text{in}[n] \) and \( \text{out}[n] \) to \{set of all nodes\}
  - Iterate the update equations until a fixed point is reached

- The algorithm terminates because \( \text{in}[n] \) and \( \text{out}[n] \) decrease monotonically
  - At most to a minimum of the empty set

- The algorithm is precise because it finds the largest sets that satisfy the constraints.
Contrasting RD/AE

Reaching Defs

\[
\begin{align*}
in[n] & := \bigcup_{n' \in \text{pred}[n]} \text{out}[n'] \\
\text{out}[n] & := \text{gen}[n] \cup (\text{in}[n] / \text{kill}[n])
\end{align*}
\]

Which definitions *may* reach \( n \)?

Initialize to \( \emptyset \)

“May” analysis

Available Expressions

\[
\begin{align*}
in[n] & := \bigcap_{n' \in \text{pred}[n]} \text{out}[n'] \\
\text{out}[n] & := \text{gen}[n] \cup (\text{in}[n] / \text{kill}[n])
\end{align*}
\]

Which expressions *must* reach \( n \)?

Initialize to all expressions

“Must” analysis
Contrasting RD/Liveness

<table>
<thead>
<tr>
<th>Reaching Defs</th>
<th>Liveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>in[n] := ( \bigcup_{n' \in \text{pred}[n]} \text{out}[n'] )</td>
<td>out[n] := ( \bigcup_{n' \in \text{succ}[n]} \text{in}[n'] )</td>
</tr>
</tbody>
</table>

Propagate information **forward**  Propagate information **backward**

Forward analysis  Backward analysis