CSE 504: Compiler Design

Data Flow Analysis

Pradipta De
pradipta.de@sunykorea.ac.kr
Current Topic

- Iterative Data Flow Analysis
- LiveOut sets
- Static Single Assignment (SSA) Form
Data Flow Analysis

• Techniques to reason about runtime flow of values in the program
  – Uses static analysis (or compile time reasoning) of the code to infer runtime behavior

• Useful in global optimization

• Data Flow Analysis techniques solves a set of simultaneous equations defined over sets associated with the nodes and edges of a CFG
In a flow graph with entry node $b_0$, node $b_i$ dominates node $b_j$, written $b_i \gg b_j$, if and only if $b_i$ lies on every path from $b_0$ to $b_j$. By definition, $b_i \gg b_i$

Nodes $B_0$, $B_1$, $B_5$, $B_6$ lie on every path from $B_0$ to $B_6 \Rightarrow \text{DOM}(B_6)$ is \{B_0, B_1, B_5, B_6\}

<table>
<thead>
<tr>
<th></th>
<th>$B_0$</th>
<th>$B_1$</th>
<th>$B_2$</th>
<th>$B_3$</th>
<th>$B_4$</th>
<th>$B_5$</th>
<th>$B_6$</th>
<th>$B_7$</th>
<th>$B_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{DOM}(n)</td>
<td>{0}</td>
<td>{0,1}</td>
<td>{0,1,2}</td>
<td>{0,1,3}</td>
<td>{0,1,3,4}</td>
<td>{0,1,5}</td>
<td>{0,1,5,6}</td>
<td>{0,1,5,7}</td>
<td>{0,1,5,8}</td>
</tr>
</tbody>
</table>
How to compute DOM sets

Solve the equation: \[ \text{Dom}(n) = \{n\} \cup \left( \bigcap_{m \in \text{preds}(n)} \text{Dom}(m) \right) \]

**Initial Condition:** \( \text{Dom}(n_0) = \{n_0\} \), and \( \text{Dom}(n) = \{N\} \), where \( n \) not = \( n_0 \)

\( \text{Dom}(n) \) is computed as a function of its predecessors \( \Rightarrow \) forward data flow problem

**Computing Dom Sets:**
1. Build a CFG (Control Flow Graph)
2. Gather initial information for each block
   1. Only information required is the block id, e.g. id i for a block Bi
3. Solve the equations to produce the DOM sets for each block
### Iterative Solver for DOM sets

Traverse the nodes in ascending order of their id, compute DOM iteratively till there is no change.

```plaintext
n ← |N| - 1
Dom(0) ← {0}
for i ← 1 to n
    Dom(i) ← N
changed ← true
while (changed)
    changed ← false
    for i ← 1 to n
        temp ← \{i\} ∪ (∩\_j∈\text{preds}(i) Dom(j))
        if temp ≠ Dom(i) then
            Dom(i) ← temp
            changed ← true
```

<table>
<thead>
<tr>
<th></th>
<th>B₀</th>
<th>B₁</th>
<th>B₂</th>
<th>B₃</th>
<th>B₄</th>
<th>B₅</th>
<th>B₆</th>
<th>B₇</th>
<th>B₈</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>{0}</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>1</td>
<td>{0}</td>
<td>{0,1}</td>
<td>{0,1,2}</td>
<td>{0,1,2,3}</td>
<td>{0,1,2,3,4}</td>
<td>{0,1,5}</td>
<td>{0,1,5,6}</td>
<td>{0,1,5,6,7}</td>
<td>{0,1,5,8}</td>
</tr>
<tr>
<td>2</td>
<td>{0}</td>
<td>{0,1}</td>
<td>{0,1,2}</td>
<td>{0,1,3}</td>
<td>{0,1,3,4}</td>
<td>{0,1,5}</td>
<td>{0,1,5,6}</td>
<td>{0,1,5,7}</td>
<td>{0,1,5,8}</td>
</tr>
<tr>
<td>3</td>
<td>{0}</td>
<td>{0,1}</td>
<td>{0,1,2}</td>
<td>{0,1,3}</td>
<td>{0,1,3,4}</td>
<td>{0,1,5}</td>
<td>{0,1,5,6}</td>
<td>{0,1,5,7}</td>
<td>{0,1,5,8}</td>
</tr>
</tbody>
</table>
```
Algorithm Property

- Does the algorithm always terminate?
  - Iterative calculation of the Dom sets halts because the sets that approximate Dom shrink monotonically throughout the computation

- Does it compute the correct DOM sets?
  - Forms a fixed point solution to the dominance equations
    - Can be proved that fixed point exists for each equation and it is unique

- Does the order of traversal in the algorithm matter?
  - Depending on the order of traversal, the algorithm can take less number of iterations
Liveness Analysis

• Finding potential use of uninitialized variables can be found using liveness analysis
  – A variable \( v \) is live at point \( p \) if and only if there exists a path in the CFG from \( p \) to a use of \( v \) along which \( v \) is not redefined

• Liveout(b): encode live information by computing all variables that are live on exit from a block \( b \)
  – For each node \( n \) in the CFG, Liveout(\( n \)) is defined by an equation that uses Liveout sets of \( n \)'s successors in the CFG, and two sets UEVar(\( n \)) and VarKill(\( n \))
Liveout Equation

\[ \text{LiveOut}(n) = \bigcup_{m \in \text{succ}(n)} (\text{UEVar}(m) \cup (\text{LiveOut}(m) \cap \overline{\text{VarKill}(m)})) \]

UEVar(m) contains variables used in m before any redefinition in m → upward exposed variables

VarKill(m) contains all variables that are defined in m → the equation contains the complement of the set, i.e. all variables that are NOT defined in m

Liveout(n) is union of those variables that are live at the head of some block m, that immediately follows n, in the CFG → backward data flow problem

Three steps:
1. Build a CFG
2. Gather initial information
   1. Compute UEVar and VarKill for each block b
3. Solve the Liveout equations for each block b
Computing UEVar and VarKill

(a) Example Control-Flow Graph

(b) Initial Information

<table>
<thead>
<tr>
<th>Iteration</th>
<th>UEVar</th>
<th>VarKill</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>$\emptyset$</td>
<td>${i}$</td>
</tr>
<tr>
<td>1</td>
<td>${i}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>2</td>
<td>${s,i}$</td>
<td>${s,i}$</td>
</tr>
<tr>
<td>3</td>
<td>${s,i}$</td>
<td>${s,i}$</td>
</tr>
</tbody>
</table>

(c) Progress of the Solution

CSE 504: Compiler Design
Finding Uninitialized Variables

• If \( v \in \text{Liveout}(n0) \), then there is a path from \( n0 \) to a use of \( v \) along which \( v \) is not defined
  \( \rightarrow \) Uninitialized variable \( v \)

• May lead to false positives \( \rightarrow \) identify variables as uninitialized although they are initialized
Limitations

• If v is accessible through another name and initialized through that name, live analysis cannot connect the two names

• If v exists before the current procedure is invoked, then it may have been initialized before
  – Static variables within current scope
  – Variables declared outside current scope

• Static analysis can discover a path from procedure entry to use of v, but the path never used at runtime
  – In C, variables are initialized by default
Use of Live variables

• Useful in global register allocation
  – Need not keep value in register if not live

• Compiler can use live variable information to discover useless store operations
  – If v is not live, no need to store v back to memory