Program Analysis

- The compiler needs to understand properties of a program (e.g. the set of variables “live” at a program point).
- This information should be computed at compile time, with incomplete information on the values the program computes, and without executing the program itself!
- This information is likely to be approximate: in general, at compile time, we will not know which sequence of instructions will be executed.
- Data-Flow Analysis is a standard way to formulate intra-procedural program analysis.
Control Flow Graphs

When we try to deduce properties of a procedure, we first build a control flow graph (CFG).

- Nodes of a CFG are Basic Blocks.
- Edges indicate which blocks can follow which other blocks.

A Basic Block is a sequence of instructions such that:
- There are no jumps/branches in the sequence except as the last instruction.
- For all jumps/branches in the program, the target is the first instruction in some basic block.
  - In other words, no jump lands in the middle of a basic block.

Example of CFGs

- Branches only at the end of a block.
- Branch destinations only at beginning of a block.
Live Variables

Consider the problem of finding the set of live variables at some program point.

- A variable is live after a statement $s$ in the program, if it is used in a statement $s'$,
- and there is a control flow path from $s$ to $s'$.

Example:
1. $i = 1$
2. $j = 1$
3. $t1 = 10 \times i$
4. $t2 = t1 + j$
5. $t3 = 4 \times t2$
6. $a[t3] = 0$
7. $j = j + 1$

- Variable $t3$ is live after statement 5 since it is used in statement 6.
- Variable $j$ is also live after statement 5 since it is used in statement 7.

Live Variable Analysis — (1)

- Let $\text{def}(s)$ be the set of all variables defined by statement $s$ (e.g. the lhs variable in an assignment statement).
- Let $\text{use}(s)$ be the set of all variables used by statement $s$ (e.g. the variables on the rhs of an assignment statement).
- $\text{succ}(s)$: the set of statements that immediately follow statement $s$.
- The above definitions for $\text{def}$, $\text{use}$, and $\text{succ}$ can be extended for whole blocks as well.
  - $\text{def}(B)$: set of variables defined in block $b$.
  - $\text{use}(B)$: set of variables used, but not defined earlier, in block $b$.
  - $\text{succ}(B)$: set of blocks that immediately succeed block $B$. 
Live Variables

Live Variable Analysis — (2)

Block | Succ | Def | Use  
--- | --- | --- | --- 
1 | \{2\} | \{i\} | {} 
2 | \{3\} | \{j\} | {} 
3 | \{3,4\} | \{t1, t2, t3, j\} | \{a,i, j\} 
4 | \{2,5\} | \{i\} | {} 
5 | \{6\} | \{i\} | {} 
6 | \{6,Exit\} | \{t4,t5,i\} | \{a,i\} 

Live Variable Analysis — (3)

- **Out(\(s\))**: the set of variables live just after statement \(s\).
- **In(\(s\))**: the set of variables live just before statement \(s\).
- The above definitions for Out and In can be readily extended for blocks.
- Observe that:
  - If a variable is used by a statement, then it must be live before the statement.
  - If a variable is live immediately after a statement, then it must be live before the statement as well, unless it is defined by the statement.
  - For a statement \(s\), if a variable is live before any of its successors, then it must be live after \(s\).
  - From these observations, we get:
    
    \[
    \text{In}(s) = \text{use}(s) \cup (\text{Out}(s) - \text{def}(s)) \\
    \text{Out}(s) = \bigcup_{t \in \text{succ}(s)} \text{In}(t)
    \]
Live Variable Analysis — (4)

\[ \text{In}(s) = \text{use}(s) \cup (\text{Out}(s) - \text{def}(s)) \]
\[ \text{Out}(s) = \bigcup_{t \in \text{succ}(s)} \text{In}(t) \]

Let \( a \) be a variable that is needed after the procedure exits (e.g. it is a global variable). Then, \( \text{In}(\text{Exit}) = \{a\} \).

<table>
<thead>
<tr>
<th>Block</th>
<th>Succ</th>
<th>Def</th>
<th>Use</th>
<th>In</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{2}</td>
<td>{i}</td>
<td>{}</td>
<td>\text{Out}(1) - {i}</td>
<td>\text{In}(2)</td>
</tr>
<tr>
<td>2</td>
<td>{3}</td>
<td>{}</td>
<td>{}</td>
<td>\text{Out}(2) - {j}</td>
<td>\text{In}(3)</td>
</tr>
<tr>
<td>3</td>
<td>{3,4}</td>
<td>{t1, t2, t3, j}</td>
<td>{a, i, j}</td>
<td>{a, i, j} \cup \text{Out}(3) - {t1, t2, t3, j}</td>
<td>\text{In}(3) \cup \text{In}(4)</td>
</tr>
<tr>
<td>4</td>
<td>{2,5}</td>
<td>{i}</td>
<td>{}</td>
<td>{i} \cup \text{Out}(4) - {i}</td>
<td>\text{In}(2) \cup \text{In}(5)</td>
</tr>
<tr>
<td>5</td>
<td>{6}</td>
<td>{i}</td>
<td>{}</td>
<td>\text{Out}(5) - {i}</td>
<td>\text{In}(6)</td>
</tr>
<tr>
<td>6</td>
<td>{6, Exit}</td>
<td>{t4, t5, i}</td>
<td>{a, i}</td>
<td>{a, i} \cup \text{Out}(6) - {t4, t5, i}</td>
<td>\text{In}(6) \cup \text{In}(\text{Exit})</td>
</tr>
<tr>
<td>Exit</td>
<td>{}</td>
<td>{}</td>
<td>{}</td>
<td>{}</td>
<td>{}</td>
</tr>
</tbody>
</table>

Live Variable Analysis — (5)

- The equations for \( \text{In} \) and \( \text{Out} \) form a set of simultaneous set equations.
- For this analysis, we require the least solution to these equations.
- Consider the equations relating \( \text{In}(6) \), \( \text{Out}(6) \) and \( \text{In}(\text{Exit}) \):

\[ \text{In}(6) = \{a, i\} \cup \text{Out}(6) - \{t4, t5, i\} \]
\[ \text{Out}(6) = \text{In}(6) \cup \text{In}(\text{Exit}) \]
\[ \text{In}(\text{Exit}) = \{a\} \]

- There are many solutions to these equations:
  1. \( \text{In}(6) = \text{Out}(6) = \{a, i\} \), and \( \text{In}(\text{Exit}) = \{a\} \).
  2. \( \text{In}(6) = \text{Out}(6) = \{a, i, t3\} \), and \( \text{In}(\text{Exit}) = \{a\} \).
  3. ...
- Of these, (1) is the least. In fact, it can be shown that every solution will contain (1).
Data Flow Equations

Solutions to Data Flow Equations

- Data flow analysis is formulated in terms of finding the least (or sometimes, the greatest) solution to a set of simultaneous equations.
- The flow equations can be written as $X = F(X)$, where $X$ is a vector of $In$'s and $Out$'s.
- Solutions $X$ such that $X = F(X)$ are fixed points of $F$.
- The smallest $X$ such that $X = F(X)$ is called the least fixed point of $F$.

Partial Orders

- Let $U$ be a finite set, and let $D = P(U)$, i.e. the powerset of $U$. Let $D^n = D \times D \times \cdots \times D$, i.e., an $n$-dimensional cartesian space over $P(U)$.
- We can define partial order among vectors of sets such that $X \subseteq X'$ if, and only if, for all components of the vector, $X_i \subseteq X'_i$.
  - It is easy to verify that “$\subseteq$” is a partial order: it is reflexive, transitive and anti-symmetric.
- Let $\perp$ be a $n$-vector of empty sets. Clearly, $\perp \subseteq X$ for all $X \in D^n$.
- Let $\top$ be a $n$-vector of $U$. Observe that $X \subseteq \top$ for all $X \in D^n$.
- $(D^n, \subseteq)$ is a complete lattice with $\perp$ as the least element and $\top$ as the greatest element.
- Vectors $X^{(0)}, X^{(1)}, \ldots, X^{(i)}$ is called a chain if $X^{(0)} \subseteq X^{(1)} \subseteq \cdots \subseteq X^{(i)}$.
- Note all chains in $(D^n, \subseteq)$ are finite, since $U$ is finite.
Monotone Functions

- Let \( F : D^n \rightarrow D^n \) (i.e. a function from \( D^n \) to \( D^n \)).
- A function \( F \) is monotone over partial order “\( \sqsubseteq \)" if, for every \( \overline{X} \) and \( \overline{X}' \) such that \( \overline{X} \sqsubseteq \overline{X}' \), we have \( F(\overline{X}) \sqsubseteq F(\overline{X}') \).
  - Note the definition of monotonicity. It says the function returns smaller values if it is given smaller argument values.
  - It is not necessary that the returned values must be smaller than the argument values!
- It is easy to see that the flow equations for live variable analysis defines a monotone function.
- There is a simple way to show the existence of fixed points, and to compute the Least/Greatest Fixed Points of a monotone function.
- Tarski-Knaster Theorem: Given a complete lattice \( L \) and a monotone function \( G : L \rightarrow L \), the fixed points of \( G \) form a complete lattice. Consequently, there exist both least and greatest fixed points.

Kleene’s Fixed Point Theorem:

- Construct a sequence \( \overline{X}^{(0)}, \overline{X}^{(1)}, \ldots, \overline{X}^{(i)}, \ldots \), where \( \overline{X}^{(0)} = \bot \) and \( \overline{X}^{(i+1)} = F(\overline{X}^{(i)}) \).
- This sequence forms a chain.
  - \( \overline{X}^{(0)} = \bot \sqsubseteq \overline{X}^{(1)} \).
  - If \( \overline{X}^{(i)} \sqsubseteq \overline{X}^{(i+1)} \), then \( \overline{X}^{(i+1)} \sqsubseteq \overline{X}^{(i+2)} \).
    - \( \overline{X}^{(i+1)} = F(\overline{X}^{(i)}) \)
    - Since \( \overline{X}^{(i)} \sqsubseteq \overline{X}^{(i+1)} \), by monotonicity of \( F \), \( F(\overline{X}^{(i)}) \sqsubseteq F(\overline{X}^{(i+1)}) \).
    - \( \overline{X}^{(i+2)} = F(\overline{X}^{(i+1)}) \)
- Since all chains over “\( \sqsubseteq \)” are finite, consider the last element of the chain \( \overline{X}^{(n)} \).
  - \( \overline{X}^{(n)} = F(\overline{X}^{(n)}) \), otherwise it is not the last element.
  - So, \( \overline{X}^{(n)} \) is a fixed point of \( F \).
Computing Least Fixed Point —(2)

- Consider the sequence $X^{(0)}, X^{(1)}, \ldots, X^{(i)}, \ldots, X^{(n)}$, where $X^0 = \bot$ and $X^{(i+1)} = F(X^{(i)})$.
- $X^{(n)}$ is the least fixed point of $F$.
  - We already know that $X^{(n)}$ is a fixed point of $F$.
  - Let $Y$ be any fixed point of $F$.
  - Clearly, $X^{(0)} = \bot \subseteq Y$.
  - If $X^{(i)} \subseteq Y$, since $F$ is monotone, $X^{(i+1)} = F(X^{(i)}) \subseteq F(Y) = Y$ (since $Y$ is a fixed point).
  - Hence, by induction, for all elements of the chain $X^{(i)} \subseteq Y$.
  - In particular, $X^{(n)} \subseteq Y$, is at least as small as any fixed point $Y$ of $F$, and hence is the least fixed point.

Computing the Greatest Fixed Point

- Consider the sequence $X^{(0)}, X^{(1)}, \ldots, X^{(i)}, \ldots, X^{(n)}$, where $X^0 = \top$ and $X^{(i+1)} = F(X^{(i)})$.
- Note the starting point of this sequence: the greatest element in the lattice.
- By an argument similar to the one we used for the least fixed point, $X^{(n)}$ can be shown to be the greatest fixed point of $F$. 
### Live Variable Analysis Revisited

<table>
<thead>
<tr>
<th>Set</th>
<th>Eqn</th>
</tr>
</thead>
<tbody>
<tr>
<td>In(1)</td>
<td>Out(1)−{i}</td>
</tr>
<tr>
<td>Out(1)</td>
<td>In(2)</td>
</tr>
<tr>
<td>In(2)</td>
<td>Out(2)−{j}</td>
</tr>
<tr>
<td>Out(2)</td>
<td>In(3)</td>
</tr>
<tr>
<td>In(3)</td>
<td>Out(3)−{t1,t2,t3,j} \cup Out(3)−{i}</td>
</tr>
<tr>
<td>Out(3)</td>
<td>In(3) ∪ In(4)</td>
</tr>
<tr>
<td>In(4)</td>
<td>Out(4)−{i}</td>
</tr>
<tr>
<td>Out(4)</td>
<td>In(2) ∪ In(5)</td>
</tr>
<tr>
<td>In(5)</td>
<td>Out(5)−{i}</td>
</tr>
<tr>
<td>Out(5)</td>
<td>In(6)</td>
</tr>
<tr>
<td>In(6)</td>
<td>Out(6)−{t4,t5,i} \cup Out(6)−{i}</td>
</tr>
<tr>
<td>Out(6)</td>
<td>In(6) ∪ In(Exit)</td>
</tr>
<tr>
<td>In(Exit)</td>
<td>{a}</td>
</tr>
</tbody>
</table>

### Reaching Definitions

- An assignment of the form \(x = e\) for some expression \(e\) is said to define \(x\).
- A definition at statement \(s_1\) reaches another statement \(s_2\) if:
  - there is some control flow path from \(s_1\) to \(s_2\), such that
  - there is no other definition of \(x\) on the path from \(s_1\) to \(s_2\).
- Let \(\text{In}(s)\) be the set of all definitions that reach \(s\).
- Let \(\text{Out}(s)\) be the set of all definitions that reach all the immediate successors of \(s\).
- Then \(\text{Out}(s) = \text{gen}(s) \cup (\text{In}(s) − \text{kill}(s))\), where
  - \(\text{gen}(s)\) is the set of definitions generated by \(s\), and
  - \(\text{kill}(s)\) is the set of definitions with the same lhs variables as those in \(s\).
- \(\text{In}(s) = \bigcup_{t \in \text{pred}(s)} \text{Out}(t)\)
Reaching Definitions vs. Live Variables

- **Live Variables:** \( In \) and \( Out \) are the smallest sets such that
  \[
  In(s) = use(s) \cup (Out(s) - def(s))
  \]
  \[
  Out(s) = \bigcup_{t \in succ(s)} In(t)
  \]

- **Reaching Definitions:** \( In \) and \( Out \) are the smallest sets such that
  \[
  In(s) = \bigcup_{t \in pred(s)} Out(t)
  \]
  \[
  Out(s) = gen(s) \cup (In(s) - kill(s))
  \]

The form of equations is identical, and they can be computed using the same procedure, except:

- Live Variables are best computed backwards through the flow graph (information goes from successors to predecessors).
- Reaching Definitions are best computed forwards through the flow graph (information goes from predecessors to successors).

Available Expressions

- An expression \( e \) is **available** at statement \( s \) if, for **every path** that reaches \( s \), there is **some** statement \( s' \) where \( e \) is evaluated.
- Let \( In(s) \) be the set of all expressions available immediately before \( s \) is evaluated.
- Let \( Out(s) \) be the set of all expressions available immediately after \( s \) is evaluated.
- Then \( Out(s) = gen(s) \cup (In(s) - kill(s)) \), where
  - \( gen(s) \) is the set of all expressions evaluated in \( s \), and
  - \( kill(s) \) is the set of all expressions that use the lhs variables defined in \( s \).
- \( In(s) = \bigcap_{t \in pred(s)} Out(t) \)
- \( In \) and \( Out \) are the **greatest sets** that satisfy the above equations.