CSE443
Compilers

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall
# Semester plan

(probably wildly optimistic)

<table>
<thead>
<tr>
<th></th>
<th>M</th>
<th>W</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROS</td>
<td></td>
<td>9.2 Data-flow analysis</td>
<td>9.3 Data-flow foundations</td>
</tr>
<tr>
<td>9.1 Overview</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9.4 Constant propagation</td>
<td>9.4 Constant propagation</td>
<td>9.5 Redundancy elimination</td>
<td></td>
</tr>
<tr>
<td>9.6 Loops in flow graphs</td>
<td>9.7 Region-based analysis</td>
<td>9.8 Symbolic analysis</td>
<td></td>
</tr>
</tbody>
</table>

Kris Schindler Architecture talk
Phases of a compiler

Figure 1.6, page 5 of text

Optimizations
Data-flow analysis

- View program execution as a sequence of state transformations.
- Each program state consists of all the variables in the program along with their current values.
State transformation

input state

intermediate instruction

output state
Program states are called program points.

A sequence of program points are called a path.
Data-flow analysis

- Begin by considering only the flow graph for a single function.
Properties

- Within a basic block:
  - Program point after a statement is same as program point before the next statement.
  - Why?
Properties

- Between basic blocks:
  
  "If there is an edge from block B1 to block B2, then the program point after the last statement of B1 may be followed immediately by the program point before the first statement of B2."

[p. 597]
An execution path (or just path) from point $p_1$ to point $p_n$ [is] a sequence of points $p_1, p_2, \ldots, p_n$ such that for each $i = 1, 2, \ldots, n-1$, either

1. $p_i$ is the point immediately preceding a statement and $p_{i+1}$ is the point immediately following that same statement, or

2. $p_i$ is the end of some block and $p_{i+1}$ is the beginning of a successor block.
Example 9.8 (p. 598)

```
Example 9.8 (p. 598)

(1)
\[ d1: a = 1 \]

(2)

(3)
\[ \text{if read()} \leq 0 \text{ goto } B4 \]

(4)

(5)
\[ d2: b = a \]
\[ d3: a = 243 \]
\[ \text{goto } B2 \]

(6)

(7)

(8)

(9)
```

---

B1

Path: (1,2,3,4,9)

Path: (1,2,3,4,5,6,7,8,3,4,9)

B2

a has value 1 first time (5) is executed.

d1 reaches (5) on the first iteration.

B3

a has value 243 at (5) on the second and subsequent iterations.

d3 reaches (5) on those iterations.

B4
Reaching definitions

"The definitions that may reach a program point along some path are known as reaching definitions."

[p. 598]
Gathering different data

"... at point (5) ... the value of a is one of \{ 1 , 243 \} and ... it may be defined by one of \{ d1 , d3 \}.

[p. 598]

"... at point (5) ... there is no definition that must be the definition of a at that point, so this set is empty for a at point (5). Even if a variable has a unique definition at a point, that definition must assign a constant to the variable. Thus, we may simply describe certain variables as 'not a constant', instead of collecting all their possible values or all their possible definitions."

[p. 599]
9.2.2 Data-flow analysis schema

"...associate with every program point a data-flow value that represents an abstraction of the set of all possible program states that can be observed at that point." [p. 599]

"The set of possible data-flow values is the domain..." [p. 599]

"We denote the data-flow values before and after each statement s by IN[s] and OUT[s], respectively." [p. 599]
"The data-flow problem is to find a solution to a set of constraints on the IN[s]'s and OUT[s]'s, for all statements s. There are two sets of constraints: those based on the semantics of the statements ("transfer functions") and those based on the flow of control." [p. 599]
Transfer functions

Information can flow forwards or backwards.

Forward flow: $\text{OUT}[s] = f_s \left( \text{IN}[s] \right)$

Backward flow: $\text{IN}[s] = g_s \left( \text{OUT}[s] \right)$
Control flow constraints

In a sequence $s_1, s_2, \ldots, s_n$ without jumps, $\text{IN}[s_{i+1}] = \text{OUT}[s_i]$ for all $i=1,2,\ldots,n-1$.

For data-flow between blocks, take "the union of the definitions after last statements of each of the predecessor blocks." [p. 600]
9.2.3 Data-flow schemas on basic blocks

Suppose a basic block $B$ consists of the sequence of statements $s_1, s_2, ..., s_n$. Define $\text{IN}[B] = \text{IN}[s_1]$ and $\text{OUT}[B] = \text{OUT}[s_n]$.

The transfer function of $B$:

$$f_B = f_{s_n} \circ ... \circ f_{s_2} \circ f_{s_1}$$

The transfer function of $B$:

$$\text{OUT}[B] = f_B(\text{IN}[B])$$
9.2.3 Data-flow schemas on basic blocks

**Forward flow problem**

\[ \text{IN}[B] = \bigcup_{P \text{ a predecessor of } B} \text{OUT}[P] \]

**Backward flow problem**

\[ \text{IN}[B] = g_B \left( \text{OUT}[B] \right) \]

\[ \text{OUT}[B] = \bigcup_{S \text{ a successor of } B} \text{IN}[S] \]
9.2.3 Data-flow schemas on basic blocks

"...data-flow equations usually do not have a unique solution. Our goal is to find the most 'precise' solution that satisfies the two sets of constraints: control-flow and transfer constraints. That is, we need a solution that encourages valid code improvements, but does not justify unsafe transformations..."
9.2.4 Reaching definitions

“A definition d reaches a point p if there is a path from the point immediately following d to p, such that d is not ‘killed’ along that path.” [p. 601]

“We kill a definition of a variable x if there is any other definition of x anywhere along the path.” [p. 601]
9.2.4 Reaching definitions

"A definition of a variable x is a statement that assigns, or may assign, a value to x."

What is meant by "may assign"?
9.2.4 Reaching definitions

"Procedure parameters, array accesses, and indirect references all may have aliases, and it is not easy to tell if a statement is referring to a particular variable x." [p. 601]

"Program analysis must be conservative" [p. 601]
Transfer equations for reaching definitions

For this definition:

\[ d: u = v + w \]

The transfer equation is:

\[ f_d(x) = \text{gen}_d \cup (x - \text{kill}_d) \]

Where \( \text{gen}_d = \{d\}\). \( \text{kill}_d \) is the set of all other definitions of \( u \) in the program.

The argument of a transfer function is a data-flow value, which "represents an abstraction of the set of all possible program states that can be observed for that point." [p. 599]

Recall too that a program state consists of all the variables in the program along with their current values.
Figure 9.13 (p. 604)

\[
\begin{align*}
d1: & \quad i = m - 1 \\
d2: & \quad j = n \\
d3: & \quad a = u1 \\
d4: & \quad i = i + 1 \\
d5: & \quad j = j - 1 \\
d6: & \quad a = u2 \\
d7: & \quad i = u3
\end{align*}
\]

\text{ENTRY} \\
\downarrow \\
\text{B1} \\
\downarrow \\
\text{B2} \\
\downarrow \\
\text{B3} \\
\downarrow \\
\text{B4} \\
\downarrow \\
\text{EXIT} \\

\text{gen}_{B1} = \{ ? \} \\
\text{kill}_{B1} = \{ ? \} \\
\text{gen}_{B2} = \{ ? \} \\
\text{kill}_{B2} = \{ ? \} \\
\text{gen}_{B3} = \{ ? \} \\
\text{kill}_{B3} = \{ ? \} \\
\text{gen}_{B4} = \{ ? \} \\
\text{kill}_{B4} = \{ ? \}
d1: i = m - 1
    d2: j = n
    d3: a = u1

    d4: i = i + 1
    d5: j = j - 1

    d6: a = u2

    d7: i = u3

ENTRY

B1

gen_{B1} = \{ d1, d2, d3 \}
kill_{B1} = \{ ? \}

B2

gen_{B2} = \{ ? \}
kill_{B2} = \{ ? \}

B3

gen_{B3} = \{ ? \}
kill_{B3} = \{ ? \}

B4

gen_{B4} = \{ ? \}
kill_{B4} = \{ ? \}

EXIT
Figure 9.13 (p. 604)

\[
d1: i = m - 1
\]
\[
d2: j = n
\]
\[
d3: a = u1
\]
\[
d4: i = i + 1
\]
\[
d5: j = j - 1
\]
\[
d6: a = u2
\]
\[
d7: i = u3
\]
**Figure 9.13**

(p. 604)

\[
d4: i = i + 1
\]
\[
d5: j = j - 1
\]
\[
d7: i = u3
\]

**gen**

\[
B1 = \{ d1, d2, d3 \}
\]

**kill**

\[
B1 = \{ d4, d5, d6, d7 \}
\]

Q: Why kill d4 - d7 here, since they are not on a path to B1?

\[
d6: a = u2
\]

**gen**

\[
B2 = \{ ? \}
\]

**kill**

\[
B2 = \{ ? \}
\]

\[
d4: i = i + 1
\]
\[
d5: j = j - 1
\]

**gen**

\[
B3 = \{ ? \}
\]

**kill**

\[
B3 = \{ ? \}
\]

\[
d6: a = u2
\]

**gen**

\[
B4 = \{ ? \}
\]

**kill**

\[
B4 = \{ ? \}
\]

**ENTRY**

\[
\]

**EXIT**
Figure 9.13 (p. 604)

d1: i = m - 1
  d2: j = n
  d3: a = u1

d4: i = i + 1
  d5: j = j - 1

d6: a = u2

d7: i = u3

Q: Why kill d4 - d7 here, since they are not on a path to B1?

A: Here we are looking just at this block, and not trying to account for flow between blocks.

Inter-block flow is taken into account later.

\[ \text{gen}_{B1} = \{ d1, d2, d3 \} \]
\[ \text{kill}_{B1} = \{ d4, d5, d6, d7 \} \]
d1: i = m - 1
  d2: j = n
  d3: a = u1

B1

d4: i = i + 1
  d5: j = j - 1

B2

B3

d6: a = u2

B4

d7: i = u3

ENTRY

EXIT

\[ \text{gen}_{B1} = \{ d1, d2, d3 \} \]
\[ \text{kill}_{B1} = \{ d4, d5, d6, d7 \} \]

\[ \text{gen}_{B2} = \{ ? \} \]
\[ \text{kill}_{B2} = \{ ? \} \]

\[ \text{gen}_{B3} = \{ ? \} \]
\[ \text{kill}_{B3} = \{ ? \} \]

\[ \text{gen}_{B4} = \{ ? \} \]
\[ \text{kill}_{B4} = \{ ? \} \]
Figure 9.13 (p. 604)

\[
\begin{align*}
\text{d1: } & i = m - 1 \\
\text{d2: } & j = n \\
\text{d3: } & a = u_1 \\
\text{d4: } & i = i + 1 \\
\text{d5: } & j = j - 1 \\
\text{d6: } & a = u_2 \\
\text{d7: } & i = u_3
\end{align*}
\]

\[\text{gen}_{B1} = \{ \text{d1, d2, d3} \} \]
\[\text{kill}_{B1} = \{ \text{d4, d5, d6, d7} \} \]

\[\text{gen}_{B2} = \{ \text{d4, d5} \} \]
\[\text{kill}_{B2} = \{ ? \} \]

\[\text{gen}_{B3} = \{ ? \} \]
\[\text{kill}_{B3} = \{ ? \} \]

\[\text{gen}_{B4} = \{ ? \} \]
\[\text{kill}_{B4} = \{ ? \} \]
\[
\text{gen}_{B1} = \{ d1, d2, d3 \} \\
\text{kill}_{B1} = \{ d4, d5, d6, d7 \}
\]

\[
\text{gen}_{B2} = \{ d4, d5 \} \\
\text{kill}_{B2} = \{ d1, d2, d7 \}
\]

\[
\text{gen}_{B3} = \{ ? \} \\
\text{kill}_{B3} = \{ ? \}
\]

\[
\text{gen}_{B4} = \{ ? \} \\
\text{kill}_{B4} = \{ ? \}
\]
\begin{center}

Figure 9.13 (p. 604)

\begin{itemize}
  \item \textbf{d1: } \( i = m - 1 \)
  \item \textbf{d2: } \( j = n \)
  \item \textbf{d3: } \( a = u_1 \)
  \item \textbf{d4: } \( i = i + 1 \)
  \item \textbf{d5: } \( j = j - 1 \)
  \item \textbf{d6: } \( a = u_2 \)
  \item \textbf{d7: } \( i = u_3 \)
\end{itemize}

\textbf{gen} \( B_1 \) = \{ d1, d2, d3 \}
\textbf{kill} \( B_1 \) = \{ d4, d5, d6, d7 \}

\textbf{gen} \( B_2 \) = \{ d4, d5 \}
\textbf{kill} \( B_2 \) = \{ d1, d2, d7 \}

\textbf{gen} \( B_3 \) = \{ d6 \}
\textbf{kill} \( B_3 \) = \{ ? \}

\textbf{gen} \( B_4 \) = \{ ? \}
\textbf{kill} \( B_4 \) = \{ ? \}
\end{center}
d1: \( i = m - 1 \)
d2: \( j = n \)
d3: \( a = u_1 \)
d4: \( i = i + 1 \)
d5: \( j = j - 1 \)
d6: \( a = u_2 \)
d7: \( i = u_3 \)

\[ \text{ENTRY} \]

\[ \text{EXIT} \]

\[ \text{gen}_{B1} = \{ \text{d1, d2, d3} \} \]
\[ \text{kill}_{B1} = \{ \text{d4, d5, d6, d7} \} \]

\[ \text{gen}_{B2} = \{ \text{d4, d5} \} \]
\[ \text{kill}_{B2} = \{ \text{d1, d2, d7} \} \]

\[ \text{gen}_{B3} = \{ \text{d6} \} \]
\[ \text{kill}_{B3} = \{ \text{d3} \} \]

\[ \text{gen}_{B4} = \{ ? \} \]
\[ \text{kill}_{B4} = \{ ? \} \]
Figure 9.13 (p. 604)

**B1**

\[ \text{d1: } i = m - 1 \]
\[ \text{d2: } j = n \]
\[ \text{d3: } a = u1 \]

\[
\text{gen}_{\text{B1}} = \{ \text{d1, d2, d3} \}
\]
\[
\text{kill}_{\text{B1}} = \{ \text{d4, d5, d6, d7} \}
\]

**B2**

\[ \text{d4: } i = i + 1 \]
\[ \text{d5: } j = j - 1 \]

\[
\text{gen}_{\text{B2}} = \{ \text{d4, d5} \}
\]
\[
\text{kill}_{\text{B2}} = \{ \text{d1, d2, d7} \}
\]

**B3**

\[ \text{d6: } a = u2 \]

\[
\text{gen}_{\text{B3}} = \{ \text{d6} \}
\]
\[
\text{kill}_{\text{B3}} = \{ \text{d3} \}
\]

**B4**

\[ \text{d7: } i = u3 \]

\[
\text{gen}_{\text{B4}} = \{ \text{d7} \}
\]
\[
\text{kill}_{\text{B4}} = \{ \text{?} \}
\]
Figure 9.13 (p. 604)

\[ \begin{align*}
  d1: & \quad i = m - 1 \\
  d2: & \quad j = n \\
  d3: & \quad a = u1 \\
  d4: & \quad i = i + 1 \\
  d5: & \quad j = j - 1 \\
  d6: & \quad a = u2 \\
  d7: & \quad i = u3
\end{align*} \]

\[ \begin{align*}
  \text{gen}_{B1} & = \{ d1, d2, d3 \} \\
  \text{kill}_{B1} & = \{ d4, d5, d6, d7 \} \\
  \text{gen}_{B2} & = \{ d4, d5 \} \\
  \text{kill}_{B2} & = \{ d1, d2, d7 \} \\
  \text{gen}_{B3} & = \{ d6 \} \\
  \text{kill}_{B3} & = \{ d3 \} \\
  \text{gen}_{B4} & = \{ d7 \} \\
  \text{kill}_{B4} & = \{ d1, d4 \}
\]
Extending transfer equations from statements to blocks

Composition of $f_1$ and $f_2$:

$$f_1(x) = \text{gen}_1 \cup ( x - \text{kill}_1 )$$

$$f_2(x) = \text{gen}_2 \cup ( x - \text{kill}_2 )$$

$$f_2( f_1(x) ) = \text{gen}_2 \cup ( (\text{gen}_1 \cup ( x - \text{kill}_1 )) - \text{kill}_2 )$$

$$= \text{gen}_2 \cup ( (\text{gen}_1 - \text{kill}_2) \cup (( x - \text{kill}_1 ) - \text{kill}_2) )$$

$$= \text{gen}_2 \cup (\text{gen}_1 - \text{kill}_2) \cup ( x - (\text{kill}_1 \cup \text{kill}_2) )$$
Extending transfer equations from statements to blocks

In general:

\[ f_B(x) = \text{gen}_B \cup (x - \text{kill}_B) \]

\[ \text{kill}_B = \bigcup_{i=n} \text{kill}_i \]

\[ \text{gen}_B = \text{gen}_n \cup \bigcup_{i=n-1} \text{gen}_{n-i} \cup \bigcup_{i=n-2} \text{gen}_{n-2} - \text{kill}_{n-1} - \text{kill}_n \cup \cdots \cup (\text{gen}_1 - \text{kill}_2 - \text{kill}_3 - \cdots - \text{kill}_n) \]
Extending transfer equations from statements to blocks

"The gen set contains all the definitions inside the block that are "visible" immediately after the block - we refer to them as downwards exposed. A definition is downwards exposed in a basic block only if it is not "killed" by a subsequent definition to the same variable inside the same basic block." [p. 605]
Iterative algorithm for reaching definitions

Algorithm [p. 606]

INPUT: A flow graph for which kill_B and gen_B have been computed for each block B.

OUTPUT: IN[B] and OUT[B], the set of definitions reaching the entry and exit of each block B of the flow graph

METHOD:

\[
\text{OUT}[\text{ENTRY}] = \emptyset \\
\text{for (each basic block B other than ENTRY) } \{ \text{OUT}[B] = \emptyset \} \\
\text{while (changes to any OUT occur) } \{ \\
\quad \text{for (each basic block B other than ENTRY) } \{ \\
\quad\quad \text{IN}[B] = \bigcup_{\text{a predecessor of } B} \text{OUT}[P] \\
\quad\quad \text{OUT}[B] = \text{gen}_B \cup (\text{IN}[B] - \text{kill}_B) \\
\quad \} \\
\} \\
\]

See footnote 4 on page 606
Example 9.12 - building off figure 9.13

Represent \( d_i \) as a bit vector.

**Union of sets** \( A \cup B: A \text{ OR } B \)

**Difference of sets** \( A - B: A \text{ AND } B' \)

Compute in order \( B_1, B_2, B_3, B_4, \text{ EXIT} \)

\[
\text{IN}[B2]^1 = \text{OUT}[B1]^1 \cup \text{OUT}[B4]^0 = 111\ 0000 \cup 000\ 0000 = 111\ 0000
\]

\[
\text{OUT}[B2]^1 = \text{gen}_{B2} \cup (\text{IN}[B2]^1 - \text{kill}_{B2})
\]

\[
= 000\ 1100 + (111\ 0000 - 110\ 0011)
\]

\[
= 000\ 1100 + 001\ 0000 = 001\ 1100
\]

<table>
<thead>
<tr>
<th>( B_1 )</th>
<th>( \text{OUT}[B]^0 )</th>
<th>( \text{IN}[B]^1 )</th>
<th>( \text{OUT}[B]^1 )</th>
<th>( \text{IN}[B]^2 )</th>
<th>( \text{OUT}[B]^2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( B_2 )</td>
<td>000 0000</td>
<td>000 0000</td>
<td>111 0000</td>
<td>000 0000</td>
<td>111 0000</td>
</tr>
<tr>
<td>( B_3 )</td>
<td>000 0000</td>
<td>111 0000</td>
<td>001 1100</td>
<td>111 0111</td>
<td>001 1100</td>
</tr>
<tr>
<td>( B_4 )</td>
<td>000 0000</td>
<td>001 1100</td>
<td>000 1110</td>
<td>001 1110</td>
<td>000 1110</td>
</tr>
<tr>
<td><strong>EXIT</strong></td>
<td>000 0000</td>
<td>001 0111</td>
<td>001 0111</td>
<td>001 0111</td>
<td>001 0111</td>
</tr>
</tbody>
</table>