CSE443
Compilers

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall
**Semester plan**
(probably wildly optimistic)

<table>
<thead>
<tr>
<th>M</th>
<th>W</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PRo5</strong></td>
<td>9.2 Data-flow analysis</td>
<td>9.3 Data-flow foundations</td>
</tr>
<tr>
<td>9.1 Overview</td>
<td>9.4 Constant propagation</td>
<td><strong>Kris Schindler Architecture talk</strong></td>
</tr>
<tr>
<td>9.4 Constant propagation</td>
<td><strong>9.6 Loops in flow graphs</strong></td>
<td>9.5 Redundancy elimination</td>
</tr>
<tr>
<td>9.7 Region-based analysis</td>
<td>9.8 Symbolic analysis</td>
<td></td>
</tr>
</tbody>
</table>
Phases of a compiler

Figure 1.6, page 5 of text
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
State transformation

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 $p_1, p_2, ..., p_n$ such that for each $i = 1, 2, ..., n-1$, 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."

[p. 597]
Example 9.8 (p. 598)

```
(1) d1: a = 1
    B1
    Path: (1,2,3,4,9)

(2)

(3) if read() <= 0 goto B4
    B2
    Path: (1,2,3,4,5,6,7,8,3,4,9)

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

(5)
    d2: b = a
    B3
    d1 reaches (5) on the first iteration.

(6) d3: a = 243
    goto B2
    B3
    a has value 243 at (5) on the second and subsequent iterations.

(7) d3 reaches (5) on those iterations.

(8)

(9) ...
    B4
```
"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 \( \text{IN}[s] \) and \( \text{OUT}[s] \), respectively." [p. 599]
9.2.2 Data-flow analysis schema

"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: \( \text{OUT}[s] = f_s (\text{IN}[s]) \)

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

In a sequence $s_1, s_2, ..., s_n$ without jumps,

$$IN[s_{i+1}] = OUT[s_i]$$ 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

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

\[ \text{OUT}[B] = \bigcup_{S \text{ a successor of } B} \text{IN}[S] \]
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...”

[p. 601]