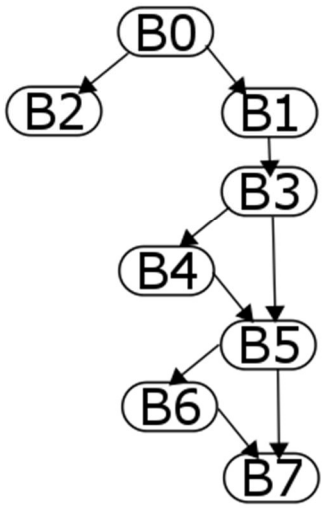
**Assignment 1 Write-Up**

Milijana Surbatovich and Emily Ruppel

**3.1**

B0
```
    x = 50
    y = 8
    z = 234
    if (x < z) { goto L1 }
```
B1
```
    y = 89
    goto L2
```
B2
```
L1: z = 65
    return z
```
B3
```
L2: y = x + 1
    if (z < x) { goto L3 }
```
B4
```
    x = 25
```
B5
```
L3: y = x + z
    switch (y) { 334: goto L4 | default: goto L5 }
```
B6
```
L4: print("failure")
```
B7
```
L5: y = 65
    return y
```



**3.2**

| BB | Gen | Kill | In | Out |
|---|---|---|---|---|
| 1 | b+c<br>c+d<br>a*b | b+c | Empty | c+d<br>a*b |
| 2 | i+d<br>c+d | c+d<br>a*b | c+d<br>a*b | i+d<br>c+d |
| 3 | b+d | a+d | i+d<br>c+d | i+d<br>c+d<br>b+d |
| 4 | b*b<br>b+d | Empty | i+d<br>c+d | b*b<br>b+d<br>c+d<br>i+d |
| 5 | Empty | i+2 | i+d<br>c+d<br>b+d | i+d<br>c+d<br>b+d |

**3.3**

1. Variables

2. Backwards analysis

3. $\text{Out}[B] = \cup\, \text{In}[s], \quad s \ni \text{Succ}[B]$
   $\text{In}[B] = \text{Out}[B] - (\text{Control}[B] + \text{Relevant}[B])$
   $\text{Relevant}[B] = \text{variables used to define LHS} \notin \text{Out}[B] \mid \text{Control}[B]$

4. Union

5. ENTRY is initialized to the bottom of the call graph, and EXIT is initialized to the top.

6. OUT is initialized to a bit vector of all zeros, IN is uninitialized.

7. Block traversal will not impact correctness, but post-order traversal will reduce the number of iterations to find a fixed-point solution because predecessors will not need to be analyzed again after their successors, since faint variables in their successors will be determined first.

8. The analysis will converge because once variables are removed from the faint set, they are never returned. Without returning variables to the faint set, there is only ever a finite number of variables to analyze, so at some point the analysis will converge on faint/not faint values for every expression.

9. The following pseudo code assumes that a bit vector the length of the total number of variables in the program is initialized to all zeros, where zero indicates that the variable is faint. Further, “is control instruction” refers to all control flow instructions as well as function calls and function returns. For each instruction, the LHS (left hand side) of the instruction, the variable being defined, is checked and if it is not faint, then the variables used to define the LHS are also set to NOT FAINT in the bit vector.

```
faints[number of variables] = {FAINT};

for each(basic block in post-order traversal)
        for each(instruction in backwards analysis)
                // Removes variables used to calculate control flow, returns or calls from faint list
                if(instruction is control instruction)
                        for each (variable in instruction)
                                faints[variable] = NOT FAINT
                // Removes variables used to calculate variables that are not faint from faint list
                if(faints[LHS] == NOT FAINT)
                        for each (RHS variable in instruction)
                                faints[RHS variable] = NOT FAINT
```