Abstract Program Representation

In this section we will be looking at structures that represent a program’s flow of control. A flow graph is essentially a finite-state automaton whose nodes represent various locations in a program and whose edges indicate possible transitions among these locations. The nodes of a flow graph represent sites where optimization information can be generated or consumed. The graph’s edges are conduits for combining and further propagating such information.

**Def:** A flow graph \( G = (N, E, \text{root}) \) is a directed graph where \( N \) is a set of nodes and \( E \) is a binary relation on \( N \). The root node is the distinguished entry node of the flow graph \( \forall X \in N, (X, \text{root}) \notin E \).

A *Control Flow Graph* \( G_{CF} \) represents potential execution paths within a procedure. Generally, each node in \( N_{CF} \) corresponds to a straight-line sequence of operations. Each edge in \( E_{CF} \) represents potential transfer of execution from the end of one sequence to the beginning of another linear sequence. Control flow graphs are used in *intraprocedural analyses*.

A *Procedural Call Graph* \( G_{PC} \) represents potential execution paths between the procedures of a program. Each node in \( N_{PC} \) corresponds to a procedure in the program. Each edge in \( E_{PC} \) represents a potential procedure call. Procedure call graphs are used in *interprocedural analyses*.

In practice, the behavior of a program is captured by multiple flow graphs, each of differing granularity, according to the expected benefits of optimization at various levels. For example, each node of a procedure call graph is a procedure that is represented by an intraprocedural control flow graph. Theory and practice suggest the benefits of this organization, and a compiler designer must consider the expense and expected benefits of representing program behavior at any given level.

**Def:** All programs can be represented by a *Control Flow Graph (CFG)* where the nodes of the graph are called *basic blocks* and the edges represent the flow between the basic blocks.

**Def:** A *basic block* is a program segment that is a maximal linear sequence of statements that has only one entry point and only one exit point.

An alternative way of looking at basic blocks is that a basic block is a sequence of program statements organized in such a way that all transfers to the block are to the first statement in the block and that if any statement in the block is executed, then all statements in the block are executed.

**Def:** An *extended basic block* is a maximal sequence of code that only one entry point and may have several exit points.

The extended basic block will be important when we look at folding and constant propagation and register allocation. In construction our control flow graph we will concentrate on basic blocks.

**Example**

Consider the following program segment
DECLARE
    INTEGER i, a, b, d3, e, d1, d, c, d2;
    INTEGER VECTOR s WITH 100 ELEMENTS;
END

COMPUTE
    i <- 1;
    a <- 0;
    b <- 10;
    DO
        c <- i * (a + 1 * b);
        d3 <- b + 1 + i;
        d <- c + d1;
        e <- a + c * (1 + i);
        d1 <- a * c * (1 + i);
        IF (c < 10)
            THEN
                e <- e + 1;
                d1 <- b + i;
                s[1 + a] <- s[1 + b] + s[2 + b];
                c <- i * (b * d1);
            END;
    END
    e <- d + e;
    DO
        d2 <- c + s[i];
        c <- b + i;
        k <- 10;
        IF (c < 20)
            THEN
                DO
                    s[k] <- e + (c + d);
                    s[e] <- 10 + i;
                    k <- e + 1;
                UNTIL (k < 20);
                k <- 25;
            ELSE
                k <- 15;
                DO
                    d <- b + 10 + i;
                    s[k] <- 3 + k;
                    k <- 1 + k;
                UNTIL (k < 30);
                k <- 25;
            END;
        e <- k + 3 * a;
        k <- c * (1 + i);
    UNTIL (k != 30);
    d <- b * (a * e);
    k <- i + (c + d);
    i <- 1 + i;
    UNTIL (i < 10);
    s[b] <- c + k;
    i <- i + d3;
END
We will be considering a sequence of 4-tuples and constructing a control flow graph after identifying the basic blocks in the 4-tuples. From the above program segment we have the following 4-tuple sequence:

```plaintext
1  i = 1
2  a = 0
3  b = 10
4  L$1 label
5  L$1 * 1 b
6  L$2 + a L$1
7  L$3 * i L$2
8  c = L$3
9  L$4 + 1 i
10 L$5 * b L$4
11 d3 = L$5
12 L$6 + 1 i
13 L$7 * c L$6
14 d = L$7
15 L$8 + 1 i
16 L$9 * c L$8
17 L$10 + a L$9
18 e = L$10
19 L$11 * a L$9
20 d1 = L$11
21 B$12 < c 10
22 L$2 JPC B$12
23 L$3 + e i
24 e = L$13
25 L$14 * b i
26 d1 = L$14
27 L$15 + 1 b
28 L$16 LD a L$15
29 L$17 + 2 b
30 L$18 LD a L$17
31 L$19 + L$16 L$18
32 L$20 + 1 a
33 s = L$19 L$20
34 L$21 * b d1
35 L$22 * i L$21
36 c = L$22
37 L$2 label
38 L$23 + d e
39 e = L$23
40 L$3 label
41 L$24 LD a L$24
42 L$25 + c L$24
43 d2 = L$25
44 L$26 + 1 i
45 L$27 * b L$26
46 c = L$27
47 k = 10
48 B$28 < c 20
49 L$4 JPC B$28
```

The sequence of 4-tuples represents the control flow graph, where each 4-tuple is a set of instructions that can be executed sequentially. The basic blocks are identified by the labels L$1, L$2, L$3, etc., and the control flow is determined by the jumps (JPC) and conditional branches (B$).
Given a sequence of 4-tuples, you can identify the basic blocks and control flow graph by

1. Determine the set of leaders, i.e. first statement in a basic block.
   a. The first statement is a leader,
   b. Any statement that is a target of a conditional or unconditional jump, i.e. any label statement, and
   c. Any statement that immediately follows a jump or conditional jump.

2. For each leader, its basic block consists of the leader and all statements up to but not including the next leader or the end of the program.

Using the above sequence of 4-tuples, we have the following set of leaders:

\{
\}

Click [here](#) for the leaders.

<table>
<thead>
<tr>
<th>Basic Block</th>
<th>Beginning 4-tuple</th>
<th>Ending 4-tuple</th>
<th>Next 4-tuples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2

3

4

5

6

7
The complete set of basic blocks for the 4-tuples is

<table>
<thead>
<tr>
<th>Basic Block</th>
<th>Beginning 4-tuple</th>
<th>Ending 4-tuple</th>
<th>Next 4-tuples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>22</td>
<td>23, 37</td>
</tr>
<tr>
<td>3</td>
<td>23</td>
<td>36</td>
<td>37</td>
</tr>
<tr>
<td>4</td>
<td>37</td>
<td>39</td>
<td>40</td>
</tr>
<tr>
<td>5</td>
<td>40</td>
<td>49</td>
<td>50, 62</td>
</tr>
<tr>
<td>6</td>
<td>50</td>
<td>59</td>
<td>50, 60</td>
</tr>
<tr>
<td>7</td>
<td>60</td>
<td>61</td>
<td>75</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Basic Block</th>
<th>Beginning 4-tuple</th>
<th>Ending 4-tuple</th>
<th>Next 4-tuples</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>62</td>
<td>63</td>
<td>64</td>
</tr>
<tr>
<td>9</td>
<td>64</td>
<td>73</td>
<td>64, 74</td>
</tr>
<tr>
<td>10</td>
<td>74</td>
<td>74</td>
<td>75</td>
</tr>
<tr>
<td>11</td>
<td>75</td>
<td>83</td>
<td>40, 84</td>
</tr>
<tr>
<td>12</td>
<td>84</td>
<td>93</td>
<td>4, 94</td>
</tr>
<tr>
<td>13</td>
<td>94</td>
<td>97</td>
<td></td>
</tr>
</tbody>
</table>

The resulting control flow graph is
The set of leaders is

\{ 1, 4, 23, 37, 40, 50, 60, 62, 64, 74, 75, 84, 94 \}

Return