CS5214 Design of Optimizing Compilers

Professor Weng-Fai Wong
Assignment Two

February 27, 2016

Abstract

This is the answer presented by Pan An. My student number is A0134556A. After tryin all the cool tools in Python, GNUPlot, and some books and papers, I finally figured out that this assignment is not about programming. Alright so I'm just gonna present some results that are partially calculated with my hand. Some of the math greatly damaged my brain(kidding). Also I wanted to put a lot of references but it seems that time is not enough for that. The folder contains some of the rough calculation results performed by Python and some other tools.

1 The LengauerTarjan algorithm

For the first question I simply assumn that in this diagram it does not have very terrible effect by counting the **start** and **stop** nodes into the algorithm.

1.1 Depth First Search

I simply used **networkx** in Python to do some of the graph operation for me. The depth first search tree is stored in a picture in **img folder**.

After performing DFS on the tree, the DFT numbering is listed below in the form of (n_{DFT}, Node). Node with number 0 and 100 are the start and stop node.

```
(1, 0)
(2, 1)
(3, 2)
(4, 8)
(5, 31)
(6, 4)
(7, 5)
(8, 6)
(9, 7)
(10, 9)
(11, 10)
(12, 11)
(13, 12)
```

```
(14, 13)
(15, 14)
(16, 15)
(17, 16)
(18, 33)
(19, 18)
(20, 34)
(21, 21)
(22, 100)
(23, 3)
(24, 17)
(25, 19)
```

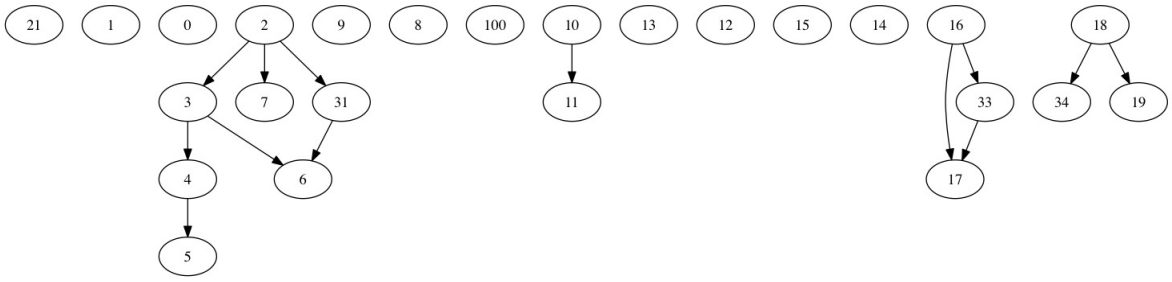
Noticing that there could be multiple answers to DFS tree numbering, I am using this one as the answer for the next couple questions.

1.2 Dominators

Most of the time semi-dominators of nodes in a graph is the immediate dominator. The following picture shows the semi-dominator of the given diagram:

1.3 Control Dependency

Actually I am a little skeptical about this one since I actually read another source that it does not have to be strict dominance in the definition of control dependence.



1.4 Strongly Connected Componants

A directed graph is called strongly connected if there is a path in each direction between each pair of vertices of the graph.

Strongly connected graphs are(all edges amongs these nodes included):

- Nodes with self loop: 9, 13, 15, 19
- Graph composed with node 16 and 33
- Graph composed with node 3, 4, 6, 7
- Graph composed with node 3, 4, 6, 7, 5
- Graph composed with node 9, 10, 11

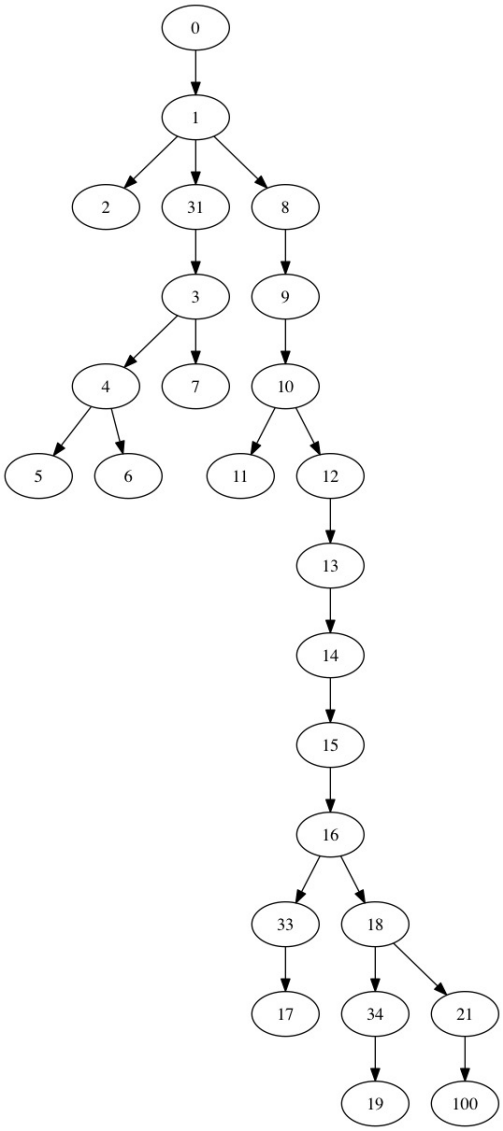


Figure 1: Semi Dominator Graph

1.5 Question 7

Dominant frontiers of the three nodes are:

- 31: [8]
- 11: [9, 12]
- 33: [16, 18]

Block 8, 9, 12, 16, 18 should be inserted with ϕ function.

2 Dataflow analysis

To be honest, I lost it when it comes to this part in the class so I Wikipediaed a little. Also I read the slides but the slides seem not so fullfilled in this part(no offence). So here I will just use the expression in terms of data flow equations.

The problem is a typical backward data flow analysis problem. Let $def[n]$ be the set of variables that are used in node n before any assignment. The function $use[n]$ with lower case letters does the exact same thing as $USE[n]$. Let $succ[n]$ be the set of successor nodes after node n successfully excutes. Dataflow equations can be expressed as the following:

- $live_{in}[n] = def[n] \cup (live_{out}[n] - use[n])$
- $live_{out}[final] = \Phi$
- $live_{out}[n] = \bigcup_{p \in succ[n]} live_{in}[p]$

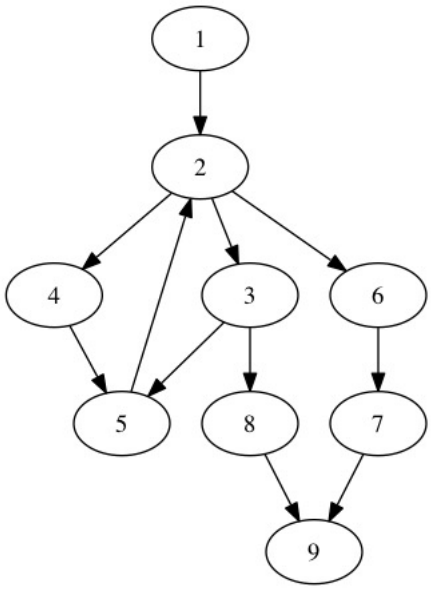
The algorithm for solving data flow equation is presented below:
for each node n in CFG:
$live_{in}[n] = \Phi, live_{out}[n] = \Phi$
repeat
for each node n in CFG:
$live'_{in}[n] = live_{in}[n]$
$live'_{out}[n] = live_{out}[n]$
$live_{in}[n] = use[n] \cup (live_{out}[n] - def[n])$
$live_{out}[n] = \bigcup_{p \in succ[n]} live_{in}[p]$
until $live'_{in}[n] = live_{in}[n] and live'_{out}[n] = live_{out}[n] for all n$
Consider the following graph:

Assuming that in the graph node **3 and 5** are with a postive $USE(n)$ value(variable is used) and the variable is **defined** in block 1. The algorithm will first traverse the graph and find the variable usage, and after this process:

$$S_u = \{3, 5\}$$

Here I assumn there are couple things happening:

1. In node 1:
   - a = 2;
   - b = 1;
   - c = 1;
2. Node 2: a = 1, 2, 3, means switch to 4, 3, 6;



3. Node 5:
   - c = a - b;
   - d = 2;
4. Node 3:
   - d = c + 1;
   - if a > d then go to 8, else go to 5;
5. Nothing concerns a, b, c, d happens in other nodes;

The program flow will be pretty simple:

$$1, in = \{\}, out = \{\} \rightarrow 2, in = a, out = \{\} \rightarrow 3, in = acd, out = \{\} \rightarrow$$

$$5, in = abcd, out = a \rightarrow 2in = a, out = cd \rightarrow 3in = acd, out = \{\} \rightarrow 8in = \{\}tillstop$$

So it actually only concerns nodes 1, 2, 3, 5, 8. And the last uses of each variable can be tracked from the flow above.