Exam in Optimising Compilers (DAT230/EDA230)

October 17, 2007, 8.00 — 13.00

Examinator: Jonas Skeppstedt, tel 0733 549 314

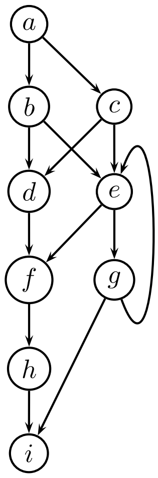


Figure 1: Control flow graph.

1. (10p) Explain how the Lengauer-Tarjan algorithm (the $O(N^2)$-version) finds the dominator tree in the control flow graph in Figure 1. For each vertex, your solution should explain:

   - when is the vertex put in a bucket?
   - in which bucket?
   - when is it deleted from the bucket?
   - when does the algorithm find the immediate dominator for the vertex?

   Answer: see book.

2. (10p) Consider again the control flow graph in Figure 1. Suppose there is a use of variable x in each vertex and an assignment to x in

vertices $a,\ c$ and e. **In vertices a and c the definition is before the use and in vertex e the definition is after the use.**

Translate the program to SSA form. Show the contents of the rename stack and when the stack is pushed and popped. *You do not have to show how you compute the dominance frontiers.*

Answer: see book.

3. (10p) Again refer to Figure 1. For each of vertex e, f, g and i, which vertices (if any) is that vertex control dependent on? You do not have to show how you arrived at that result, but you should explain in a few sentences how it is done in a compiler.

   Answer: A vertex v is control dependent on a vertex u if u is a member of the dominance frontier of v in the reverse control flow graph.

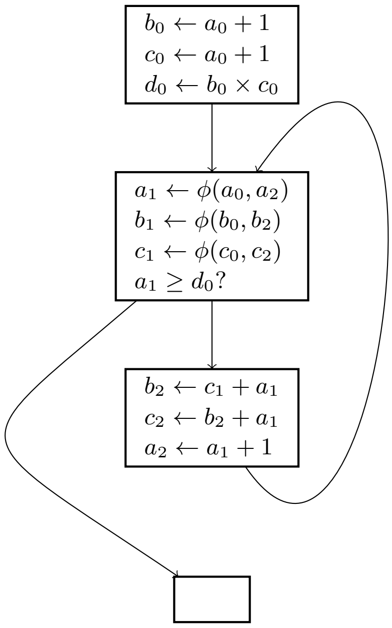
$$
\begin{aligned}
CD^{-1}(e) &= \{b, c, g\} \\
CD^{-1}(f) &= \{b, c, e\} \\
CD^{-1}(g) &= \{e\} \\
CD^{-1}(i) &= \emptyset
\end{aligned}
$$

```
int f(int a)
{
        int     b, c, d;

        b = a + 1;
        c = a + 1;
        d = b * c;

        while (a < d) {
                b = c + a;
                c = b + a;
                a = a + 1;
        }

        return b + c;
}
```

Figure 2: C function for question on partition-based global value numbering.

4. (10p) How does partition-based global value numbering (GVN) on SSA form optimise the program in Figure 2? Show how the algorithm proceeds.

Answer:



The instructions are partitioned into an initial set of blocks π_0:

$$B_0 = \{a_0\}$$
$$B_1 = \{b_0 \leftarrow a_0 + 1, c_0 \leftarrow a_0 + 1, b_2 \leftarrow c_1 + a_1, c_2 \leftarrow b_2 + a_1, a_2 \leftarrow a_1 + 1\}$$
$$B_2 = \{a_1 \leftarrow \phi(a_0, a_2), b_1 \leftarrow \phi(b_0, b_2), c_1 \leftarrow \phi(c_0, c_2)\}$$

We show the N^2-version of GVN. When B_1 is checked, it is split into B_1' and B_1''. First one member from B_1 is put in B_1' and then the others are compared with it, and either also are put in B_1' if they are equivalent, or otherwise are put in B_1''.

The first new block is thus B_1' which becomes:

$$
\begin{aligned}
B_1' &= \{b_0 \leftarrow a_0 + 1, c_0 \leftarrow a_0 + 1\} \\
B_1'' &= \{b_2 \leftarrow c_1 + a_1, c_2 \leftarrow b_2 + a_1, a_2 \leftarrow a_1 + 1\}
\end{aligned}
$$

When B_2 is checked, none of the second and third members are equivalent to the first since a_0 belongs to a singleton block:

$$
\begin{aligned}
B_2' &= \{a_1 \leftarrow \phi(a_0, a_2)\} \\
B_2'' &= \{b_1 \leftarrow \phi(b_0, b_2), c_1 \leftarrow \phi(c_0, c_2)\}
\end{aligned}
$$

For the next iteration, we rename the blocks as follows:

$$
\begin{aligned}
B_0 &= \{a_0\} \\
B_1 &= \{b_0 \leftarrow a_0 + 1, c_0 \leftarrow a_0 + 1\} \\
B_2 &= \{b_2 \leftarrow c_1 + a_1, c_2 \leftarrow b_2 + a_1, a_2 \leftarrow a_1 + 1\} \\
B_3 &= \{a_1 \leftarrow \phi(a_0, a_2)\} \\
B_4 &= \{b_1 \leftarrow \phi(b_0, b_2), c_1 \leftarrow \phi(c_0, c_2)\}
\end{aligned}
$$

Now B_2 will be split into:

$$
\begin{aligned}
B_2' &= \{b_2 \leftarrow c_1 + a_1\} \\
B_2'' &= \{c_2 \leftarrow b_2 + a_1, a_2 \leftarrow a_1 + 1\}
\end{aligned}
$$

B_4 will also be split:

$$
\begin{aligned}
B_4' &= \{b_1 \leftarrow \phi(b_0, b_2)\} \\
B_4'' &= \{c_1 \leftarrow \phi(c_0, c_2)\}
\end{aligned}
$$

Renaming the blocks for the next iteration we get:

$$
\begin{aligned}
B_0 &= \{a_0\} \\
B_1 &= \{b_0 \leftarrow a_0 + 1, c_0 \leftarrow a_0 + 1\} \\
B_2 &= \{b_2 \leftarrow c_1 + a_1\} \\
B_3 &= \{c_2 \leftarrow b_2 + a_1, a_2 \leftarrow a_1 + 1\} \\
B_4 &= \{b_1 \leftarrow \phi(b_0, b_2)\} \\
B_5 &= \{c_1 \leftarrow \phi(c_0, c_2\}\}
\end{aligned}
$$

After that also B_3 will be split and only B_1 contains multiple members, of which the first dominates the second which will be removed. Thus, only c_0 is optimized away by GVN in this code.

5. (10p) What is partial redundancy elimination (PRE)? Explain an an algorithm for doing PRE on SSA form. Show an example code which PRE can optimise which partion-based global value numbering cannot.

   Answer: see book. For an example of code, we need a partial *redundancy such as in:*

   ```
   if (a < b)
           c = a * b;
   d = a * b;
   ```

6. (10p) Explain the principles behind Chaitin's algorithm. Among other things, you should explain what the purpose of *coalescing* is and why some caution should be observed when coalescing.

   Answer: see George/Appel article. With too much coalescing, the IG may not be possible to color due to too many nodes have too many neighbors and cannot find an available color.