Compiler Theory CS447b: Quiz 8. **UWO, April 7, 2004.**

Student ID number:

Guidelines. The quiz is closed book and all notes are forbidden. The duration of the quiz is 40 minutes. All answers should be written in the *answer boxes*. No justifications for the answers are needed. You are expected to do this quiz on your own without assistance from anyone else in the class. If possible, please avoid pencils and use pens with dark ink. Thank you.

Exercise 1. Consider the following three-address code fragment.

```
L0  a := 15
    b := 12
L1  if b = 0 goto L6
L2  q := 0
    r := a
L3  if r < b goto L5
L4  r := r - b
    q := q + 1
    goto L3
L5  a := b
    b := r
    goto L1
L6  g := a
```

1. Give the flow graph of this program.

2. Identify the back edges and the loops.

3. Is there any propagation of a copy statement that can be performed. Justify your answer briefly.

4. Explain how the techniques developed in the course can help discover that two lines of the above program can be eliminated.

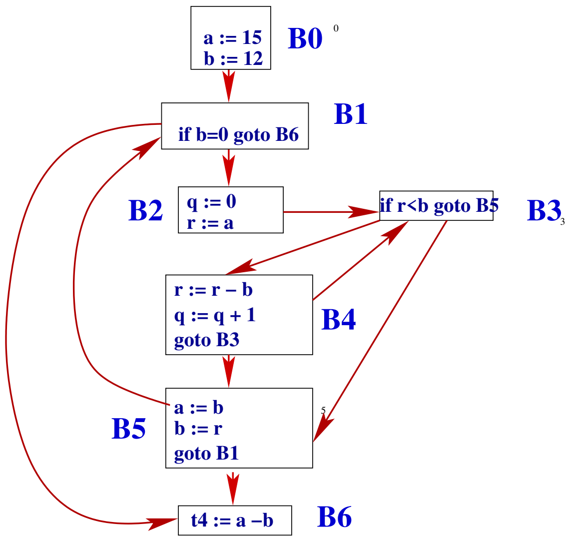

Figure 1: The control flow graph.

.

Answer 1

The node B_1 in the control flow graph G of Figure 1 dominates the nodes B_2, B_3, B_4, B_5, B_6. Therefore the edge $[B_5, B_1]$ is a back-edge. Clearly, the nodes B_1, B_2, B_3, B_4, B_5 induce a strongly connected subgraph L_1 of G. Since there is only one port of entry from the outside of L_1 into L_1, the subgraph L_1 is a loop. Similarly, B_1, B_2, B_3, B_5 induce a loop L_2 of G. The node B_3 dominates the node B_4, together they induce a strongly connected subgraph L_3 of G into which there is only one port of entry. Therefore L_3 is a loop.

It is not hard to see that the whole program computes the gcd of 15 and 12 by the Euclidean Algorithm. The loops L_2, L_3 and L_1 corresponds respectively the Euclidean Algorithm when no division is needed, the division algorithm, the general case for the Euclidean Algorithm (where divisions may be needed).

No propagation of copy statement is possible. For instance `a := 15` does **not** reach the statement `r := a`. Indeed, there are pathes from `a := 15` to `r := a` that go through `a:= b`.

Observe that after the statement `q := q + 1`, the variable `q` is never refereed (it is just overwritten by `q := 0`). Hence the variable `q` is dead after `q := q + 1` and this statement is dead-code that can be eliminated. Then `q := 0` becomes dead-code too.

Exercise 2. Consider the following three-address code fragment.

```
L0  a := 15
    b := 12
L1  d := a - b
    if d = 0 goto L5
L2  if d > 0 goto L4
L3  b := a - b
    b := - b
    goto L1
L4  a := a - b
    goto L1
L5  g := b
```

1. Give the flow graph of this program.
2. Identify the back edges and the loops.
3. Is there any propagation of a copy statement that can be performed. Justify your answer briefly.
4. Explain how the techniques developed in the course can help discover that two arithmetic operations of the above program can be eliminated (and replaced by a copy statement).
5. After this transformation, is there any propagation of a copy statement that can be performed?

Answer 2

I give here only indications for the answers.

1. No particular difficulty. One can observe that this program computes also the gcd of 12 and 15 (but with a different algorithm).

2. Three loops again: B_1, B_2, B_3, B_4, B_1, B_2, B_3 and B_1, B_2, B_4.

3. No propagation of a copy statement in the initial progam.

4. The expression `a - b` from the statement `d := a - b` reaches both statements `b := a - b` and `a := a - b`. Hence one can add a copy statements `t := d` just after `d := a - b` in B_1 and replace `b := a - b` and `a := a - b` by `b := t` in B_3 and `a := t` in B_4 respectively.

5. Then `b := t` in B_3 can be propagated to `b := - b` leading to the transformation `b := -t` and the elimination of `b := t` in B_3.

Exercise 3. Translate the following C-like piece of code into three-address code.

```
a = x * y + u * v - w * t;
b = w * t - m * n;
while (b != 0) {
   tmp = a;
   a = b;
   while (tmp > b) {
      tmp = tmp - b;
   }
   b = tmp;
}
```

The three-address code instructions you may need are of one of the forms:

- `goto L` where is a symbolic label (like `L1, L2, ...`)
- `if x relop y goto L` where `relop` can be = or <
- `x:= y op z` where `op` can be −, − and `*`

Moreover your solution does **not** need to be obtained by one of the algorithms of the course. Just consider that you are programming a very basic calculator.

Answer 3

See Quiz 4 of last year.