Compilers

Spring 2023

Code Optimizations
(Applications of Data-Flow and Control-Flow Analysis)

Sample Exercises and Solutions

Prof. Pedro C. Diniz

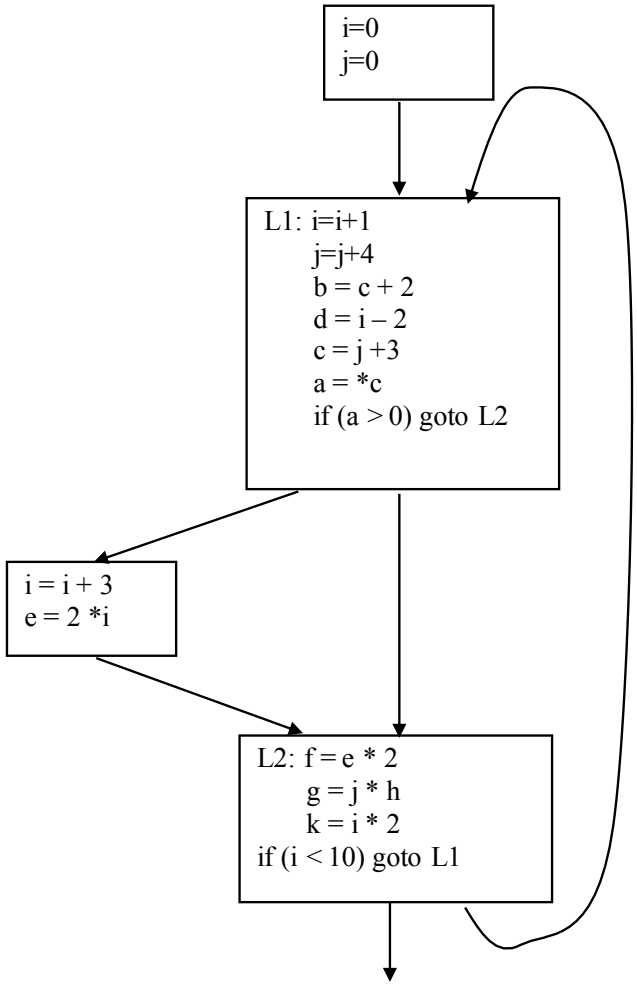
Faculdade de Engenharia da Universidade do Porto
Departamento de Engenharia Informática
pedrodiniz@fe.up.pt

Problem 1

For the program depicted below do the following:

1. Find the induction variables in this program. Show the results after applying strength reduction and loop test replacement.
2. Show the resulting code from applying loop unrolling with an unrolling factor of 2.

Note: The "a = *c" assigns the contents of the memory location pointed to by c to a.



Solution:

Assuming that there is no alias among the variables, so h is not changed within the loop. Basic induction variables: {i, j} Induction variable families: {{i, d, e, k}, {j, g, c}} (We can consider h as a constant within the loop, and as a result, g will be an induction variable in the same family with j) Introduce s1 for d, s2 for e and k, s3 for g, s4 for c, and eliminate basic induction variables i and j because they are only used for calculation of other induction variables and loop test. After strength reduction and loop test replacement:

```
    s1 = -2
    s2 = 0
    s3 = 0
    t3 = 4 * h
    s4 = 3

L1: s1 = s1 + 1
    s2 = s2 + 2
    s3 = s3 + t3
    s4 = s4 + 4
    b = c + 2
    d = s1
    c = s4
    a = *c
    if a > 0 goto L2

    s1 = s1 + 3
    s2 = s2 + 6
    e = s2

L2: f = e * 2
    g = s3
    k = s2
    if s1 < 8 goto L1

    i = d + 2
    j = c – 3
```

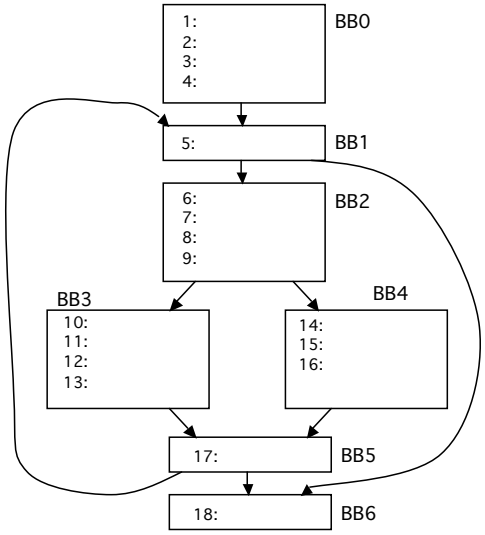
1) After loop unrolling:

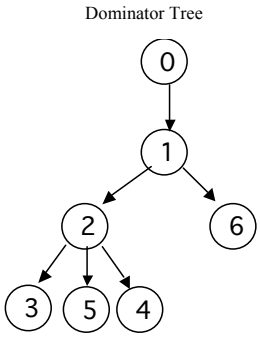
```
    s1 = -2
    s2 = 0
    s3 = 0
    t3 = 8 * h
    s4 = 3

    if s1 >= 4 goto L4
L1: s2 = s2 + 2
    s4 = s4 + 4
    c = s4
    a = *c
    if a > 0 goto L2

    s1 = s1 + 3
    s2 = s2 + 6
    e = s2

L2: s1 = s1 + 2
    s2 = s2 + 2
    s3 = s3 + t3
    b = c + 2
    d = s1

    s4 = s4 + 4
    c = s4
    a = *c
    if a > 0 goto L3

    s1 = s1 + 3
    s2 = s2 + 6
    e = s2

L3: f = e * 2
    g = s3
    k = s2

    if s1 < 4 goto L1

L4: s1 = s1 + 1
    s2 = s2 + 2
    s3 = s3 + t3
    s4 = s4 + 4
    b = c + 2
    d = s1
    c = s4
    a = *c
    if a > 0 goto L5

    s1 = s1 + 3
    s2 = s2 + 6
    e = s2

L5: f = e * 2
    g = s3
    k = s2
    if s1 < 8 goto L4

    i = d + 2
   j = c –
```

Problem 2

Consider the code depicted below for a function with integer local variables i, a, b, c, d and an integer input parameters p and n.

```
int i, a, b, c, d;

 1:     i = 0;
 2:     a = 1;
 3:     b = i * 4;
 4:     c = a + b;
 5: L1: if(i > n) goto L2
 6:     c = a + 1;
 7:     i = i + 1;
 8:     b = i * 4;
 9:     if (c <= p)goto L3
10:     e = 1;
11:     c = e - b;
12:     a = e;
13:     goto L4
14: L3: d = 2;
15:     c = d + b;
16:     a = d;
17: L4: goto L1
18: L2: return;
```





For this code fragment determine:

1. The Control Flow Graph (CFG).
2. The dominator tree and the loops (identify the back edges).
3. Use constant propagation to the statements in the body of the loop identified in (2).
4. Are the statements in line 8 and 16 loop invariants? Explain and describe where can they be moved to if they can in fact be moved outside any of the loops.

Solution:

1. See the CFG on the left above. In the CFG each node is a basic block. Recall that a basic block is a maximal sequence of instructions such that if the first instruction of the sequence is executed so are all the instructions in the basic block. Conversely if the first instruction is not executed none of the instructions in the basic block are executed.
2. See the dominator tree on the right above. In this tree an edge indicates the notion of immediate dominance. Recall that a node p dominates another node q if and only if every path from the entry node of the code to node q has to go through node p.
3. Back edge is (5,1) since the node pointed to by its head dominates the node pointed to by its tail. The natural loop is (1,2,3,4,5). The basic blocks in this natural loop are found tracing back the back-edge against the control flow until the node pointed to by the head. In this case and regarding the edge (5,1) we would find nodes 3 and 4, then node 2 and finally node 1. The basic blocks of the natural loop induced by the back edge (5,1) would then be {1,2,3,4,5}.
4. Could remove the statement `a = p1` to the head of the loop. This is only valid because a is not live at the end of the loop so even if the original loop were never to execute and since the head of the loop would execute once this transformation would still be correct.

Problem 3

For the code below apply the following code transformations: constant propagation, constant folding, copy-propagation, dead-code elimination and strength reduction.

```
          t1 = t1 + 1
L0:       t2 = 0
          t3 = t1 * 8 + 1
          t4 = t3 + t2
          t5 = t4 * 4
          t6 = *t5
          t7 = FP + t3
          *t7 = t2
          t8 = t1
          if (t8 > 0) goto L1
L1:       goto L0
L2:       t1 = 1
          t10 = 16
          t11 = t1 * 2
          goto L1
```

Solution:

The code in red is unreachable and can thus be eliminated. The goto L1 in the end is a goto to another goto instruction and thus can be converted into a goto L0 instruction.

```
          t1 = t1 + 1
L0:       t2 = 0
          t3 = t1 * 8 + 1
          t4 = t3 + t2
          t5 = t4 * 4
          t6 = *t5
          t7 = FP + t3
          *t7 = t2
          t8 = t1
          if (t8 > 0) goto L0
L1:       goto L0
L2:       t1 = 1
          t10 = 16
          t11 = t1 * 2
          goto L1
```

after dead-code elimination

```
          t1 = t1 + 1
L0:       t2 = 0
          t3 = (t1 << 3) + 1
          t4 = t3
          t5 = (t4 << 2)
          t6 = *t5
          t7 = FP + t3
          *t7 = t2
          t8 = t1
          if (t8 > 0) goto L0
```

after symbolic propagation for (t2 = 0) and strength reduction of the various multiplications by 4 and 8.

```
          t1 = t1 + 1
L0:       t2 = 0
          t3 = (t1 << 3) + 1
          t4 = t3
          t5 = (t3 << 2)
          t6 = *t5
          t7 = FP + t3
          *t7 = t2
          t8 = t1
          if (t1 > 0) goto L0
```

after copy propagation; now we can eliminate the variables t8 and t4 and t2 as they are never used in this code.

In reality you can also see that t1 is now loop invariant and so is the sub-expression `(t1 <<3) + 1` which can now be hoisted outside the loop. Along the same reasoning t5 is also loop invariant and so is everything else. This is a very uninteresting example.

Problem 4

Consider the 3-address code depicted below already with specific register allocation and assignments.

```
01:      r0 = 0
02:      r1 = $fp - 4
03:      r1 = *r1
04:      r2 = 0
05:      r4 = 1
06:      r5 = r1
07: L1:  if r0 = r1 goto L5
08:      if r2 = r5 goto L3
09:      goto L2
10: L2:  r3 = 1
11:      r0 = r0 + 1
12:      goto L6
13: L3:  r4 = 0
14: L4:  goto L7
15: L5:  r3 = 1
16:      r0 = r0 + 1
17:      r4 = 0
18: L6:  r4 = r0 * 2
19:      goto L8
20: L7:  r1 = FP + 4
21:      *r1 = r4
22:      r1 = $gp + 20
23:      *r1 = r5
24:      return
25: L8:  r5 = r5 - r3
26:      if r5 != 0 goto L1
27:      r4 = 1
28:      goto L7
```

For this code fragment determine:

a. [10 points] The corresponding Control Flow Graph (CFG) identifying the basic blocks with their instructions.

b. [10 points] The dominator tree and the natural loops (identify the back edges). The dominator tree is a simple relationship between nodes. A node in this tree is directly connected to the nodes it immediately dominates.

c. [10 points] Check and transform the code taking advantage of loop-invariant code motion arguing about the correctness of your transformations.

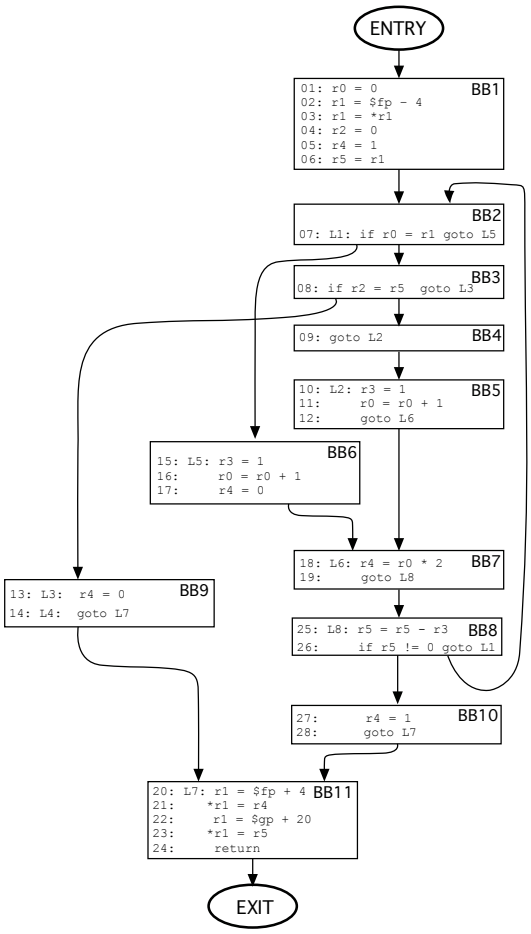
d. [10 points] Check and transform the code taking advantage of induction variables in this loop arguing about the correctness of your transformations.

Solution

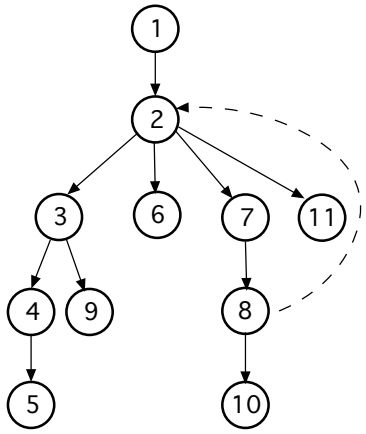
a) The control-flow graph is as shown below on the left-hand-side.

b) The dominator tree is shown on the right-hand side. The natural loop induced by the back-edge (8-2). Tracing backwards this edge from the basic block BB8 towards BB2 we encounter the basic blocks {2, 3, 4, 5, 6, 7, 8}

c) There are really no loop-invariant instructions in the only loop in this code. The only candidate instruction which one could move to the head of the loop would be the instructions "r3 = 1" in the basic blocks BB5 and BB6. Unfortunately, none of these basic blocks by themselves dominate the basic blocks where there are uses of the variable r3 in BB8. Similarly, we can see that the assignment " r4 = 0" in BB6 would be a candidate for loop invariant code motion. Again, this statement does not dominate the exit of the loop (BB2). In fact that are other assignment statement to variable r4.



Control-Flow Graph (CFG)



loop = { 2, 3, 4, 5, 6, 7, 8 }

Dominator Tree, Back-edge and Natural Loop

d) There are some opportunities for induction variables in this loop, which are not trivial to grasp, and for which the help of data-flow analysis (as seen in the next problem) are required. There is an increment on r0 by 1 at every iteration of the loop although the increment occurs in different basic blocks. At each iteration of the loop, however, only one of them is executed. This means that r0 is a *basic induction variable*. As such the variable r4 is a *derived* induction variable with increment by 2. The fact that we have an assignment to r4 in basic block BB6 is irrelevant as that assignment is dead in the sense that is immediately overwritten in BB7.