

Homework 1 Answers

CS 322 Compiler Construction
Winter Quarter 2006

Problem 1

```

    m := 0                <-- LEADER
    i := 0
L1:  if i<n == false goto L2  <-- LEADER
    s := 0                <-- LEADER
    j := i
L3:  if j<n == false goto L4  <-- LEADER
    x := j                <-- LEADER
    s := s+x
    if s>m == false goto L5
    m := s                <-- LEADER
L5:  // end of if-statement
    j := j+1              <-- LEADER
    goto L3
L4:  // end of inner for-loop
    i := i+1              <-- LEADER
    goto L1
L2:  // end of outer for-loop
    return m              <-- LEADER
```

The reason some labels are on lines of their own is to emphasize the fact that they are not generated at the same time as the code that follows them. Think about how an IR code generator would work. Given a for-statement, it would first emit code for the initializer (`i:=0`). Then, it would generate a label (`L1:`) and emit the condition (`if i<n==false`). If the condition is false, control will continue at some other point, so it should generate a label for that destination and emit a goto statement (`goto L2`). Then, go on to generate code for the body of the loop. At the end of that, place the destination label (`L2:`). What comes next is the code for the statement following the loop and will be generated separately. See the SDT for the while-loop in problem 2 for more info. This is very similar to what you will have to do in PA4.

The first step in the SSA conversion is to find out in which blocks each variable is defined and insert a ϕ -function in the dominance frontiers of those blocks. The ϕ -functions are also definitions of the variable.

This information is shown in figure 2.

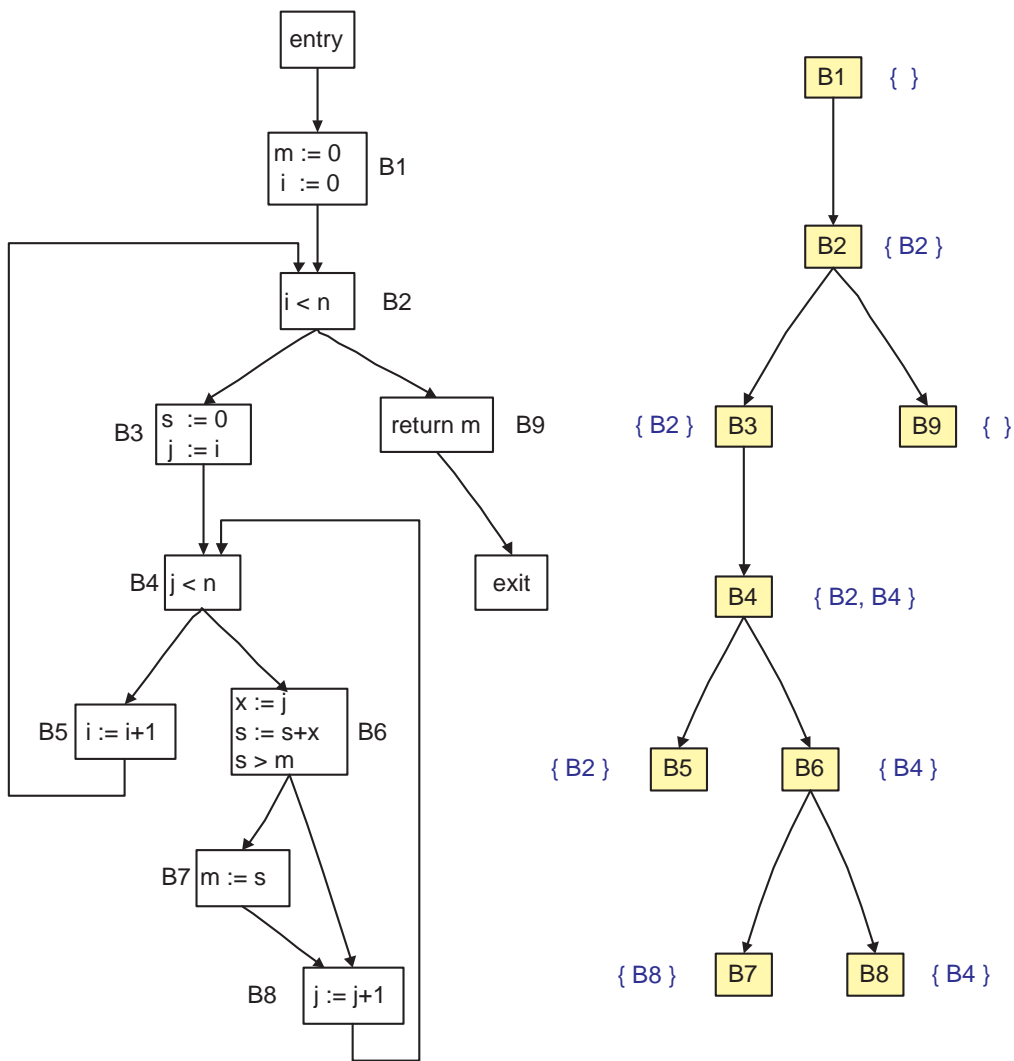


Figure 1: *The CFG and dominator tree. The nodes in the dominator tree are labeled with their dominance frontiers.*

Variable	BB where it is defined	BB with ϕ -functions
m	B1, B7, B8, B4, B2	B8, B4, B2
i	B1, B5, B2	B2
s	B3, B2, B6, B4,	B2, B4
j	B3, B2, B8, B4	B2, B4
x	B6, B4, B2	B4, B2

Figure 2: *Finding where ϕ -functions should be placed*

After inserting the ϕ -functions, we need to rename. The final result is shown in figure 3.

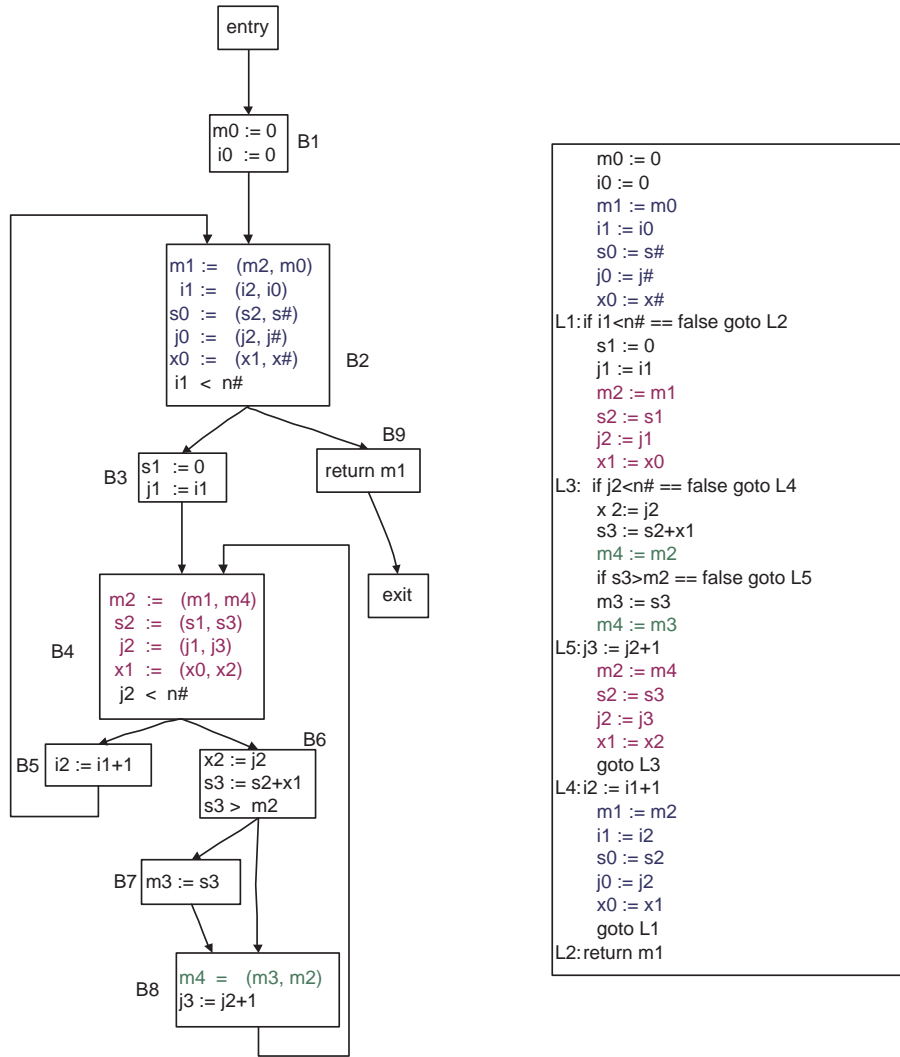


Figure 3: *Final SSA Form and conversion back to MIR*

Note how each argument i of a ϕ -function is taken from the corresponding predecessor i as ordered in the figure. Furthermore, n doesn't have an initial definition, so we use the pound symbol as a subscript.

The conversion from SSA back to MIR is shown in figure 3. $m4 := m2$ provides an interesting challenge. Technically, it should be at the end of the block, right after the evaluation of $s3 > m2$ and before the `goto`. Nevertheless, its current location is safe. A better representation would have created a temporary variable to hold the result of the test and then used that variable in a branching statement.

Problem 2

In the SDT that follows, \parallel stands for “concatenate” and $newTemp()$ is a function that generates a new temporary variable every time it is called.

The SDT follows.

```
< S >  →  while < E > do { < S1 > }
{
S.begin =  genLabel()
S.end   =  genLabel()
S.code  =  S.begin —— “.”
          || E.code
          || “ if ” || E.temp || “ == FALSE goto ” || S.end
          || S1.code
          || “ goto ” || S.begin
          || S.end || “.”
}
```

```
< S >  →  if < E > then { < S1 > }
{
S.end   =  genLabel()
S.code  =  E.code
          || “ if ” || E.temp || “ == FALSE goto ” || S.end
          || S1.code
          || S.end || “.”
}
```

```
< S >  →  < S1 > ; < S2 >
{
S.code  =  S1.code || S2.code
}
```

```
< S >  →  < V > := < E >
{
S.code  =  E.code
          || V.name || “:=” || E.temp
}
```

```
< E >  →  < E1 > == < E2 >
{
E.temp  =  newTemp()
E.code  =  E1.code || E2.code
          E.temp || “:=” || E1.temp “==” E2.temp
}
```

```

< E >  →  < E1 > + < E2 >
{
E.temp  =  newTemp()
E.code  =  E1.code || E2.code
          E.temp || “:=” || E1.temp “+” E2.temp
}

< E >  →  id
{
E.temp  =  id.name
E.code  =  “”
}

```

Now let us consider the addition of a break statement and how this will affect the code. A sample syntax tree for a possible nested-loop configuration is shown in figure 4.

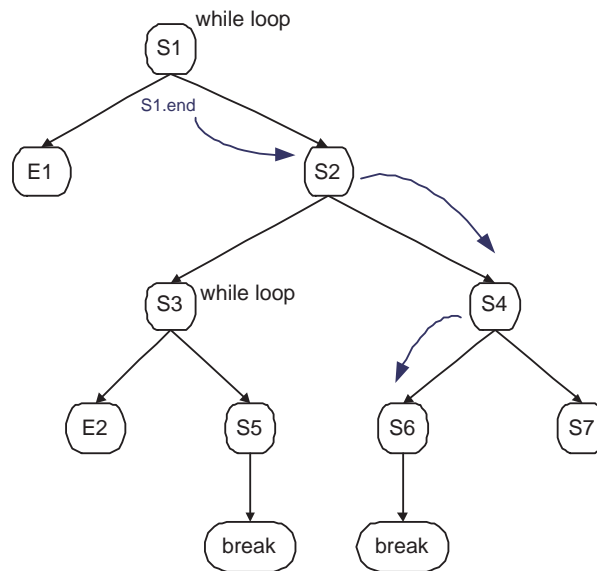


Figure 4: The blue arrows show how the value of attribute $S_1.end$ needs to travel in the parse tree

The first break statement (S5) should generate a branch just outside loop S3. The label used in this goto statement is S3.end. Things become more interesting when we consider the other break (S6). This should generate a `goto S1.end`. The label S1.end was created when we started generating code for S1, yet now we need it in S6. This piece of information needs to pass from S1 to S2 to S4 to S6. Clearly, it's an inherited attribute that will have to be passed along all kinds of statements that may consist of other statements (eg. $S \rightarrow S S$).

Below are the rules we need to add to the existing SDT:

$\langle S \rangle \rightarrow \text{while } \langle E \rangle \text{ do } \{ \langle S_1 \rangle \}$
{
 $S_1.in = S.end$
}

$\langle S \rangle \rightarrow \text{if } \langle E \rangle \text{ then } \{ \langle S_1 \rangle \}$
{
 $S_1.in = S.in$
}

$\langle S \rangle \rightarrow \langle S_1 \rangle ; \langle S_2 \rangle$
{
 $S_1.in = S.in$
 $S_2.in = S.in$
}

$\langle S \rangle \rightarrow \text{break}$
{
 $S.code = \text{"goto"} \parallel S.in$
}