

198:515 Programming Languages and Compilers I

Problem Set 3

Problem 1 - Attribute Grammars and Syntax-Directed Translation Schemes

The following context-free grammar describes a simple imperative programming language. As a matter of notation, non-terminals are in **CAPITALS**, and terminals are in **lower case**. “id” represents an identifier and “const” represents an integer constant.

```
PROGRAM ::= procedure STMT_LIST
STMT_LIST ::= STMT ; STMT_LIST
           | STMT
STMT ::= FOR_STMT
      | IF_STMT
      | A_STMT
      | READ_STMT
      | WRITE_STMT
FOR_STMT ::= for id := const to const begin STMT_LIST end
IF_STMT ::= if EXPR > 0 then STMT_LIST else STMT_LIST endif
A_STMT ::= id := EXPR
READ_STMT ::= read(id)
WRITE_STMT ::= write(EXPR)
EXPR ::= EXPR + EXPR
      | EXPR * EXPR
      | id
      | const
```

1. Formally specify an attribute grammar that determines the maximal and minimal number of variable references that will occur during the program’s execution. All non-terminal symbols in the grammar have the integer-valued attribute `count` associated with them.

The token `const` has the attribute `val` which is a synthesized attribute and assigned by the scanner. It contains the integer value of the constant.

The number of times that the body of a `for-loop` executes can be computed from the two constants that specify the range of the loop variable.

Example:

```
procedure
  read(a);           // 1 ref
  a := b + 5;       // 2 refs
  if a > 0 then     // 1 ref
    c := a + b;     // 3 refs
  else
    c := 3;         // 1 ref
  endif;
  write(c)           // 1 ref
  for a := 1 to 2   // 1 ref
  begin
    write(a)        // 2 * 1 = 2 refs
  end
```

The above `if` statement has at most 4 variable references, one for the test and three for the true branch, and at least 2 variable references, one for the test and one for the false branch. The `for` statement executes its body exactly twice, resulting in two variable references in its body. Overall, the program has at most 11 variable references, and at least 9 variable references.

2. Define a syntax-directed translation scheme for the attribute grammar. Use a YACC-like notation to access the attribute instances.

Problem 2 - Common Subexpression Elimination

What's the problem when you apply the common subexpression elimination algorithm for basic blocks on the following code?

```
a = c + 1
c = q + 2
q = a
```

Any idea how to fix the problem?