

## CS 515 Problem Set 3 Sample Solution

### Problem 1 — Attribute Grammars and Syntax-Directed Translation Schemes

1. Formal attribute grammar.

PRODUCTION	SEMANTIC RULES
$PROGRAM \rightarrow \text{procedure } STMT\_LIST$	$PROGRAM.min := STMT\_LIST.min$ $PROGRAM.max := STMT\_LIST.max$
$STMT\_LIST \rightarrow STMT ; STMT\_LIST_1$	$STMT\_LIST.min := STMT.min + STMT\_LIST_1.min$ $STMT\_LIST.max := STMT.max + STMT\_LIST_1.max$
$STMT\_LIST \rightarrow STMT$	$STMT\_LIST.min := STMT.min$ $STMT\_LIST.max := STMT.max$
$STMT \rightarrow FOR\_STMT$	$STMT.min := FOR\_STMT.min$ $STMT.max := FOR\_STMT.max$
$STMT \rightarrow IF\_STMT$	$STMT.min := IF\_STMT.min$ $STMT.max := IF\_STMT.max$
$STMT \rightarrow A\_STMT$	$STMT.min := A\_STMT.min$ $STMT.max := A\_STMT.max$
$STMT \rightarrow READ\_STMT$	$STMT.min := READ\_STMT.min$ $STMT.max := READ\_STMT.max$
$STMT \rightarrow WRITE\_STMT$	$STMT.min := WRITE\_STMT.min$ $STMT.max := WRITE\_STMT.max$
$FOR\_STMT \rightarrow \begin{array}{l} \text{for id := const to const} \\ \text{begin } STMT\_LIST \text{ end} \end{array}$	$FOR\_STMT.min := 1 + (\text{const.val} - \text{const.val}) * STMT\_LIST.min$ $FOR\_STMT.max := 1 + (\text{const.val} - \text{const.val}) * STMT\_LIST.max$
$IF\_STMT \rightarrow \begin{array}{l} \text{if } EXPR > 0 \\ \text{then } STMT\_LIST_1 \\ \text{else } STMT\_LIST_2 \text{ endif} \end{array}$	$IF\_STMT.min := EXPR.min + \min(STMT\_LIST_1.min, STMT\_LIST_2.min)$ $IF\_STMT.max := EXPR.max + \max(STMT\_LIST_1.max, STMT\_LIST_2.max)$

*continued on next page*

continued from previous page

PRODUCTION	SEMANTIC RULES
$A\_STMT \rightarrow id := EXPR$	$A\_STMT.min := 1 + EXPR.min$ $A\_STMT.max := 1 + EXPR.max$
$READ\_STMT \rightarrow read(id)$	$READ\_STMT.min := 1$ $READ\_STMT.max := 1$
$WRITE\_STMT \rightarrow write(EXPR)$	$WRITE\_STMT.min := EXPR.min$ $WRITE\_STMT.max := EXPR.max$
$EXPR \rightarrow EXPR_1 + EXPR_2$	$EXPR.min := EXPR_1.min + EXPR_2.min$ $EXPR.max := EXPR_1.max + EXPR_2.max$
$EXPR \rightarrow EXPR_1 * EXPR_2$	$EXPR.min := EXPR_1.min + EXPR_2.min$ $EXPR.max := EXPR_1.max + EXPR_2.max$
$EXPR \rightarrow id$	$EXPR.min := 1$ $EXPR.max := 1$
$EXPR \rightarrow const$	$EXPR.min := 0$ $EXPR.max := 0$

2. Syntax-directed translation scheme for attribute grammar.

```

PROGRAM ::= procedure STMT_LIST
        { $$ .min = $2 .min; $$ .max = $2 .max; }
STMT_LIST ::= STMT ; STMT_LIST
        { $$ .min = $1 .min + $3 .min; $$ .max = $1 .max + $3 .max; }
        | STMT
        { $$ .min = $1 .min; $$ .max = $1 .max; }
STMT ::= FOR_STMT
        { $$ .min = $1 .min; $$ .max = $1 .max; }
        | IF_STMT
        { $$ .min = $1 .min; $$ .max = $1 .max; }
        | A_STMT
        { $$ .min = $1 .min; $$ .max = $1 .max; }
        | READ_STMT
        { $$ .min = $1 .min; $$ .max = $1 .max; }
        | WRITE_STMT
        { $$ .min = $1 .min; $$ .max = $1 .max; }
FOR_STMT ::= for id := const to const begin STMT_LIST end
        { $$ .min = 1 + ($6 .val - $4 .val) * $8 .min;
          $$ .max = 1 + ($6 .val - $4 .val) * $8 .max; }
IF_STMT ::= if EXPR > 0 then STMT_LIST else STMT_LIST endif
        { $$ .min = $2 .min + min ($6 .min, $8 .min);
          $$ .max = $2 .max + max ($6 .max, $8 .max); }
A_STMT ::= id := EXPR
        { $$ .min = 1 + $3 .min; $$ .max = 1 + $3 .max; }
READ_STMT ::= read(id)
        { $$ .min = 1; $$ .max = 1; }

```

```

WRITE_STMT ::= write(EXPR)
            { $$ .min = $3.min; $$ .max = $3.max; }
EXPR       ::= EXPR + EXPR
            { $$ .min = $1.min + $3.min; $$ .max = $1.max + $3.max; }
            | EXPR * EXPR
            { $$ .min = $1.min + $3.min; $$ .max = $1.max + $3.max; }
            | id
            { $$ .min = 1; $$ .max = 1; }
            | const
            { $$ .min = 0; $$ .max = 0; }

```

## Problem 2 — Common Subexpression Elimination

The problem with the “standard” CSE algorithm is that it does not consider order restrictions due to dependences. Statement “ $a = c + 1$ ” has to be executed before statement “ $c = q + 2$ ” due to a write-after-read dependence on variable  $c$ . Statement “ $c = q + 2$ ” has to be executed before statement “ $q = a$ ” due to a write-after-read dependence on variable  $q$ .

This situation can be handled by removing these write-after-read dependences through renaming (see Figure 1):

```

a1 = c0 + 1
c1 = q0 + 2
q1 = a1

```

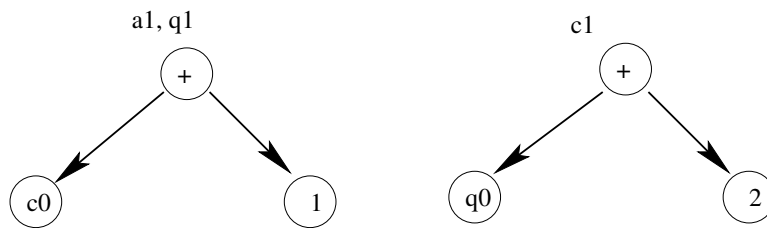


Figure 1: The resulting DAG.

Note: renaming may require additional code to be introduced to copy the initial value of a variable “ $x$ ” into “ $x_0$ ” at the beginning of the basic block, and if  $x$  is used after exit of the block, i.e., in some part of the program that can be reached after the basic block is exited, the final value of  $x$  has to be restored, for instance through “ $x = x_i$ ”, where “ $x_i$ ” is the final renamed variable of  $x$  in the basic block.