CS335: COMPILER DESIGN - Assignment 3

Harsh Bihany (210406)

Problem 1

Consider the following SDD, where V, W, X, Y, and Z are non-terminals, where V is the starting non-terminal. The attribute val in the semantic rules represents a numeric value.

$V \rightarrow Y\#W$	$\{V.val = W.val \% Y.val\}$
$W \rightarrow X@Y$	$\{W.val = X.val + Y.val\}$
$X \rightarrow X_1 Z$	$\{X.val = X_1.val + 3 \times Z.val\}$
$X \rightarrow Z$	$\{X.val = Z.val\}$
$Y \rightarrow ZY_1$	$\{Y.val = 2 \times (Z.val + Y_1.val)\}$
$Y \rightarrow Z$	$\{Y.val = 3 \times Z.val\}$
$Z \rightarrow 3$	$\{Z.val = 3\}$
$Z \rightarrow 4$	$\{Z.val = 4\}$

(i) Show the annotated parse tree for the input 43#43@443.

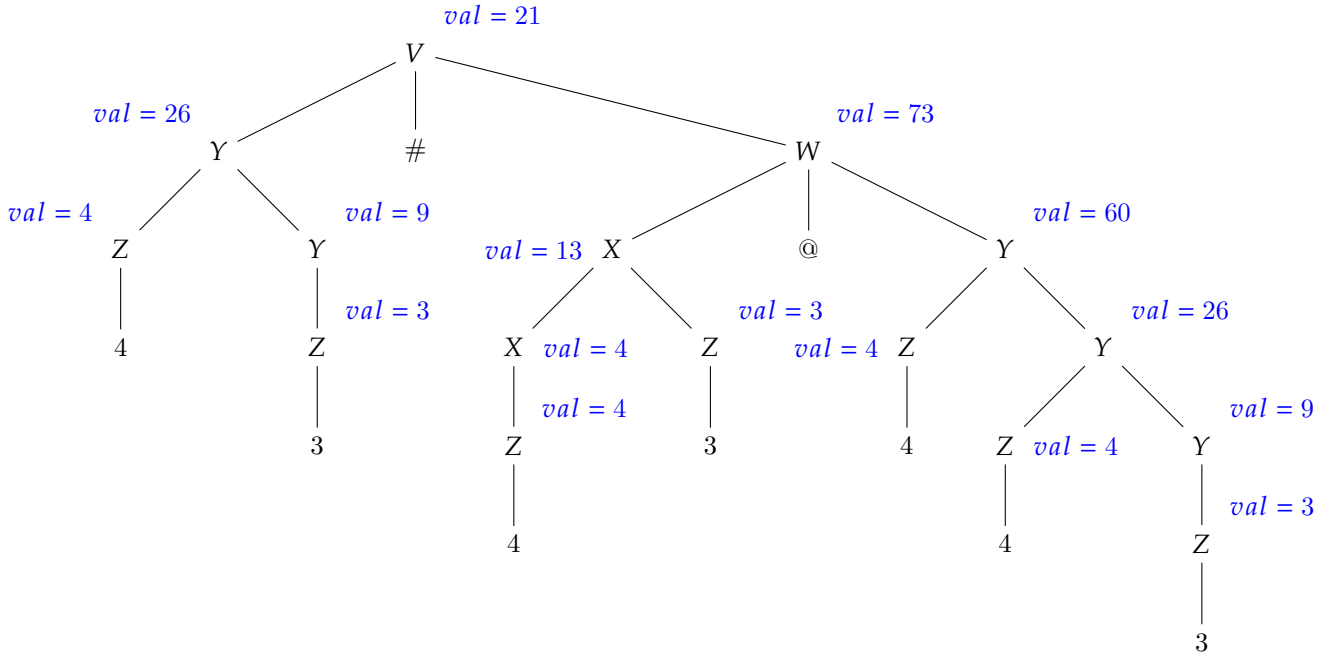

Figure 1.1: Annotated Parse Tree: Problem 1

(ii) What is the value at V computed by the translation scheme for the above input string.

As shown in the figure the value of V is 21.

(iii) Explain whether the grammar is S-attributed or L-attributed.

Each semantic rule associated with a production $(A \rightarrow X_1 \dots X_n)$ is of the form $A.s = f(X_1.s, \dots, X_n.s)$. Hence it is a S-attributed grammar. But since each S-attributed grammar is L-attributed as well, implies the grammar is L-attributed.

Problem 2

We have discussed generating 3AC for array accesses using semantic translations. Consider the following extended grammar with semantic translation.

$$
\begin{array}{cl}
S \rightarrow \textbf{id} = E & \{gen(symtop.get(id.lexeme) \text{ “ = ” } E.addr)\} \\
S \rightarrow L = E & \{gen(L.array.base \text{ “[” } L.addr \text{ “]” “ = ” } E.addr)\} \\
E = E_1 - E_2 & \{E.addr = \textbf{new}Temp();\ gen(E.addr \text{ “ = ” } E_1.addr \text{ “ − ” } E_2.addr)\} \\
E = E_1 / E_2 & \{E.addr = \textbf{new}Temp();\ gen(E.addr \text{ “ = ” } E_1.addr \text{“/”} E_2.addr)\} \\
E \rightarrow \textbf{id} & \{E.addr = symtop.get(\textbf{id}.lexeme)\} \\
E \rightarrow L & \{E.addr = \textbf{new}Temp();\ gen(E.addr \text{ “ = ”} L.array.base \text{ “[”} L.addr \text{ “]”})\} \\
E \rightarrow *E_1 & \{E.addr = \textbf{new}Temp();\ gen(E.addr \text{ “ = ” “ * ” } E_1.addr)\} \\
L \rightarrow \textbf{id}[E] & \{L.array = symtop.get(\textbf{id}.lexeme);\ L.type = L.array.type.elem; \\
 & L.addr = \textbf{new}Temp();\ gen(L.addr \text{ “ = ” } E.addr \text{ “ * ” } L.type.width)\} \\
L \rightarrow L_1[E] & \{L.array = L1.array;\ L.type = L1.type.elem; \\
 & t = \textbf{new}Temp();\ L.addr = \textbf{new}Temp(); \\
 & gen(t \text{ “ = ” } E.addr \text{ “ * ” } L.type.width);\ gen(L.addr \text{ “ = ” } L1.addr \text{ “ + ” } t);\}
\end{array}
$$

Assume the size of integers to be four bytes, and that the arrays are zero-indexed. Let A, B, and C be integer arrays of dimensions 11×8, 12×6, and $10 \times 10 \times 6$, respectively. Construct an annotated parse tree for the expression $C[i][j][k] - A[i][k]/B[i][j]$ and show the 3AC code sequence generated for the expression.

The 3AC generated is:

```
t1 = i * 240
t2 = j * 24
t3 = t1 + t2
t4 = k * 4
t5 = t3 + t4
t6 = C[t5]
t7 = i * 32
t8 = k * 4
t9 = t7 + t8
t10 = A[t9]
t11 = i * 24
t12 = j * 4
t13 = t11 + t12
t14 = B[t13]
t15 = t10 / t14
t16 = t6 - t15
```

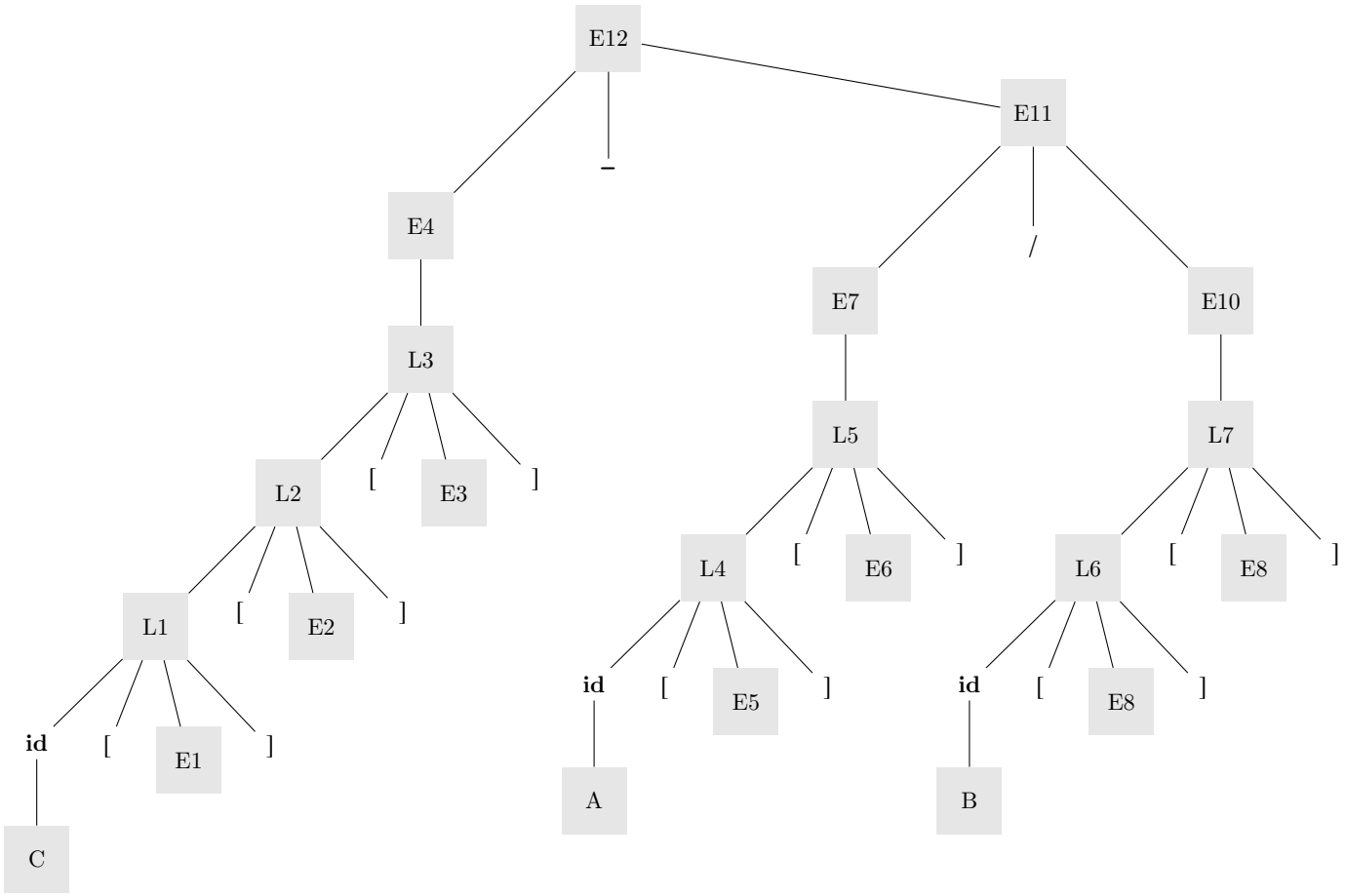

Figure 2.1: Annotated Parse Tree: Problem 2

The associated attributes are depicted in the following table 2.1:

C	C.type = array(10, array(10, array(6, integer)))
L1	L.array = C L.type = array(10, array(6, integer)) L.addr = t1
E1	E.addr = i
L2	L.array = C L.type = array(6, integer) L.addr = t3
E2	E.addr = j
L3	L.array = C L.type = integer L.addr = t5
E3	E.addr = k
E4	E.addr = t6
A	A.type = array(11, array(8, integer))
E5	E.addr = i
L4	L.array = A L.type = array(8, integer) L.addr = t7
E6	E.addr = k
L5	L.array = A L.type = integer L.addr = t9
E7	E.addr = t10;
B	B.type = array(12, array(6, integer))
E8	E.addr = i
L6	L.array = B L.type = array(6, integer) L.addr = t11
E6	E.addr = j
L7	L.array = B L.type = integer L.addr = t13
E10	E.addr = t14
E11	E.addr = t15

Table 2.1: Attributes of the nodes

Problem 3

Construct an SDT translation scheme for array expressions using column-major organization of arrays. Use the following grammar.

$$
\begin{aligned}
S &\rightarrow \textbf{id} = E \\
S &\rightarrow L = E \\
E &\rightarrow E_1 + E_2 \\
E &\rightarrow L \\
L &\rightarrow \textbf{id} \\
L &\rightarrow \textbf{id}\,[\ Elist \\
Elist &\rightarrow E\ ] \\
Elist &\rightarrow E, Elist
\end{aligned}
$$

(i) Show the semantic actions for your proposed translation.

The semantic actions for the column major form of array reference is as shown below

$S \rightarrow \textbf{id} = E$	$gen(\textbf{id}.lexeme\text{“}=\text{”}E.addr)$
$S \rightarrow L = E$	$\texttt{if}(L.offt)\ gen(L.addr\text{“[”}L.offt\text{“]”“}=\text{”}E.addr$ $\texttt{else}\ gen(L.addr\text{“}=\text{”}E.addr)$
$E \rightarrow E_1 + E_2$	$E.addr = \textbf{new}\ Temp();$ $gen(E.addr = E_1.addr + E_2.addr)$
$E \rightarrow L$	$E.addr = L.offt == \texttt{nullptr}?L.addr : \textbf{new}\ Temp();$ $\texttt{if}\ (L.offt)\ gen(E.addr = L.addr\text{“[”}L.offt\text{“]”})$
$L \rightarrow \textbf{id}$	$L.addr = symtop.get(\textbf{id}.lexeme);$ $L.offt = \texttt{nullptr};$
$L \rightarrow \textbf{id}\ [\ Elist$	$currId.push(\textbf{id});$ $L.addr = \textbf{new}\ Temp();$ $L.offt = \textbf{new}\ Temp();$ $gen(L.addr\text{“}=\text{”}baseAddr(\textbf{id}));$ $t = \textbf{new}\ Temp();$ $gen(t = Elist.addr\text{“}*\text{”}widthBase(\textbf{id}));$ $gen(L.offt = L.addr\text{“}+\text{”}t)$
$Elist \rightarrow E\ ]$	$Elist.addr = E.addr;$ $Elist.order = 1;$ $currId.pop();$
$Elist \rightarrow E, Elist_1$	$t = \textbf{new}\ Temp();$ $Elist.order = Elist_1.order + 1$ $gen(t\text{“}=\text{”}Elist_1.addr\text{“}*\text{”}numElem(Elist.order, currId.top()));$ $gen(t\text{“}=\text{”}t\text{“}+\text{”}E.addr);$ $Elist.addr = t;$

(ii) Explain the attributes and auxiliary functions in your SDT.

Here follows the description of attributes and helper functions used in the translation scheme:

(a) Attribute $offt$ is the relative offset in bytes with respect to the base address of an array on memory in colum-major form.

(b) Attribute $addr$ is the same reference to the address of a variable, constant or a compiler-generated temporary.

(c) Function gen incrementally appends the generated 3AC code and either pushes it into a storage or prints it.

(d) Symbol $symtop$ is the current Symbol-table in scope.

(e) Function $widthBase(\cdot)$ takes in a identifier and outputs the width of the base data type of the same.

(f) Function $baseAddr(\cdot)$ takes in an identifier and gives the relative offset of the identifier on the symbol table.

(g) Symbol $currId$ is a stack that keeps track of the current lexeme (the array name) whose array reference is being done.

(h) Function $numElem(\cdot)$ takes in an order, and the array/matrix (lexeme) in scope, and outputs the size of the matrix in that order (by checking the symbol table entries). Do note that this order is deliberately from the rightmost to the left most.

(iii) Show the annotated parse tree for the expression `x = c + A[i, j]`. Assume that A is a 10×20 array of integers, the size of an integer is 4 bytes, and the arrays are zero-indexed.

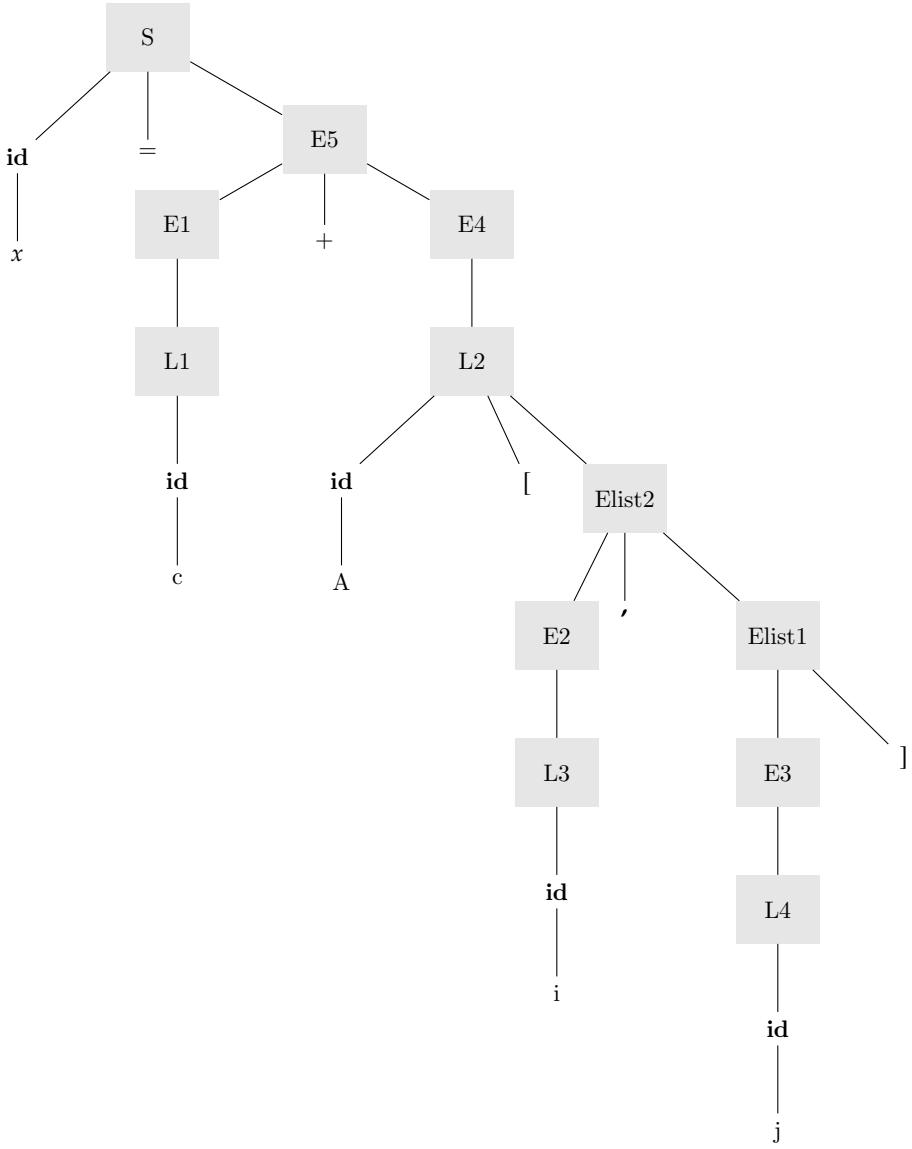

Figure 3.1: Annotated Parse Tree: Problem 3

The attributes are shown in the following table 3.1

L1	L.addr = c L.offt = nullptr;
E1	E.addr = c
L3	L.addr = i L.offt = nullptr;
E2	E.addr = i
L4	L.addr = j L.offt = nullptr;
E3	E.addr = j
Elist1	Elist.addr = j Elist.order=1
Elist2	Elist.order = 2 Elist.addr = t1
L2	L.addr = t2 L.offt = t3
E4	E.addr = t5
E5	E5.addr = t6

Table 3.1: Attributes of the nodes

(iv) Show the generated 3AC for the above expression.

```
t1 = j * 10
t1 = t1 + i
t2 = A          // base address of A
t4 = t1 * 4
t3 = t2 + t4
t5 = t2[t3]
t6 = c + t5
x = t6
```