CS 335 Semester 2023–2024-II
Assignment 3

Divyansh
(210355)

5th April 2024

Question 1

(i)

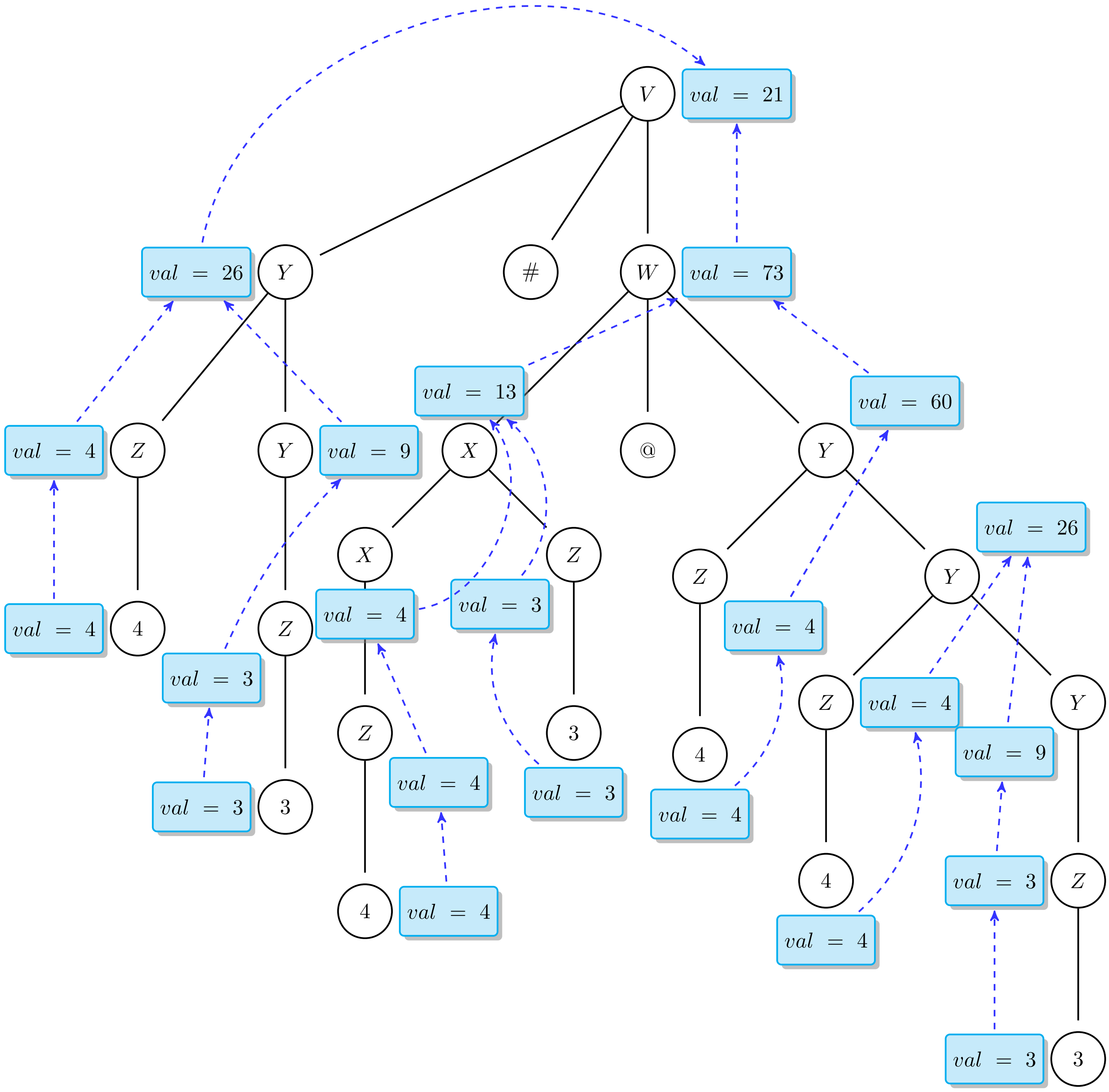
Annotated parse tree for the input string 43#43@443 is on page 2

(ii)

The Value of V computed by the translation scheme for the provided input string is `21`.

(iii)

The Grammar is S-attributed since the attribute of each of the node in the parse tree is the function of the attributes of the its children.
And, all the S-attributed grammar is L-attributed grammar.

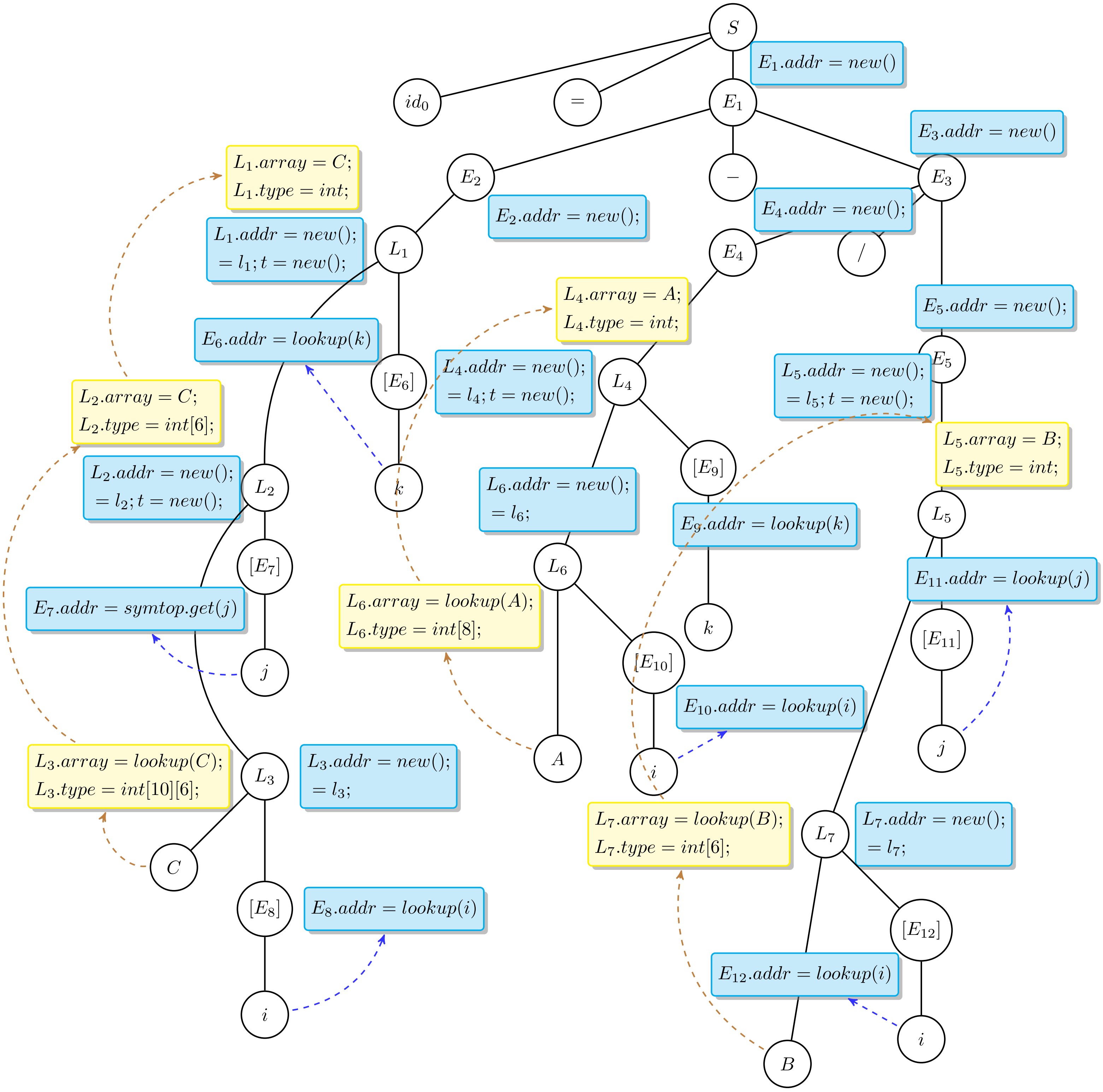

V
val = 21
val = 26
Y
#
W
val = 73
val = 13
val = 60
val = 4
Z
Y
val = 9
X
@
Y
val = 26
X
Z
Z
Y
val = 4
4
Z
val = 4
val = 3
val = 4
val = 3
Z
val = 4
Y
3
4
val = 9
val = 3
3
val = 4
val = 3
val = 4
4
val = 4
val = 3
Z
val = 4
val = 3
3

Question 2

Annotated parse for the input string $C[i][j][k] - A[i][k]/B[i][j]$ is on the page 4 and `3AC` for the same is as follows:

Note: For the sake of compactness the `new Temp()` function in the semantic action is replace with just `new()` in the annoted parse tree, similarly the `symtop.get()` function is replaced with `lookup()`. Also, some of the temporaries generated are implicit (or the numbering) in the parse tree while they have been mentioned in the generated 3AC code.

```
/* code for computing e2 */
/* Indexing C */
l_3 = i * 240;

t = j * 24;
l_2 = l_3 + t;

t = k * 4;
l_1 = l_2 + t;

e_2 = C[l_1];

/* code for computing e3 */
/* Indexing A */
l_6 = i * 32;

t = k * 4;
l_4 = l_6 + t;

e_4 = A[l_4];

/* Indexing B */
l_7 = i * 24;

t = j * 4;
l_5 = l_7 + t;

e_5 = B[l_5];

/* Arithmetic computations */
e_3 = e_4 / e_5;

e_1 = e_2 - e_3;
```


S
$E_1.addr = new()$
id_0
$=$
E_1
$E_3.addr = new()$
$L_1.array = C;$
$L_1.type = int;$
E_2
$-$
E_3
$E_2.addr = new();$
$E_4.addr = new();$
$L_1.addr = new();$
$= l_1; t = new();$
L_1
E_4
$/$
$L_4.array = A;$
$L_4.type = int;$
$E_5.addr = new();$
$E_6.addr = lookup(k)$
$[E_6]$
$L_4.addr = new();$
$= l_4; t = new();$
L_4
E_5
$L_5.addr = new();$
$= l_5; t = new();$
$L_2.array = C;$
$L_2.type = int[6];$
$L_5.array = B;$
$L_5.type = int;$
$L_2.addr = new();$
$= l_2; t = new();$
L_2
k
$L_6.addr = new();$
$= l_6;$
$[E_9]$
$E_9.addr = lookup(k)$
L_5
$[E_7]$
L_6
$E_{11}.addr = lookup(j)$
$E_7.addr = symtop.get(j)$
$L_6.array = lookup(A);$
$L_6.type = int[8];$
k
$[E_{11}]$
j
$[E_{10}]$
$E_{10}.addr = lookup(i)$
$L_3.array = lookup(C);$
$L_3.type = int[10][6];$
L_3
$L_3.addr = new();$
$= l_3;$
A
i
j
$L_7.array = lookup(B);$
$L_7.type = int[6];$
L_7
$L_7.addr = new();$
$= l_7;$
C
$[E_8]$
$E_8.addr = lookup(i)$
$[E_{12}]$
$E_{12}.addr = lookup(i)$
i
i
B

Question 3

(i)

Semantic action of the translation is as follows:

$$
\begin{array}{lr}
S \rightarrow \textbf{id} = E & \{gen(id \text{ '=' } E.addr);\} \\
S \rightarrow L = E & \{L.addr = E.addr;\} \\
E \rightarrow E_1 + E_2 & \{E.addr = new\ Temp(); \\
 & gen(E.addr \text{ '=' } E_1.addr \text{ '+' } E_2.addr);\} \\
E \rightarrow L & \{E.addr = L.addr;\} \\
L \rightarrow \textbf{id} & \{L.addr = symtop.get(id.lexeme);\} \\
L \rightarrow \textbf{id}[Elist & \{L.addr = new\ Temp(); \\
 & Elist.type = symtop.get(id).type; \\
 & Elist.order = 0; \\
 & gen(L.addr \text{ '=' } Elist.array.base \text{ '['} Elist.addr\text{']'});\} \\
Elist \rightarrow E] & \{Elist.addr = new(); \\
 & t = new\ Temp(); \\
 & gen(t = get\_dim(Elist.type, Elist.order); \\
 & gen(Elist.addr = t * E.addr);\} \\
Elist \rightarrow E, Elist_1 & \{Elist_1.array = Elist.array; \\
 & Elist_1.order = Elist.order + 1; \\
 & Elist.addr = new\ Temp(); \\
 & t_1 = new\ Temp(); \\
 & gen(t_1 = get\_dim(Elist.type, Elist.order)); \\
 & gen(Elist.addr = E.addr * t_1); \\
 & gen(t_1 = Elist_1.addr * t_1); \\
 & gen(Elist.addr = Elist.addr + t_1);\}
\end{array}
$$

(ii)

The attributes and auxiliary functions used in thise translation is very similar to the one used in question-2. Here they are:

- **Attributes** :
  - `order`: This attributes helps to know current dimension of direfrencing.
  - `addr`: stores the address where the value of corresponding node is stored.
  - `type`: stores the type of the value in the corresponding node.
  - `array`: stores the defined array to which the node corresponds.

- `type.width`: number of bytes needed to store an element of this `type`
- `type.array.base`: base address of corresponding array

- **Auxiliary Functions** :
  - `symtop.get()`: Looksup the symbol table for the passed lexeme argument and returns the address of the variable related to the lexeme.
  - `new Temp()`: Returns a temporary `3AC` address used for intermediates.
  - `gen()`: Generates `3AC` code for the corresponding production and also appends to current stream of code being generated.
  - `get_dim(type,order)`: This is returns the `order`th dimension of the `type` and `0`th order is width of the datatype. For example: if a given type is `int[3][5][7]` then
    1. `0`th order will be 4.
    2. `1`th order will be 3.
    3. `2`th order will be 5.
    4. `3`th order will be 7.

    This function helps in detemining the offset factor of the current production which helps in generating the current offset while dereferencing.

(iii)

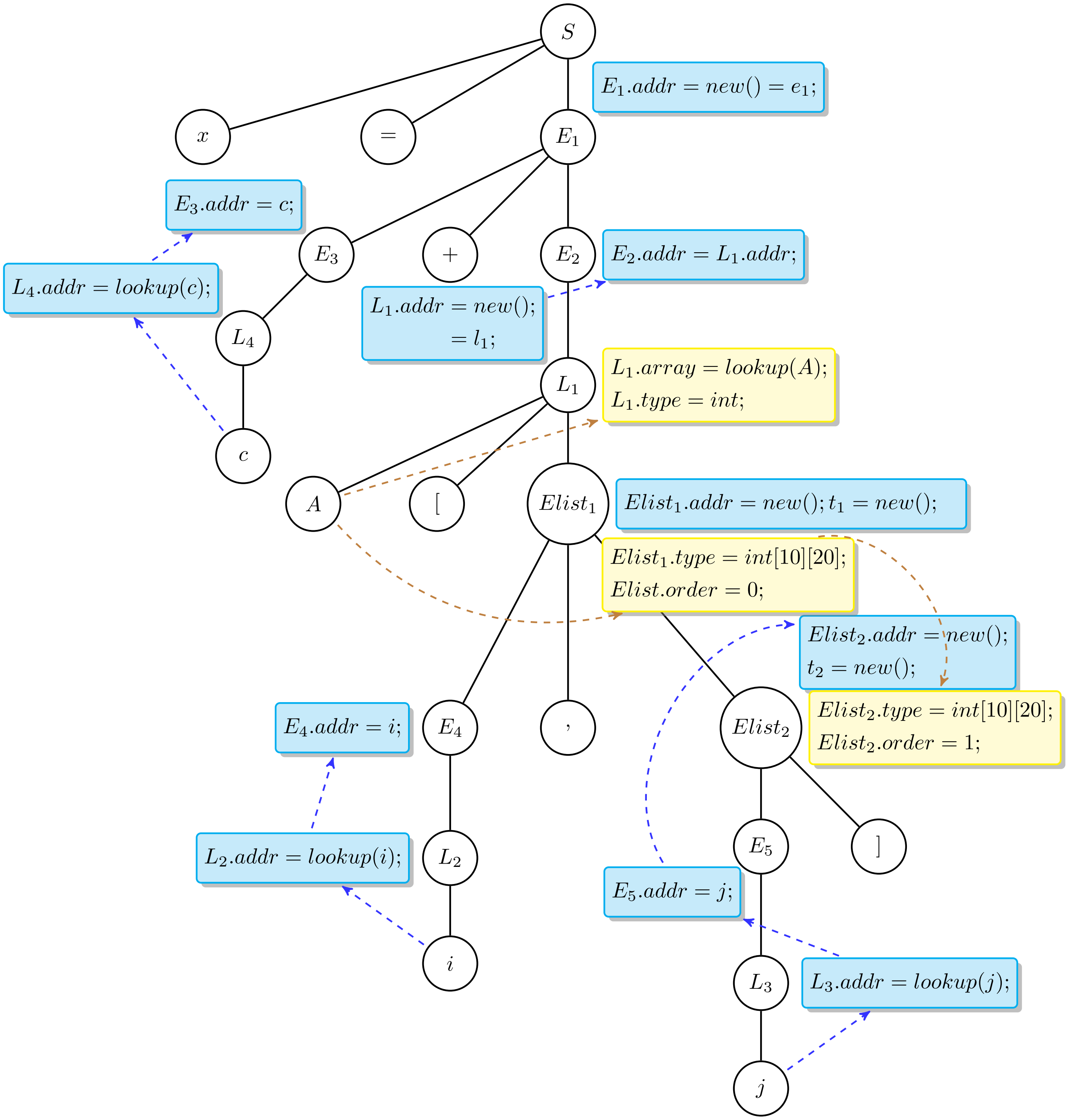
The annotated parse tree for input string `x = c + A[i,j]` is on the page 7

(iv)

The `3AC` code for the above expressions is as follows:

```
/* indexing array A */
t_2 = 10;
elist_2 = t_2 * j;

t_1 = 4;
elist_1 = t_1 * i;
t_1 = elist_2 * t_1;
elist_1 = elist_1 + t_1;

l_1 = A[elist_1];

/* Arithmetic computation */
e_1 = c + l_1;
x = e_1;
```

Note: For the sake of compactness the `new Temp()` function in the semantic action is replace with just `new()` in the annoted parse tree, similarly the `symtop.get()` function is replaced with `lookup()`. Also, some of the temporaries generated are implicit in the parse tree while they have been mentioned in the generated 3AC code.


S
x
$=$
E_1
$E_1.addr = new() = e_1;$
$E_3.addr = c;$
E_3
$+$
E_2
$E_2.addr = L_1.addr;$
$L_4.addr = lookup(c);$
L_4
$L_1.addr = new();$
$= l_1;$
L_1
$L_1.array = lookup(A);$
$L_1.type = int;$
c
A
$[$
$Elist_1$
$Elist_1.addr = new(); t_1 = new();$
$Elist_1.type = int[10][20];$
$Elist.order = 0;$
$Elist_2.addr = new();$
$t_2 = new();$
$E_4.addr = i;$
E_4
$,$
$Elist_2$
$Elist_2.type = int[10][20];$
$Elist_2.order = 1;$
$L_2.addr = lookup(i);$
L_2
E_5
$]$
$E_5.addr = j;$
i
L_3
$L_3.addr = lookup(j);$
j