CS335 Assignment 3

Dhruv Garg (210339)

April 5, 2024

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.

1. **Show the annotated parse tree for the input string 43#43@443.**

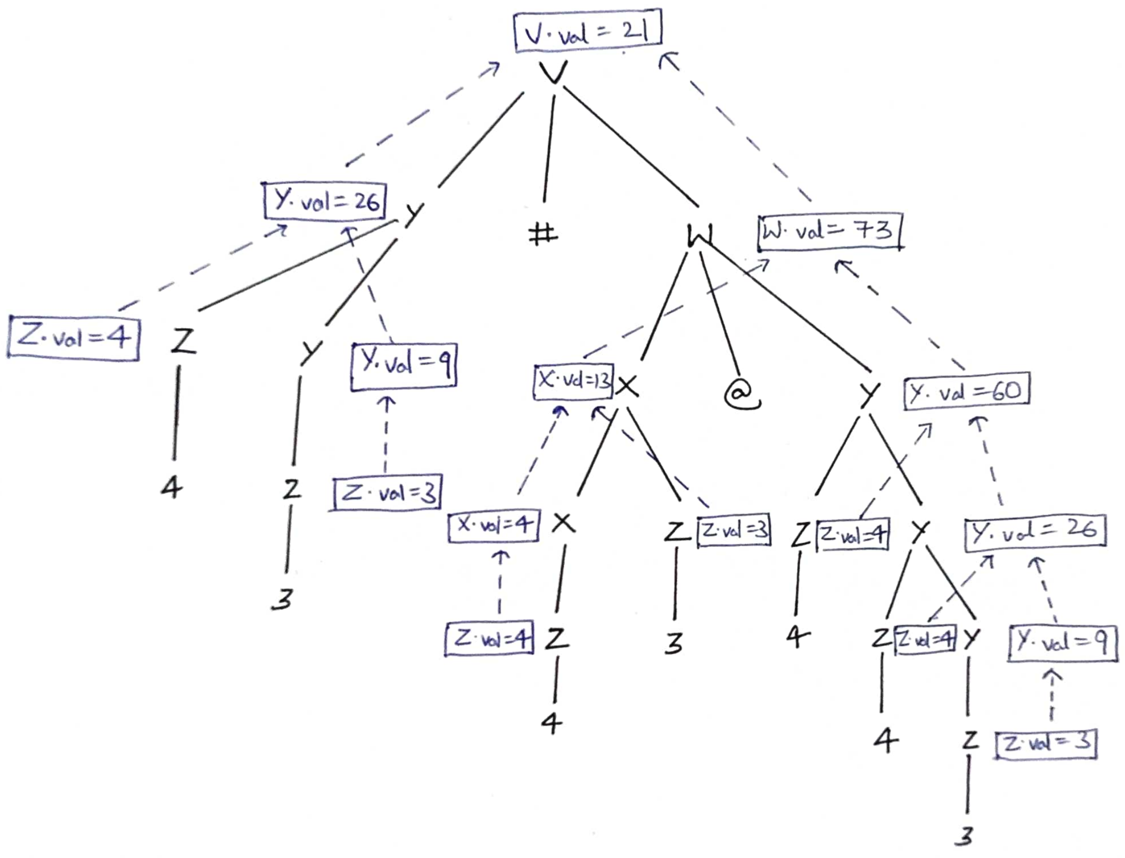

Figure 1: Annotated Parse Tree

2. **What is the value at V computed by the translation scheme for the above input string.**

   The value of V computer by the translation scheme for the above input string is $\boxed{21}$. The flow is shown above in the figure.

3. **Explain whether the grammar is S-attributed or L-attributed.**

   The grammar is both S-attributed and L-attributed. Each non-terminal has the attribute "val" and in each production rule, the attribute is synthesized from the child nodes. This makes it a synthesized attribute for all non-terminals. Since all S-attributed grammars are L-attributed, we can conclude that this grammar is both S-attributed and L-attributed.

Problem 2

We have discussed generating 3AC for array accesses using semantic translations. Consider the following extended grammar with semantic translation. 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 × 10 × 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.

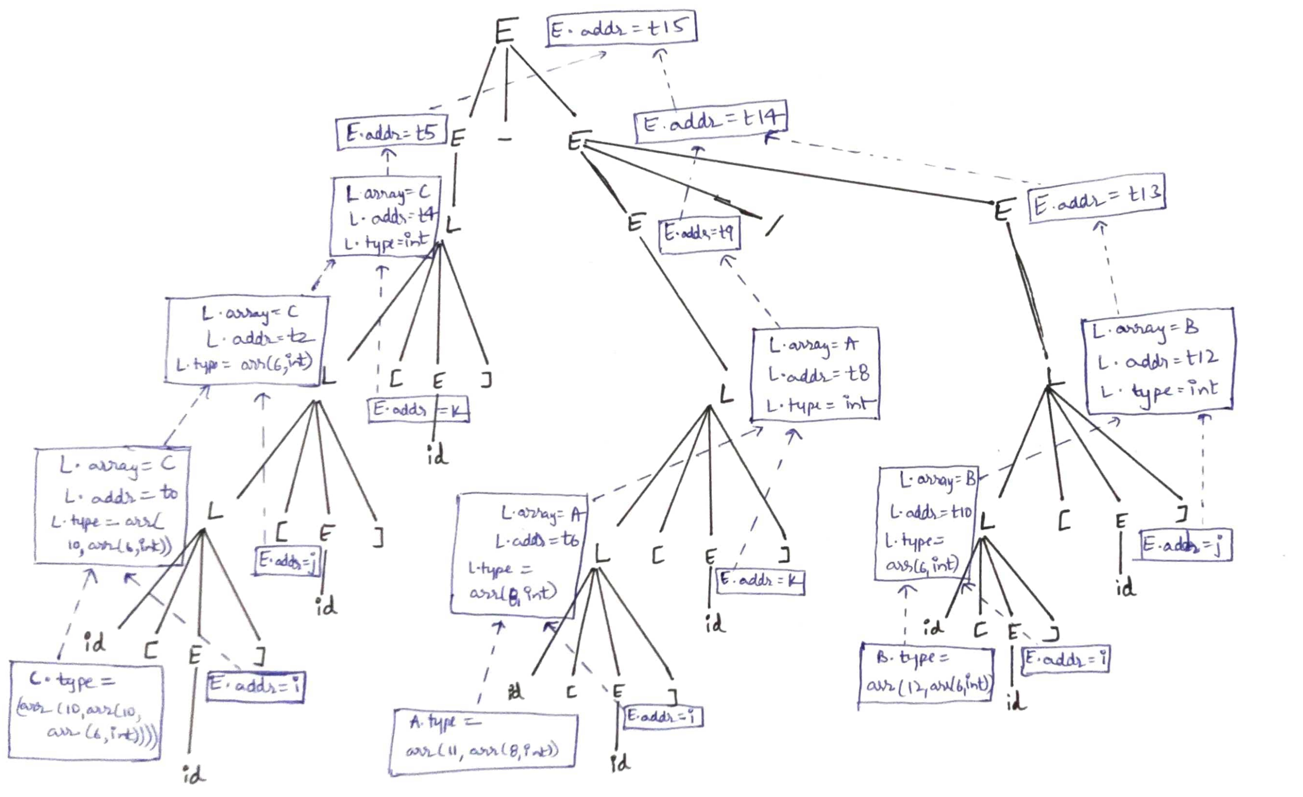

Figure 2: Annotated Parse Tree

3 AC code:

I0: $t_0 = i \times (60 \times 4)$	I8: $t_8 = t_6 + t_7$
I1: $t_1 = j \times (6 \times 4)$	I9: $t_9 = A[t_8]$
I2: $t_2 = t_0 + t_1$	I10: $t_{10} = i \times (6 \times 4)$
I3: $t_3 = k \times 4$	I11: $t_{11} = j \times 4$
I4: $t_4 = t_2 + t_3$	I12: $t_{12} = t_{10} + t_{11}$
I5: $t_5 = C[t_4]$	I13: $t_{13} = B[t_{12}]$
I6: $t_6 = i \times (8 \times 4)$	I14: $t_{14} = \frac{t_9}{t_{13}}$
I7: $t_7 = k \times 4$	I15: $t_{15} = t_5 - t_{14}$

Problem 3

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

1. **Show the semantic actions for your proposed translation.**

Production	**Semantic Actions**
$\mathrm{S} \rightarrow \mathrm{id} = \mathrm{E}$	`gen( symtop.get(id.lexeme) = E.addr )`

$\mathrm{E} \rightarrow \mathrm{E}_1 + E_2$	`E.addr = newTemp()`
	`gen( E.addr = E1.addr + E2.addr )`

$\mathrm{L} \rightarrow \mathrm{id}$	`L.addr = symtop.get(id.lexeme)`
	`L.offset = NULL`

$\mathrm{S} \rightarrow \mathrm{L} = \mathrm{E}$	`if L.offset == NULL:  gen( L.addr = E.addr)`
	`else:  gen(L.addr[L.offset] = E.addr)`

$\mathrm{E} \rightarrow L$	`if( L.offset == NULL ):    E.addr = L.addr`
	`else:`
	`    E.addr = newTemp()`
	`    gen( E.addr = L.addr[L.offset]`

$\mathrm{Elist} \rightarrow \mathrm{E}\ ]$	`if( E.addr >= DimSize(1,E.array)):`
	`    Error("Index out of bound!"); exit;`
	`Elist.addr = newTemp()`
	`Elist.addr = E.addr`
	`Elist.dim = 1`

$\mathrm{Elist} \rightarrow \mathrm{E},$
 	
Elist_1
	`{ Elist1.array = Elist.array}`
	`{    if( E.addr >= DimSize(Elist1.dim,Elist1.array)):`
	`         Error("Index out of bound!"); exit;`
	`     Elist.addr = newTemp()`
	`     Elist.dim = Elist1.dim + 1`
	`     gen( Elist.addr = Elist1.addr * DimSize(Elist.dim,Elist1.array) )`
	`     gen( Elist.addr = Elist.addr + E.addr )`
	`}`

$\mathrm{L} \rightarrow \mathrm{id}\ [$
 	
Elist
	`{Elist.array = symtop.get(id.lexeme)}`
	`{    if( Elist.dim != getDim(id) ):`
	`         Error("Dimensions doesn't match"); exit;`
	`     gen( L.addr = getBase(id) )`
	`     L.offset = newTemp()`
	`     gen( L.offset = Elist.addr * id.BaseType.width )`
	`}`

2. **Explain the attributes and auxiliary functions in your SDT.**

   Lets first see the attributes:

   - addr: hold the name / value / temporaries
   - dim: It tells the position of current dimension of the list from reverse order while parsing.
   - array: It stores the symbol table entry of the array we are currently parsing through. It is an inherited attribute.
   - offset: current offset and is used for referencing
   - BaseType: Gives the type of the base element. It is integer if it is array of integers and similarly.
   - width: Gives size of the type of element.

   Lets see the helper functions now:

   - getDim(A): gets the dimension of array A whether is a 3D array or 4D array or what so ever.
   - getBase(A): gets the Base address of array A
   - DimSize(m,A) : Gives the maximum size of m^{th} dimension from back of array A. For example for array of size 2*3*4*5, we have DimSize(1,A) = 5, DimSize(2,A) = 4, DimSize(3,A) = 3, DimSize(4,A) = 2.
   - gen: generates the 3AC instruction
   - newTemp(): generates a new temporary
   - symtop.get(): Directly accesses symbol table and completes the required operation.

   Note: The helper functions getDim, getBase and DimSize(m,A) are implemented simply. They directly access symbol table and get the required information from the symbol table.

3. **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.**

   Present on next page

4. **Show the generated 3AC for the above expression.**

   I1: t0 = j

   I2: t1 = t0 * 10

   I3: t1 = t1 + i

   I4: t2 = t1 * 4

   I5: t3 = A [t2]

   I6: t4 = c + t3

   I7: x = t4

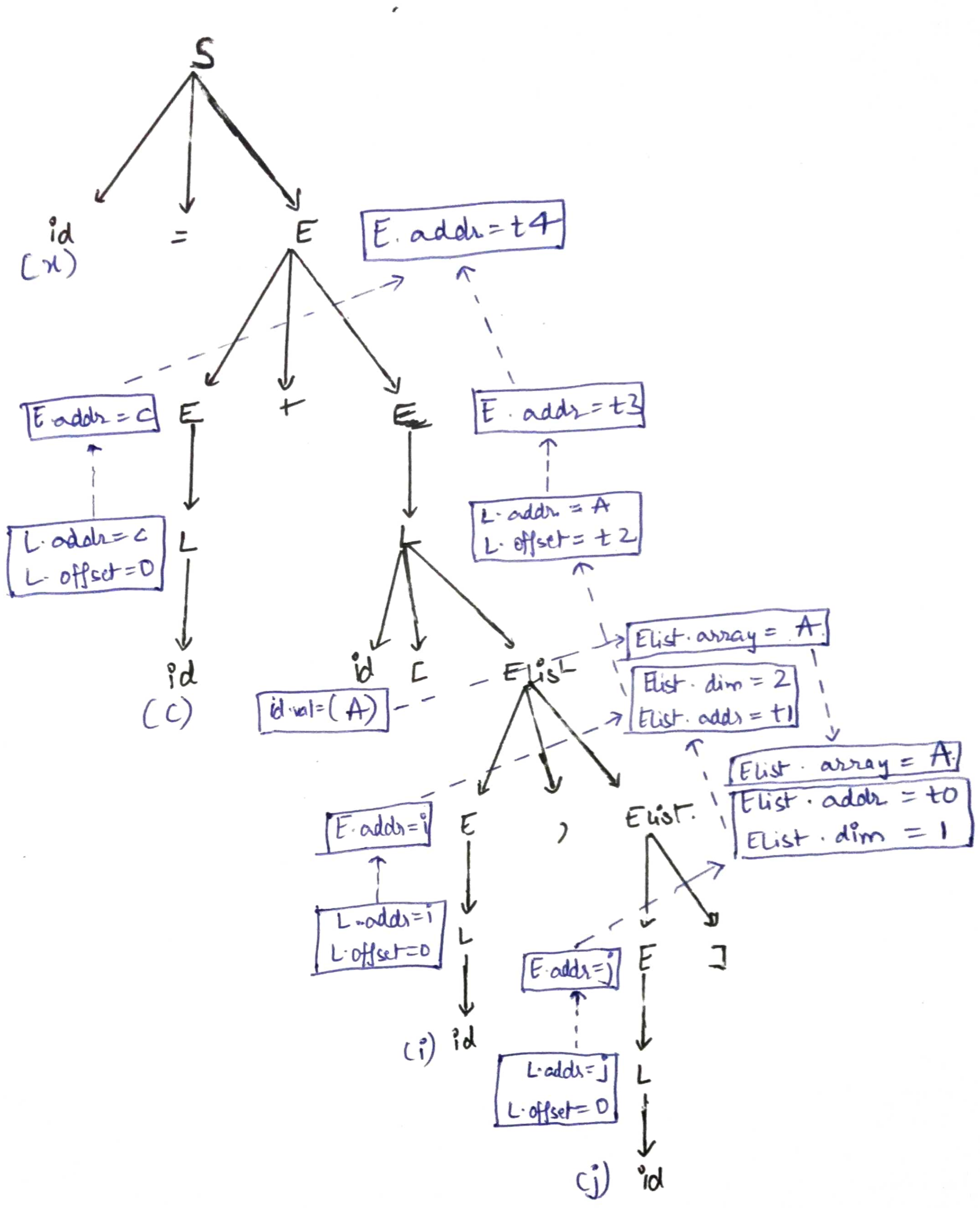

Figure 3: Annotated Parse Tree