April 10, 2025

Syntax Directed Definition (SDD):

Syntax directed definitions are a generalization of Context Free Grammars in which:

1) Grammar symbols have an associated set of attributes
2) Productions are associated with semantic rules for computing values of the attributes.

eg:

CFG: $E \rightarrow E+T$

SDD: $E \rightarrow E+T$ { $E.val \rightarrow E.val + T.val$ }

attribute associated with E/E/T

Semantic Rule (প্রোডাকশন রুলের সাথে সম্পর্কিত)

এখানে val গুলোটাই same value বোঝায় কোনো রকম নাই, So no connection between the E.val's other than this SDD formula.

Syntax Directed Translation:

Syntax Directed Translation is done by attaching rules or program fragments to productions in grammar.

eg:

SDT : $E \rightarrow E+T \{ \text{print "+"} \}$

Q) What's the difference betⁿ SDD & SDT?

eg: **SDD:**

CFG	Semantic Rules
$S \rightarrow E$	$\{ S.val = E.val \}$
$E \rightarrow E1+T$	$\{E.val = E1.val + T.val\}$
$E \rightarrow T$	$\{E.val = T.val\}$
$T \rightarrow T1*F$	$\{T.val = T1.val * F.val\}$
$T \rightarrow F$	$\{T.val = F.val\}$
$F \rightarrow digit$	$\{F.val = digit.lexval\}$

1 দিয়ে separate করতে হবে

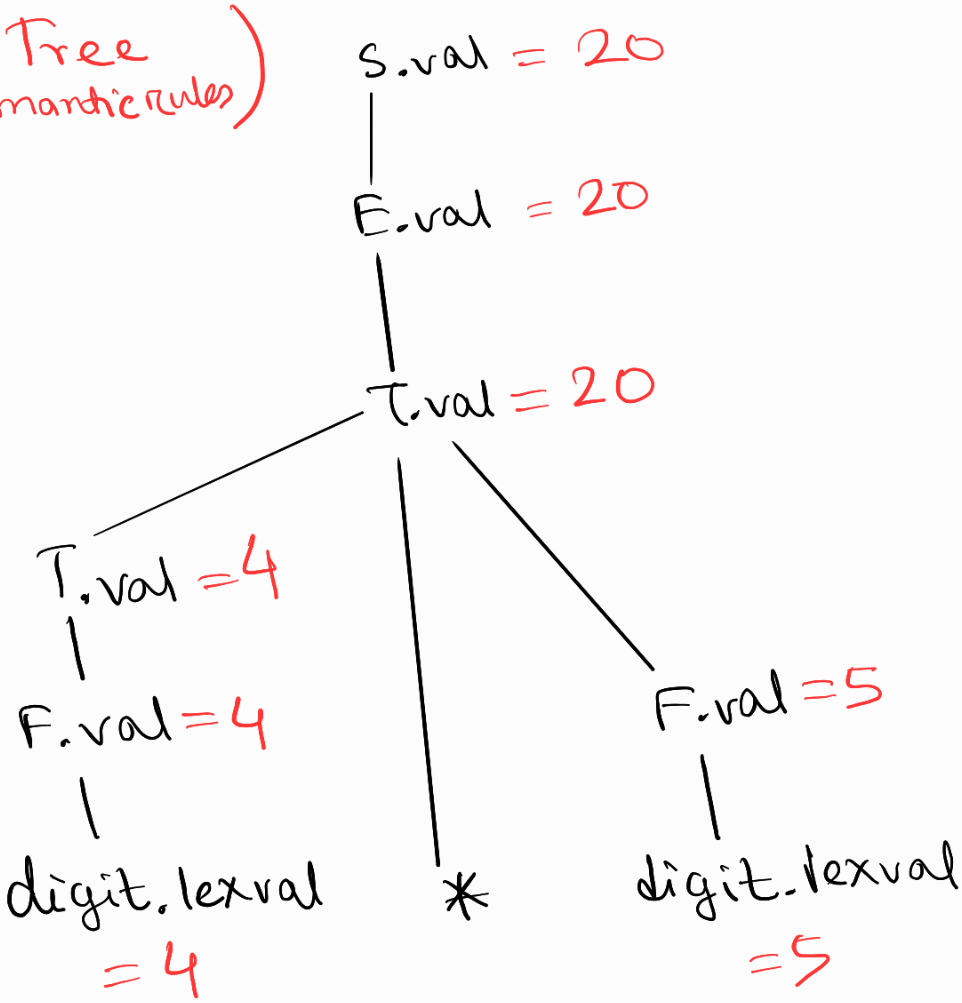
প্রতিটা CFG rule এর জন্য Semantic rule থাকতে হবে এমন নয়

→ inputstring: 4 * 5

∴ tokens: digit * digit

∴ Annotated Parse Tree:

(Parse Tree with semantic rules)



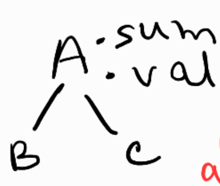
> **Q) Write the different types of attributes**

Two Types of Attributes:

1) **Synthesized Attribute:** A synthesized attribute at Node N is defined in terms of attribute values at the children of N and at N itself.

eg: A → BC {A. ______ }
↓
B বা C

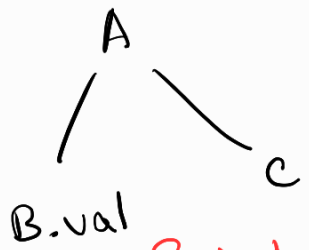
so like



→ val একটি synthesized attribute হবে if it uses any attribute from child nodes B/C or from another attribute of A like sum.

2) Inherited Attribute: An inherited attribute at Node N is defined only in terms of attribute values at N's parent, N itself and N's siblings.

eg: $A \rightarrow BC$



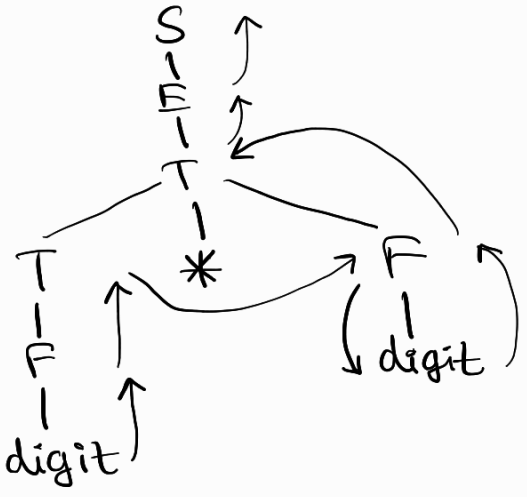
একটা inherited attribute হবে যদি B এর parent A এর কোনো attribute বা more than one attribute used হয়, বা B এর অন্য attribute used হয়, বা sibling node C এর attribute used হয়।

So like B.val = B.val + C.val + A.val একটা example

Dependency Graph:

A dependency graph helps us to determine how the attribute values can be computed. It tells us about which order we should evaluate the tree.

eg:



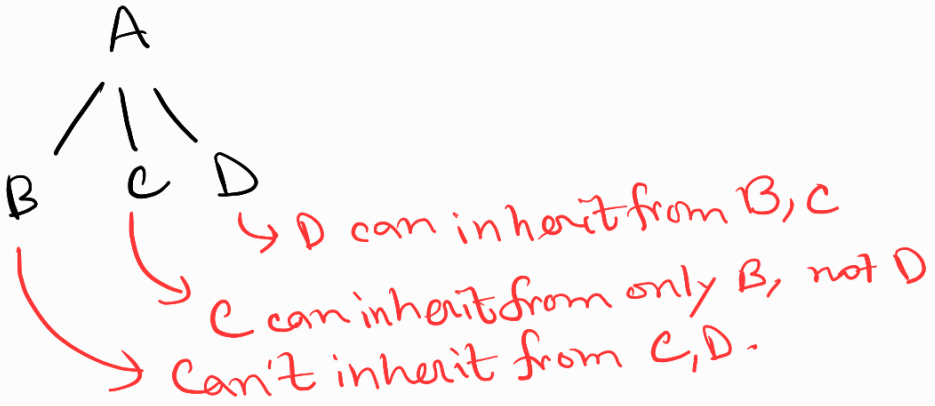
Q) Draw a dependency graph for a CFG

Evaluation orders for SDD :

- **S attributed definitions** → A SDD is S-attributed if every attribute is synthesized.

- **L attributed definitions** → If a SDD uses both Synthesized and Inherited attribute with a restriction that inherited attributes can inherit from the left sibling only, then we can call that SDD a L-attributed SDD.

eg:



3:30 class
April 12

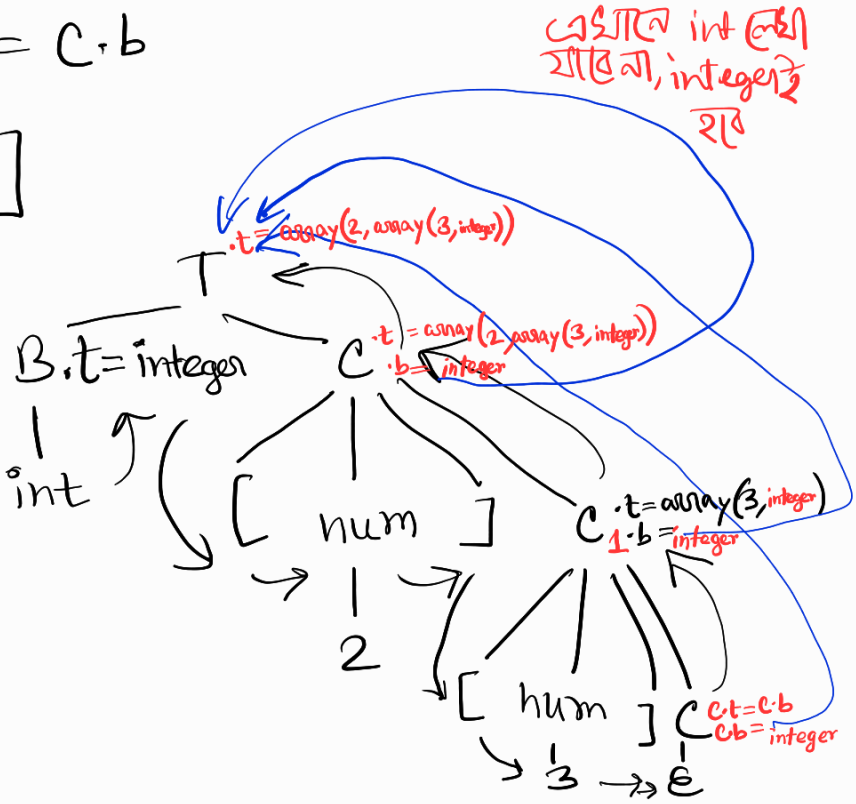
Type expression:

eg SDD

CFG	Semantic Rules
$T \to BC$	$\{T.t = C.t$ $C.b = B.t\}$
$B \to int$	$\{B.t = 'integer'\}$
$B \to float$	$\{B.t = float\}$
$C \to [num]C_1$	$\{C.t = array(num.val, C_1.t)$ $C_1.b = C.b\}$
$C \to \epsilon$	$C.t = C.b$

Input string: int[2][3]

Annotated Parsetree:

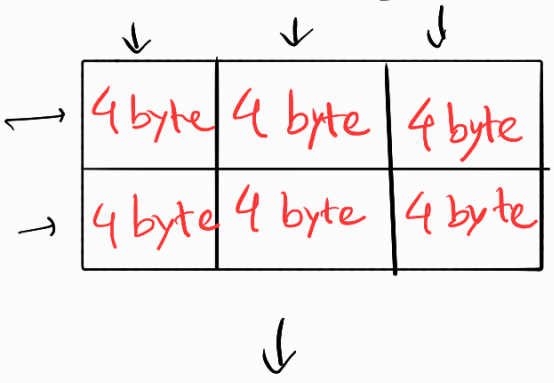


ϵ → C reduction এ একটা problem হচ্ছে C.b exist করছে না, তাই C.t = C.b হচ্ছে না, hence root node এ referred. Only after T এ Sem. Rule C.b = B.t যখন execute হয়, নিচের গুলো একটা একটা update হয়।

Thus,

T.t তে যাকে "array(2, array(3, integer))" is the "type expression" for input string "int[2][3]".

- With the help of this type expression, computer can allocate memory in your system.



অন্য কিছু বলা না থাকলে,
for each integer 4 bytes are allocated.
(Usually float হলে 8 bytes)

For 1 row: 3 * 4 byte = 12 bytes
columns

For both rows: 2 * 12 bytes = 24 bytes

Total: 24 bytes

(Calculation questions আসতে)

(Draw Dependency graph for given tree)

(Draw parse tree from given SDD and code)

Types:

- Basic Type → int, float, double, char
- Composite Type → array linked list

eg:

CFG	Semantic Rules
$T \to BC$	$\{ T.t = C.t \; C.b = B.t \}$
Basic Type: $B \to int$	$\{B.t = integer\}$
Basic Type: $B \to float$	$\{B.t = float\}$
Composite Type: $C \to [num]C_1$	$\{C.t = array(num.val, C_1.t) \; C_1.b = C.b\}$
Composite Type: $C \to \epsilon$	$C.t = C.b$

- ϵ কোনো Type
- 2 বা
- একটি variable both Basic and Composite হতে পারে যদি আলাদা rules এ থাকে উল্লেখ

Sequence of Declarations:

Refers to how declarations are handled within a grammar's production rules.

eg

$P \to \{offset = 0;\} \; D$

P produces offset = 0 and then D.

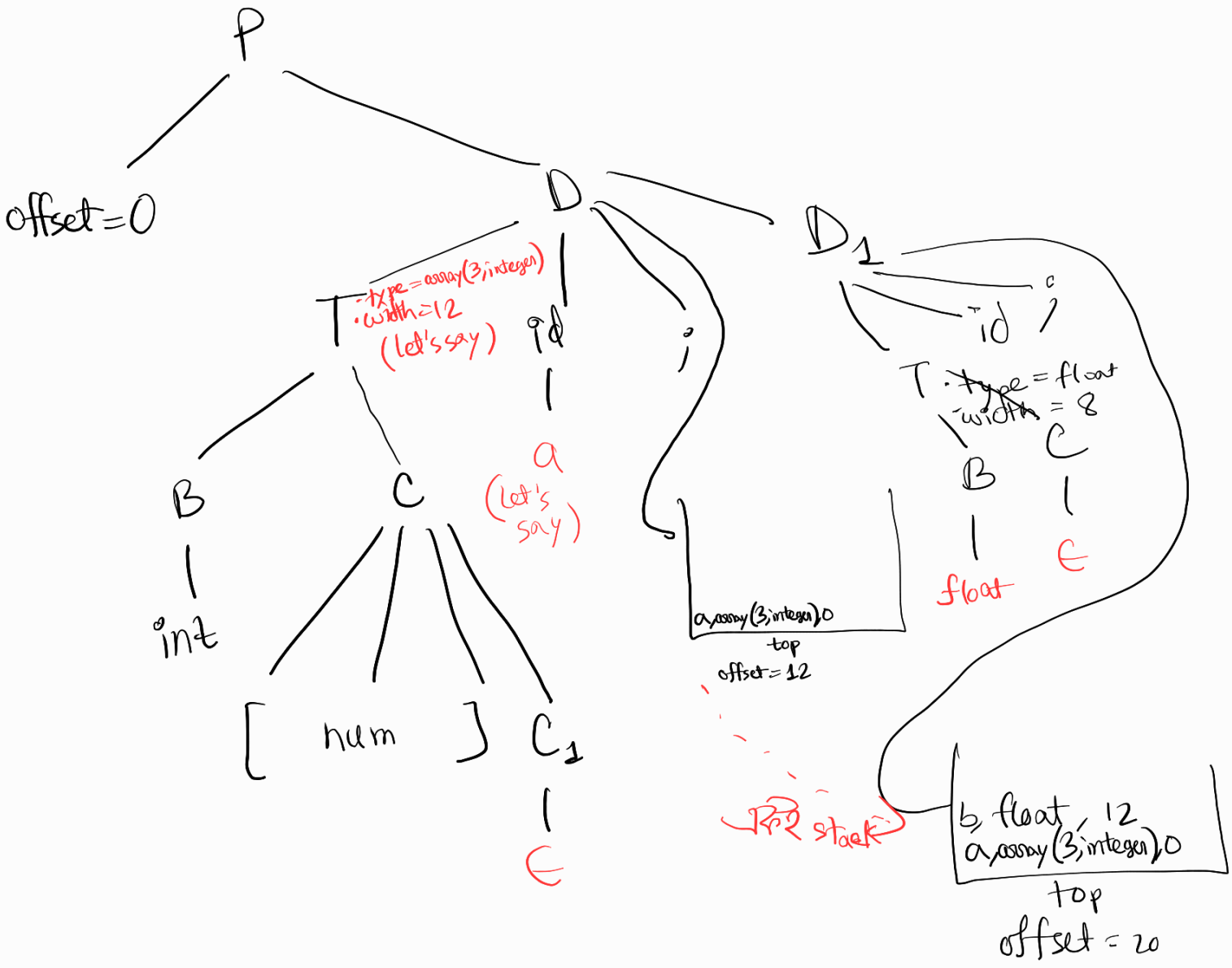
$D \to T \; id \; ; \; \{top.put(id.lexeme, T.type, offset); \; offset = offset + T.width;\} \; D_1$

D produces T id; {....} D_1

sequence ঠিক থাকবে

$D \to \epsilon$

D produces epsilon


P
offset=0
D
D1
T
type=array(3,integer)
width=12
(let's say)
id
a
(let's say)
B
int
C
[num]
C1
ε
;
id
c
T
type=float
width=8
B
float
C
ε
a, array(3,integer), 0
top
offset=12
पहिले stack
b, float, 12
a, array(3,integer), 0
top
offset=20

April 17

Quiz 3: May 1st/3rd

Syllabus:

~~SDD~~:

$P \rightarrow \{offset = 0\} D$

$D \rightarrow T\ id\ ; \{top.push(id.lexeme, T.type, offset);$
$offset = offset + T.width;\}$

D_1

$D \rightarrow \epsilon$

$T \rightarrow B\ C \{T.type =$
$T.width =$ $\}$

$B \rightarrow int \{B.type =$ $\}$

$C \rightarrow [num]\ C_1 \{C.type =$

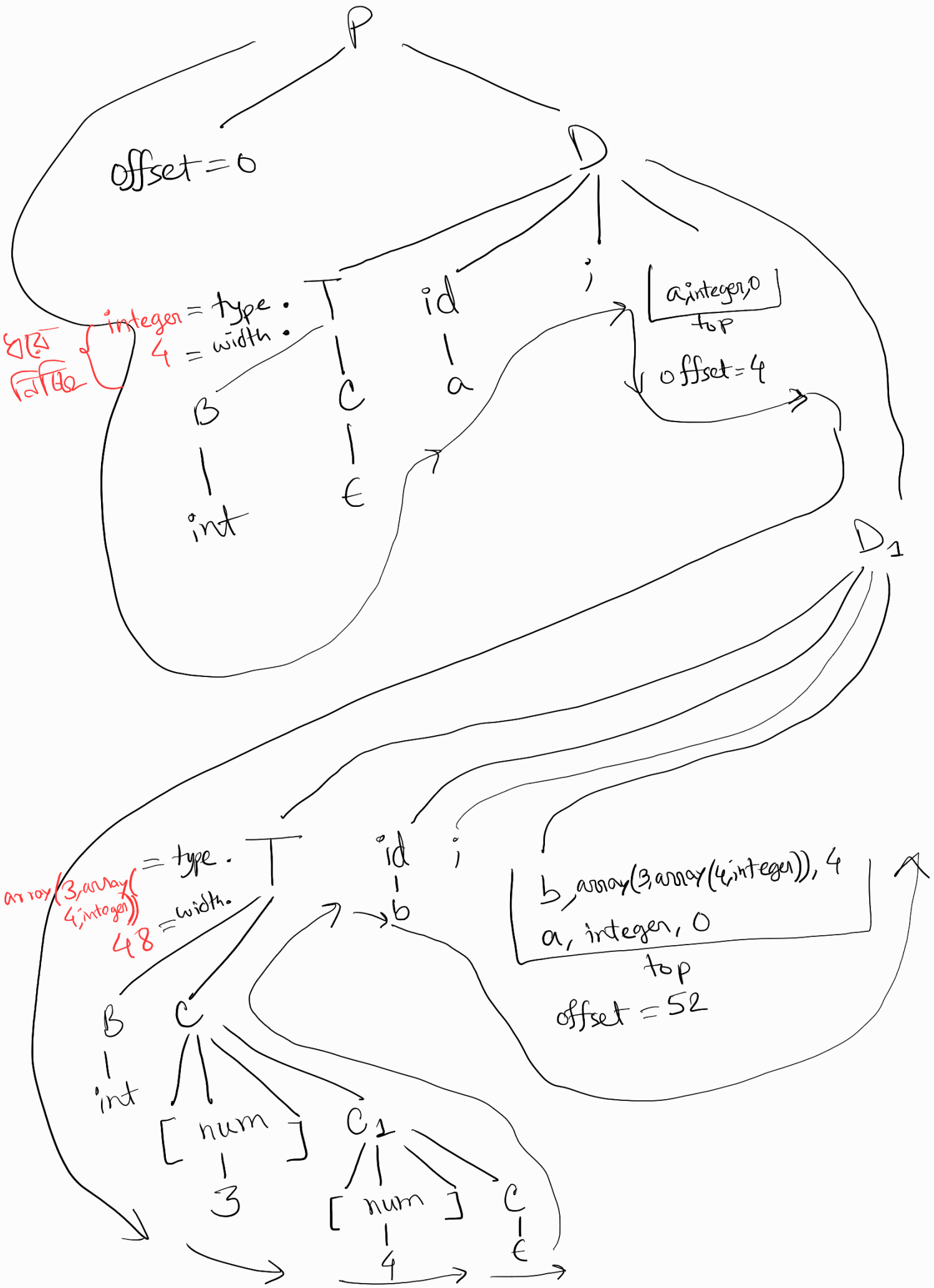
$C \rightarrow \epsilon$

Assignment deadline:
May 29

inp str:

```
int a;
int [3][4] b;
```

Tree:


P
offset = 0
D
T
id
;
a, integer, 0
top
integer = type
4 = width
B
C
a
offset = 4
int
ϵ
D1
T
id
;
= type
= width
b
b, array(3, array(4, integer)), 4
a, integer, 0
top
offset = 52
B
C
int
[num]
3
C1
[num]
4
C
ϵ

SDD:

$D \rightarrow T\ id;\ D_1 \mid \epsilon$

$T \rightarrow BC \mid record\ '\{'\ D_1\ '\}'$

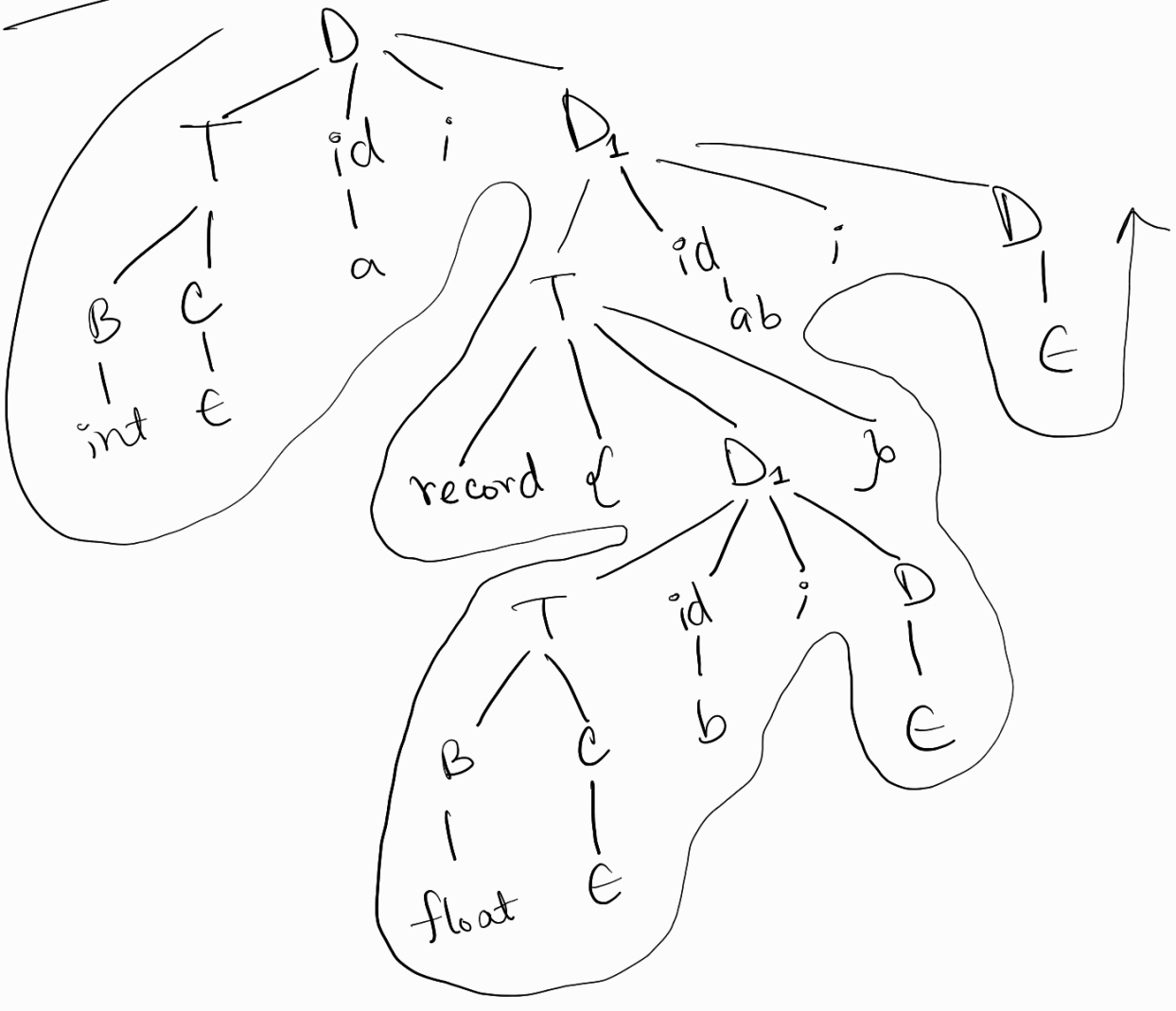
$B \rightarrow int \mid float$

$C \rightarrow \epsilon \mid [num]C_1$

input string:

```
int a;
record {
    float b;
}ab;
```

Tree:



Assignment SDD grammar:

$P \rightarrow \{offset = 0;\}\ D$

$D \rightarrow T\ id;\ \{top.put$ $\}D_1$

$D \rightarrow \epsilon$

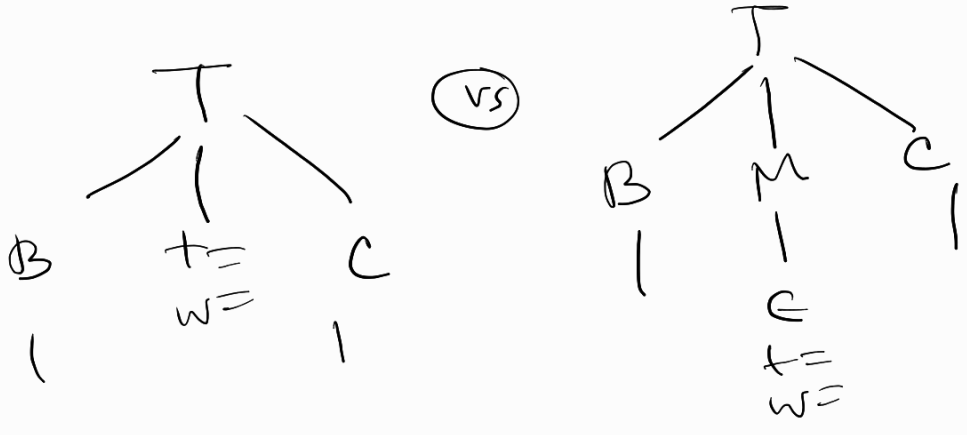
$T \rightarrow B$

Using Dummy Nonterminals for semantic rules SDD:

$T \rightarrow B\ C\ \{t = B.type;\ w = B.width\}$

$T \rightarrow B\ M\ C$ → Dummy NT

$M \rightarrow \epsilon\ \{t = B.type,\ w = B.width\}$



No difference!

Same day, বিকালের class:

SDD:

$P \rightarrow \{offset = 0\}\ D$

$D \rightarrow T\ id\ ;\ \{top.push(id.lexeme, T.type, offset);$
$offset = offset + T.width\}\ D_1$

$D \rightarrow \in$

$T \rightarrow BC\ \{T.type = C.type\ \ T.width = C.width\}$

$B \rightarrow int\ \{B.type = integer\ \ B.width = 4\}$

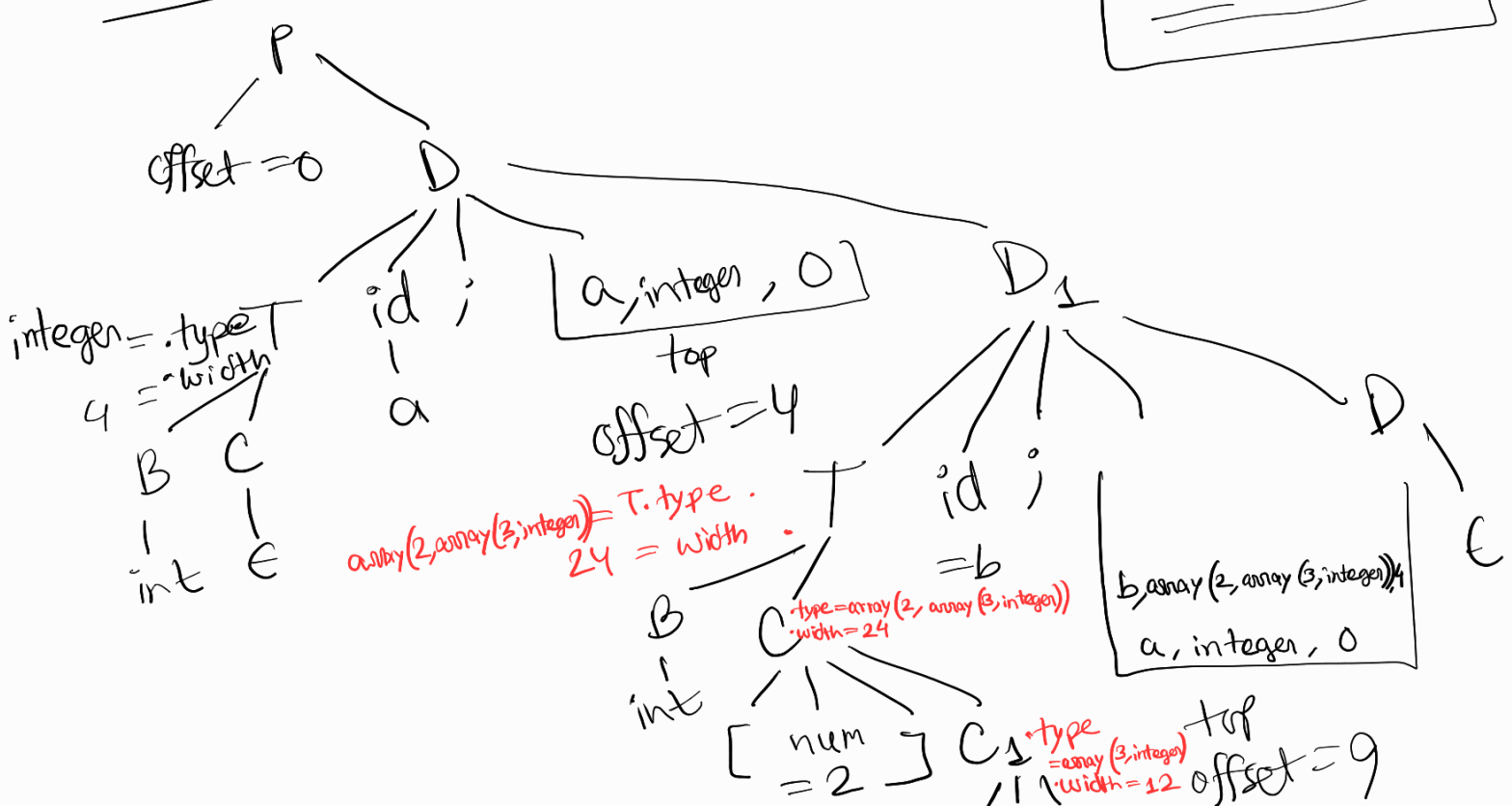
$C \rightarrow [num]\ C_1\ \{C.type = array(num.value, C_1.type);$
$C.width = C_1.width \times num.val\}$

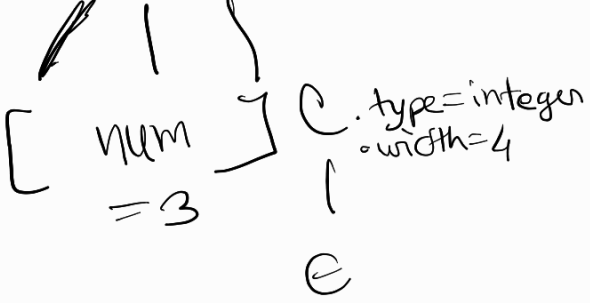
$C \rightarrow \in\ \{C.type = B.type;$
$C.width = B.width\}$

inputstr:

```
int a;
int [2][3]b;
```

Tree:





SDD:

$D \rightarrow T\ id;\ D \mid \epsilon$

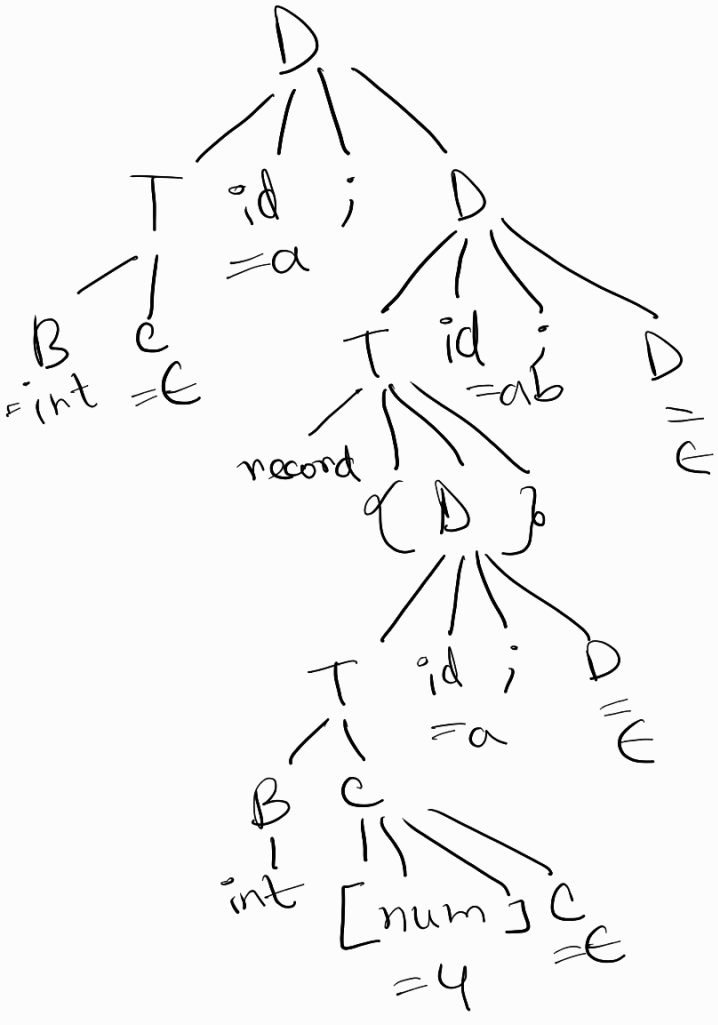
$T \rightarrow BC \mid record\ '\{'\ D\ '\}'$

$B \rightarrow int \mid float$

$C \rightarrow \epsilon \mid [num]\ C$

input string

```
int a;
record{
  int[4] a;
}ab;
```



April 19 [Online Class @ 7 PM]

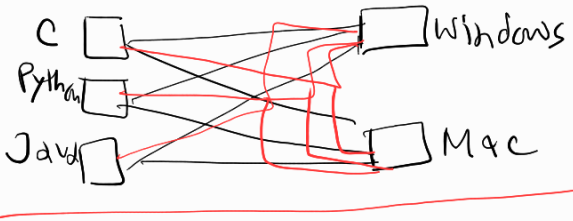
Intermediate Code Generation

→ Generation of T.A.C (Three Address Code) that's machine independent.

Q) Why do we generate Machine Independent Code? What are the advantages?

m X n Compiler

Q) কী এটা? see book



Benefits of dividing Compilation steps into Frontend & Backend:
যেকোনো no. এর Frontend কে যেকোনো no এর Backend এর সাথে Connect করা যায়

In every line highest ৩টি address থাকবে highest ১টি operator থাকবে

Three Address Code Generation:

Input code: `a = b+c+d`

Temporary variables used হয় অনেক

T.A.C :

```
t1 = b+c
t2 = t1+d
a = t2
```

Optimized T.A.C:

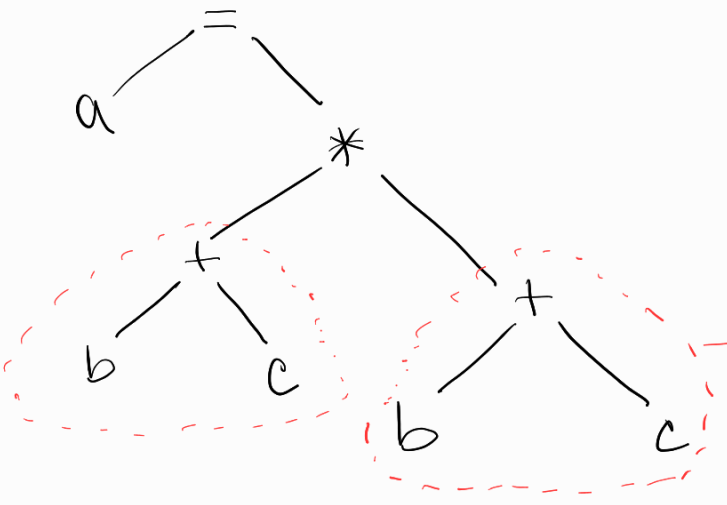
```
t1 = b+c
a = t1+d
```

Question এ Optimized চাইলে এটা দিয়ে দেওয়া লাগবে, না বললে your choice

DAG → Directed Acyclic Graph

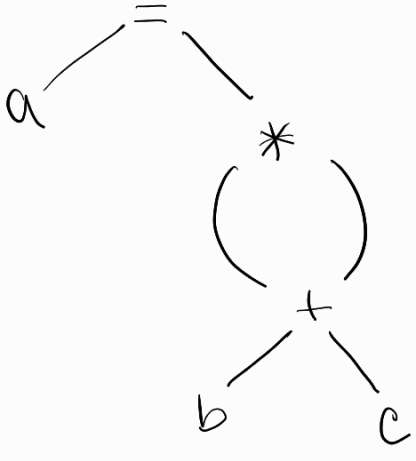
Input str: $a = (b + c) * (b + c)$ → Brackets বুঝানোর জন্য ব্যবহৃত, not part of inp. str.

Syntax Tree:



DAG ও no subtrees can be repeated!

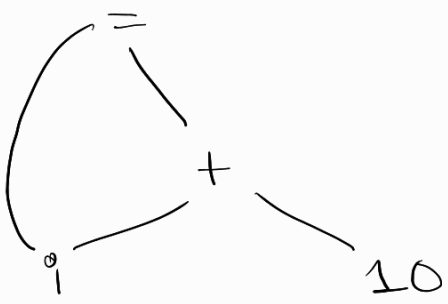
DAG:



No Repetition!

eg inp str: i = i + 10

DAG:



Array of Records:

0	id		
1	num		
2	+	0	1
3	=	0	2
4			

Index 0 এর স্থানে value এর স্থানে 1 " " " " addition হচ্ছে

Index 0 কে set করছে to what's stored in index 2.

H/W: a + a * (b - c) + (b - c) * d

Draw the DAG for this expression before next class!
Also draw Array of Records

Three Address Code generation Instructions/Rules:

1. Assignment Instructions:

$$x = y \ op \ z$$

2. Assignment Instructions for Unary Operations:

- a = -c
- a = int(c)

Operands in total number in 2

$$x = op \ y$$

3. Copy Instructions:

$$x = y$$

4. Unconditional Jump:

```
.
.
goto L
.
.
L: ----
   ----
   ----
```

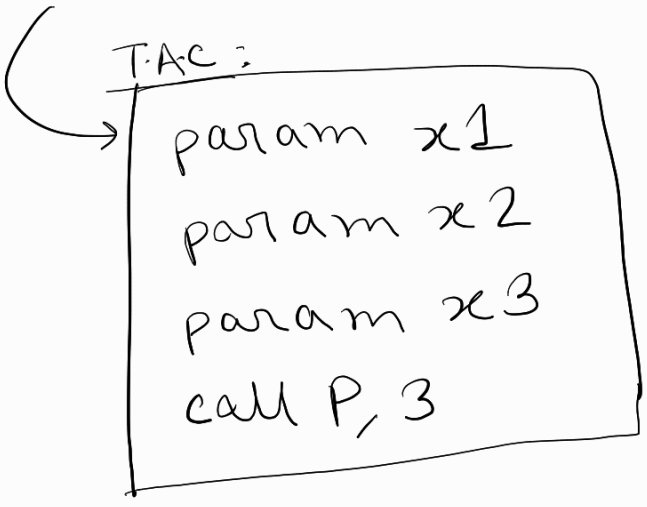
5. Conditional Jump:

if False x goto L

6. For Loop (ফর লুপ)

7. Function :

eg :- P (x1, x2, x3)



April 24

TAC Representations

- Quadruples
- Triples
- Indirect Triples

$x = \&y$ → x equals address of y.

$x = *y$ → x equals value in the variable stored by pointer y

$*x = y$ → The variable that's being pointed by x will have the value stored in y.

eg $a = b * -c + b * -c$

Non-optimized Three Address Code:

unary minus কে আলাদাভাবে নিচ্ছে

$t1 = -c$ বা $t1 = \text{minus } c$

$t2 = b * t1$

$t3 = \text{minus } c$

$t4 = b * t3$

$t5 = t2 + t4$

$a = t5$

" Uminus c " হলে

For this,

Quadruple's Representation:
(Array of Records)

এখানে arg1 এর মান বসাতে হবে, not 2.

Quadruple's Repr. Numbers

	op	arg1	arg2	result
0	minus	c		t1
1	*	b	t1	t2
2	minus	c		t3
3	*	b	t3	t4
4	+	t2	t4	t5
5	=	t5		a

Triple's Representation:

এখানে Triple's Representation Numbers (indices গুলো) দিয়ে T.A.C এর instruction number denote করে, যাতে result column টা না লাগে।

	op	arg 1	arg 2
0	minus	c	
1	*	b	(0)
2	minus	c	
3	*	b	(2)
4	+	(1)	(3)
5	=	a	(4)

Instead of t1, we write operation(0)

Brackets লাগে

Assign operation এ LHS (result) টাই প্রথম argument হয়

Quiz 4 syllabus:
The 3 Representations

Indirect Triples :

Triple's Rep:

	op	arg 1	arg 2
0	minus	c	
1	*	b	(0)
2	minus	c	
3	*	b	(2)
4	+	(1)	(3)
5	=	a	(4)

Memory এর location

	Pointer
35	(0)
36	(1)
37	(2)
38	(3)
39	(4)
40	(5)

Static Single Assignment:

With the help of Static Single Assignment, we can understand whether a variable's value has already changed or not.

eg:
$t_1 = \text{minus } c$
$t_2 = b * t_1$
$t_3 = \text{minus } c$
$t_4 = b * t_3$
$t_5 = t_2 + t_4$
$a = t_5$

Numbers গুলো না দিলে আলাদা করা অসম্ভব।

SDD for the TAC:

Semantic Rule (No brackets)

$S \rightarrow id = E;$ $S.code = E.code \,||\, gen(top.get(id.lexeme)\text{'='}E.addr)$

concatenation ↑ ↳ generates TAC line, like "x = 0x11" বা "x = t1"

$E \rightarrow E_1 + E_2$ $E.addr = new\ Temp()$ একটা করে temporary variable (t1, t2 ...) create হচ্ছে।

$E.code = E_1.code \,||\, E_2.code \,||\, gen(E.addr\text{'='}E_1.addr\text{'+'}E_2.addr)$

$E \rightarrow -E_1$ $E.addr = new\ Temp()$

$E.code = E_1.code \,||\, gen(E.addr\text{'='}\text{'minus'}E_1.addr)$

$E \rightarrow (E_1)$ $E.addr = E_1.addr$

$E.code = E_1.code$

$E \rightarrow id$ $E.addr = top.get(id.lexeme)$

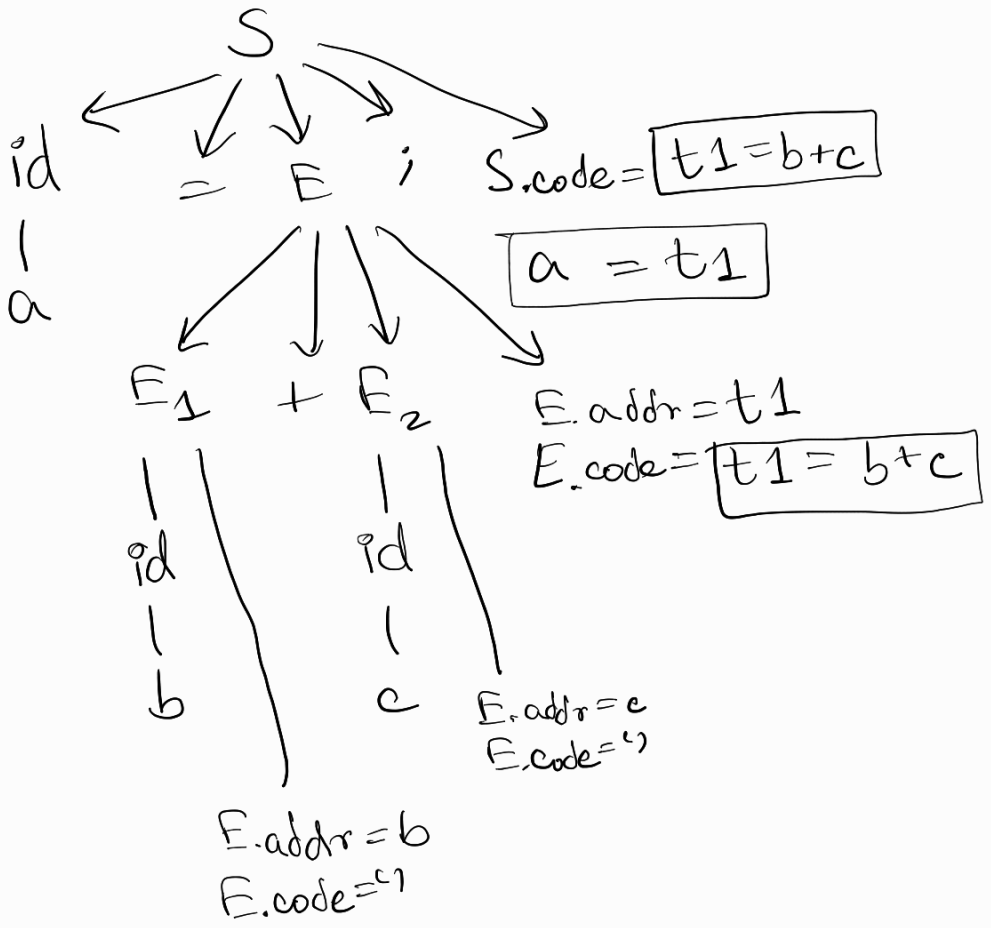
$E.code = ''$ ↳ empty string

একটা input string of code কে কে sdd parse করতে Using its syntax tree and convert to a T-A-C

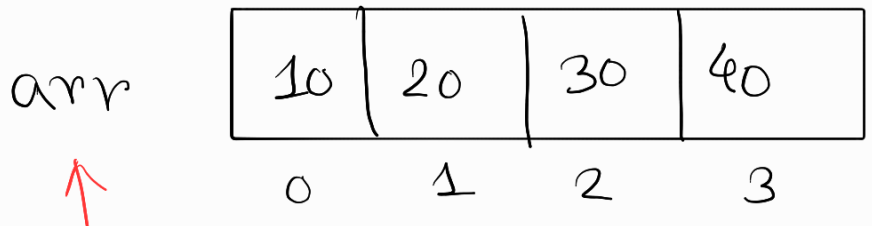
April 26

Input string: $a = b + c;$

Parse Tree:



Relative address Calculation for Array:



base address = 0 → (Let's say)

data type = integer

'For each integer, 4 bytes memory will be allocated'

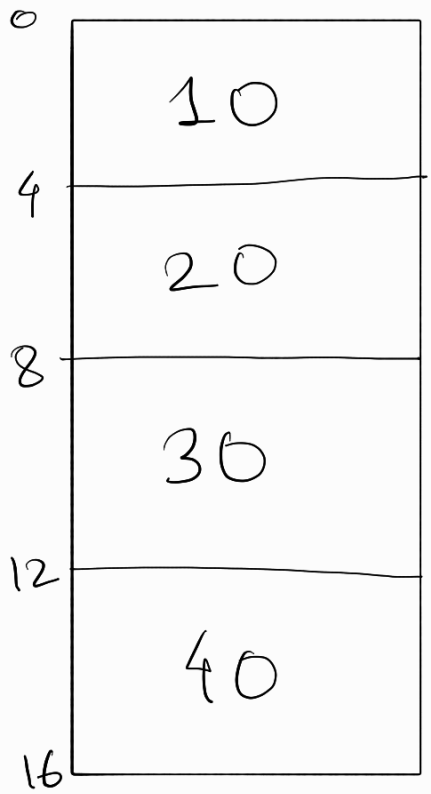
2 Representations:

1) Row Major
2) Column Major

↑ Representation আলাদা হয়ে in 2D and higher D arrays.

1D Arrays:

(Row Major Rep)



a[2] তাহলে compiler directly 8 byte skip করে to fetch value of index 2. → 30 (Jumps to byte 8 from 0.)

Calculation is like

$$0 + 4 * 2 = 0 + 8 = 8$$

So the formula is,

$$\text{Rel. Address for jump} = \text{base} + (\text{Size of each integer} \times \text{Index No.})$$

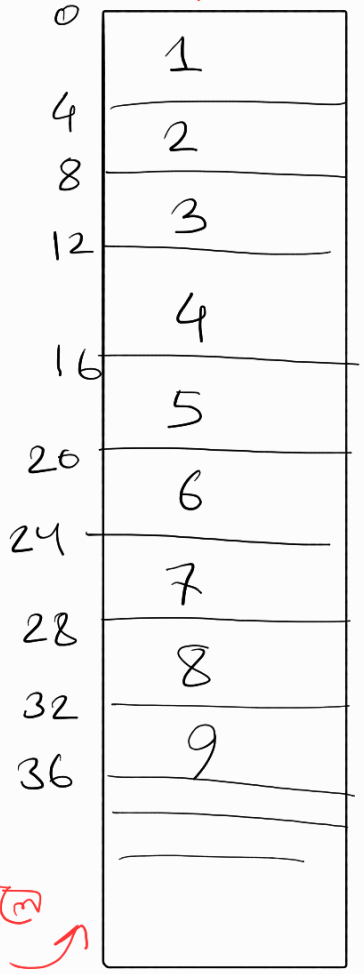
< For One Dimensional Arrar >

2D Arrays:

	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20
4	21	22			

arr ←

Row Major Repr: → (Used More)



base address = 0
data type = integer
for each integer → 4 bytes ...

একটা Row শেষ হলে পরেরটায় যায় →

Column major : (Used less)

Address	Value
0	1
4	6
8	11
12	16
16	21
20	2
24	7
28	12
32	17
36	22
40	3
44	

Column একেকটা ক্রমে হলে সাজিয়ে যায় →

arr[1][3] লিখলে 9 fetched হবে, skipping 32 bytes (in the row major representation), এটার calculation:

- Row size = 4 * 5 = 20 bytes (4 = bytes, 5 = cols)
- Since arr[1]..., we've to skip row 0 and land on row 1, and then 3 indices more:

$$0 + (20 \times 1) + (4 \times 3) = 32$$

(20 = Row size, 1 = Idx 1, 4 = cols on the row, 3 = Idx 2, 32 = Relative address)

So, the formula for Relative address calc. for 2D array:

> Relative Address = Base + (Size of a row × Your row number) + (Column number × Value size)

SDD:

$S \rightarrow id = E;$ {gen(top.get(id.lexeme)'='E.addr);}

| L=E; {gen(L.addr.base'['L.addr']''='E.addr);}

$E \rightarrow E_1 + E_2$ {E.addr = new Temp();
gen(E.addr'='E_1.addr'+'E_2.addr);}

| id {E.addr = top.get(id.lexeme);}

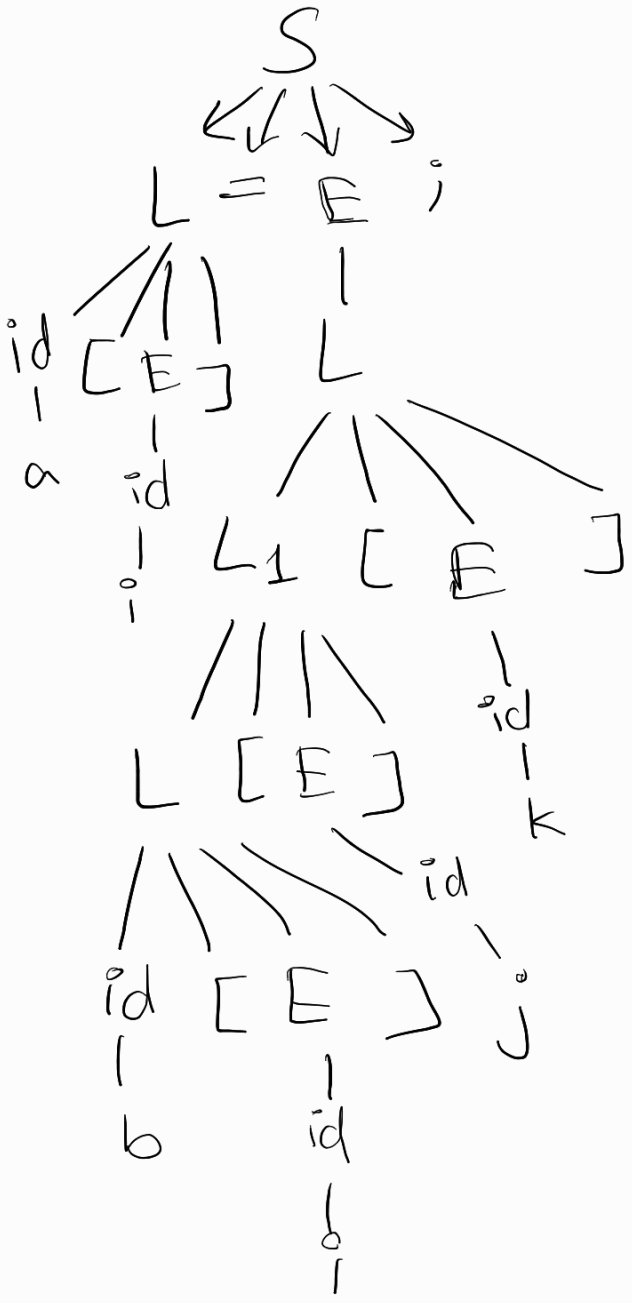
| L {E.addr = new Temp();
gen(E.addr'='L.array.base'['L.addr']');}

$L \rightarrow id[E]$ {L.array = top.get(id.lexeme);
L.type = L.array.type.elem;
L.addr = new Temp();
gen(L.addr'='E.addr'*'L.type.width);}

| L1[E] {L.array = L1.array;
L.type = L1.type.elem;
t = new temp();
L.addr = new Temp();
gen(t'='E.addr'*'L.type.width);
gen(L.addr'='L1.addr'+'t);}

Input String :

a[i] = b[i][j][K];


S
L = E ;
id [E]
L
a
id
i
L1 [E]
id
k
L [E]
id
id [E]]
j
b
id
i

May 3

Saturday

Q) SDD:

$S \rightarrow id = E;$ {gen (top.get(id.lexeme) '=' E.addr);}

| $L = E;$ {gen (L.addr.base '[' L.addr ']' '=' E.addr);}

$E \rightarrow E_1 + E_2$ {E.addr = new Temp();
gen(E.addr '=' E_1.addr '+' E_2.addr);}

| id {E.addr = top.get(id.lexeme);}

| L {E.addr = new Temp();
gen(E.addr '=' L.array.base '[' L.addr ']');}

$L \rightarrow id[E]$ {L.array = top.get(id.lexeme);
L.type = L.array.type.elem;
L.addr = new Temp();
gen(L.addr '=' E.addr '*' L.type.width);}

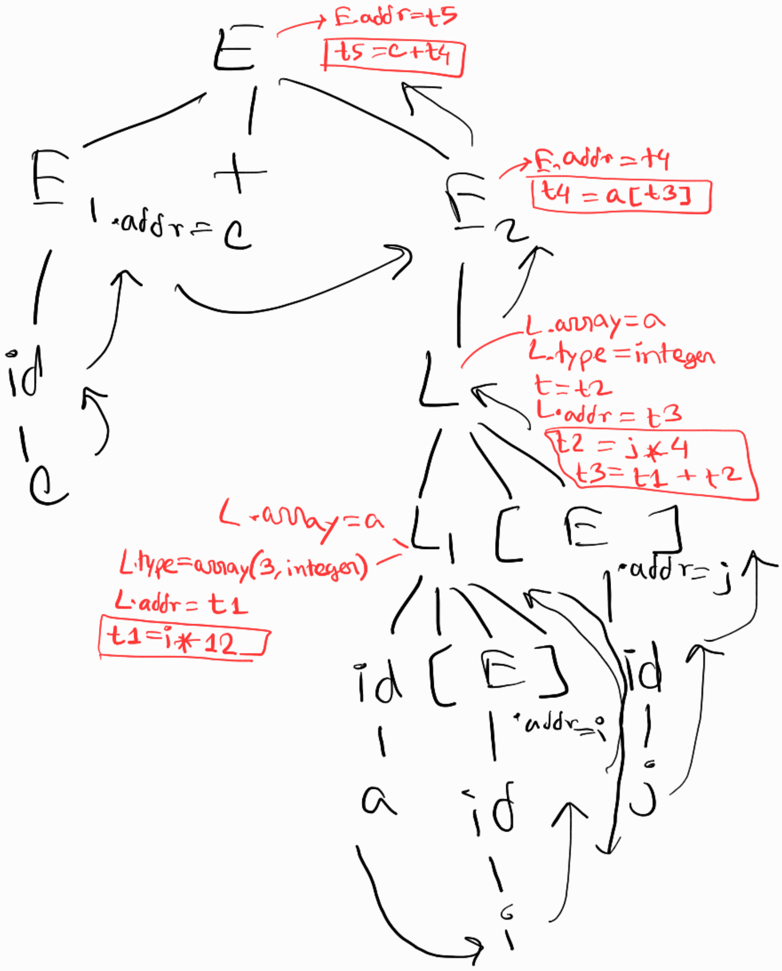
| $L_1[E]$ {L.array = L_1.array;
L.type = L_1.type.elem;
t = new Temp();
L.addr = new Temp();
gen(t '=' E.addr '*' L.type.width);
gen(L.addr '=' L_1.addr '+' t);}

Given,

Let a denote a 2X3 array of integers, and let c, i and j all denote integers. Then, the type of a is array(2, array(3, integer)). Its width is 24, assuming that the width of an integer is 4. The type of a[i] is array(3, integer) of width w1 = 12. The type of a[i][j] is integer.

Input String: c+a[i][j]

Soln:



Base address দেওয়া না থাকলে variable নামই base address ধরবো

Generated code:

t1 = i * 12
t2 = j * 4
t3 = t1 + t2
t4 = a[t3]
t5 = c + t4

May 8

Control Flow

- We will use control flow statements such as if-else statements, while statements. Translation of control flow statements are tied with Boolean expressions.

For writing Boolean expression we use Boolean operators.

Boolean operators are applied to—

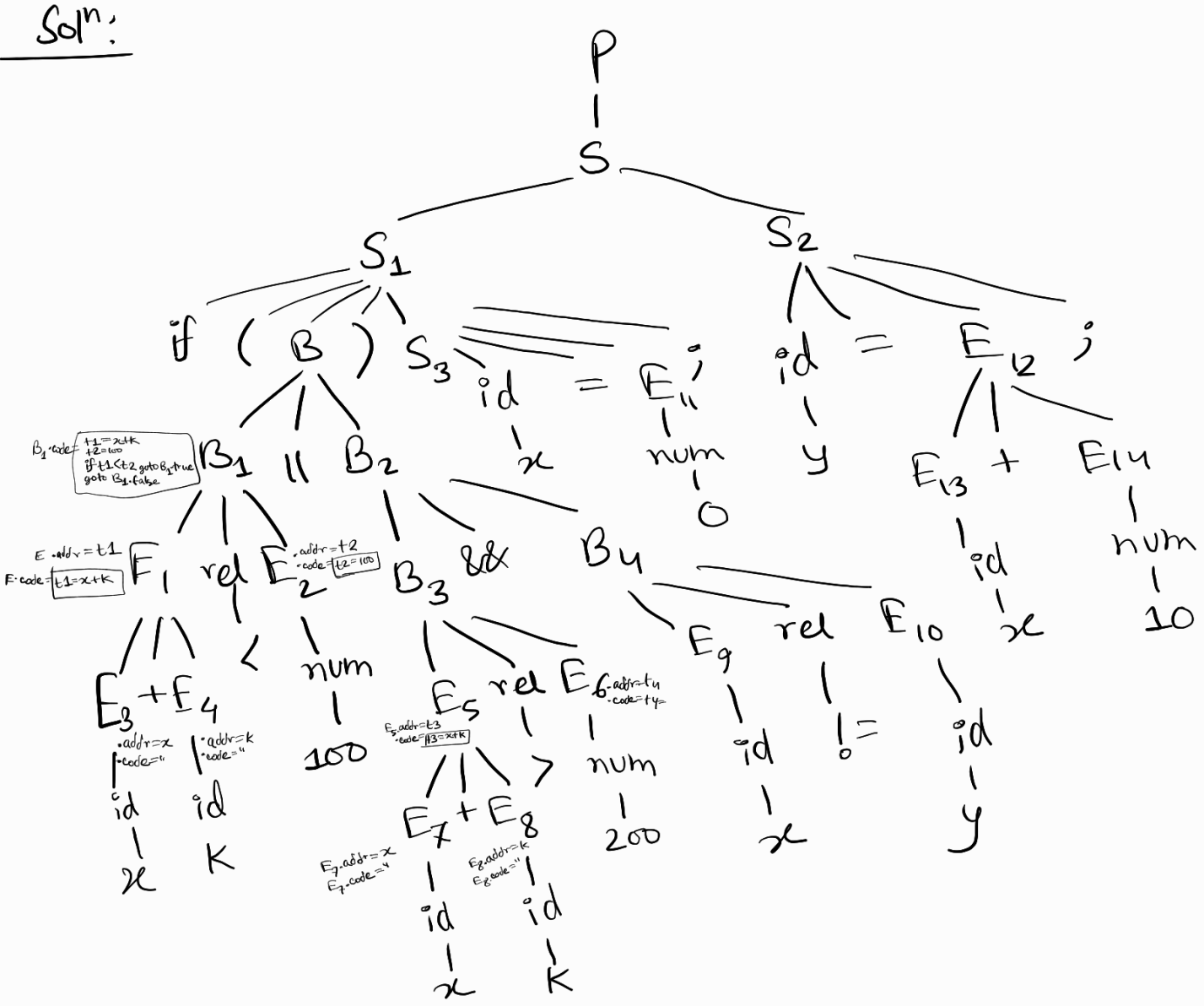
1. Boolean variables
2. Relational expressions with relational operators

In given SDD:

- $S \rightarrow \text{if } (B) S_1$
  $S \rightarrow \text{if } (B) S_1 \text{ else } S_2$
  $S \rightarrow \text{while } (B) S_1$

- এখানে B is Boolean expressions
  S is statement
- বিভিন্ন blocks এ যাবে using labels.
- integer হলে num এ যায়

Soln:



Code values

$t1 = x + K$ (E_1)
$t2 = 100$ (E_2)

$t1 = x + K$
$t2 = 100$
if $t1 < t2$ goto $B_1.true$
goto $B_1.false$ (B_1)

$t3 = x + K$ (E_5)

- synthesized attribute গুলো আগে compute করবো, inherited পরে