Final Assessment: Fall 20-21
Course: Compiler Design
Course code: CSE 323

Submitted by:
Syeda Nowshin Ibnat
ID: 17183103020
Intake: 39
Section: 1
Program: B.Sc. in CSE

1(a) Question Answer

Part-1

Soln:

(i) Three Address Code

$t_1 = a + b$

$t_2 = c + d$

$t_3 = t_1 * t_2$

$t_4 = b + c$

$t_5 = t_3 + t_4$

(ii) Triples

Soln:

	Operator	arg_1	arg_2
0	Add (+)	a	b
1	Add (+)	c	d
2	Mult (*)	0	1
3	Add (+)	b	c
4	Add (+)	2	3

(iii) Indirect Triples

M/A				Op	Arg_1	Arg_2
1000	0	→	1000	+	a	b
1002	1	→	1002	+	c	d
1004	2	→	1004	*	1000	1002
1006	3	→	1006	+	b	c
1008	4	→	1008	+	1004	1006

Part-2

Indirect Triple representation makes use of pointers to the listing of # all references to computations which is made seperately and ~~sorted~~ stored. It's similar in utility as compared to quadruple representation but requires less space.

Advantages

(1) The pointers are smaller than the triples and hence move faster. And this could be used b for quads and many other recordings (e.g sorting large records).

(2) Since the triples do not move, the reference they contain to past result remain accurate.

(3) Temporaries are implicit and easier to rearrange code.

Disadvantages

(1) One can ~~not~~ easily access value of temporary variables using the table. But it's not easier to manipulate on a reorder.

(2) It's not easy to rearrange code for global optimization. Because the result is positional.

(3) It is more complex than the other two representation of three address code, namely → ~~trip~~ Quadruple and Triple.

(4) changing order of statements may cause problems.

1(b) Question Answer

Given Gnammar,

1. $S \rightarrow aTRe$
2. $T \rightarrow Tbc$
3. $T \rightarrow b$
4. $R \rightarrow d$

Input string

abbcde

Stack	Input	Action
$	abbcde $	shift
$ a	bbcde $	shift
$ ab	bcde $	reduce 3
$ aT	bcde $	shift
$ aTb	cde $	shift
$ aTbc	de $	reduce 2
$ aT	de $	shift
$ aTd	e $	reduce 4
$ aTR	e $	shift
$ aTRe	$	~~Re~~ reduce 1
$ S	$	accept

Ans:

2(a) Question Answer

(i)

From the given SDD we can say that the syntax directed definition is propagating the infonmation ~~using~~ using inherited on synthesized attributes.

From the given Syntax-directed definition we can see, this definition associates an integer-valued synthesized attribute called "Val" with some of the nonterminals: F, T, . For each F and T production. The semantic rule computes the value of attribute "Val". ~~For the non~~

Productions	Rules	
1. $T \to F\,T'$	$T'.inh = F.val$	Here, T' = inherited
	$T.val = T'.syn$	Because here child is taking the property of ~~parent~~ sibling ($F.val$).
	$T.val = T'.syn$	child is taking the value of parent. So, it's synthesized.
2. $T' \to *\,F\,T_1'$	$T_1'.inh = T'.inh \times F.val$	
	$T'.syn = T_1'.syn$	

Here, T_1' is a child which is taking the property of parent and sibling both. So, it is inherited.

T' is synthesized, because it is taking the property of child.

It is also ~~true for~~ true for other two rules that SDD is propagating the infonmation.

Here, the terminal digit has a synthesized attribute lexval. F generates a digit, but the operator * is generated by T'. Thus, the left operand appears in a different subtree. So, we can say semantic rules are applied here.

(ii)

Annotated parse tree:

My ID = 20
↳ last digit

Expression: 3 * 5 * 0
= 15 * 0
= 0

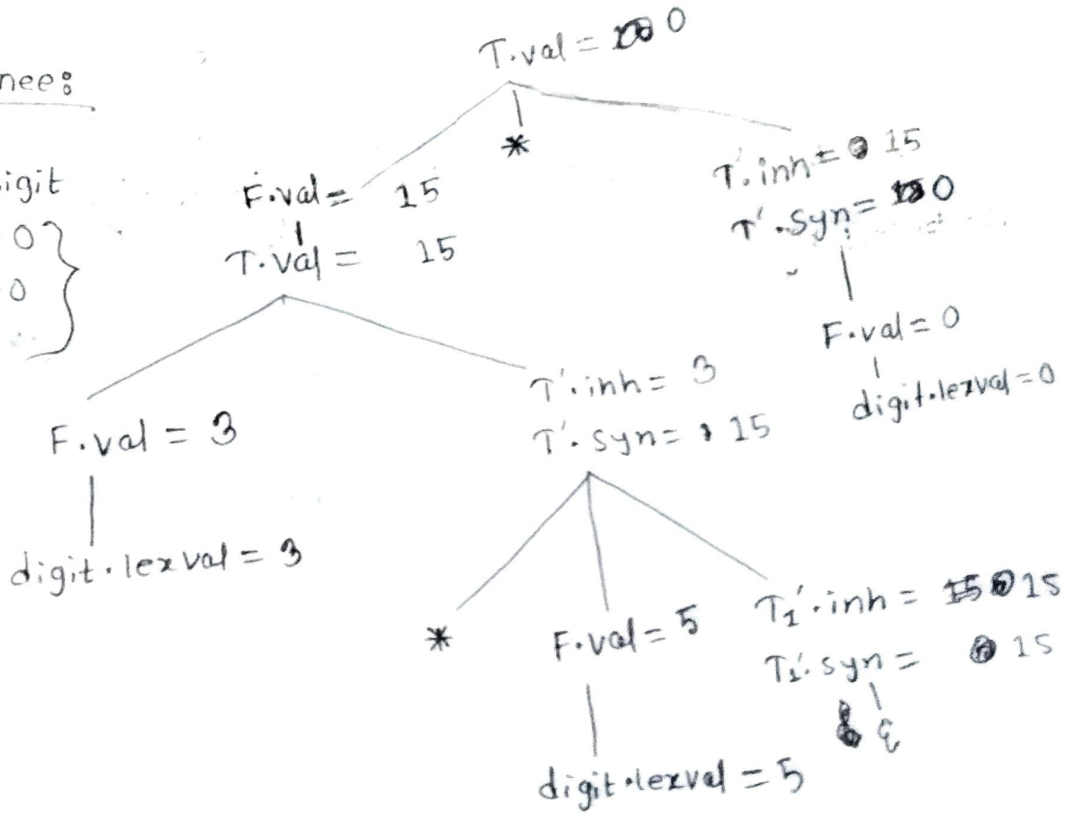


Fig: Annotated parse tree

(iii)

Dependency Graph

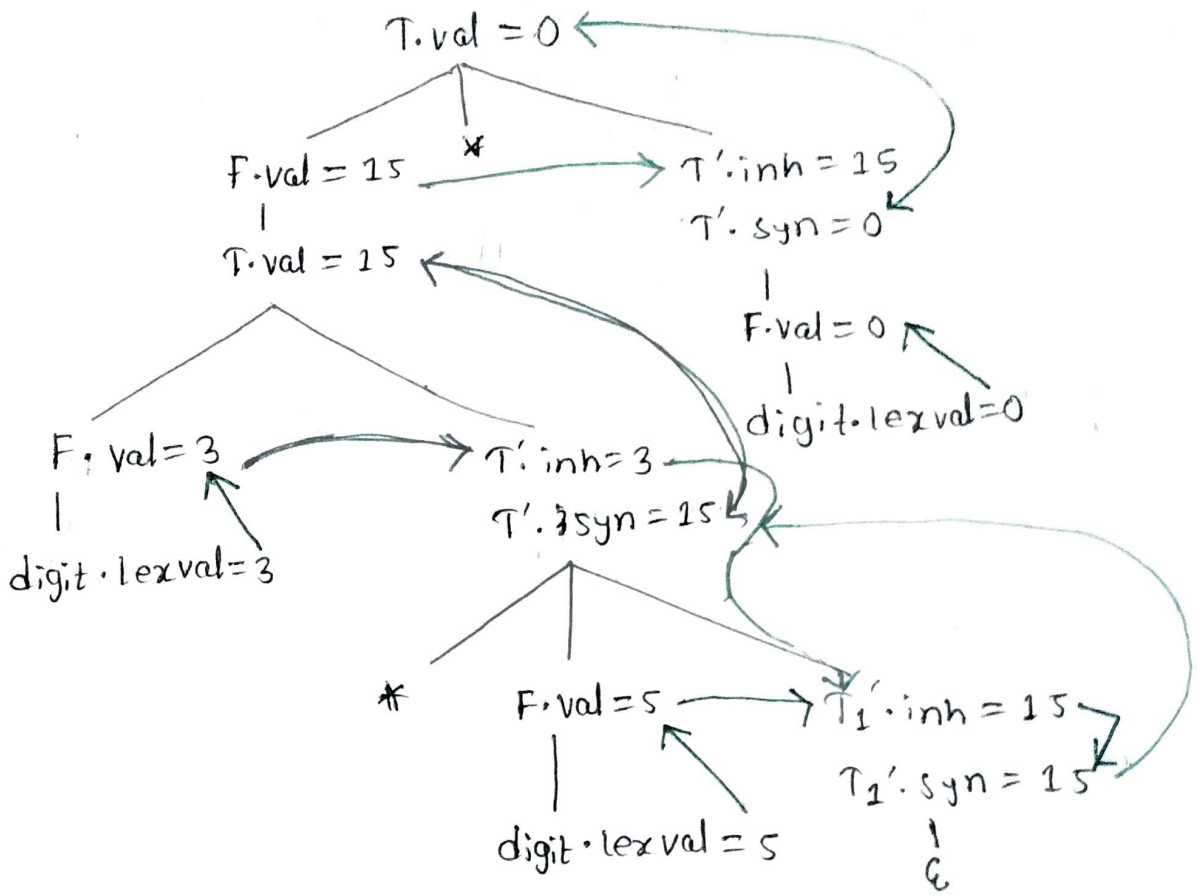


Fig: Dependency graph

2(b) Question Answer

LR Parsing

For this - We ~~have~~ check the ambiguity. In LR(0) We don't have to check the input. But in SLR(1) we have to check the input. LR(0) and SLR(1) both uses LR(0) items. If LR(0) Parsing table has Shift Reduce conflicts then it's not a LR(0) grammar.

SLR Parsing

SLR(1) refers to simple LR parsing. It is same as LR(0) parsing. The only difference is in the parsing Table. To construct SLR(1) parsing Table, we use canonical collection of LR(0) item. If there is no confliction on LR(0) then there will be no confliction on SLR(1). But, if there is a confliction on LR(0) then in SLR(1), there may or may not be a confliction. For this we have to draw the parse tree.

LALR

If there is no conflict in CLR(1) then there may on may not be ~~conflict~~ RR conflict in LALR(1). But if there is no SR conflict in CLR(1) then there is no SR conflicts in LALR(1). ~~Here~~, Number of states are same as SLR(1). If a grammar is CLR it may on may not be LALR.

~~LAL~~ CLR

more

Number of States are ~~lesser~~ here then LR, SLR. But CLR is more powerful. Because here, the number of Blank Space is more - i.e. confliction probability is less.

So, We can write : LR < SLR < LALR < CLR

Ans:

3(a) Question Answer

Part-1

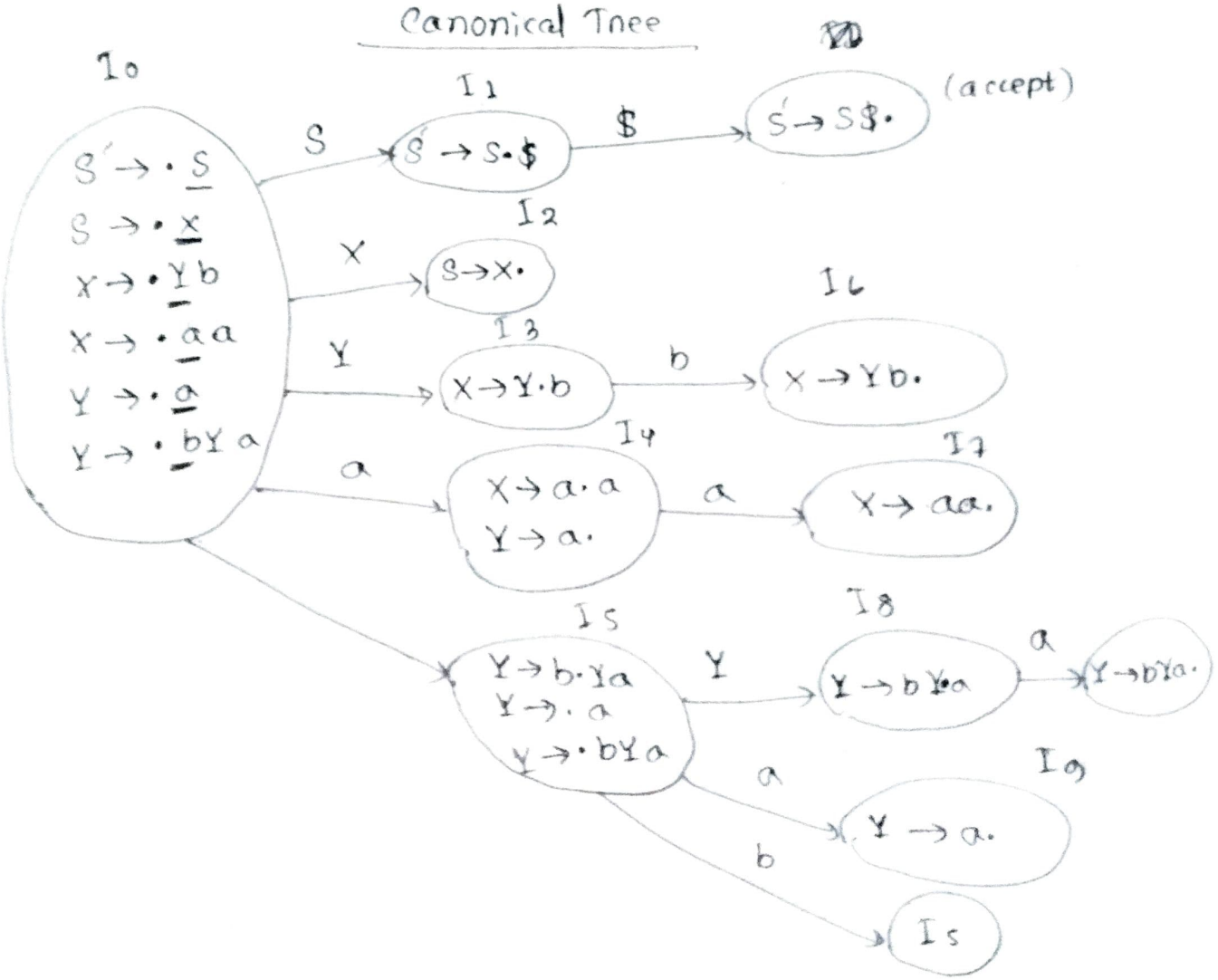
Given, Gnamman, (i)

0. $S \to X$
1. $X \to Yb$
2. $X \to aa$
3. $Y \to a$
4. $Y \to bYa$

Augmented Gnamman

0. $S' \to S$
1. $S \to X$
2. $X \to Yb$
3. $X \to aa$
4. $Y \to a$
5. $Y \to bYa$

Canonical Tree



LR(0) Parsing Table

	Action			GoTo		
	a	b	$	X	Y	S
0	S_4/Π_4	S_5/Π_4	Π_4	2	3	1
1			accept			
2	Π_2	Π_2	Π_2			
3		S_6				
4	S_7					
5	S_9	S_5			8	
6	Π_2	Π_2	Π_2			
7	Π_3	Π_3	Π_3			
8	S_{10}					
9	Π_4	Π_4	Π_4			
10	Π_5	Π_5	Π_5			

Here, S/R conflict exists. So, this grammar is not a LR(0) Grammar.

SLR(1) Parse Table

Follow (S) = {$}
Follow (X) = {$}
Follow (Y) = {a, b}

	Action			Goto		
	a	b	$	X	Y	S
0	S_4/Π_4	S_4/Π_4		2	1	3
1			accept			
2		S_6	Π_1			
3		S_6				
4	S_7					
5	S_9	S_5			8	
6			Π_2			
7			Π_3			
8	S_{10}					
9	Π_4	Π_4				
10	Π_5	Π_5				

Here, S/R conflict exists. So, this grammar is n't a SLR(1) Grammar.

Part-2

(ii)

The given Statement is true that any Grammar parsed by an LR(0) parser can be parsed by an SLR(1) parser.

SLR(1) Parsers can parse a large number of grammers than LR(0). When any grammar LR(0) grammar has no S/R (Shif reduce) on R/R (Reduce reduce) conflict then it can be parsed. So, then SLR(1) ~~grammar~~ can easily parsed this. ~~LR(0)~~ grammar. Because SLR(1) Resolve S/R and R/R conflict problem.

So we can say the given Statement is true.

3(b) Question Answer

A context-free grammar is said to be ambiguous if it permits more than one phrase structure to describe a single input text. Most conflicts are the result of such ambiguities, and there are three ways of resolving them:

(1) change the grammar so that only one phrase structure is possible.

(2) Provide additional information that causes the parser to select one of the set of phrase structures.

(3) change the form of the input text to avoid the ambiguity.

Here, all of these methods results in the parser recognizing a different language than one that is described by the original grammar.

Ans:

4(a) Question Answer

Leader:
1) Target of GOTO
2) Follows of GOTO

Basic Blocks

:

Label-20: i = b+c — B_1
b = 10
K = 9

Label-21: a = b+c
d = e+f
K = j+1 — B_2
if P > 10 goto ~~Label-22~~ B_3

---

label-22: goto label ~~2~~4 B_5 — B_3

---

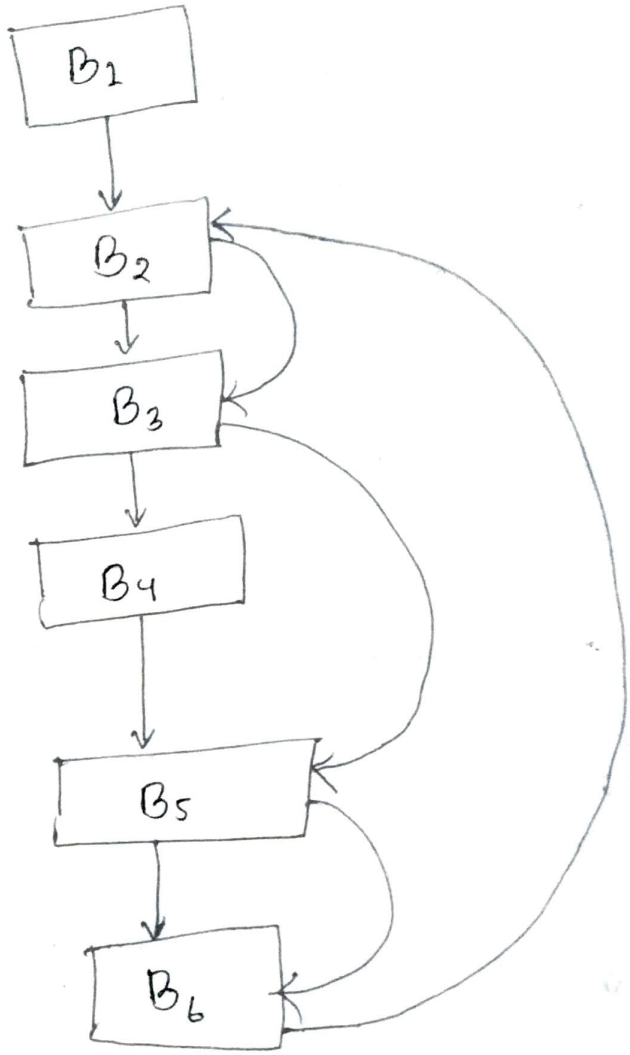
label-23: e = 5 — B_4

Label-~~2~~4: goto Label ~~2~~25 B_6 — B_5

Label-25: g = -K
F = d+4 — B_6
c = 25
h = e+10
j = b+j
if q > 10 goto ~~lebel-21~~ B_2

:

Control Flow Graph



4b Question Answer

(i)

Common Subexpression Elimination

■ → Eleminated expression