FA17-BCS-062 final exam

Rimsha Bilal

a) Calculate S5 if "&" has higher Priority as compared To '#' by modifying CFG G3 with Semantic rule.

Calculate S5

if & has higher priority as Compare Compared to # then it should be below in Grammer i.e

Our Grammer will be

E → T# F

E → T

T → E& T

T → f

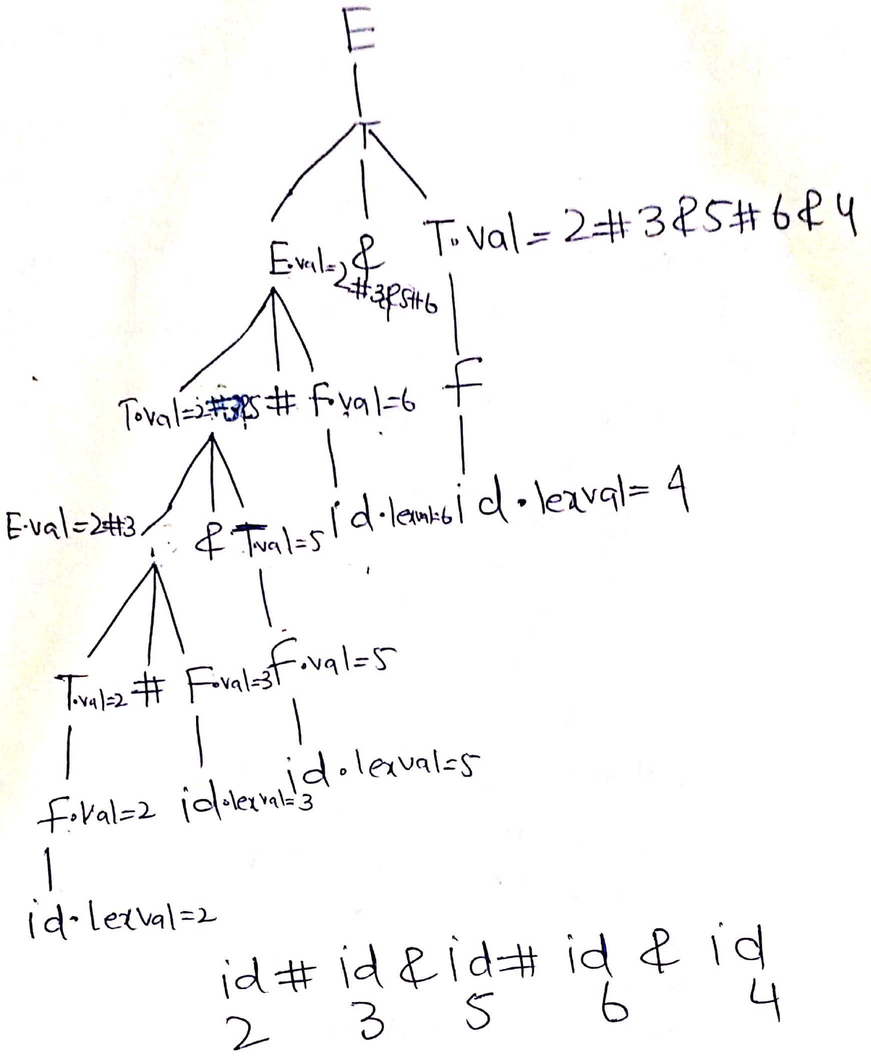
F → id

Calculating S5

S5 → 2#3& 5# 6& 4

E → T# F	E.Val → T.Val # F.Val
E → T	E.Val → T.Val
T → E&T	T.Val → E.Val & T.Val
T → f	T.Val → F.Val
f → id	f.Val → id.lexval

Tree


E
T
E.val=2#3&5#6
&
T.val=2#3&5#6&4
T.val=2#3&5
#
f.val=6
f
E.val=2#3
&
T.val=5
id.lexval=6
id.lexval=4
T.val=2
#
F.val=3
f.val=5
f.val=2
id.lexval=3
id.lexval=5
id.lexval=2
id # id & id # id & id
2 3 5 6 4

FA17-BCS-062-B (Termina Exam)
Rimsha Bilal
Compiler construction

Question 1

(b) write type-checking Semantic rule for true or false reasoning of

$E_1 \rightarrow E_2 = E_3$

If (E.type == E3.type) and (E.type = int|boolean)

then E1.type = boolean error

C (Part)

= Representation which convert the Source languages is called intermediate code write name of Those representation Types and Subtype.

The representation that represents the Source code in efficient way, called as intermediate code

Type of intermediate code

1- Polish Notations
- Infix to prefix
- Infix to postfix
- Prefix to postfix

like these

2- Three address code
Quadruples
Tripples
indirect triples ⟹ Abstract Syntax tree

it represents upcoming code in efficient ways by using solutions.

Symentic Analysis → intermediate code → code optimization

intermediate code

```
temp1 = m to realval(10)
temp2 = id3 * tem1
temp3 = Id2+temp2
Id1 = temp3
```

Code Generation ←

```
Id1   Id2   Id3
a  =  b  +  c*10
```

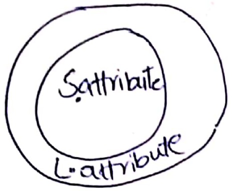
(inefficient way)

d):-Is L-attributed grammer better than S-attributed? How? Describe with example

So, L attribute is better than S-attribute As it involve both Synthesized and inherited attribute.

In S attribute in Grammer Semantic is only placed at right position.

In L attribute we placed Semantic rule at only where at start at end at middle.



e) part

What are two main factors That make code inefficient discussed in topic code optimization.

The code become inefficient due to two factors

1 Programmer

2 compiler

f (Part)

Convert S_2 by using G_2 to an equivalent decimal Number.

$S_2 \rightarrow 1100\text{l}\cdot 1001$

$1001 = 9$

$$
\begin{array}{l}
1 \times 2 + 1 \\
\hline
\quad 3 * 2 + 0 \\
\hline
\qquad 6 * 2 + 0 \\
\hline
\qquad\quad 12 * 12 + 1 \\
\hline
\qquad\qquad 25
\end{array}
$$

so $\frac{9}{2^4} = 0{\cdot}5625$

So Then answer is $= 25{\cdot}5625$

$N \rightarrow L_1 \cdot L_2$	$\{N\cdot dval = L_1\cdot dval + \frac{L_2\cdot dval}{2^\wedge(L_2\cdot count)}\}$
$\rightarrow LB$	$\{L\cdot C = L_1\cdot ct\ B\cdot C,\ L\cdot dval = L_1\cdot dval + B\cdot dval\}$
$L \rightarrow B$	$\{L\cdot C = B\cdot C;\ L\cdot dval = B\cdot dval\}$
$B \rightarrow 0$	$\{B\cdot C = 1,\ B\cdot dval = 0\}$
$B \rightarrow 1$	$\{B\cdot C = 1,\ B\cdot dval = 1\}$

9 (Part)

S → f = ex

ex → ex addopterm | term

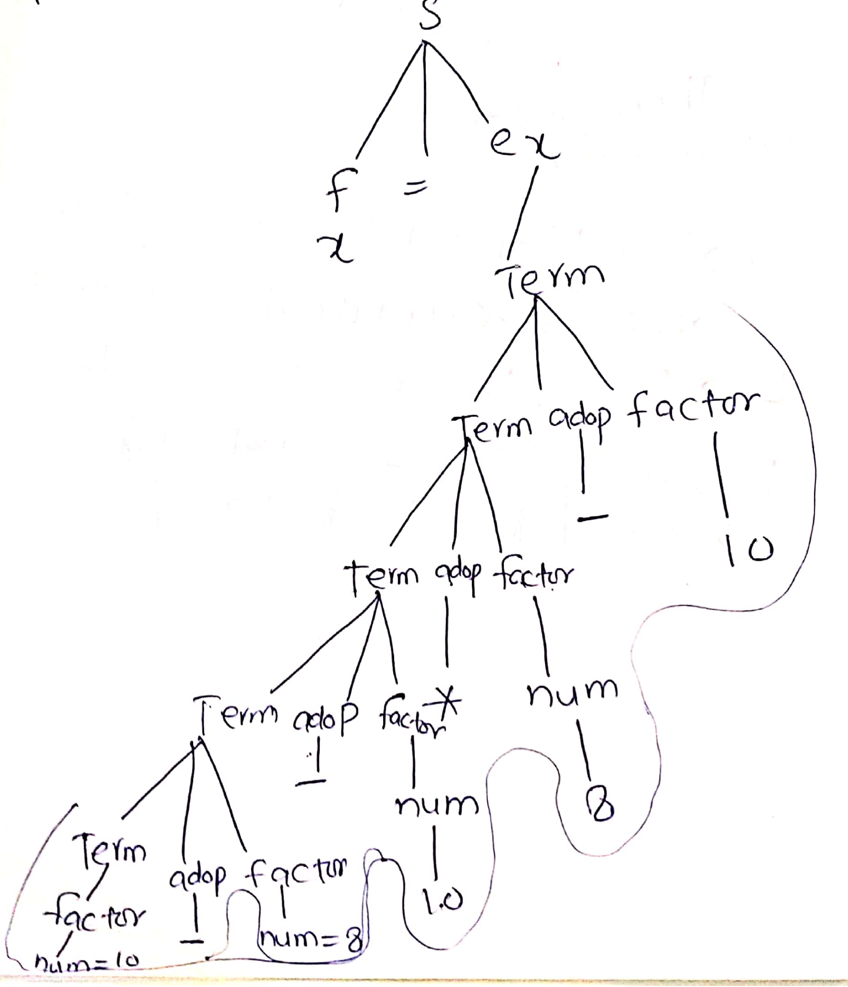
term → Term adop factor | factor

addop → + | - | * | /

factor → X | num

String 3 = x=10-8-10* 8-10

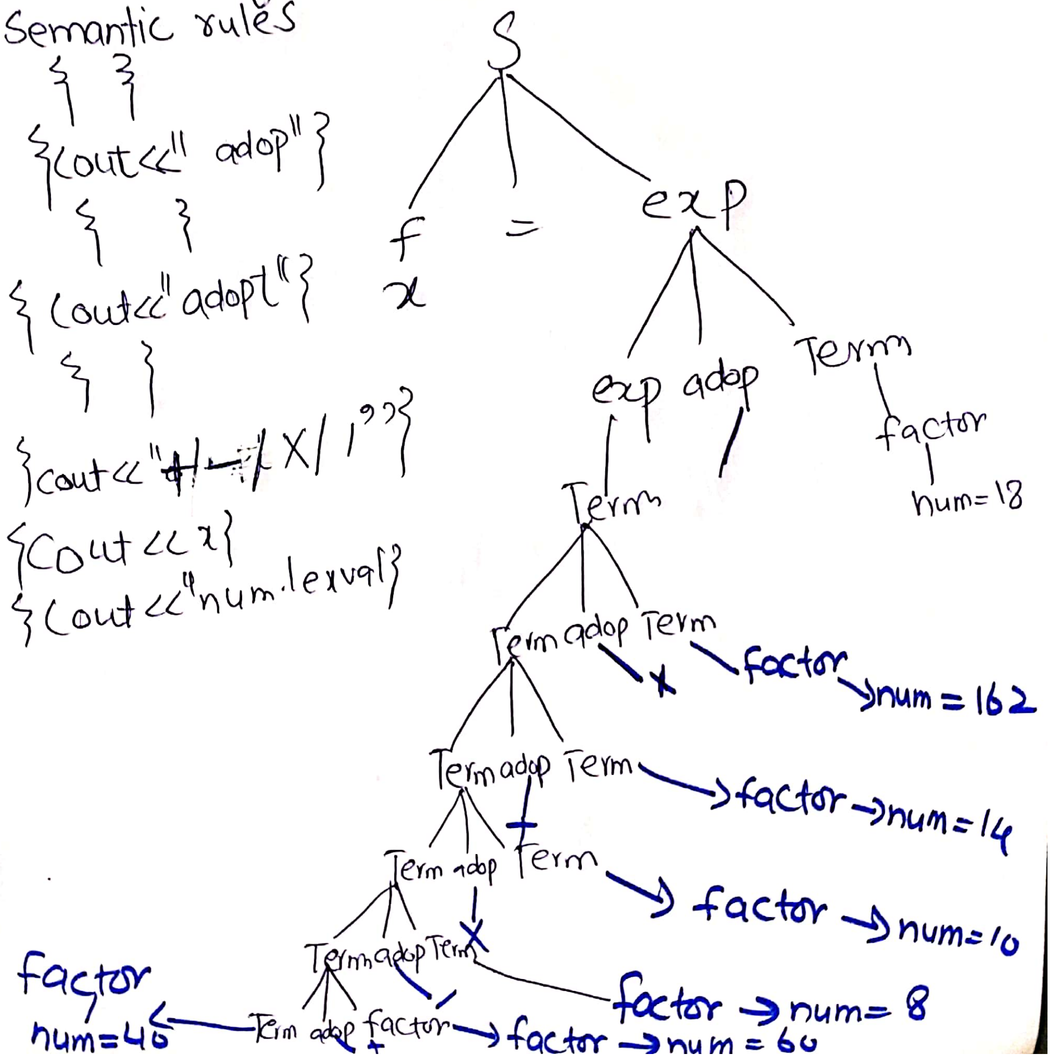
Annotated parse tree



h (Part)

Convert S1 into Postfix operator Through G1

Postfix X= 4060+810*-8414/162*18/+



Semantic rules

{ }

{cout<<" adop"}

{ }

{cout<<"adopt"}

{ }

{cout<<"~~+/-~~*/X/1"}

{Cout<<x}

{Cout<<"num.lexval}

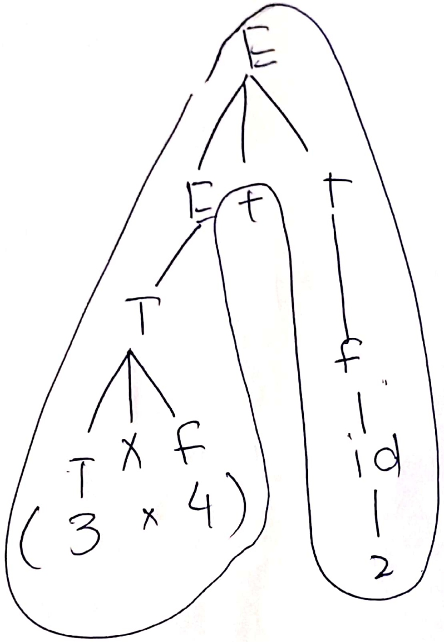
Infix= 40+60 - 8*10+ 84/14 *
Postfix = 4060 + 810 * - 8414/162 * 18/+

I (Part)

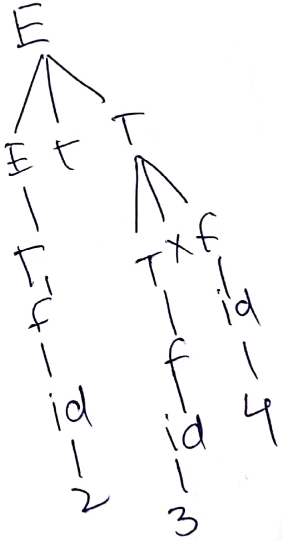
C.f.G

$E \rightarrow E+T \mid T$
$T \rightarrow T*F \mid F$
$F \rightarrow id$

Here we have
$S_4 \rightarrow 2+3*4$

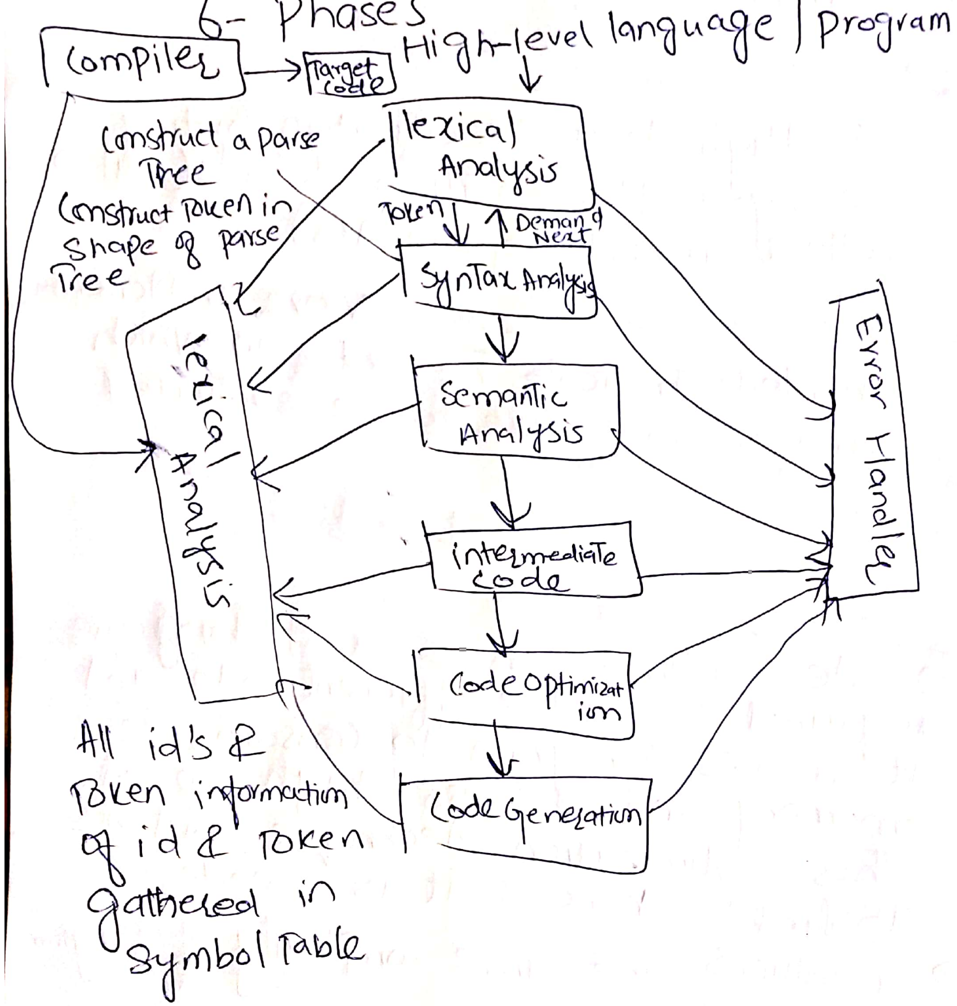


(3×4)+2=14



J) Show workflow of source program to Target code diagrammatically by dividing into Three main Phases. write Their phases and sub phases.

6- Phases



K: Discuss notations for attaching Semantic rules?

There are two notations for attaching Semantic rules.

1. Syntax directed Definition
High-level specification hiding many implementation details (also called attribute grammars)

2 Translation Schemes. More implementati oriented: Indicate the order in which Semantic rule are to be evaluated.

l (Part)
what is code optimization?.....

code optimization:-
→ Code optimization is a technique required to produce efficient code and it makes program to consume less memory and delivers high speed. This optimization technique will be applied whenever it is needed.
→ Reduces the time complexity of Program

Issues:-
There are two issues
→ Meaning of The Source code Should be Changed

→ The efficiency of The Source code must be gained without changing the algorithm.

Techniqúes:-

→ Dead code Elimination

→ Common Sub Expression Elimination

→ Strength Reduction

→ Code Movement.

Question:2 :-

Convert S6 To 3AC

$S_6 \longrightarrow a + b * c / e ^ f + -b + a$

Solve :

$t1 = -b$ (uminus b)

$t2 = e \wedge f$

$t3 = b \times c$

$t4 = t3 / t2$

$t5 = a + t4$

$t6 = t5 + t1$

$t7 = t6 + a$

Question: 3

Semantic Rule

(1) {Cout<<" = "}

(2) { Cout <<" adop"}

(3) {}

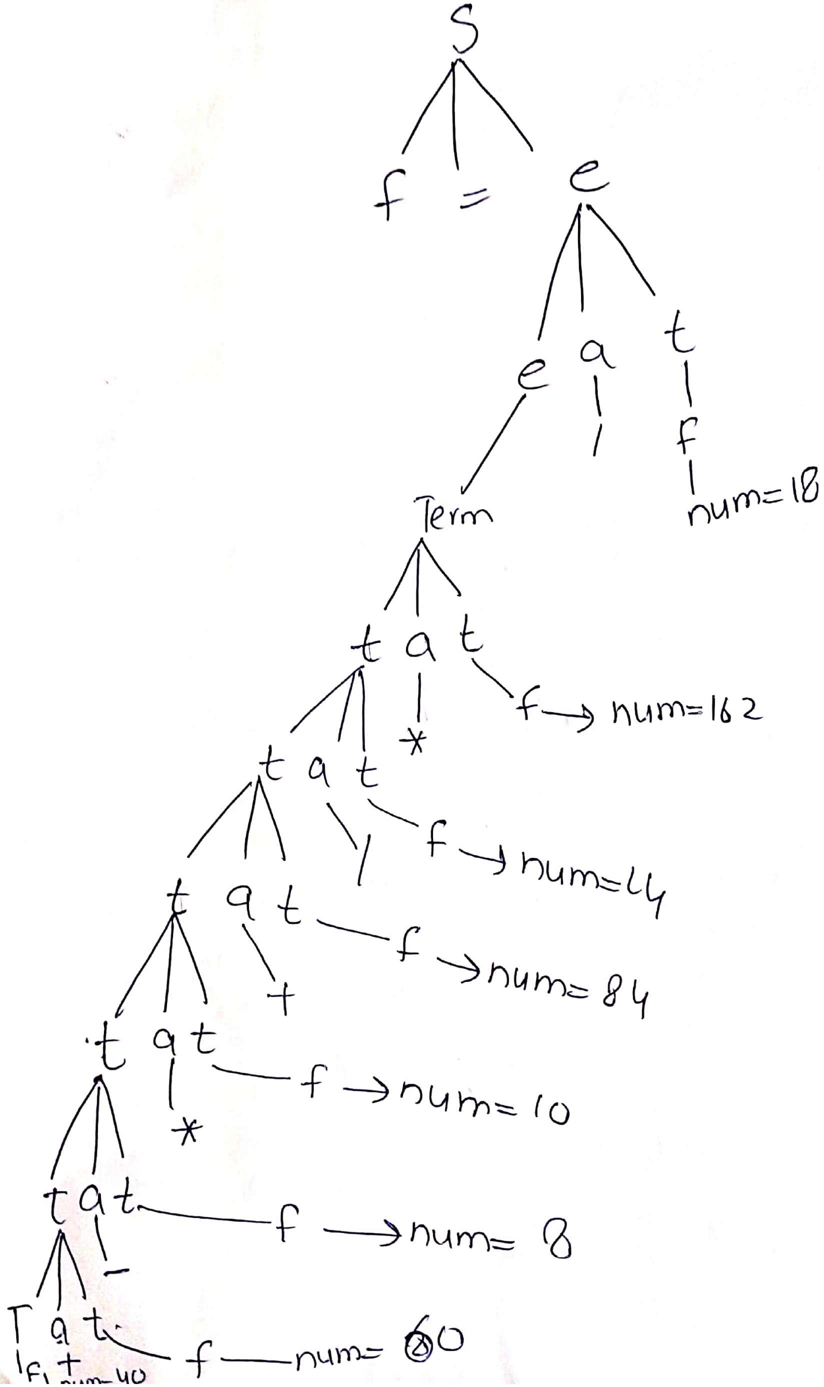
(4) {cout<<" adop"}

(5) {}

(6) {}

7 {}

(8) { }


S
f
=
e
e
a
t
/
f
num=18
Term
t a t
*
f
num=162
t a t
/
f
num=44
t a t
+
f
num=84
t a t
*
f
num=10
t a t
-
f
num= 8
T a t
F
+
num=40
f
num= 60

Prefix= x=40+*60-810/84x14/162

(4) Solution:

Muqsit → Bahadur Aqib|Ali
Bahadur → Yasir Awais
Yasir → Daud zaid | waqas
Daud → Daud Zaid| Muqsit Aqib|
Yasir waqas

Removing Indirect Recursion:-

DAUD → DAUDzaid | Bahadur Aqib Aqib|
Ali Aqib| Yasir waqas
DAUD→ DAUD Zaid| DAUD zaid Aqib Aqib|
waqas Aqib Aqib| Ali Aqib| Yasir waqas
DAUD→ DAUDzaid| DauD zaid Aqib Aqib|
waqas Aqib Aqib| Ali Akib| Daud zaid|
waq as|

Now Remove left Recursion

DAUD→ waqas Akib Akib DAUD' |Ali
Kqib DAUD' /waqas waqas DAUD'
DAUD'→ Zaid DAUD' |Zaid Aqib Aqib
DAUD' |Zaid waqas DAUD' |ε

write Gramer correctly

Muqsit → Bahadur Aqib|Ali
Bahadur → Yasir Awais
Yasir → DAUD zaid /waqas
DAUD → waqas AqibAqib DAUD' /Ali Akib
Daud' | waqas waqaS/DauD'

DAUD' → zaid DAUD' / zaid Aqib Aqib Daud / zaid waqas Daud / ε

First	follow
{Ali, waqas}	{$}
{waqas, Ali}	{Aqib}
{waqas, Ali}	{AWAIS}
{waqas, Ali}	{zaid}
{zaid, ε}	{zaid}