KATHMANDU UNIVERSITY

COMPILER.

2nd Internal

Submitted by
Prabhat Neupane
CS
32.

Submitted to:
Mr. Nabin Ghimire.

①

Ans. The weakness of Simple LR parser ~~...~~ is that, ~~...~~ procedure of LR parser produce reduce action and all reduce action ~~might not~~ must occupy an entire table row, causing the reduction to occur regardless of next symbol in input stream.

The canonical LR(1) parsing table:

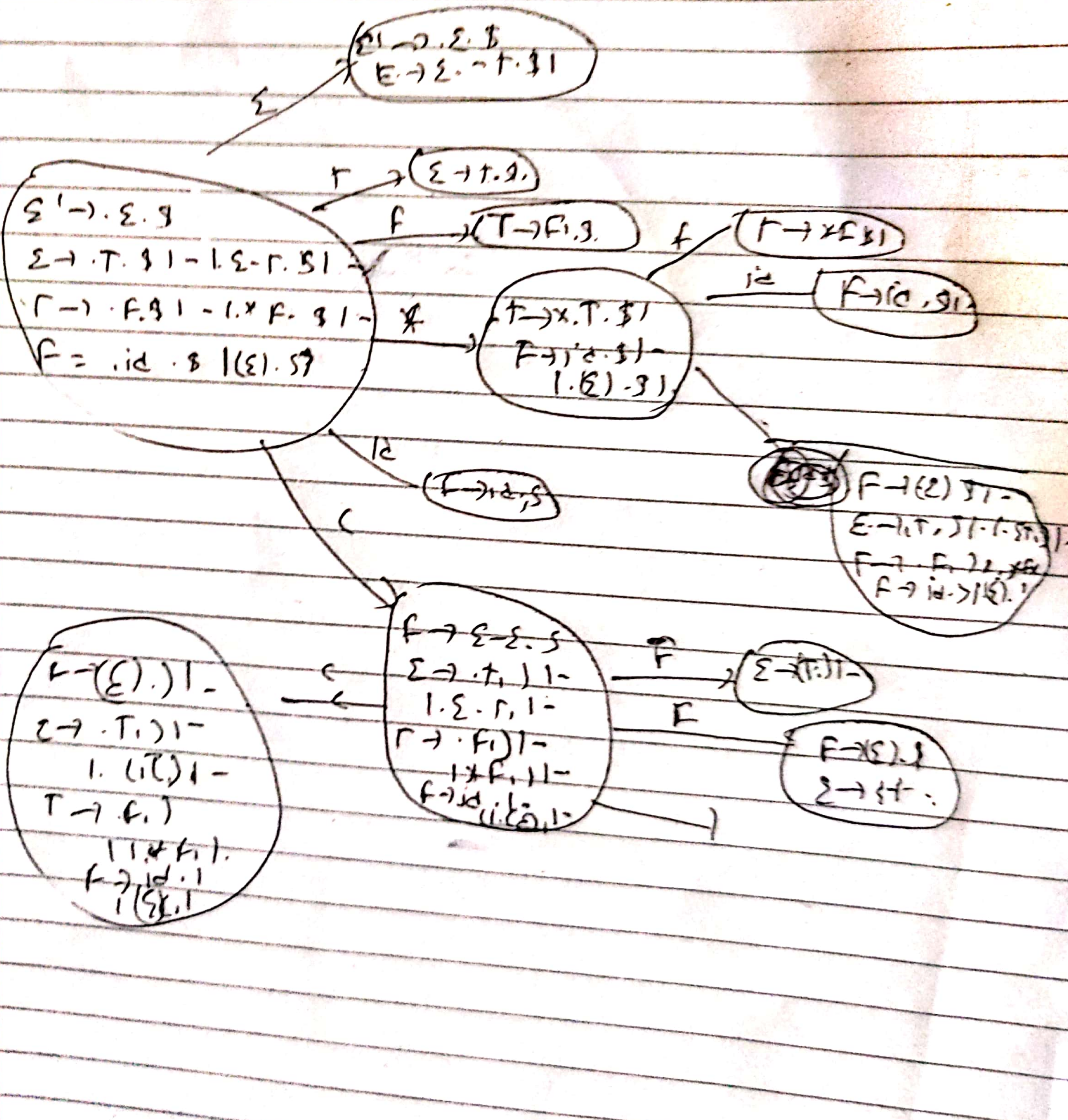
$E \rightarrow T \mid E - T.$

$T \rightarrow F \mid {}^{*}F.$

$F \rightarrow id \mid (E).$

Agumnented grammar:

$E' \rightarrow E$	$E' \rightarrow .E$
$E \rightarrow T$	$E \rightarrow .T.$
$E \rightarrow E - T.$	$E \rightarrow .E - T.$
$T \rightarrow F.$	$T \rightarrow .F$
$T \rightarrow {*}F$	$T \rightarrow .{*}F$
$F = id.$	$F \rightarrow .id.$
$F = (E)$	$F = .(E)$



S' → . E . $
E → . T . $ | - | . E - T . $ | -
T → . F . $ | - | . * F . $ | -
F = . id . $ | (E) . $
E
S' → . E . $
E → E . - T . $ |
T
E → T . $,
F
T → F . $
*
T → * . T . $ |
F → . id . $ | - | . (E) . $ |
F
T → * F . $ |
id
F → id . $ |
id
F → id . $
(
F → (E) $ | -
E → . T ,) | - | . E - T ,) | -
T → . F ,) | * F
F → id .) | (E) . |
F → E - E . $
E → . T ,) | -
| . E - T , | -
T → . F ,) | -
| * F ,) | -
F → id , | . (E) , | -
(
F → (E) .) | -
E → . T ,) | -
| . (T) | -
T → . F ,)
| | * F ,) .
F → id .)
| (E) .)
F
E → (T .) | -
E
F → (E) . $
E → E - .
)

Date / /
Page No.

(2)

Answer

A syntax-directed defination binds a set of semantic rule to production. They are high level specifications for the translation schemes i.e. They hide the implementation details and do not necessitate the consideration of translation order. A compiler can exploit type information for better performance. Syntax directed defination can be used in type checking because it can hold the value/rules set for certain types to occur. For example, if we are taking a number and type-checking it as integer, in this case, SDD will match rule of a number to be integer and when all rules are satisfied it resolve it as an integer.

Example of type-checking expression in SDD:

E → literal. {E.type = char},
E → num {E.type = integer},
E → id. {E.type = lookup(id.entry)},
E → E1 [E2]. {E.type = if E2.type = integer and E1.type = array(S,t), then t else error}.
E → E1* {if. E1.type = pointer(t) then t else error}.

③

Optimization is often required ~~by s~~ in the code generator by simple code generator because optimized codes has following features:-

① maximize the efficiency of executable program.
② transformation done by compiler as well as programmer.
③ preserve the meaning of programs.

Thus a simple code generator:
① converts the intermediate representation of source code into form that can be readily executed by machine.
② is expected to generate the correct code.
③ product/results should be easily implemented, tested and maintained.

Thus, optimization is required in code generated by simple code generator.

Unreachable code are the part of source code that will never be executed due to inappropriate exit point/ control flow, or the code is refferred as dead code, i.e. code might get executed but has no effect on the functionality of the system. Such codes are optimized by:

① Code elimination: - inherent to each of the optimization passes.
② Constant folding and Combining: precalculating the constant expression during compiling that execution.

③ Loop-Jamming: if two loops have same number of iterations and they use same indices.

④ loop optimization: to reduce overheads associated with executing loops.

~~Example!~~

~~If (a = b*c)~~
~~x = a~~
~~elseif (b = a*c)~~
~~y = b~~
~~elseif (a == b == c)~~
~~x = 0~~
else
~~x = 1~~

~~optimized~~
~~if (a = b*c)~~
~~x = a~~
~~elseif (~~

source

```
if (a<b)
  x=a
elseif (a>b)
  x=b
elseif (a==b)
  x=0
else
  x=1
```

optimized

```
if (a<b)
  x=a
elseif (a>b)
  x=b
else
  m=0.
```

Date / /

Page No.

(4)

Answer.

Syntax-Directed translation is the combination of context-free grammar and ~~symen~~ semantic rule. Here we can place semantic in any way in right hand of the production. It involves passing the information bottom-up and/or top-down the parse tree in form of attributes attached to the nodes. They uses lexical value of nodes, constants and attributes associated to non-terminal in their definition. Its general approach is to construct a parse tree or a syntax tree and compute the value of attributes at the node of trees by visiting them in some order.

Syntax-directed translation = grammar + semantic rule.

Now for the SDT to convert infix ~~expre~~ expression to postfix expression, we have.

Infix notation: operand operator operand.

postfix notation: operand operand operator.

Example:

$E \rightarrow E + T$	{E.val = E.val + T.val}.
$E \rightarrow T$	{E.val = T.val}.
$T \rightarrow T * F$.	{T.val = T.val + F.val}
$T \rightarrow F$.	{T.val = F.val}
$F \rightarrow (F)$.	{F.val = F.val}.
$F \rightarrow num$	{F.val = Lex.number}

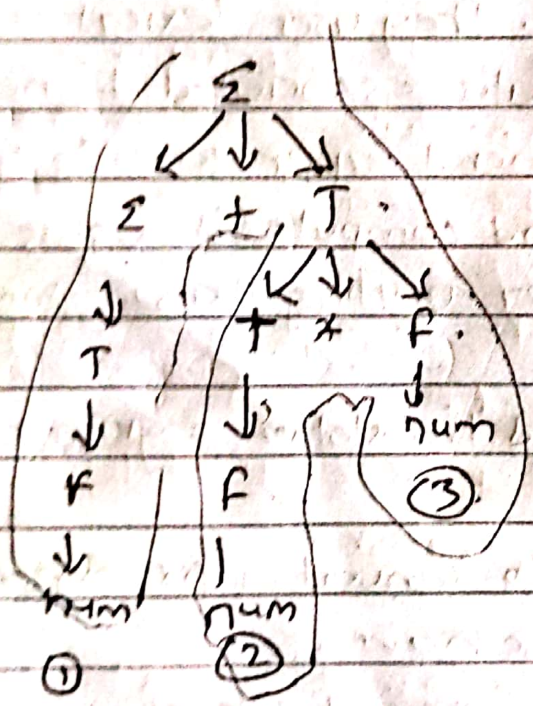
Scanned with CamScanner

Date / /
Page No.

Now, E → E+T {print '+'}.
E → T {}
T → T*F {print '*'}
T → F. {}
F → num {print lex.value}.

~~Now, compute 4+9*5~~

If 1+2*3 be infix then using SDT.



Traversing top to bottom and left to right, we have, 1 2 3 * + which is required postfix expression.