Lecture -14

Context Free Grammar and Context Free Langu

CFG

$G = (V, T, P, S)$

V = finite non empty variable / non-terminal.

$\{V_1, V_2, V_3, S\}$ এখানে S → যেসবগুলো variable দ্বারা overall CFG টা যেই language কে represent করছে।

T = set of terminal symbols. $V \cap T = \phi$ [variable & Terminal common কিছু থাকবে না]

P = set of Production or grammer rules.

$A \to \alpha$ where [head] $A \in V$

[body] $\alpha \in (V \cup T)^*$

[production symbol] '→' means "could take the value"

S = one of the variables represents the language being defined. it is called the start symbol.

$S \in V$

Capital letter → variable/non terminal

small letter → terminal / symbol

Q: Grammar দেয়া থাকবে আর তাকলকে চমস্কি- normal form-এ নাও।

Simplification of CFGs

→ Eliminating Useless symbol:

* Use full symbol ছাড়া কতগুলো eliminate করব।

$S \rightarrow * \alpha X \beta \rightarrow * w$ ← it must be terminal.

* Generating symbol → X is generating symbol.

if $X \rightarrow * w$ [terminal symbol অবশ্যই generating symbol]

* Reachable Symbols → $S \rightarrow * \alpha X \beta$

$\alpha \rightarrow (V \cup T)^* (V \text{ Union } T)^*$ তাই α terminal ও হতে পারে আবার variable ও।

α যদি generating symbol এর combination হয় তাহলে α ও generative তাই A ও।

(*) Let assume A is reachable then we have to find A কাদের head. অর্থাৎ A এর multiple rules থাকে $A \rightarrow 0$ $A \rightarrow \dots$ $A \rightarrow$ তাহলে A যদি reachable হয় তাহলে তার body র সব symbolই reachable.

→ Eliminating ε Productions:

* A will be nullable if $A \rightarrow *\varepsilon$

Example: $A \rightarrow B \mid \varepsilon$ এখানেও A nullable.

$S \rightarrow AB$
$A \rightarrow aAA \mid \varepsilon$ Here, A, B both nullable.
$B \rightarrow bBB \mid \varepsilon$

nullable হলেই যে always ε হয়ে পৌঁছাবে এমন নাও হতে পারে। তাই আমাদের দুইটা case ই consider করতে হবে। nullable হলে কি হবে এবং না হলে কি হবে।

Let assume none is nullable,

$S \rightarrow AB$ ✓ ⓐ so, $S \rightarrow AB \mid A \mid B$
$S \rightarrow A$ if B nullable
$S \rightarrow B$ if A

Again,
$A \rightarrow aAA$ $A \rightarrow aAA \mid aA \mid aA \mid a$

same for ⓑ B.

তাহলে Simplified Grammar হবে,

$S \rightarrow AB \mid A \mid B$.
$A \rightarrow aAA \mid aA \mid a$
$B \rightarrow bBB \mid bB \mid b$

→ Eliminate Unit Productions

* Unit Production হচ্ছে যেই grammer-এ Production (→) র দুই পাশেই Variable অথবা non terminal থাকে।

Here $E \to a$ not Unit Production

$A \to AB$ is " " [কোনটা কোনটা থাকবে]

* Unit Production Pair (A,A) (B,B) ....

Grammer-এ না থাকলেও use করা যায়।

$A \to ABC$ → এটা unit P না

$A \to B$ → এটা unit P

$A \to A$ unit

$C \to \epsilon$

বাকিটুকু lecture-16 এবেক -

Simplify করতে order maintain করতে হবে,

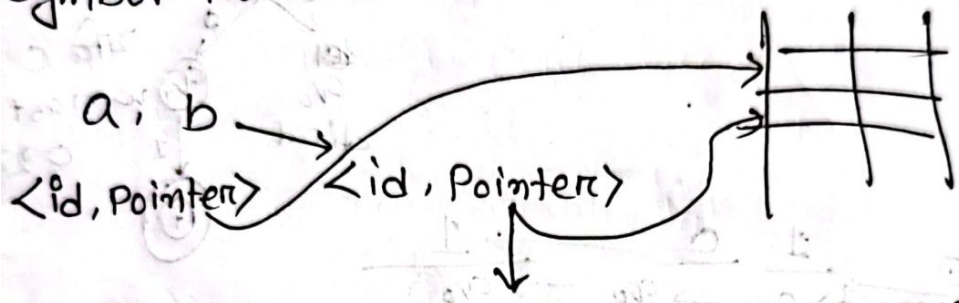
1. Eliminate ϵ-production
2. " Unit "
3. " useless symbols.

Lecture-11

Lexical Analysis & Symbol Table

int a ;
↓ ↓
<kw> <id> <sep>

Symbol Table →



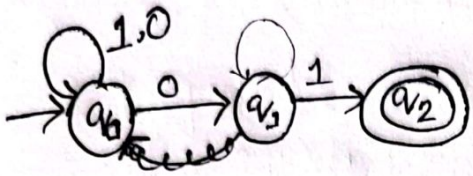
<id, attribute value> symbol table-এ কোথায় যাবে,

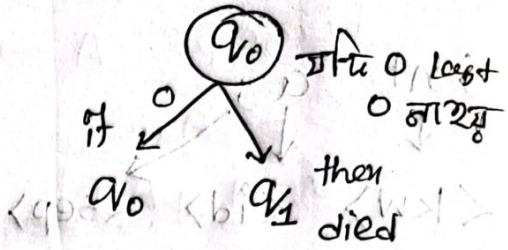
Attribute value টা optional করুন suppose যদি

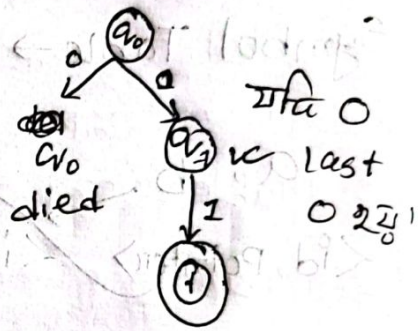
a+b
<id, -> <+> <id, -> এভাবে রাখা যায়,

NFA design

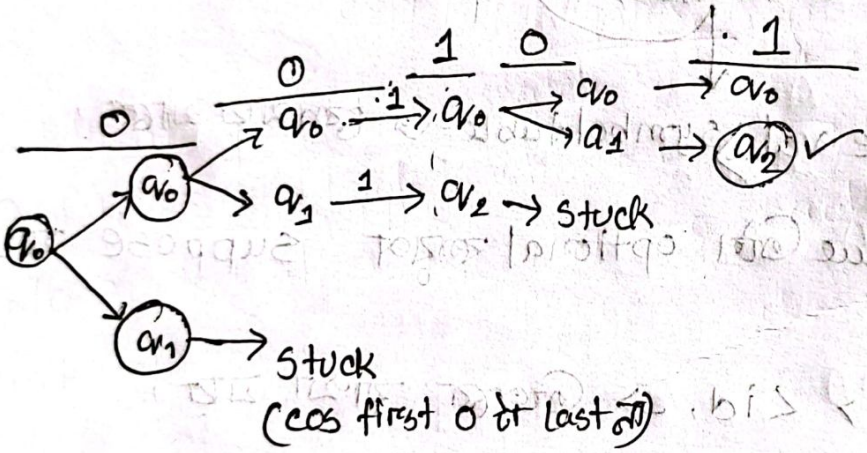
Last - ও (01) হলে হবে।







00101



$Q = \{q_0, q_1, q_2\}$

$\Sigma = \{0, 1\}$

$q_0 = q_0$

$F = \{q_2\}$

Transition Table

	0	1
$\rightarrow q_0$	$\{q_0 q_1\}$	$\{q_0\}$
q_1	$\emptyset$	$\{q_2\}$
q_2	$\emptyset$	$\emptyset$

Bye... Bye.... 2023

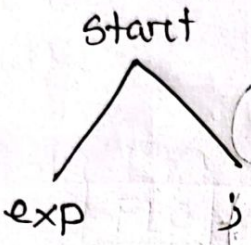
31. Dec. 23 ~~Friday~~ Sunday

FLC

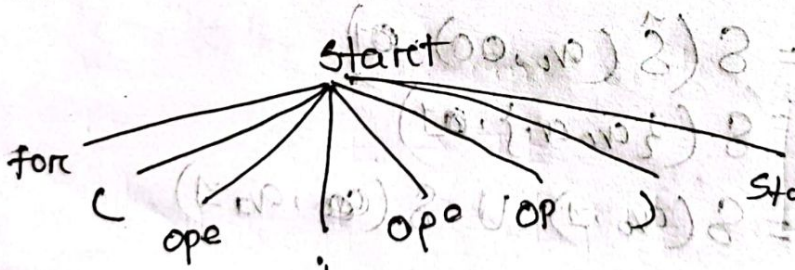
start → exp ;

for (; expr; exp) ... other.

↑
lookat symbol
root-ও আসবে



lookat symbol-ও match হয় কি না তা দেখবো।
match হয় না। backtrack করে পরেরটা নিয়ব। 2nd rules-ও lookat ও মিলল না। so backtrack & check 3rd Rule



for → পেয়ে নিয়েছি তাই lookat symbol update হয়ে (হবে। ope → ε | exp ε ছাড়া বাকি থাকলো আগে match করাবো then lookat exp. ε check করবো।

* Problem হচ্ছে বার বার backtrack করতে হয়।

→ To solve this ~~Predicted value~~ - Predictive Parsing.

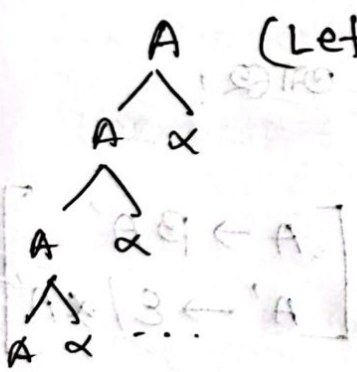
look at symbol → কোন starting symbol এর সাথে match হচ্ছে তখন (if/for/other/expr)

```
stmt → expr;
     | if(expr)stmt
     | for(ope ; ope ; ope)stmt
     | other
ope  → ∈
     | expr
```

এটা কাজ করবে না কারণ stmt এর মধ্যে যদি start symbol similar থাকবে তখন ambiguity তখাবে।
অর্থাৎ for (;;) এটাও যদি থাকতো তখন এটার ভেতরও কাজ করবে।

Recursive algo problem:

$A \rightarrow A\alpha / \beta$



ever ~~reading~~ চলমান process..

To solve:

$A \rightarrow A\alpha$ or $A \rightarrow \beta$ or $A \rightarrow \beta\alpha\alpha$

$\rightarrow \boxed{\beta\alpha}$ or $A \rightarrow A\alpha$

$\rightarrow A\alpha\alpha$

$\rightarrow \boxed{\beta\alpha\alpha}$

to summarize or Regular express $[\beta\alpha^*]$

So we can write new grammar;

$A \to \beta A'$

$A' \to \varepsilon / \alpha A'$

this grammar is exactly works like $A \to A\alpha / \beta$ so we can replace it by new one and then draw the Parse tree.

FLC Syntax 10 Jan 24 Wednessday.

Left Recursion Elimination:

$E \to E+T|T$

Same হচ্ছে Left Recur Problem আছে।

$E \approx A$
$\beta \approx T$
$\alpha = +T$

So we can write,

$E \to TE'$
$E' \to \varepsilon \mid +TE'$

$\left[\begin{array}{l} A \to \beta A' \\ A' \to \varepsilon / \alpha A' \end{array}\right]$

Book (4.3.4) Left Factoring:

$A \to \alpha\beta_1$
$\to \alpha\beta_2$

to solve Parsing Problem;

$A \to \alpha A'$
$A' \to \beta_1 | \beta_2$

[যদি Grammare এ common starting symbol থাকবে তখন এই rule use করতে হবে]

→ Book (chapter-2
chapter-4)
→ 4.3.4
→ 4.4.2

FIRST & FOLLOW

First (Gs) = { } Grammar symbol-র first set return করবে।
(Terminal/Non terminal)

Follow (Gs) = { } Follow set return করবে।
(non terminal) follow set-এ কখনো ε থাকবে না।

Why need this?

Non-recursive Predictive Parsing-র জন্য
(next class-এ পড়ব)

(*) E → TE'
E' → +TE'/ε
T → FT'
T' → *FT'/ε
F → (E) / id

Check/calculate first follow

First

আগে terminal গুলো বের করব।

$Fr(+) = \{+\}$

$Fr(*) = \{*\}$

$Fr(() = \{(\}$

$Fr()) = \{)\}$

$Fr(id) = \{id\}$

[যেকোন terminal এর first set ঐ terminal ই হয়]

$Fr(F) = \{(, id\}$

$Fr(T') = \{*, \varepsilon\}$ সোজাসুজি ε বসবে

$Fr(T) = \{(, id\}$

$Fr(E') = \{+, \varepsilon\}$

$Fr(E) = \{(, id\}$.

*Non terminal গুলোর জন্য Last থেকেই শুরু করব Grammar

Exception

$Fr(A) = \{\#, \varepsilon\}$

$Fr = \{*, \varepsilon\}$

$Fr(@) = \{+\}$

$F \rightarrow ABC$

$Fr(F) \rightarrow \{\#, \varepsilon, *, +\}$ এখানে $A \rightarrow \varepsilon$ আছে তাই A না বসে BC র কাজ হবে তাই C র কাজ করেছে। যতক্ষণ পর্যন্ত ε পাবো ততক্ষণ সব নিতে থাকব।

Follow

① $Follow(E) = \{\$,)\}$

$Follow(E') = \{\$,)\}$ [দুইটা case ই calculate করব]

$Follow(T) = \{+, \$,)\}$

$Follow(T') = \{+, \$,)\}$

$Follow(F) = \{*, +, \$,)\}$

②

* যেখানে হবে E production body র কোথায় কোথায় আছে।
* যেখানে E পাবো তারপরের symbol র first(Fr) নিতে হবে।
* যেহেতু E starting symbol তাই first-এ $ symbol add করব।
* E' -র পর কিছু না থাকে তাহলে সেই production body র Head র follow set

Sunday
Date: 14 Jan 24

LL(1) Grammar

L → Left to right traverse input

L → Left most derivation আছে হবে → then Non terminal কে বলবে.

1 → Lookhead symbol points to the same symbol.

* $A \to \alpha | \beta$

(*) যখনই α, β derivation করবে তখনই minimize টা পাবো যেটার beginning terminal symbol same হওয়া যাবে না.

$\alpha \to a \cdots$
$\beta \to a \cdots$ } both not accepted.

(*) $\alpha \to \varepsilon$ & $\beta \to \varepsilon$ (⊗) Ambiguity যেকোন একটাজনের derivation এ ε আসলে LL(1), দুইটা হলে not follow LL(1).

(*) $\alpha \to \varepsilon$ β যেমন তেমন string হবে না, যার beginning এ follow of A আসবে -

Xm এ এই properties গুলো আসবে।

Non-Recursive Predictive Parser

Parsing Table: (Previous Grammar)

M[E', id]
↓ ↓
row col

• $E' \to \varepsilon$ হলে E' এর follow table-এ বসাবো

* যেসব empty index indicate করবে এগুলো যেই index-এ দুটি rule থাকবে এরকম index গুলোতে error generate হবে।

→ output-এ production rule.

→ Algo কেমন করে input string derive করতে কি কি rules use হয়েছে সেটা output-এ আসবে।

$A \to aB$
$B \to \varepsilon | b$

input string: ab

① $A \to aB$
$B \to b$ } output.

Bottom Up Parsing

17 Jan 24
Wednesday

Recursive Decent Parsing
↳ Predictive Parsing
↳ Recursive
↳ Non Recursive

→ Top-down Parsing

(Example)

LL(1) Grammar Handle করতে পারবে।

(4.1) → math আছে বইয়ের

$E \to E+T \mid T$
$T \to T*F \mid F$
$F \to (E) \mid id$

id * id এই input টা Grammar follow করে কিনা?

Bottom-up strategy.

⇒

$id * id \to F * id \to T * id \to T * F \to T \to E$

$id * id$
$F * id$ (F → id)
$T * id$ (T → F, F → id)
$T * F$ (F → id)
T (T → T * F)
E (E → T)

product এর Body কে head দিয়ে replace করাই Reducing. product body র যতটুকু replace করি সেটা handle.

* There is multiple option for Reducing.

LL1 grammar → Predictive Parsing 24 Jan 24

LR parse | SLR parser

Augmented grammar.

closure

GOTO (I, +)

Buttom up parser → shift Reduce Parser.

→ SLR ← Parsing table নিয়ে কাজ করে

→ Augmented grammar

→ Canonical LR(0) Automaton.

→ Closour

→ Goto.

Action & Goto → Parsing table

Algorithm → X_m -এ আরম্ভ শেষ।

$sB \rightarrow r$

↓ ↓

shift reduce

* Transition আকারে → shift
* " না " → Reduce.
*

If grammar $\boxed{A \rightarrow \alpha \cdot}$ টেবিলে হবে

Follow (A) ঘর করতে হবে।

So for our grammar.

follow(E) = {$, +,)} $E \to T$.

main $E \to \overset{1}{E} + T \mid \overset{2}{T}$

$T \to \overset{3}{T} * F \mid \overset{4}{F}$

$F \to \overset{5}{(} E) \mid \overset{6}{id}$

So $E \to T$ এটা 2 no production. তাই reduce 2 লিখব

Parsing র কাজ কি?

String (input) → Parser → Accepted / Rejected.

Grammar তাকে create করি

Compiler-এ Parser র ভূমিকা কি?

Token stream → Parser of compiler → Accepted/Rejected.
Error detection

পুরো C program এরকম একটা Grammar থাকে।
যেখানে অসংখ্য production rule থাকে।

Three Address Code

* একটা instruction কে multiple ভাগে ভাগ করে নিতে হয়

$x = y \text{ op } z$ এখানে ৩টা memory address থাকবে
(x, y, z)
আর একটাই binary operator (+, -)
এরকম থাকবে।

So if $x+y*z$ -এটাকে 3 address code-এ লিখবেন,

$t_1 = y * z$

$t_2 = x + t_1$

$y = 2 * 3$ → constant can be used. it is valid.

$x = op\ y$
$x = y$
→ valid (atmost 3 variable হতে হবে)

relop → relational operator (==, <=, >=, !=)

Procedure & call (বাদ) Syllabus-এ নাই।

⇒ $x = y[i]$ $\quad *(y+i)$
↳ ঐ address-এর value x-এ রাখব।

y ← int
int 4byte.

y[i] means = $(y + 4*i)$ ← byte define করা থাকলে এগুলো multiply করে নিতে হবে।

Example

1. $x = a + b * c + d$ [multiplication আগে করব...

$t_1 = b * c$

$t_2 = a + t_1$

$t_3 = t_2 + d$

2. $-(a * b) + (c + d) - (a + b + c + d)$

$t_1 = a * b$

$t_2 =$ unary op t_1

$t_3 = c + d$

$t_4 = t_2 + t_3$

$t_5 = a + b$

$t_6 = t_5 + c$

$t_7 = t_6 + d$

$t_8 = t_4 - t_7$

3. if $a < b$ then $a = 3$ else $b = 2$

if $(a < b)$ → Fal

$b = 2$

goto L2

L1: $a = 3$

L2:

Xm-এ Triples .. representation আসবে।

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

Statement table → pointer এর মতো কাজ করবে।

15
14
16 এভাবে হবে। ভুল আছে।
:

From book,

do i = i+1.

$t_2 = i$

6.2 → Excercise solve করতে হবে।

6.2.5 → ↑

Quize → Non-recursive -এরপর থেকে 3 address পর্যন্ত।

LR(0) automaton

Grammar দেয়া থাকলে প্রথান থেকে LR(0) বের করতে হবে।

Target

~~(1) Lexical Analysis~~ Date:

2 Perse Tree to Annotated Parse Tree

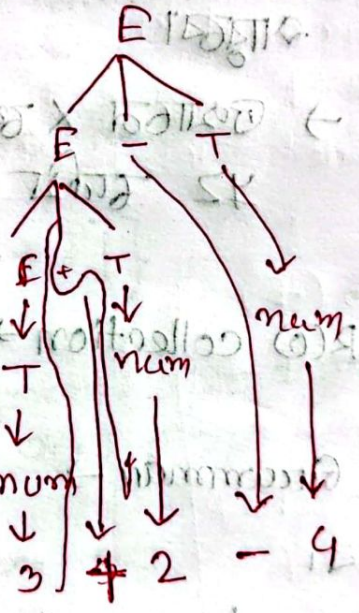
Syntax Directed Translation

CFG এর semantic rules = Syntax Directed Definition (SDD)

two Approaches

1. Top down
2. Bottom up

3+2-4



→ num-এর value TOP এসাইন

→ T এর

Quiz-4

Introduction to LR Parsing : Simple LR

LR parsers → k value 1.

LR parsers are table-driven,

Items → কোন একটা Grammar-এ dot বসিয়ে product body তাকে item পাবো।

A → .XYZ → এখানে কিছু পড়ি নাই XYZ তাকে মিলাতে পারবো

A → X.YZ → এখানে X তাকে কিছু terminal পেয়েছি YZ পড়ি নাই।

Canonical LR(0) collection → set of LR(0) item.

Augmented Grammar → start symbol কে আরও পরে দিয়ে লিখব।

$E' \to E$ } Augmented Grammar.

$E \to E+T \mid T$
$T \to T*F \mid F$
$F \to (E) \mid id$ } main Grammar

এবার যখন আমরা খুব ছোট্ট S কে reduce করতে S' → S যাওয়া যা mean করছে আমরা root-এ চলে গিয়েছি।

Closure:

$I = \{ E' \to .E \}$

Closure(I) = { $E' \to .E$ ↳ dot এর পর nonterminal থাকলে তার grammer add করতে হবে

$E \to .E + T$

$E \to .T$

$T \to .T * F$

$T \to .F$

$F \to .(E)$

$F \to .id$ ↳ terminal পাওয়া মাত্রই কাজ।

}

GoTo → kind of transition function.

$J = \{ [E' \to E.], [E \to E. + T] \}$

GoTo (I, +) ↳ grammar symbol. তখন খুঁজতে হবে + আগে কোন nonterminal আছে এমন Grammar আছে কিনা।

$= E \to E + .T$ [dot কে + এর পর নিয়ে যাবো]

তখন dot এর পর যা আছে তার closure নিতে হবে, যতক্ষণ পর্যন্ত terminal পাই।

So, $E \to E + .T$

$T \to .T * F$

$T \to .F$

$F \to .(E)$

$F \to .id$

$E \to E+T \mid T$
$T \to T*F \mid F$
$F \to (E) \mid id$

Line	STACK	Symbol	Input	ACTION
01	0	$	$id_1 * id_2 \$$	shift id to 5
2	05	$\$id_1$	$*id_2\$$	reduced by $F \to id$
3	03	$\$F$	$*id_2\$$	reduced by $T \to F$
4	02	$\$T$	$*id_2\$$	shift $* \to 7$
5	027	$\$T*$	$id_2\$$	shift $id_2 \to 5$
6	0275	$\$T*id_2$	$\$$	reduce by $F \to id$ এবারও I_7 থেকে F এর transition খুজবে
7	0270	$\$T*F$	$\$$	reduce by $T \to T*F$
8	02	$\$T$	$\$$	reduce by $E \to T$
9	01	$\$T$		

(1) I_0 থেকে দেখবো id এর জন্য transition আছে কিনা, I_5-এ আছে তাই stack state 05 হবে এবং symbol table-এ $\$id$ shift হবে।

(2) next input = *; এখন আছি 5state-এ কিন্তু এখান থেকে '*' কোন transition নেই। তাই * কে shift না করে id_1 কে reduce করব। এবং কে reduce করবে সেটা state 5 - ই বলে দেয়া আছে $F \to id$.

(3) so reduce means pop. pop করে $I_5 \to I_0$ তে আবার ফেরত যাবো কেন হাতে তখন F আছে। এখন $I_0 \to F$ এর transition জায়গায় খুঁজবো। I_3 তে so stack-এ 03 হবে, symbol-এ F হবে।

(4) এখন I_3 তে * এর transition নেই so আবার reduce করব $F \to T$ হয়ে I_0 তে ফেরত যাবো এবং I_0 থেকে T এর transition খুঁজবো I_2

Bottom-Up Parsing

A general style of bottom up parsing known as shift reduce parsing.

We can think of bottom up parsing as the proce of 'reducing' a string ω to the start symbol of the grammar. At each 'reduction step', a specific substring matching the body of a production is replaced by the non-terminal at the head of that production.

A reduction is the reverse of ~~of~~ a step in a derivation

Handle Pruning: is a substring that matches the boo of a production, and whose reduction presents on step along the reverse of a rightmost derivation

$F \rightarrow id$ → handle

Shift-Reduce Parsing: is a form of bottom-up parsing in which a stack holds grammar symbols and an input buffer holds the rest of the string to be parsed.

* the handle always appears at the top of the stack ju before it is identified as the handle.

Top down → stack का leftmost → और stack का Top.

Bottom-up → " Rightmost → " stack का Top → $a

top

4 Action
- Shift
- Reduce
- Error
- Accept

LR

The most prevalent type of bottom-up parser today is based on a concept called LR(k) parsing.

Item: An LR(0) 'item' (item for short) of a grammar G is a production of G with a dot at some position of the body.

$A \to XYZ$

$A \to \cdot XYZ$
$A \to X \cdot YZ$
$A \to XY \cdot Z$
$A \to XYZ \cdot$

} 4 item

$A \to \varepsilon$
$A \to \cdot$ (one item)

Canonical LR(0): One collection of sets of LR(0) items, called ↑ collection, provides the basis for constructing a deterministic finit automaton that is used to make parsing decisions. Such an automaton is called an LR(0) automaton.

Closure of Item sets

If I is a set of items for a grammar G, then closure of (I) is the set of items constructed from I by two rules:

1. Initially, add every item in I to closure (I)
2. If $A \to \alpha \cdot B\beta$ is an in closure (I) and $B \to \gamma$ is a production then add the item $B \to \cdot\gamma$ to closure(I), if it is not already there.

We divide all the sets of items of interest into two classes.

1. kernel items: the initial item $S' \to \cdot S$ and all items whose dots are not at the left end

2. Nonkernel items: All items with their dots at the left end. except for $S' \to \cdot S$

$S' \to S$

$E \to \cdot E * T$
$T \to \cdot F$

Function GOTO (I, X): I is a set of items and X is a grammar symbol. The goto function is used to define the transitions in the LR(0) automaton for a grammar

Left factoring

It is a grammar transformation that is useful for producing a grammar suitable for predictive, or top-down parsing.

stmt → if expr then stmt else stmt
| if expr then stmt.

অর্থাৎ উপরের মতো look at symbol হলো if তখন সিদ্ধান্ত নেয়া confusing so to resolve this use Left factor

$A \rightarrow \alpha\beta_1 | \alpha\beta_2$

Equivalent, we can write, $A \rightarrow \alpha A'$

$A' \rightarrow \beta_1 | \beta_2$

& thus we can eliminate left factoring.

$S \rightarrow iEtS | iEtSeS | a$

$E \rightarrow b$

we can write, $S \rightarrow iEtSS' | a$

$S' \rightarrow \varepsilon | eS.$

$E \rightarrow b$

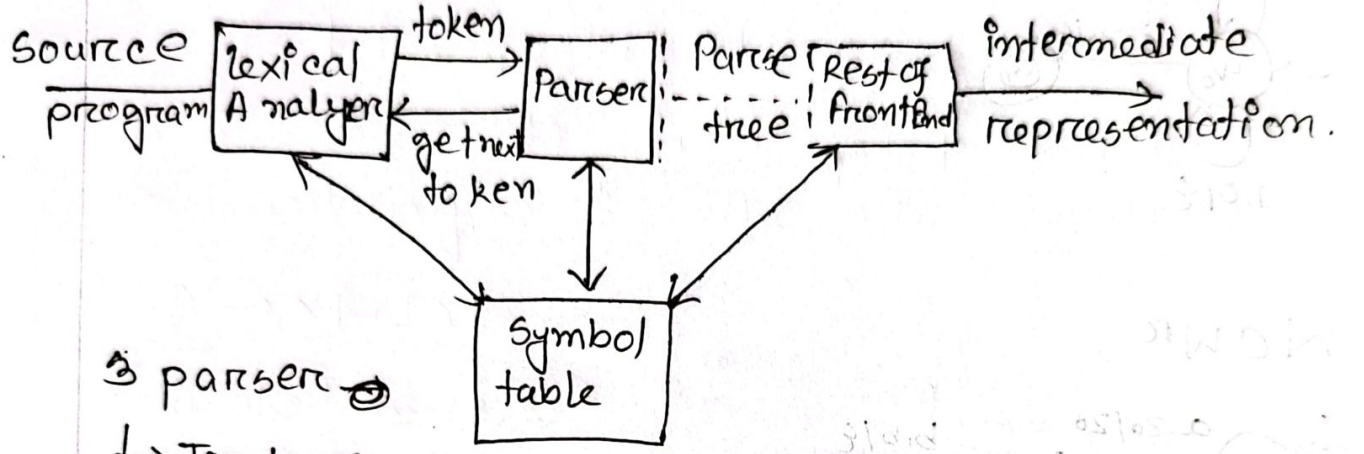
First Folloo 4.4.2

আমরা একবারে বা এক chance এ choose করতে চাই আমরা কোন production use করতে পারবো নাকি পারবো না। তাই First Folloo use করা হয়।

trailerror, Backtrack এসব কথা লাগছে না।

Parser

The parser obtains a string of tokens from the le analyzer and verifies that the string of token on can be generated by the grammar for the source la



3 parser

- Top down
- Universal → can parse any grammar.
- bottom up

(*) The input to the parser is scanned from left to ri

Top down Parsing

Recursive Descent parser is a kind of TDP

Trial and Error: that is we may have to try a producti and backtrack to try another production if first is found to be unsuitable. A production is unsuitable if, after using the production, we can not complete the tree to match the input string.

Look at symbol → input symbol currently processing.

do solve trial and Error use

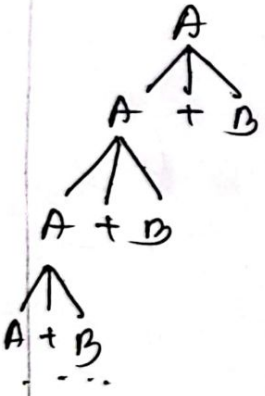
Predictive Parsing:

যতগুলো non-terminal ততগুলো ↑1 function লিখতে হবে, টেবিল ভালো বলা যায় না যখন দুটি production same terminal দিয়ে শুরু হয়।
এটা শুধু শুধু backtrack না করার জন্যই useful but same starting র জন্য কাজ করে না।

Left Recursion

(2.4.5 + 4.3.3) BOOK

① $A \to A + B$ অর্থাৎ যখন Grammar র head & production Body র first symbol same হয় তখন Left recursion problem হয়।



To solve,

$A \to A\alpha \mid B$

$A \to A\alpha$

$\to A\alpha\alpha$

$\to B\alpha\alpha$

$A \to BA'$

$A' \to \varepsilon \mid \alpha A'$

Ex: $E \to E+T \mid T$

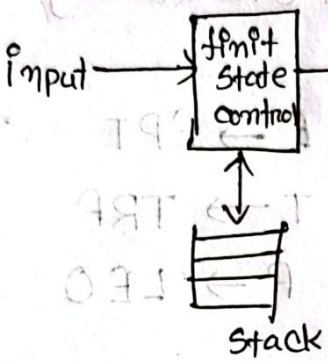
$E \to TE'$

$E \to \varepsilon \mid +TE'$

Push Docon Automata

ε-NFA + stack = PDA.

Two different version → final state -এ গ্রহণ
→ stack empty করে গ্রহণ



* PDA → input symol + current state + Symbol at the top
* PDA can make spontaneous transition অর্থাৎ ε মানে input ছাড়াও transition করতে পারে।

7-tuples

$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

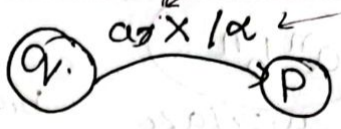
Γ (bin γ) → stack alphabet. $\Sigma \subset \Gamma$;

δ → triple (q, a, X)
- q ↓ current state
- a ↓ input
- X → stack top value.

$\gamma = X$ then the stack is unchanged.

output (p, γ)
- p ↓ next state
- γ ↳ is string of stack symbols that replaces X at the top of the stack.

$\delta(q, a, x)$ $\quad \delta(p, \alpha)$

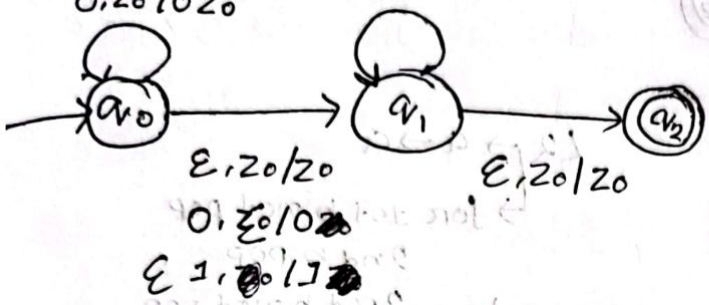


Example: 110011 $\quad \omega = 110$ $\quad \omega_R = 0111$

0,1/01
1,1/11
1,0/10
0,0/00
1,z_0/1z_0
0,z_0/0z_0

input, stack top

1, 1/ε → pop
0, 0/ε



many state

$(ID) \vdash^* (ID)$

$(q_1, a\omega, X\beta) \vdash (q_1, a\omega, X\beta)$

ID of a PDA:

q → is the state
ω → remaining input
γ → is the stack contents.

1111 → input

$(q_0, 1111, z_0)$

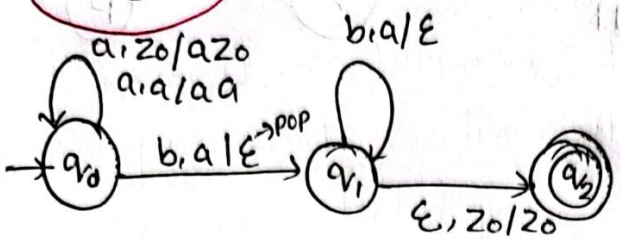
$(q_0, 111, 1z_0)$ $\quad (q_1, 111, 1z_0) \rightarrow (q_2, 111, z_0)$

$(q_0 1, 11, 11z_0)$ $\quad (q_1, 11, 11z_0) \rightarrow (q_1, 11, z_0) \rightarrow (q_2, 11, z_0)$

$(q_0 11, 1, 111z_0)$ $\quad (q_1 1, 1, 11z_0) \rightarrow (q_1 1, 1, 1z_0) \rightarrow (q_1 \varepsilon, z_0) \rightarrow (q_2, \varepsilon, z_0)$

$(q_0 111, \varepsilon, 1111z_0)$ $\quad (q_1, 1, 111z_0) \rightarrow (q_1, \varepsilon, 111z_0)$

$a^n b^n$



$a^n b^{2n}$ L1 → 1. a → 2a's
2. b → pop
3. Repeat.

L2 → a → a
↳ for 1st b not pop
2nd b pop
3rd b not pop
4th b pop.

(push 2a for 1a)

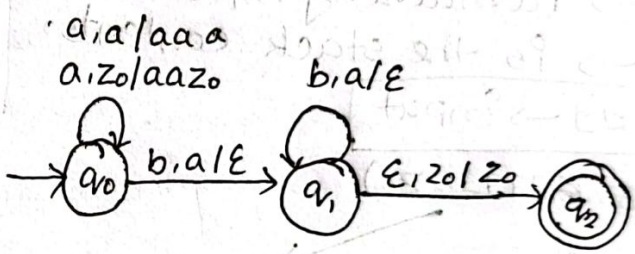
$\delta(q_0, a, z_0) = (q_0, aaz_0)$

$\delta(q_0, a, a) = (q_0, aaa)$

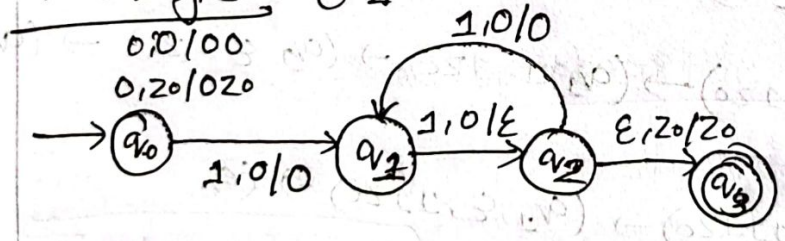
$\delta(q_1, b, a) = (q_1, \varepsilon)$

$\delta(q_1, b, a) = (q_1, \varepsilon)$

$\delta(q_1, \varepsilon, z_0) = (q_2, \varepsilon)$.



For Logic 2 $0^n 1^{2n}$



$\delta(q_0, 0, z_0) \to 0z_0$

$\delta(q_0, 0, 0) \to 00$

$\delta(q_0, 1, 0) \to q_1, 0$

$\delta(q_1, 1, 0) \to q_2, \varepsilon$

$\delta(q_2, 1, 0) \to q_1, 0$

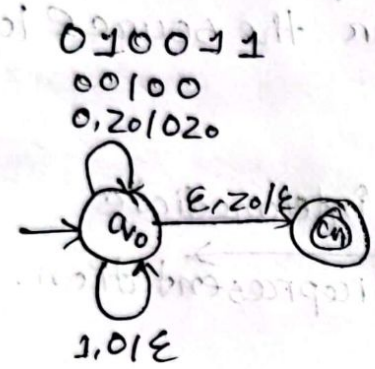
$\delta(q_2, \varepsilon, z_0) \to q_0$.

(*) $L = \{\omega \cdot \varepsilon (0,1)^* \mid n_0(\omega) = n_1(\omega)\}$

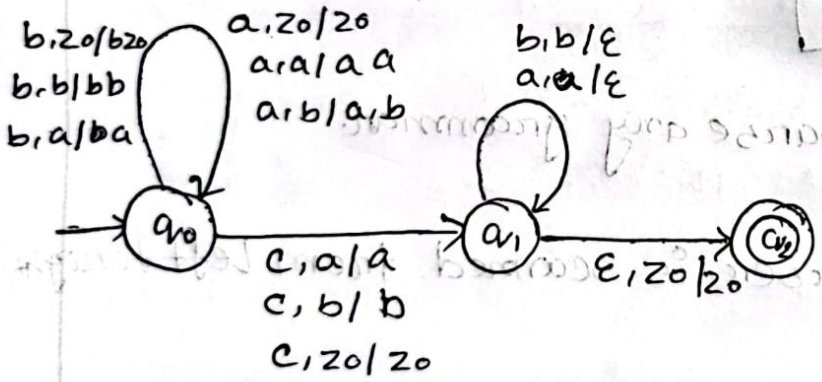
no of 0 = no of 1

01 ✓ 00 ✗ 0011 ✓

10 ✓ 11 ✗ 010011 ✓



WCW^R



Chomoskey Normal Form (CNF)

Conditions

1. $A \to BC$ A, B, C variable.
2. $A \to e$ e = terminal
3. length not exceed 2 [২ এর বেশি হবে না]

From slide,

$E \to E+T$ $\quad P \to +$ so $E \to EPT$

$T \to T*F$ $\quad R \to *$ so $T \to TRF$

$F \to (E)$ $\quad L \to ($ so $F \to LEO$

$O \to)$

$E \to EPT$ $\quad S \to PT \Rightarrow E \to ES$

$T \to TRF$ $\quad X_1 \to RF = T \to TX_1$

$F \to LEO$ $\quad X_2 \to LE = F \to X_2O$

	0	1
$\{q_0\}$	$\{q_1, q_2\}$	$\{q_3 q_5\}$
$\{q_1, q_7\}$	$\{q_6\}$	$\{q_2\}$
$\{q_3, q_5\}$	$\{q_2\}$	$\{q_6\}$
$\{q_4\}$	$\{q_7 q_1\}$	$\{q_3 q_5\}$
$\{q_6\}$	$\{q_6\}$	$\{q_4\}$
$\{q_2\}$	$\{q_0\}$	$\{q_2\}$

Simplification of CFGs.

⇒Explain the techniques to find the Generating & Reachable symbols of a CFG.

Lets, The Grammer is,

$S \to AB$
$A \to aA \mid B$
$B \to bB \mid a$
$C \to dC$

Here S → start symbol

A

$S \to aS \mid W \mid U$
$W \to aW$
$U \to a$
$V \to aa$

Here, a → Terminal so it is generating symbol.

U and V are also generating so S is generating

NOW W is not generating

W is however not generating.

Reachable Symbols:

* If A is reachable and we have a production A→α then all α symbols in α are Reachable.
* The start variable S is always Reachable.

S → aB | BC
A → aA | C | aDb
B → DB | C
C → b
D → B

Here, S is reachable.
So, a, B, C are reachable.
Then, b and D are reachable
However, A and C are not reachable.

Q: Eliminate Useless Symbol.

S → gAe | aYB | CY
B → dd | D
D → n
V → baXXX | oV
X → fV
A → bBY | ooC
C → jVB | gL
U → kW
W → c
Y → Yhm

Generating:

S → gAe
B → dd | D
D → n
C → gL
A → ooC
W → c
Y → Yhm

Reachable:

S → gAe
A → ooC
C → gL

Simplified Grammer is

S → gAe
A → ooC
C → gL

Eliminating Unit Productions:

$S \to cBh \mid D$ $\quad A \to aaC$

$B \to Sf \mid ggg$ $\quad C \to cA \mid d \mid C$

$D \to E \mid SABC$ $\quad E \to be$

(S,S) (B,B) (D,D) (A,A) (C,C) (E,E)

(S,S) → S→D unit P (S,D) — S→cBh|

(S,D) → D→E UP (S,E) — S→SABC

(S,E) → E→be UPX — S→be

(B,B) → X — B→Sf|ggg

(D,D) → D→E UP (D,E) — D→SABC

(D,E) → E→be UP X — D→be

(A,A) → X — A→aaC

(C,C) → X — C→cA|d

(E,E) → X — E→be

S = cBh | be | SABC

B = Sf | ggg

D → be | SABC

A → aaC

C → cA | d

E → be

Eliminate useless symbol:

$S \to abS \mid abA \mid abB$

$A \to cd$

$B \to aB$

$C \to dC$

G: $S \to abS \mid abA$

$A \to cd$

$C \to dC$

R: $S \to abS \mid abA$

$A \to cd$

Q: $S \to AB \mid AC$

$A \to aAb \mid bAa \mid a$

$B \to bbA \mid aaB \mid AB$

$C \to abCA \mid aDb$

$D \to bD \mid aaC$

G: $S \to AB$

$A \to aAb \mid bAa \mid a$

$B \to bbA \mid aaB \mid AB$

$D \to aC$

R: $S \to AB$

$A \to aAb \mid bAa \mid a$

$B \to bbA \mid aaB \mid AB$

Q: $A \to xyz \mid X_{yzz}$

$X \to X_z \mid xY_z$

$Y \to yY_y \mid X_z$

$Z \to Z_y \mid z$

G: $A \to xyz \mid X_{yzz}$

$Z \to Zy \mid z$

R: $A \to xyz$

$S \to a \mid bXY$

$A \to BAd \mid bSX \mid a$

$B \to aSB \mid bBX$

$X \to SBD \mid aBx \mid ad$

$Y \to SBa \mid XYb$

first-এ আমাদের দেখতে হবে কারা সরাসরি সবগুলো a Terminal symbol দিচ্ছে। এখানে A ও X সরাসরি terminal দিচ্ছে (a, ad) তাই B & Y বাদ হয়ে যাবে অর্থাৎ non generating.

G:

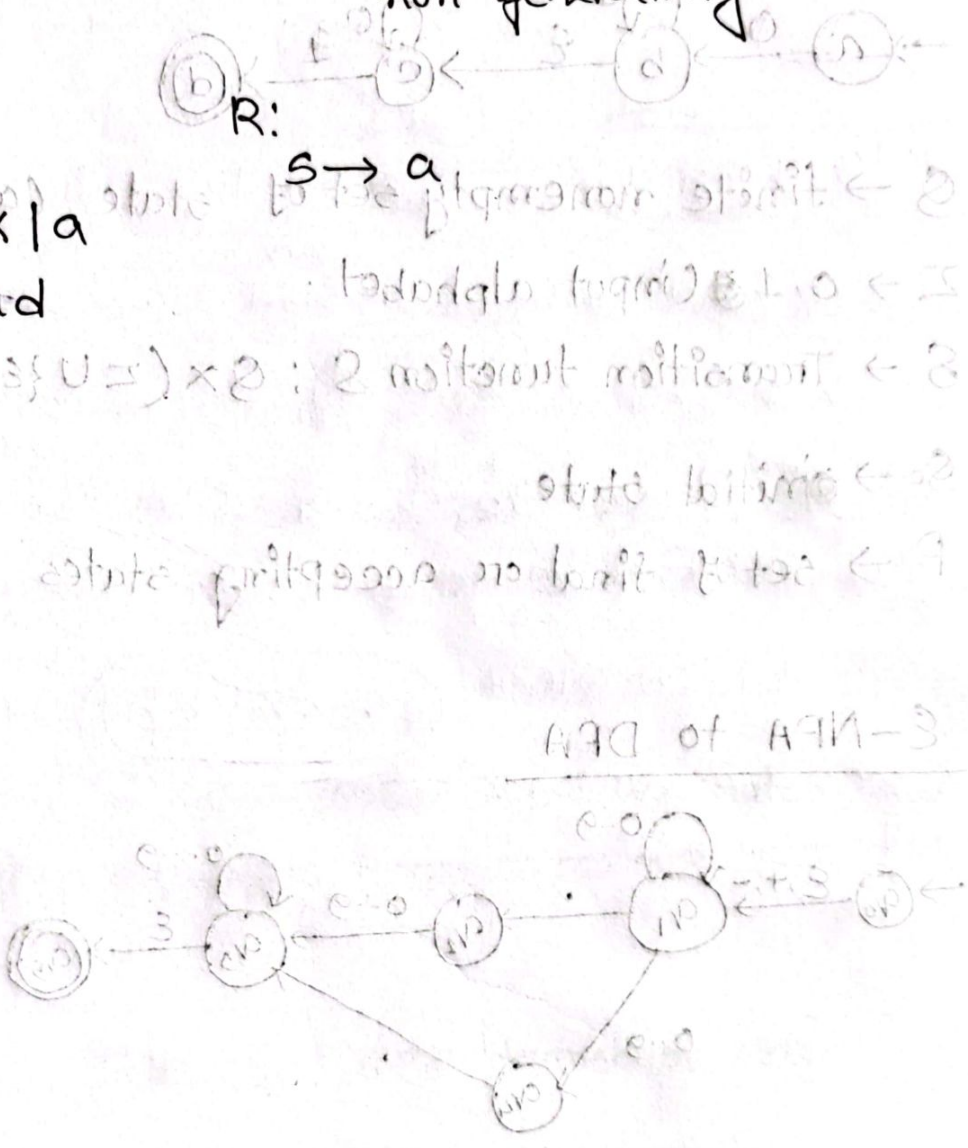
$S \to a$

$A \to bSX \mid a$

$X \to ad$

R:

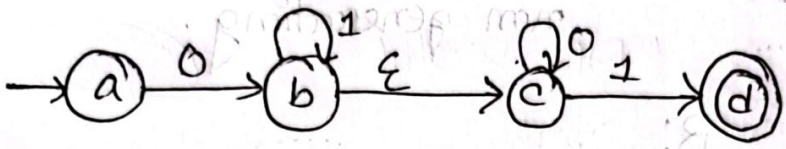
$S \to a$



ε-NFA

Another name is Extension of Finite Automaton.

ε-NFA mean যেকোন transition-এ কোন input ছাড়াই একটা state থেকে অন্য state-এ যাওয়া যাবে।



Q → finite nonempty set of state (a, b, c, d)

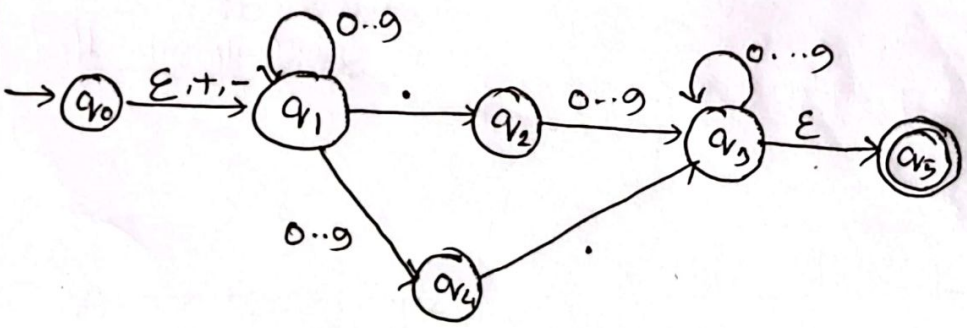
Σ → 0, 1 (input alphabet.

δ → Transition function $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$

q_0 → initial state

F → set of final or accepting states $F \subseteq Q$.

ε-NFA to DFA

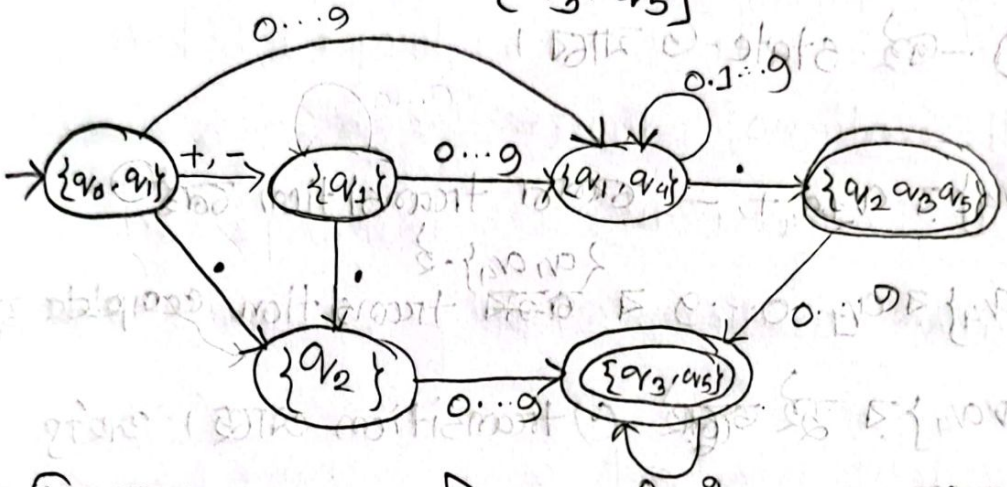


Equivalent DFA:

$\Sigma = \{+, -, ., 0 \cdots 9\}$

এখানে $\text{Eclose}(q_0) = \{q_0, q_1\}$ start নোড থেকে যাওয়া জন্য

$\text{Eclose}(q_3) = \{q_3, q_5\}$



১. প্রথমেন ε closer নিয়ে q_0 থেকে q_1 যাই তাই starting node হবে $\{q_0 q_1\}$. $\{q_0\}$-র epsilon clouser নিবো

২. $\Sigma = \{+, -, ., 0 \cdots 9\}$

৩. এখন q_0 থেকে +, - গেলে যাবো q_1 state. আরো q_1 stateর জন্য $\text{Eclose}(q_1) = \{q_1\}$ বের করতে হবে। যেহেতু একই তাই পাচ্ছি $\{q_1\}$.

④ $\{q_0 q_1\}$-র q_1 থেকে (+, -) কোন transition নেই।

⑤ $\{q_0 q_1\}$ র q_0 থেকে $0 \cdots 9$ র transition নেই

⑥ " র q_1 " $0 \cdots 9$ র " q_1 state এ + q_4 দুইটাতেই

তাই $\{q_1, q_4\}$ আরেকটা state.

⑦ $\{q_0 q_1\}$ র q_0 থেকে . কোন transition নেই

⑧ " র q_1 " . q_2 state-এ যায়।

এখন,

১. $\{q_2\}$ র (+, - ·) র কোনো transition নেই।

২০. $\{q_2\}$ র 0···9 র জন্যে q_3 state-এ যায় কিন্তু এর ε আছে তাই Eclose $(q_3) = \{q_3, q_5\}$। অতএব $\{q_2\}$ 0···9 র জন্যে $\{q_3, q_5\}$ -এই state-এ যাবে।

২১. $\{q_1, q_4\}$ র q_1, q_4 +, - কোনো transition নেই

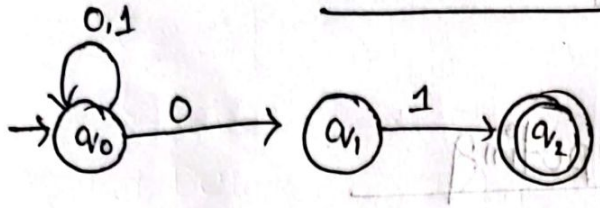
২২. $\{q_1, q_4\}$ র q_1 0···9 র $\{q_1, q_4\}$-ই transition complete হবে।

২৩. $\{q_1, q_4\}$ র দুই জায়গাই (·) transition আছে। অর্থাৎ q_1 (·) পেলে → q_2 তে যায় আবার q_4 (·) → q_3 যায় আবার ε transition আছে। অতএব $\{q_1, q_4\}$ (·) পেলে $\{q_2, q_3, q_5\}$ state-এ যাবে।

full diagram,

$\{q_0\ q_1\}$

Equivalent of NFA to DFA



Alphabet, $\Sigma = \{0,1\}$

Start state, $Q_0 = \{q_{v0}\}$

All possible state = $\emptyset$ $\{q_{v0}\}$ $\{q_{v1}\}$ $\{q_{v2}\}$ $\{q_{v0}q_{v1}\}$ $\{q_{v0}q_{v2}\}$ $\{q_{v1}q_{v2}\}$ $\{q_{v0}q_{v1}q_{v2}\}$

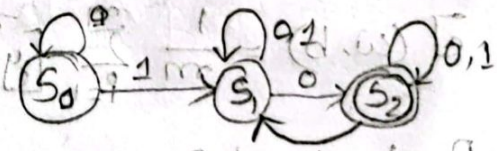
Probable final state: $\{q_{v2}\}$, $\{q_{v0}, q_{v2}\}$, $\{q_{v1}q_{v2}\}$, $\{q_{v0}, q_{v1}, q_{v2}\}$.

Transition Table:

From Lecture

Quiz

	0	1
$\rightarrow S_0$	$\{S_0\}$	$\{S_1\}$
S_1	$\{S_1, S_2\}$	$\{S_1\}$
* S_2	$\{S_2\}$	$\{S_1, S_2\}$



* $\{S_0, S_1\}$	$\{S_0, S_1, S_2\}$	$\{S_1\}$
* $S_0 S_2$	$\{S_0 S_2\}$	$\{S_1, S_2\}$
* $S_1 S_2$	$\{S_1 S_2\}$	$\{S_1, S_2\}$
* $S_0 S_1 S_2$	$\{S_0 S_1 S_2\}$	$\{S_1 S_2\}$