Theory of Computation (CS F351)

BITS Pilani
Hyderabad Campus

Prof.R.Gururaj
CS&IS Dept.

PDA, CGF and Parsing

(Chapter-3; Sec-3.4 – 3.7)

PDA and CFG

The class of languages accepted by PDA is exactly the class of Context –Free Languages.
Each CFL is accepted by some PDA.
For every grammar we must be able to construct a PDA.
The intersection of CFL with RL is a CFL.
The CFLs are closed under Union, Concatenation and Kleeene star.

The CFLs are not closed under Intersection and Complementation.

Rules for constructing PDA from CFG

If G= (V, ∑, R, S)

We must be able to construct M such that L(M)=L(G)

M= ({p,q}, ∑ , V, Δ , p, {q})

Where-

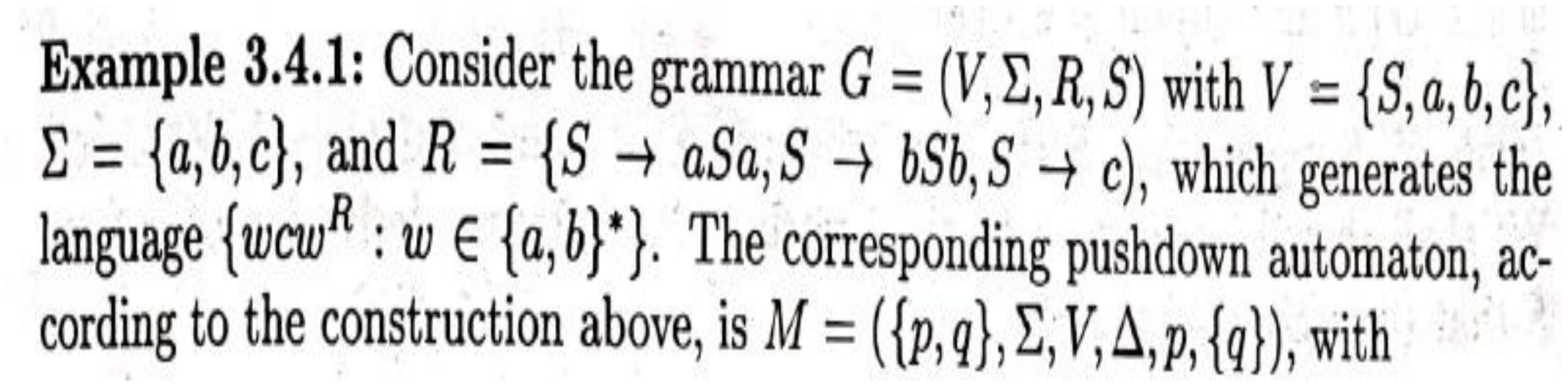
Δ = (1) ((p,e,e), (q,S))

(2) ((q,e,A), (q, x))

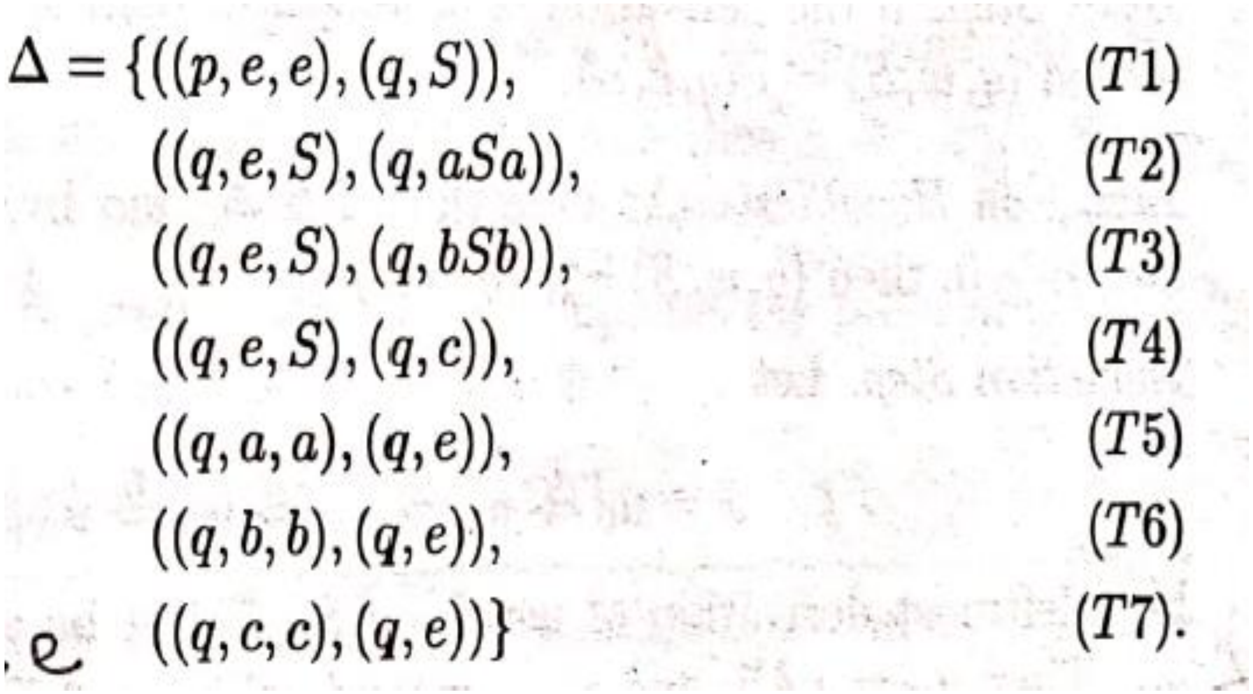
for each rule A→X in R

(3) ((q,a,a), (q,e))

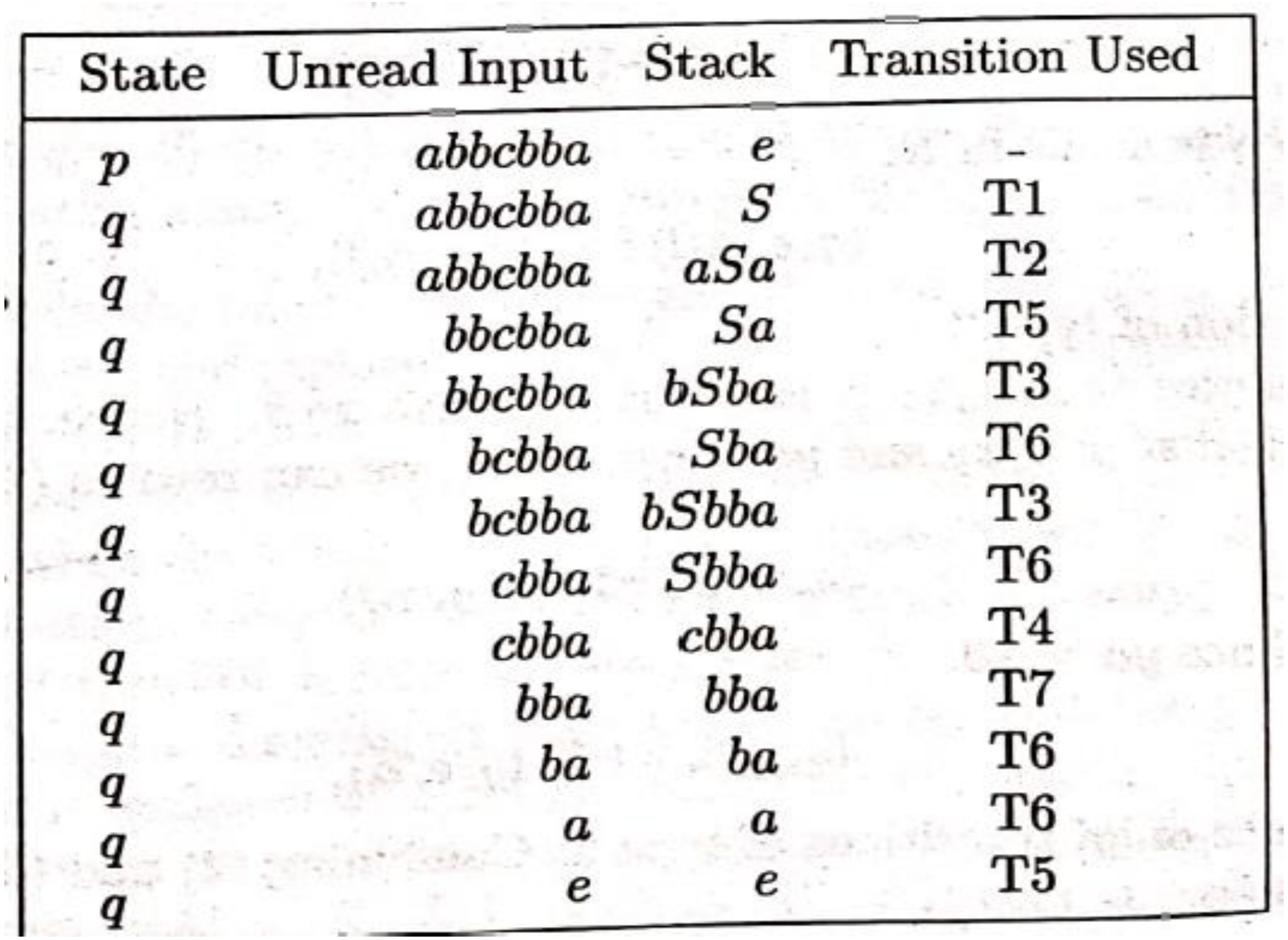
for each a∈∑



Example 3.4.1: Consider the grammar $G = (V, \Sigma, R, S)$ with $V = \{S, a, b, c\}$, $\Sigma = \{a, b, c\}$, and $R = \{S \to aSa, S \to bSb, S \to c)$, which generates the language $\{wcw^R : w \in \{a, b\}^*\}$. The corresponding pushdown automaton, according to the construction above, is $M = (\{p, q\}, \Sigma, V, \Delta, p, \{q\})$, with



$$
\begin{aligned}
\Delta = \{&((p,e,e),(q,S)), && (T1)\\
&((q,e,S),(q,aSa)), && (T2)\\
&((q,e,S),(q,bSb)), && (T3)\\
&((q,e,S),(q,c)), && (T4)\\
&((q,a,a),(q,e)), && (T5)\\
&((q,b,b),(q,e)), && (T6)\\
, e \quad &((q,c,c),(q,e))\} && (T7).
\end{aligned}
$$



State	Unread Input	Stack	Transition Used
p	$abbcbba$	e	–
q	$abbcbba$	S	T1
q	$abbcbba$	aSa	T2
q	$bbcbba$	Sa	T5
q	$bbcbba$	$bSba$	T3
q	$bcbba$	Sba	T6
q	$bcbba$	$bSbba$	T3
q	$cbba$	$Sbba$	T6
q	$cbba$	$cbba$	T4
q	bba	bba	T7
q	ba	ba	T6
q	a	a	T6
q	e	e	T5

The class of deterministic CFLs is closed under complement.
The class of deterministic CFLs are properly contained in the class of CFLs.

Determinism and Parsing (Section 3.7)

CFGs are extensively used in modeling the syntax of programming languages.

A compiler for such a programming language must then embody a parser that is an algorithm to determine whether a given string is in the language generated by a given CFG, if so, to construct a parse tree of the string.

The compiler then translate this parse tree into a program in a more basic language , such as assembly language

Many approaches to the parsing problem have been developed by compiler designers over the past four decades.
The most successful ones among them are rooted in the idea of a PDA.
The equivalence of CFG and PDA should be put to work.
A PDA is not of immediate practical use because it is a nondeterministic device.
The question is can we always make PDA to work deterministically, as we were able to do in the case of FA.

We shall see that there are some CFLs that cannot be accepted by DPDA.
This is bit disappointing to us.
Nevertheless it turns out that for most programming languages, one can construct DPDA.
Here we look at some heuristic rules for constructing DPDA.

Deterministic PDAs

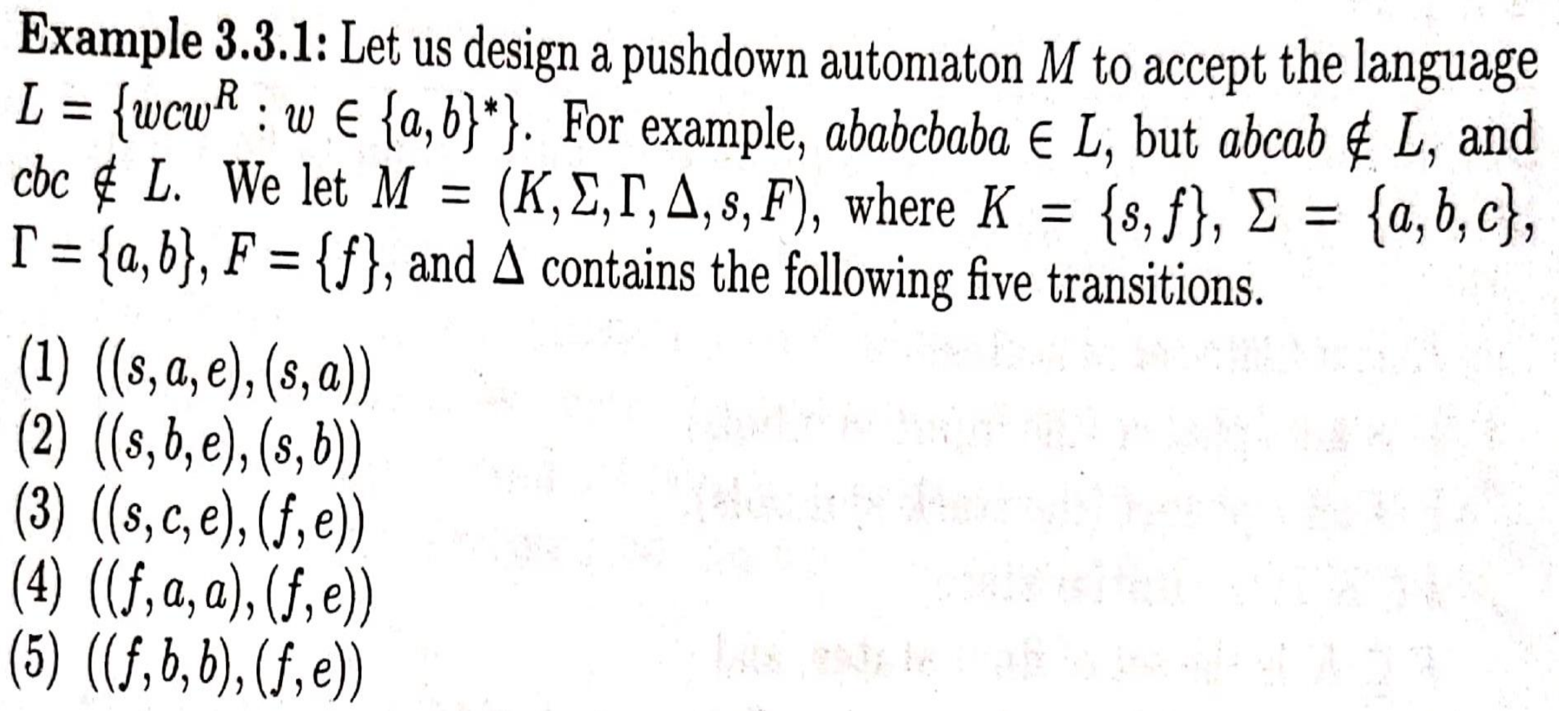
A PDA M is deterministic if for each configuration there is at most one configuration succeeding it in a computation.

We call two strings consistent if the first is the prefix of the second or vice versa.

We call two transitions

$((p, a, \beta), (q, \gamma))$ and $((p, a', \beta'), (q', \gamma'))$ compatible if a and a' are consistent and β and β' are consistent, in other words if there is a situation in which both transitions are applicable.

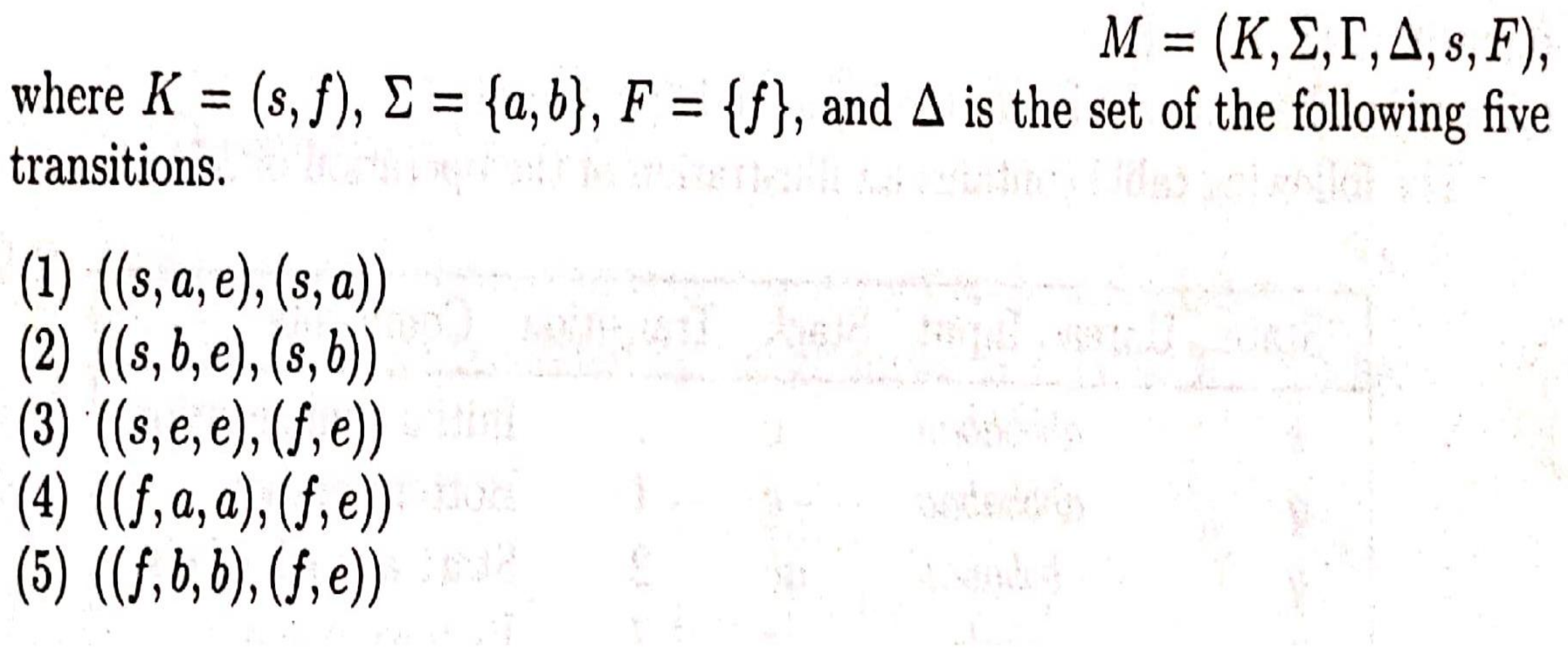
Then M is deterministic if it has no two distinct compatible transitions. Ex.



Example 3.3.1: Let us design a pushdown automaton M to accept the language $L = \{wcw^R : w \in \{a,b\}^*\}$. For example, $ababcbaba \in L$, but $abcab \notin L$, and $cbc \notin L$. We let $M = (K, \Sigma, \Gamma, \Delta, s, F)$, where $K = \{s, f\}$, $\Sigma = \{a, b, c\}$, $\Gamma = \{a, b\}$, $F = \{f\}$, and Δ contains the following five transitions.

(1) $((s,a,e),(s,a))$
(2) $((s,b,e),(s,b))$
(3) $((s,c,e),(f,e))$
(4) $((f,a,a),(f,e))$
(5) $((f,b,b),(f,e))$

Is deterministic, sine it doesn't have compatible transitions.



$$M = (K, \Sigma, \Gamma, \Delta, s, F),$$

where $K = (s, f)$, $\Sigma = \{a, b\}$, $F = \{f\}$, and Δ is the set of the following five transitions.

(1) $((s, a, e), (s, a))$
(2) $((s, b, e), (s, b))$
(3) $((s, e, e), (f, e))$
(4) $((f, a, a), (f, e))$
(5) $((f, b, b), (f, e))$

Is non-deterministic, sine (1) and (3) are compatible.
Similarly- (2) and (3) are also compatible.

Deterministic CFLs

Deterministic CFLs are essentially those that are accepted by deterministic PDA (DPDA).

A language is said to be deterministic CFL if it is recognized by a DPDA and that also has a capability to of sensing the end of the input string.

But for DPDA, we have to modify he acceptance criteria slightly.

Formally, we call a language L subset of $\sum^*$ deterministic Context-Free if $L\$=L(M)$ for some deterministic PDA M.

Here $\$$ is a new symbol not in $\sum$, which is appended to the input string for the purpose of marking its end.

Every deterministic CFL is a CFL.

If we do not adopt this new convention then many CFLs which are actually deterministic turn out to be non-deterministic. Ex. $a^* \cup (a^n b^n \mid n \geq 1)$

Rules for constructing PDA from CFG

$\Delta =$ (1) ((p,e,e), (q,S))

(2) ((q,e,A), (q, x))

for each rule A→X in R

(3) ((q,a,a), (q,e))

for each a€∑

Top-down Parsing

For the Language L= $a^n b^n$

$M_1 = (\{p, q\}, \{a,b\}, \{a,b,S\}, \Delta_1, R, \{q\})$

$S \rightarrow aSb; \; S \rightarrow e$

$\Delta_1 = \{$ ((p,e,e), (q,S)),

((q,e,S), (q,aSb)),

((q,e,S), (q,e)),

((q,a,a), (q,e)),

((q,b,b), (q,e)) $\}$

This is non-deterministic because

2 and 3 are compatible.

For the Language L= $a^n b^n$

Δ = { ((p,e,e), (q,S)),
((q,a,e), (q_a, e)),
((q_a,e,a), (q,e)),
((q,b,e), (q_b, e)),
((q_b,e,b), (q,e)),
((q,\$,e), ($q_\$$, e)),
((q_a,e,S), (q_a,aSb)),
((q_b,e,S), (q_b,e)) }

This is deterministic.

This is achieved by *lookahead*. by consuming an input symbol ahead of time and incorporating that information into its state.

Ex :Construct Look-ahead Deterministic PDA for the following Grammar

Grammar G:

S→ ABa
A→ aa|e
B→bb

Grammar G:

S→ ABa
A→ aa|e
B→bb

Deterministic Look-ahead transitions:

1. ((p ,e ,e), (q, S))
2. ((q ,a, e), (q_a, e))
3. ((q ,b, e), (q_b, e))
4. ((q_a ,e, A), (q, aa))
5. ((q_a ,e, S), (q, ABa))
6. ((q_b ,e, S), (q, ABa))
7. ((q_b ,e, B), (q, bb))
8. ((q_b ,e, A), (q, e))
9. ((q ,\$, e), ($q_\$$, e))

Grammar G: S→ ABa; A→ aa | e; B→bb

Deterministic Look-ahead transitions:

1. ((p ,e ,e), (q, S))
2. ((q ,a, e), (q_a, e))
3. ((q ,b, e), (q_b, e))
4. ((q_a ,e, A), (q, aa)) // since a is in FIRST of A
5. ((q_a ,e, S), (q, ABa)) // since a is in FRIST of S
6. ((q_b ,e, S), (q, ABa)) // since b is in FRIST of S
7. ((q_b ,e, B), (q, bb)) // since b is in FRIST of B
8. ((q_b ,e, A), (q, e)) // since b is in FOLLOWS of A
9. ((q ,\$, e), ($q_\$$, e))

FIRST (S)= FIRST(A) U FIRST(B) = {a,b}
FIRST (A)= {a} U {e}= {a,e}
FIRST (B)= {b}
FOLLOWS (S)= {\$}
FOLLOWS (A)= FIRST (B)= {b}
FOLLOWS (B)= {a}

E→T

E→E+T

T→T*F

T→F

F→(E)

F→id

F→id(E)

Construct a NDPDA using standard rules.

Δ = (1) ((p,e,e), (q,S))

(2) ((q,e,A), (q, x))

for each rule A→x in R

(3) ((q,a,a), (q,e))

for each a∊∑

Δ = { ((p,e,e), (q,E)),
((q,e,E), (q, E+T)),
((q,e,E), (q,T)),
((q,e,T), (q, T*F)),
((q,e,T), (q,F)),
((q,e,F), (q, (E))),
((q,e,F), (q,id)),
((q,e,F), (q,id(E)))

and finally ((q,a,a), (q,e)) for all a$\epsilon\sum$
}

$F \rightarrow id$

$F \rightarrow id(E)$ creates problems. Lookahead may not work here.

Whenever we have production of the form-

$A \rightarrow \alpha\beta_{1,}\ A \rightarrow \alpha\beta_{2,}\ A \rightarrow \alpha\beta_{n,}$

Replace them by

$A \rightarrow \alpha A'$

$A' \rightarrow \beta_1$

$A' \rightarrow \beta_2$ and $A' \rightarrow \beta_n$

F→id
F→id(E)

Replace the above with

F→idA
A→ e
A→ (E)

Now we can apply lookahead without any problem.

Δ = { ((p,e,e), (q,E)),

((q,e,E), (q, E+T)),

((q,e,E), (q,T)),

((q,e,T), (q, T*F)),

((q,e,T), (q,F)),

((q,e,F), (q, (E))),

$$\left.\begin{array}{l}((q,e,F),\ (q,idA)),\\ ((q,e,A),\ (q,e))\\ ((q,e,A),\ (q,(E)))\end{array}\right\}\quad \begin{array}{l}((q,e,F),\ (q,id)),\\ ((q,e,F),\ (q,id(E)))\end{array}$$

and finally ((q,a,a), (q,e)) for all a$\epsilon\sum$

}

E→E+T

E→T

T→T*F

T→F

F→(E)

F→idA

A→ e

A→ (E)

$E \rightarrow E+T$

Creates problems. This is known as *Left Recursion*.

Whenever we have production of the form-

$A \rightarrow A\alpha_{1,}\ A \rightarrow A\alpha_{2,}\ A \rightarrow A\alpha_{n,}$ and

$A \rightarrow \beta_{1,}\ A \rightarrow \beta_2$ and $A \rightarrow \beta_m$

Replace these by

$A \rightarrow \beta_1 A', A \rightarrow \beta_2 A', \ldots, A \rightarrow \beta_m A',$ and

$A' \rightarrow \alpha_1 A', A' \rightarrow \alpha_2 A', \ldots, A' \rightarrow \alpha_n A',$ and

$A' \rightarrow e$

$E \rightarrow E+T$

$E \rightarrow T$

Can be replaced by

$E \rightarrow TE'$

$E' \rightarrow +TE'$

$E' \rightarrow e$

$T \rightarrow T*F$

$T \rightarrow F$

Can be replaced by

$T \rightarrow FT'$

$T' \rightarrow *FT'$

$T' \rightarrow e$

E→T

E→E+T

T→T*F

T→F

F→(E)

F→id

F→id(E)

E→ TE'

E'→ +TE'

E'→e

T→ FT'

T'→ *FT'

T'→e

F→(E)

F→idA

A→ e

A→ (E)

$\Delta = \{$ $((p,e,e), (q,E)),$ (1)

$((q,a,e), (q_a, e)),$ (2) for each a € $\sum$ U $\{\$\}$

$((q_a ,e,a), (q,e)),$ (3) for each a € $\sum$

$((q_a,e,E), (q_a , TE')),$ (4) for each a € $\sum$ U $\{\$\}$

$((q_+,e,E'), (q_+,+TE')),$ (5)

$((q_a,e,E'), (q_a, e)),$ (6) for each a € $\{), \$\}$

$((q_a,e,T), (q_a, FT')),$ (7) for each a € $\sum$ U $\{\$\}$

$((q_*,e,T'), (q_*, FT')),$ (8)

$((q_a,e,T'), (q_a, e)),$ (9) for each a € $\{+,), \$\}$

$((q_(,e,F), (q_(, (E))),$ (10)

$((q_{id},e,F), (q_{id},idA)),$ (11)

$((q_(,e,A), (q_(,(E)))$ (12)

$((q_a,e,A), (q_a, e)),$ (13) for each a € $\{+, *,), \$\}$

$\}$

The devices like ones we have seen in the previous example Which correctly decide whether a string belongs in a CFL and in the case of positive answer, produce the corresponding *parse tree*, are called as *parsers*

The PDA we have seen is a *top-down parser* because tracing its operation at the steps where non-terminals are replaced on the stack, constructs a parse-tree in top-down order or left-to-right order.

Naturally not all CFLs have deterministic acceptors that can be derived from the standard nondeterministic one via the lookahead idea.

For certain deterministic CFLs, lookahead of just one symbol may not be sufficient to resolve all the uncertainties.

Some languages are not directly amenable to parsing by lookahead.

Top-Down Parsing

The parser discussed so far is a top-down parser.

Because during its operations it constructs the parse tree in top-down and left-to-right fashion.

We start with the starting NT on the stack.

Every time we replace the leftmost NT in the intermediate string.

Bottom-up Parsing

G

$S \rightarrow bBA$

$A \rightarrow ab$

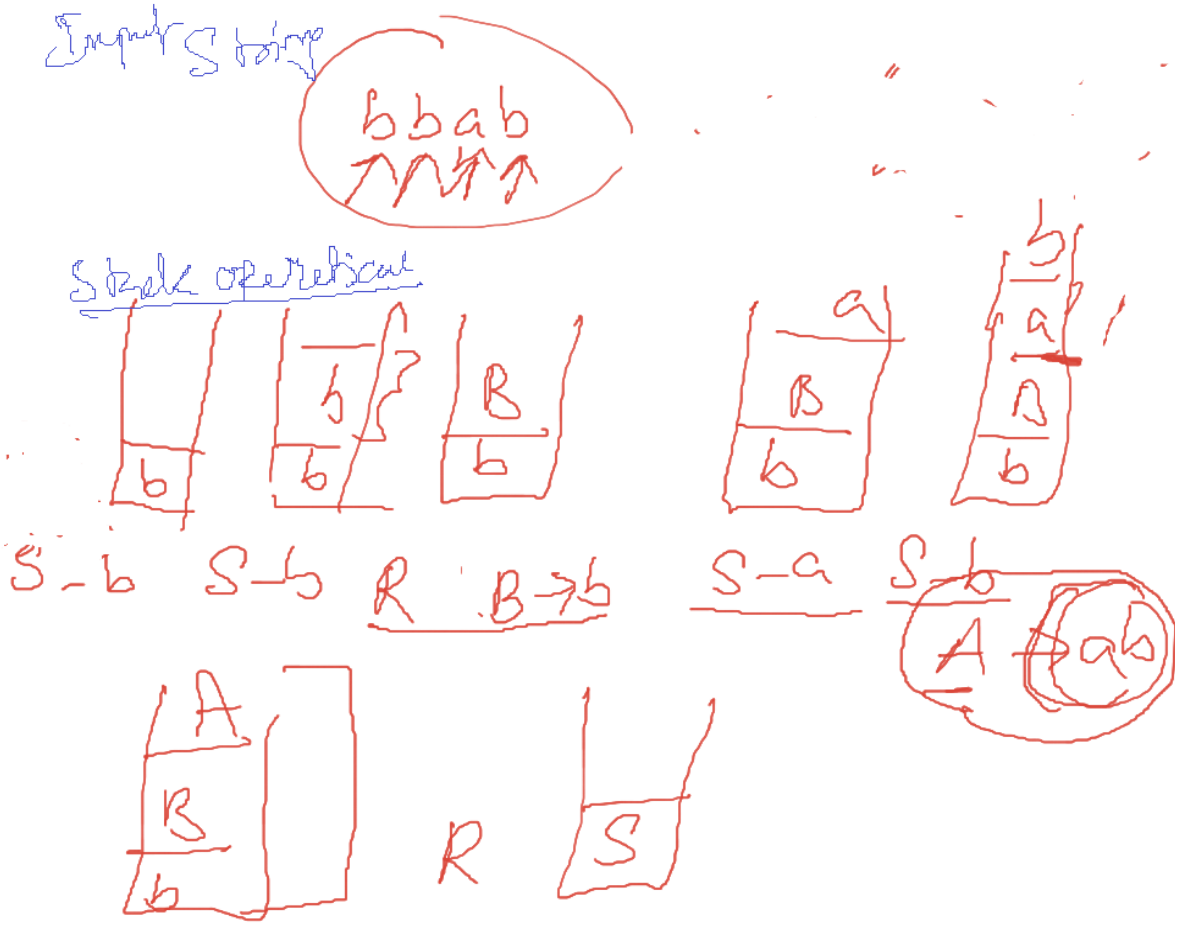
$B \rightarrow b$

$S \rightarrow bBA$

$\rightarrow bBab$

$\rightarrow bbab$

R



G

1. S → bBA
2. A → ab
3. B → b.

State	input	Stack	Rule
p	bbab	e	
p	bab	b	
p	ab	bb	
p	ab	Bb	B→b
p	b	aBb	
p	e	baBb	
p	e	ABb	A→ab
p	e	S	S→ bBA

1. ((p, a, e) (p, a))
2. ((p, b, e) (p, b))
3. ((p, e, ABb), (p, S))
4. ((p, e, ba), (p, A))
5. ((p, e, b), (p, B))
6. ((p, e, S), (q, e))

Bottom-up Parsing

$\Delta =$ (1) $((p,a,e), (p,a))$

for each $a \epsilon \sum$

(2) $((p,e,\alpha), (p, A))$

for each rule $A \rightarrow \alpha$ in R

(3) $((p,e,S), (q,e))$

Bottom-up Parsing

$\Delta =$ (1) ((p,a,e), (p,a))

for each a$\epsilon\sum$

(2) ((p,e,α^R), (p, A))

for each rule A$\rightarrow\alpha$ in R

(3) ((p,e,S), (q,e))

Transitions of type 1 move input symbols onto the stack. (Shift)

Transitions of type 3 pop terminal symbols off the stack when they match input symbols.

Transitions of type 2 replace the right-hand side of a rule on the stack by the corresponding left-hand side .(Reduce)
Transitions of type 3 here end a computation by moving to final state when only the start symbol (S) remains on the stack.

E→E+T

E→T

T→T*F

T→F

F→(E)

F→id

(p,a,e),(p,a)	0	for each a∈∑
(p,e,T+E), (p,E)	1	
(p,e,T), (p,E)	2	
(p,e,F*T), (p,T)	3	
(p,e,F), (p,T)	4	
(p,e,)E(), (p,F)	5	
(p,e,id), (p,F)	6	
(p,e,E), (q,e)	7	

Shift reduce conflict: One possible way to deciding whether to shift or reduce is –

If *a* is stack symbol

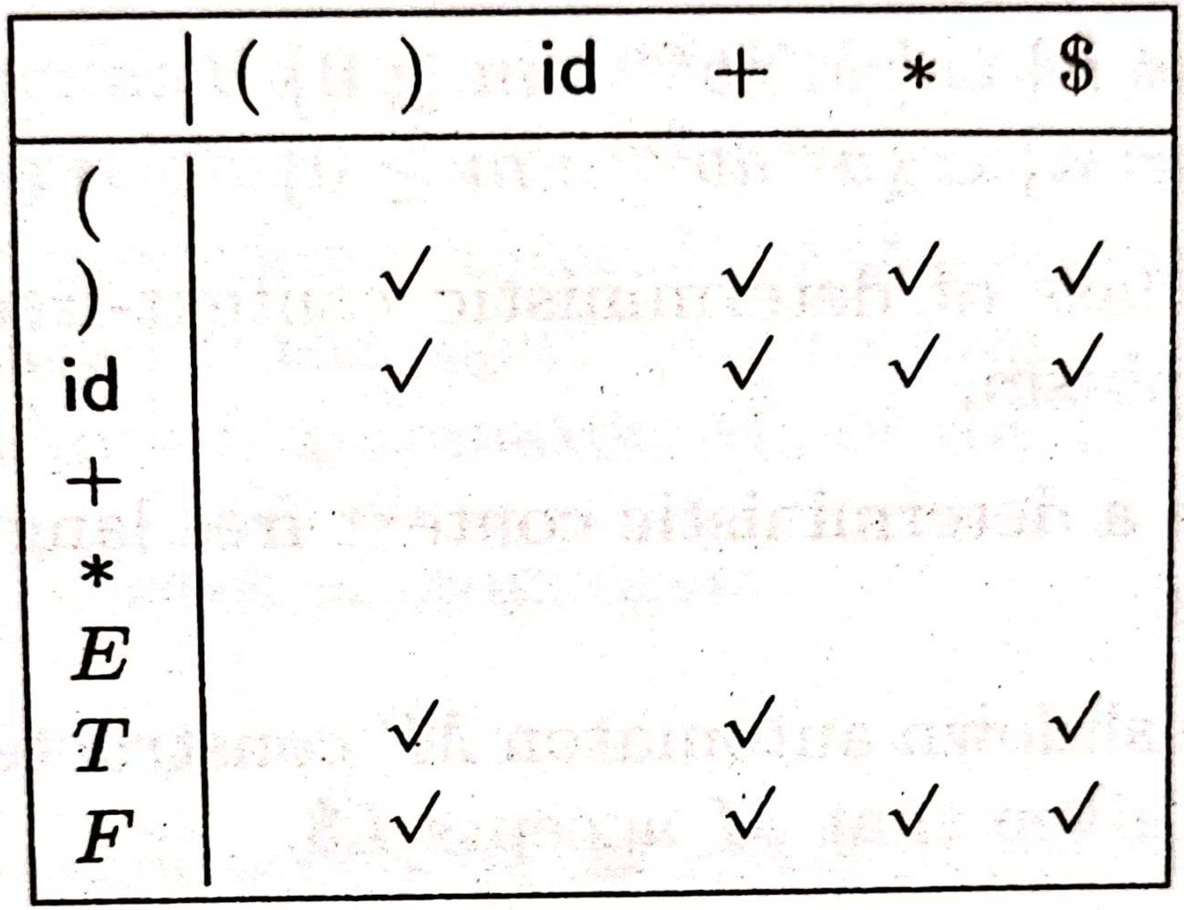
and *b* is next input symbol,

If *(a, b) ϵ P* then go with reduce else do shift.

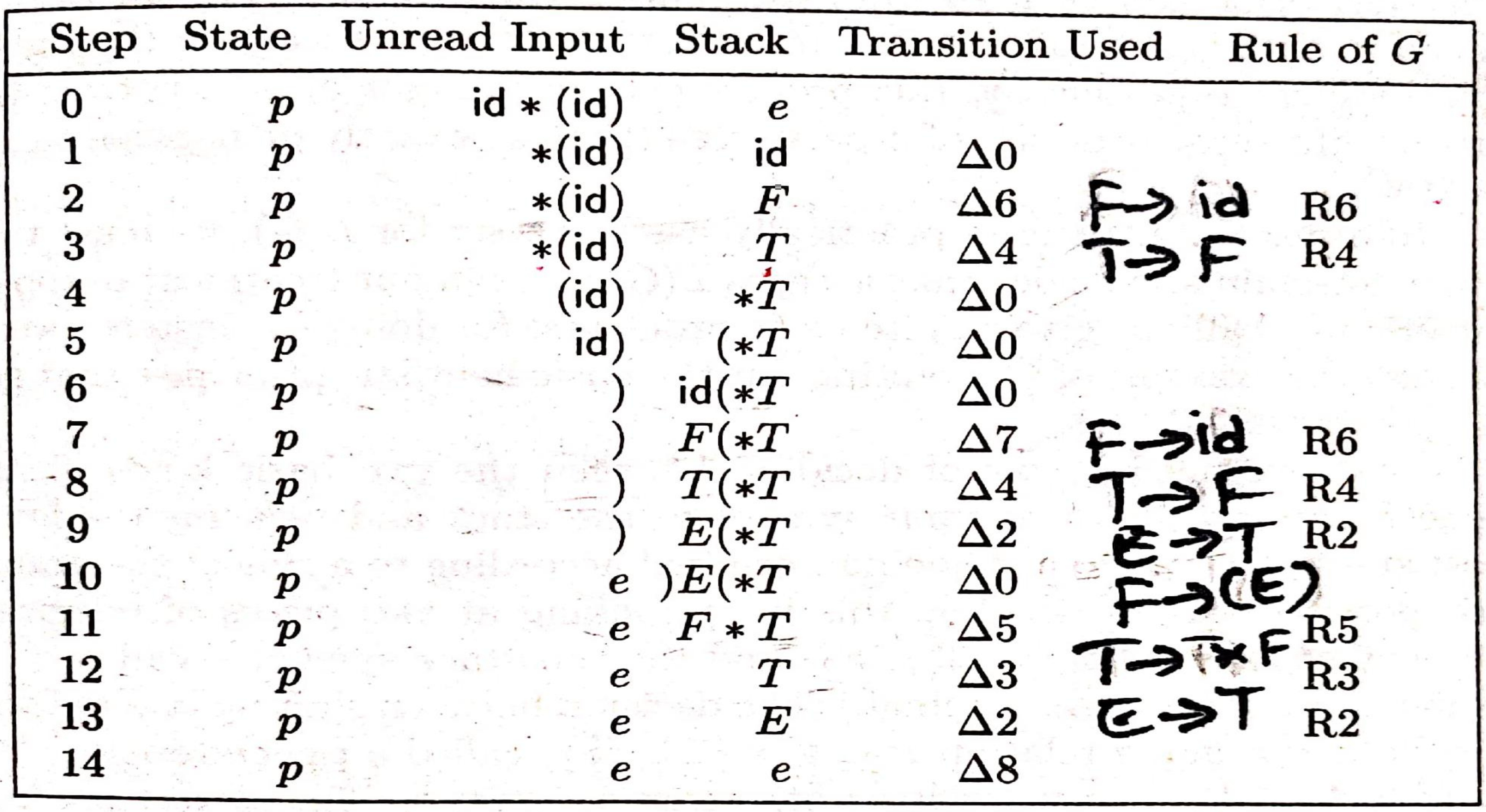
P is precedence relation.

(a, b) ϵ P if

$$S \overset{R}{\Rightarrow}{}^{*}_{G} \beta A b x \overset{R}{\Rightarrow}_{G} \beta \gamma a b x.$$



	(	)	id	+	*	$
(						
)		√		√	√	√
id		√		√	√	√
+						
*						
E						
T		√		√		√
F		√		√	√	√



Step	State	Unread Input	Stack	Transition Used	Rule of G
0	p	id $*$ (id)	e		
1	p	$*$(id)	id	$\Delta 0$	
2	p	$*$(id)	F	$\Delta 6$	F→id R6
3	p	$*$(id)	T	$\Delta 4$	T→F R4
4	p	(id)	$*T$	$\Delta 0$	
5	p	id)	$(*T$	$\Delta 0$	
6	p	)	id$(*T$	$\Delta 0$	
7	p	)	$F(*T$	$\Delta 7$	F→id R6
8	p	)	$T(*T$	$\Delta 4$	T→F R4
9	p	)	$E(*T$	$\Delta 2$	E→T R2
10	p	e	$)E(*T$	$\Delta 0$	
11	p	e	$F * T$	$\Delta 5$	F→(E) R5
12	p	e	T	$\Delta 3$	T→T*F R3
13	p	e	E	$\Delta 2$	E→T R2
14	p	e	e	$\Delta 8$	

Shift reduce conflict: Use precedence relation.
Reduce-Reduce conflict: Go with longest prefix for replacement.
The Grammars for the above heuristics work are known as *Weak precedence* Grammars.
Many Grammars for Programming languages can be converted to Weak precedence grammars.
Advantage of Bottom-up parsers:

1. Can be built for all grammars.
2. Errors can be detected as early as possible.
3. Can parse all languages parsed by Top-down parsers.

Other Properties of CFL

The Context-Free languages are closed under *union*, *concatenation*, and *Kleene* star.

The intersection of a Context-Free language with Regular Language is a Context Free Language.

CFLs are not closed under *intersection* or *complementation*.

There is a polynomial algorithm which, given a CFG, construct an equivalent PDA, and vice versa.

Let *G= (V, ∑, R, S)* be a Context-free Language.
The *fanout* of *G*, denoted by *Ø(G)*, is the largest number of symbols on the right-hand side of any rule in R.
A *path* in a parse-tree is a sequence of distinct nodes, each connected to the previous node by a line segment.
The first node is the Root of the tree, and last node is the leaf.
The *length of the path* is the number of line segments in it.
The *height of the parse tree* is the length of the longest path in it.

The yield of any parse tree of G of height h has length at most $\emptyset(G)^h$

Summary

1. PDA
2. PDA and CFG
3. CFG to PDA
4. Top-down parser
5. Deterministic parsers
6. Look-ahead parsers
7. Bottom-up parser
8. Shift/reduce & reduce/reduce conflict
9. Precedence relation
10. Some properties of CFL