Pushdown Automata
and
Context Free Language Parsing

Prof. A. Morzenti

NB: up to slide 13 notions assumed to be known from other previous courses

PUSHDOWN AUTOMATA

$$\overbrace{A_k}^{top}$$

$$A_1 A_2 \dots \overbrace{A_k}^{top}$$

$$a_1 a_2 \dots \quad \overbrace{a_i}^{curr.char.-cc} \quad \dots a_n \dashv$$

1) stack auxiliary memory +
   input string with terminator ⊣
3) operations:
   push(B), push(B₁, B₂, ... Bₙ): push symbol(s) on top of the stack
   empty test: a predicate that holds iff $k = 0$
   pop, if the stack is not empty, deletes A_k
4) Z_0 is the *initial* (*bottom*) stack symbol (can only be read)
5) configuration: current state, string portion from current character ***cc***, stack content

MOVE OF THE AUTOMATON:

- read *cc* and advance the head (***shift***), or (***spontaneous*** move) do not advance the head
- read the top stack symbol (possibly Z_0 if the stack is empty)
- based on the current char, state, and stack top symbol, go to a new state and replace the stack top symbol with a string (zero or more symbols)

DEFINITION OF PUSHDOWN AUTOMATON

A pushdown autromaton M (in general nondeterministic) is defined by:

1. Q — *finite set of states* of the control unit
2. Σ — *input alphabet*
3. Γ — *stack alphabet*
4. δ — *transition function*
5. $q_0 \in Q$ — *initial state*
6. $Z_0 \in \Gamma$ — *inital stack symbol*
7. $F \subseteq Q$ — set of *final states*

spontaneous move

TRANSITION FUNCTION:

domain: $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$

range: the powerset $\wp(Q \times \Gamma^*)$ of $Q \times \Gamma^*$

nondeterminism

READING (/ scanning / shift) MOVE:
(possibly nondeterministic)

$$\boxed{\begin{array}{l}\delta(q,a,Z)=\{(p_1,\gamma_1),(p_2,\gamma_2),...(p_n,\gamma_n)\} \\ \text{with } n\geq 1,\ a\in\Sigma, Z\in\Gamma,\ \ p_i\in Q,\gamma_i\in\Gamma^*\end{array}}$$

SPONTANEOUS MOVE:
(possibly nondeterministic)

$$\boxed{\begin{array}{l}\delta(q,\varepsilon,Z)=\{(p_1,\gamma_1),(p_2,\gamma_2),...(p_n,\gamma_n)\} \\ \text{with } n\geq 1,\ Z\in\Gamma,\ p_i\in Q,\ \ \gamma_i\in\Gamma^*\end{array}}$$

NONDETERMINISM: for a given triple (state, stack top, input) there are ≥2 possibilities among reading and spontaneous moves

INSTANTANEOUS CONFIGURATION OF MACHINE ***M***: a triple

$$\boxed{(q, y, \eta) \in Q \times \Sigma^{*} \times \Gamma^{+}}$$

current state of the control unit

unread portion of input x

stack content

INITIAL CONFIGURATION: (q_0, x, Z_0)
FINAL CONFIGURATION (q, ε, η) if $q \in F$ (NB: ε means input completely scanned)

TRANSITION FROM ONE CONFIGURATION TO THE NEXT: $\boxed{(q, y, \eta) \to (p, z, \lambda)}$

TRANSITION SEQUENCE: $\overset{*}{\to}$, $\overset{+}{\to}$

current config.	next config	applied move
$(q, az, \eta Z)$	$(p, z, \eta\gamma)$	reading move $\delta(q, a, Z) = \{(p, \gamma), ...\}$
$(q, az, \eta Z)$	$(p, az, \eta\gamma)$	spontaneous move $\delta(q, \varepsilon, Z) = \{(p, \gamma), ...\}$

NB: A ***string*** is pushed: it can be ε (just a *pop* operation)
or the same symbol previously on top (stack is unchanged)

A string x is recognized/accepted with final state if:

$$\boxed{(q_0, x, Z_0) \xrightarrow{+} (q, \varepsilon, \lambda)}$$

$q \in F$ and $\lambda \in \Gamma^*$
(no condition on λ, it can be ε, but not necessarily)

STATE-TRANSITION DIAGRAM FOR PUSHDOWN AUTOMATA

Example: even-length palindromes accepted with final state by a (nondeterministic) PDA

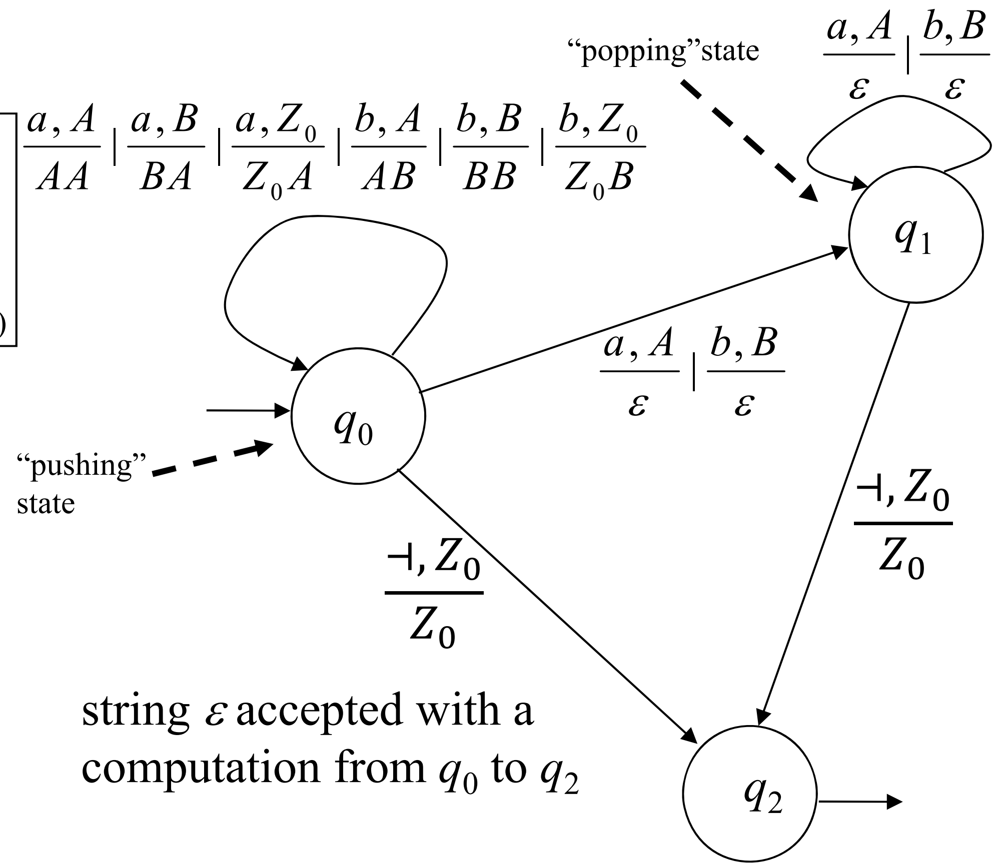
$$L = \left\{ uu^R \mid u \in \{a, b\}^* \right\}$$

Stack	x	State	Comment
Z_0	$aa\dashv$	q_0	
Z_0A	$a\dashv$	q_0	
Z_0AA	$\dashv$	q_0	reject: no move defined for (q_0, ⊣, A)

"guesses" that $|x|>2$

Stack	x	State	Comment
Z_0	$aa\dashv$	q_0	
Z_0A	$a\dashv$	q_0	
Z_0	$\dashv$	q_1	
Z_0	ε	q_2	acceptance with final state

"guesses" that $|x|=2$



string ε accepted with a computation from q_0 to q_2

FROM THE GRAMMAR TO THE PUSHDOWN AUTOMATON

1) Grammar rules seen as instructions of a ***non deterministic PDA*** with a single state; ***predictive*** (goal oriented) analysis: stack used as a list of future actions.

   Only one state ⇒ called "***daisy automaton***"

2) Stack includes terminal and nonterminal symbols.

   Stack = $A_1, \dots A_k$ goal is: read from input a string derived from A_k,

3) The goal can be restated recursively in subgoals if A_k is a nonterminal from which other symbols are derived

Initial goal is the axiom: goal is to read a sentence, i.e., a string derived from axiom $\boldsymbol{S}$

Initially stack is $\boldsymbol{Z_0}$ $\boldsymbol{S}$ and the input head on the first char of the input string

At each step the automaton chooses (nondeterministically) one of the applicable moves/rules

The string is recognized if the terminal ⊣ is read with an empty stack

FROM GRAMMAR RULES TO TRANSITIONS $A, B \in V, b \in \Sigma, A_i \in V \cup \Sigma$

Rule	Move	Comment
$A \to BA_1 \ldots A_n \quad n \geq 0$	**if** *top*= A **then** pop; push($A_n \ldots A_1 B$) **end if**	to recognize A one must recognize $B\ A_1 \ldots A_n$
$A \to bA_1 \ldots A_n \quad n \geq 0$	**if** $cc = b \wedge top = A$ **then** pop; push($A_n \ldots A_1$); shift **end if**	b is the first expected char and is read; $A_1 \ldots A_n$ still to be accepted
$A \to \varepsilon$	**if** *top*= A **then** pop **end if**	ε deriving from A is accepted
for each char. $b \in \Sigma$	**if** $cc = b \wedge top = b$ **then** pop; shift **end if**	b is the first expected char and is read;
– – –	**if** $cc = \dashv \ \wedge$ *empty stack* **then** accept **end if** halt	string completely scanned, agenda completed

Example – Rules and moves of the predictive recognizer

$$L = \{a^n b^m \mid n \geq m \geq 1\}$$

Rule	Moves
1. $S \to aS$	[$\delta(q_0, a, S)=(q_0, S)$] **if** $cc = a \wedge top=S$ **then** pop; push(S); shift **end if**
2. $S \to A$	[$\delta(q_0, \varepsilon, S) = (q_0, A)$] **if** $top= S$ **then** pop; push(A) **end if**
3. $A \to a\, A\, b$	[$\delta(q_0,a,A) = (q_0, bA)$] **if** $cc = a \wedge top = A$ **then** pop; push(bA); shift **end if**
4. $A \to ab$	[$\delta(q_0, a, A)=(q_0, b)$] **if** $cc = a \wedge top = A$ **then** pop; push(b); shift **end if**
5. --	[$\delta(q_0, b, b)=(q_0, \varepsilon)$] **if** $cc = b \wedge top = b$ **then** pop; shift **end if**
6. --	[halt] **if** $cc = \dashv \wedge$ *empty stack* **then** accept **end if**

Nondeterminism: between 1 and 2 (2 can also be chosen with input *a*); between 3 and 4

String $a^n b^m$, $n \geq m \geq 1$ analyzed as $a^{n-m} a^m b^m$,

"guessing nondeterministically" position where $a^m b^m$ starts, by choice between moves 1 and 2

"guess" of position where a^m ends and b^m starts by choice between moves 3 and 4

this automaton accepts a string iff the grammar generates it
for every accepting computation there is a derivation and viceversa
the automaton ***simulates the leftmost derivations***
the automaton must explore all derivations including nonaccepting ones
a string is accepted by several computations iff it is ambiguous

$S \Rightarrow A \Rightarrow aAb \Rightarrow aabb$

accepting computation:

	Stack	x
	Z_0S	$aabb$-|
$\delta(q_0,\varepsilon,S)=(q_0,A)$	Z_0A	$aabb$-|
$\delta(q_0,a,A)=(q_0,bA)$	Z_0bA	abb-|
$\delta(q_0,a,A)=(q_0,b)$	Z_0bb	bb-|
$\delta(q_0,b,b)=(q_0,\varepsilon)$	Z_0b	b-|
$\delta(q_0,b,b)=(q_0,\varepsilon)$	Z_0	-|

conversion grammar rules – PDA transitions is bidirectional:

It can be applied to obtain a grammar from a PDA

Therefore:

LANGUAGES DEFINED BY ***NONDETERMINISTIC*** PDA's AND CF LANGUAGES ARE A UNIQUE FAMILY

VARIETIES OF PUSHDOWN AUTOMATA

PDA can be enriched in various ways, concerning internal states and acceptance conditions

3 possible accepting modes:
- with final state (stack content immaterial)
- empty stack (current state immaterial)
- combined: (final state and empty stack)

PROPERTY – These three accepting modes are equivalent

Another important PDA feature: absence of spontaneous loops and on-line functioning

PROPERTY: any PDA can be converted into an equivalent one
- with no cycles of spontaneous moves
- which can decide acceptance right after reading the last input symbol

ELEMENTS UNTIL HERE ASSUMED KNOWN FROM PREVIOUS COURSES

INTERSECTION OF REGULAR AND CONTEXT FREE LANGUAGES

We can now justify the statement:
Intersection of a ***CF*** and a ***REG*** language is ***CF***

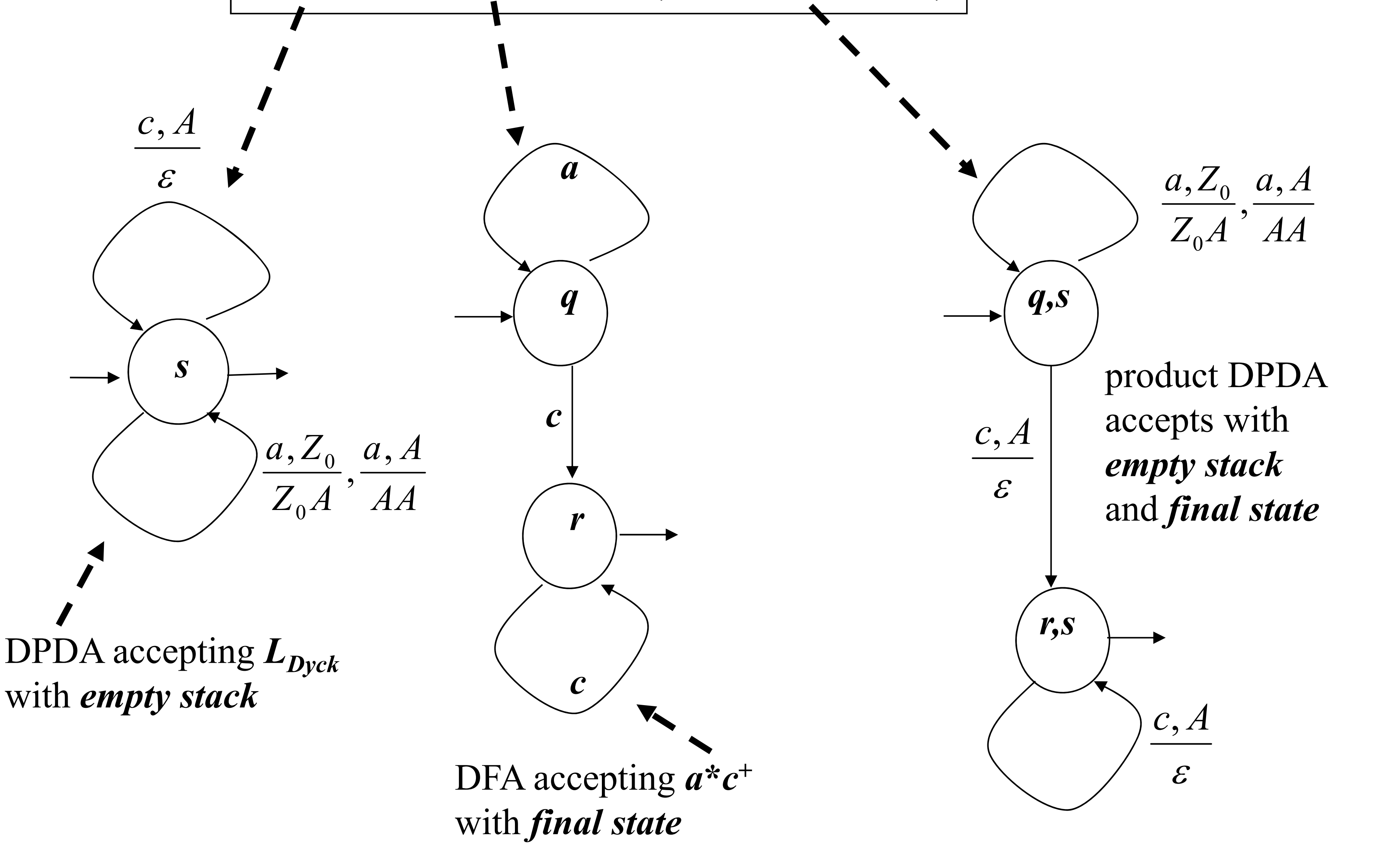
Given grammar ***G*** and automaton ***A***, obtain PDA ***M*** accepting $\boldsymbol{L(G) \cap L(A)}$ as follows:

1) build PDA ***N*** accepting ***L*(*G*) with empty stack**
2) build machine ***M*** product of machines ***N*** and ***A***,
   applying the known construction for finite automata adapted so that
   the product machine ***M*** manipulates the stack in the same way as component ***N***

The machine thus constructed:

- internal states are the product of the state sets of the component machines
- accepts with **final state and empty stack**
- final states are those including a final state of the finite automaton ***A***
- is deterministic if so are both machines ***N*** and ***A***
- accepts exactly the strings of the intersection of the two languages

Example $\boxed{L_{Dyck} \bigcap (a^* c^+) = \{a^n c^n \mid n \geq 1\}}$ ← Dyck language with one nest



PUSHDOWN AUTOMATA AND DETERMINISTIC LANGUAGES (DET)

Only deterministic CF languages (those accepted by a deterministic PDA) are considered in language and compiler design due to efficiency reasons

Nondeterminism is absent if δ is one-valued and also

if $\boldsymbol{\delta(q, a, A)}$ is defined for some $\boldsymbol{a} \in \Sigma$, then $\boldsymbol{\delta(q, \varepsilon, A)}$ is not defined
if $\boldsymbol{\delta(q, \varepsilon, A)}$ is defined then $\boldsymbol{\delta(q, a, A)}$ is not defined for any $\boldsymbol{a} \in \Sigma$

NB: therefore **a deterministic PDA *CAN* have spontaneous moves**

Relation between classe CF (Context Free Languages) and DET (deterministic ones)

Example: nondeterministic union of deterministic languages

$$\boxed{L = \left\{ a^n b^n \mid n \geq 1 \right\} \bigcup \left\{ a^n b^{2n} \mid n \geq 1 \right\} = L' \bigcup L''}$$

a PDA accepting x must push all a's and, if $x \in L'$ (e.g., *aabb*), pop one a for each b;
but if $x \in L''$ (e.g., *aabbbb*), two b's must be popped for each a in the stack
The PDA does not know which alternative holds, it must try both

L',L''$\in$DET, L',L''$\in$CF, L=L'$\cup$L'', L$\in$CF but L$\notin$DET,

hence **DET $\subseteq$ CF and DET $\neq$ CF**

CLOSURE PROPERTIES OF DETERMINISTIC CF LANGUAGES

We denote as ***L***, ***D***, and ***R*** a language in the familiy ***CF, DET*** and ***REG***

Operation	Property	(Property already known)
Reflection	$D^R \notin DET$	$D^R \in CF$
Star	$D^* \notin DET$	$D^* \in CF$
Complement	$\neg D \in DET$	$\neg L \notin CF$
Union	$D_1 \bigcup D_2 \notin DET, D \bigcup R \in DET$	$D_1 \bigcup D_2 \in CF$
Concatenation	$D_1.D_2 \notin DET, D.R \in DET$	$D_1.D_2 \in CF$
Intersection	$D \bigcap R \in DET$	$D_1 \bigcap D_2 \notin CF$

NB: the typical operations on languages (R, * , $\cup$, $\cdot$) **DO NOT** preserve determinism

SYNTAX ANALYSIS

Given a grammar ***G***, the syntax analyzer (***parser***)

- reads the source string x and
- if $x \in L(G)$,
    - accepts and possibly outputs a syntax tree or (equivalently) a derivation;
- otherwise it stops signalling an error (diagnosis)

TOP-DOWN AND BOTTOM-UP ANALYSIS

One given tree in general corresponds to various derivations (left, right, ...)

The two most important types of parsers characterized by the type of identified derivation: **left** or **right**, and **order** of the tree and derivation construction

TOP-DOWN ANALYSIS: builds
a **left** derivation in **direct order**
syntax tree: **from root to leaves**, through ***expansions***

BOTTOM-UP ANALYSIS: builds
right derivation in **reverse** order
syntax tree: **from leaves to root**, through ***reductions***

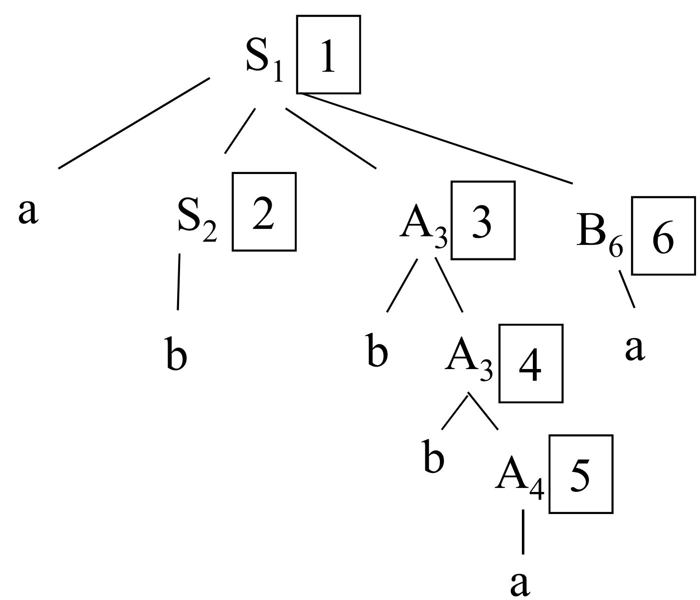
Example – top-down analysis of sentence:

a b b b a a

framed numbers = order in rule application
subscripts of nonterminals in the tree = applied rule
Leftmost: always expanded firsr nonterm. from the left

1. $S \rightarrow aSAB$	2. $S \rightarrow b$
3. $A \rightarrow bA$	4. $A \rightarrow a$
5. $B \rightarrow cB$	6. $B \rightarrow a$

S ⇒ aSAB ⇒ abAB ⇒ abbAB ⇒ abbbAB ⇒ abbbaB ⇒ abbbaa



Example – bottom-up analyis of sentence:

a b b b a a

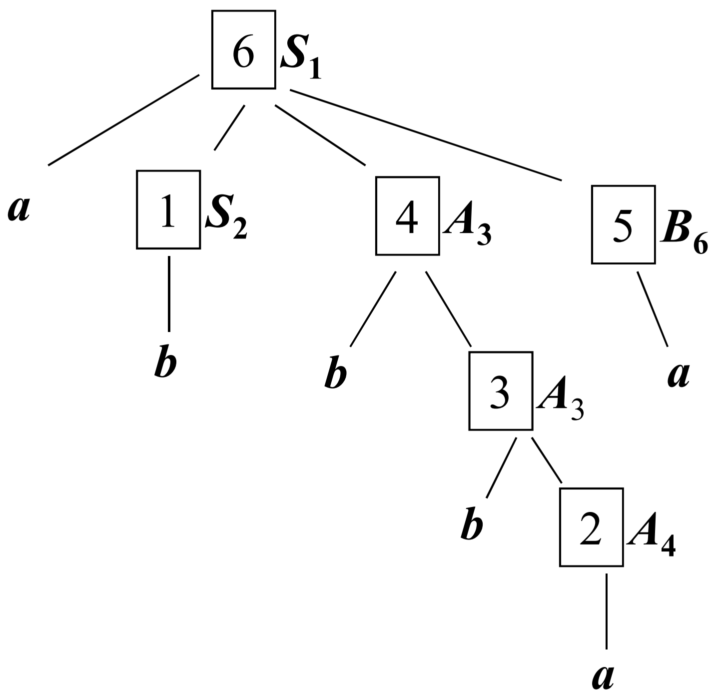
1. $S \to aSAB$	2. $S \to b$
3. $A \to bA$	4. $A \to a$
5. $B \to cB$	6. $B \to a$

Derivation (rightmost n.t. always expanded):

S ⇒ aSAB ⇒ aSAa ⇒ aSbAa ⇒ aSbbAa ⇒ aSbbaa ⇒ a b b b a a

framed numbers = order in reduction sequence

subscripts of nonterminals in the tree = applied rule



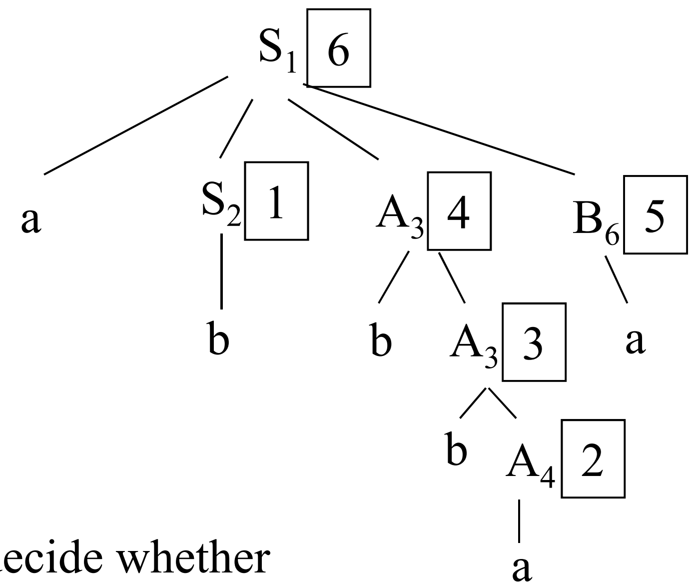
right parts of rules are ***reduced*** as they are scanned, first in the sentence, then in the phrase forms obtained after the ***reductions***. The process terminates when the entire string is reduced to the axiom

NB: reconstruction of a ***right derivation*** in ***reverse order***

Bottom-up analysis: the **reduction** operations (>rd> below) trasform part of a phrase form into a string $\alpha \in (\Sigma \cup V)^*$, called «***viable prefix***», that may include the result of previous reductions (and is stored in the parser's stack). Here below, the '^' shows the head position (right of ^ is the unread string), **underline** shows the part to be reduced, called «***handle»***

a**b**bbaa >rd> aSbb**a**a >rd> aSb**bA**a >rd> aS**bA**a >rd> aSA**a** >rd> **aSAB** >rd> S

1. $S \rightarrow aSAB$	2. $S \rightarrow b$
3. $A \rightarrow bA$	4. $A \rightarrow a$
5. $B \rightarrow cB$	6. $B \rightarrow a$



At each step of the analysis, the parser must decide whether

- to continue and read the next symbol (shift), or
- to build a subtree from a portion of the viable prefix

it chooses based on the symbol(s) coming after the current one (***lookahead***)

GRAMMARS AS NETWORKS OF FINITE AUTOMATA

Suppose $\boldsymbol{G}$ in extended form: every nonterminal has a unique rule

$\boldsymbol{A \rightarrow \alpha}$ with $\boldsymbol{\alpha}$ regular expression on terminals and nonterminals

α defines a regular language, hence there exists a finite automaton $\boldsymbol{M_A}$ that accepts $\boldsymbol{L(\alpha)}$

any transition of $\boldsymbol{M_A}$ labeled by a nonterminal $\boldsymbol{B}$

is interpreted as a «call» of an automaton $\boldsymbol{M_B}$ (if $\boldsymbol{B=A}$ then recursive call)

let us call

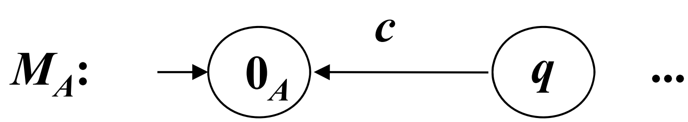
- "***machines***" the finite automata of the various nonterminals,
- "***automaton***" the PDA that accepts and parses the language $\boldsymbol{L(G)}$
- "***net***" the set of all machines

- $\boldsymbol{L(q)}$ the set of terminal strings generated along a path of a machine, starting from state $\boldsymbol{q}$ (possibly including calls of other machines) and reaching a final state (examples follow)

We set a further requirement on machines corresponding to nonterminals

The initial state $\mathbf{0}_A$ of machine $\boldsymbol{A}$ is not visited after the start of the computation

No machine $\boldsymbol{M_A}$ includes a transition like



Requirement very easy to "implement" in case it is not satisfied …

… it suffices to add one state (to be the new initial state) and a few arcs …

$\Rightarrow$ the machine is not minimal, but only one state has been added

Automata satisfying this condition are called

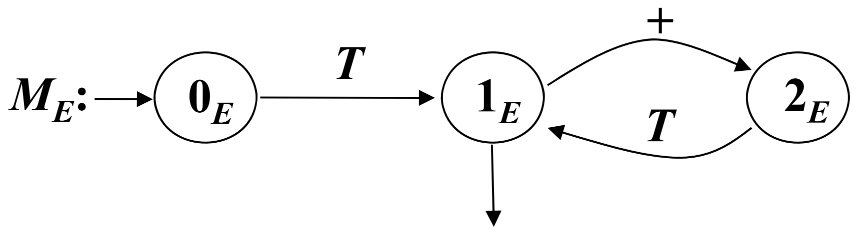
normalized or with initial state **non recirculating** or **non reentrant**

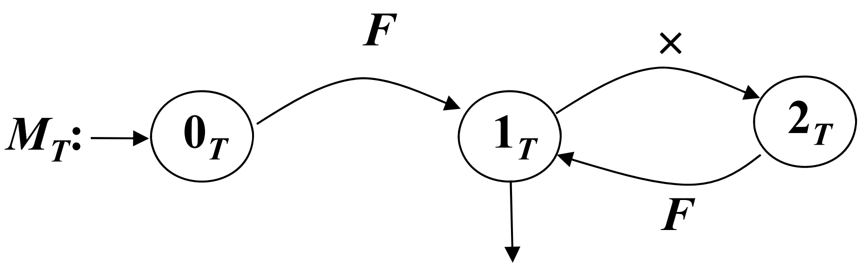
NB: it is **NOT forbidden** that the initial state be also final

(which is necessary if $\varepsilon \in \boldsymbol{L(A)}$)

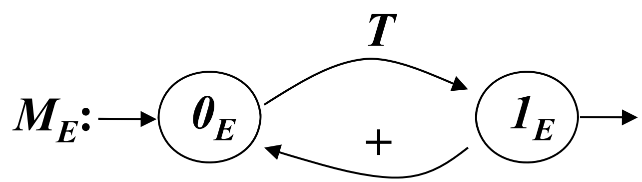
Example – Arithmetic expressions

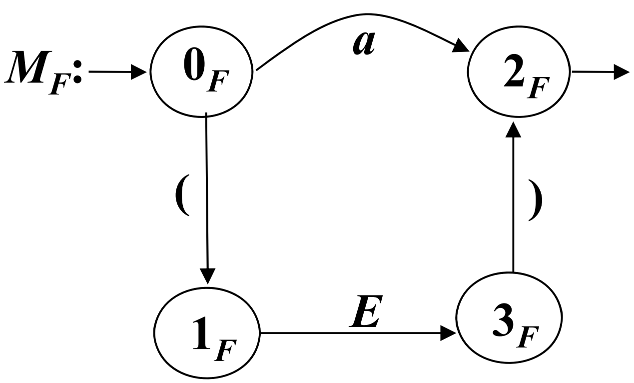
$$E \to T(+T)^*$$
$$T \to F(\times F)^*$$
$$F \to a \mid \text{‘('}\; E \;\text{‘)'}$$





NB: M_E minimal not normalized





$L(2_F) = \{ \varepsilon \}$

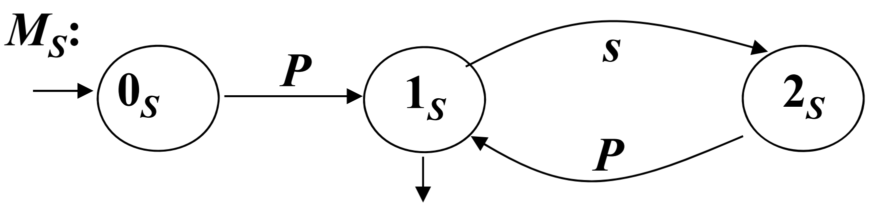
$L(3_F) = \{) \}$

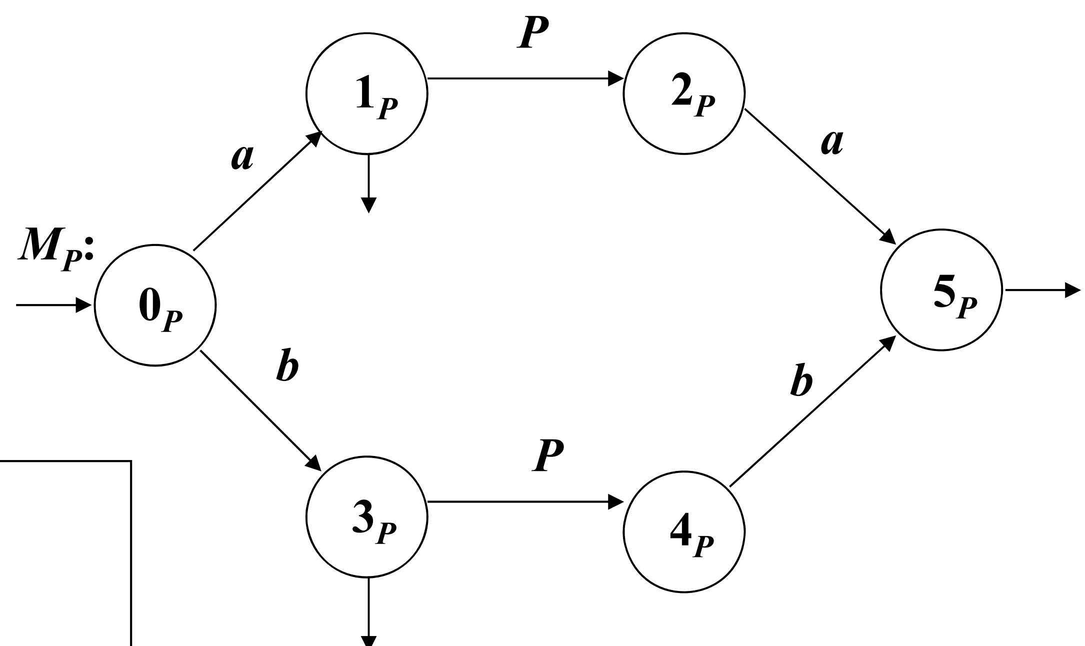
$L(0_F) = L(F)$ the language generated from F

$L(1_F) = L(E) \cdot \{) \}$

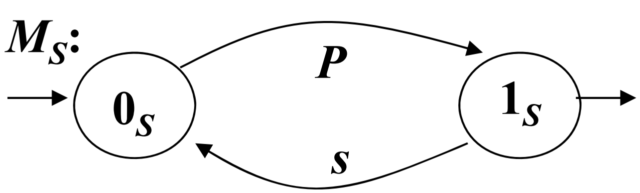
Example – List of odd-length palindromes, separated by ‘*s*’

$$S \to P\ (\ s\ P\ )^*$$
$$P \to aPa \mid bPb \mid a \mid b$$





NB: M_S minimal not normalized



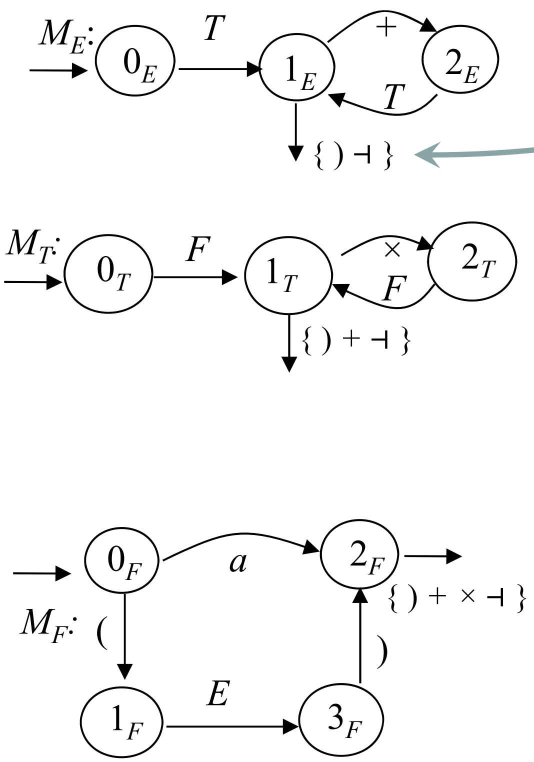
TOP-DOWN ANALYSIS WITH RECURSIVE PROCEDURES

Simple and elegant: the procedure code reflects the transitions of machines

$E \to T\,(+\,T\,)^*$
$T \to F\,(\times\,F\,)^*$
$F \to a \mid \text{‘(’}\,E\,\text{‘)’}$

Example – Arithmetic expressions

When in the automaton there are ***bifurcations***, to decide which one to follow one must consider the symbols encountered on every arc, including those encountered when «exiting the machine»



cc is current char,
symbol under input head
updated by the ***next*** procedure

```
procedure E
 call T;
 while (cc = ‘+’)
  cc:=next;
  call T;
 end;
 if (cc ∈ { ) ⊣} )
   return:
  else error;
 endif
end E;
```

```
procedure T
 call F;
 while (cc = ‘×’)
  cc:=next;
  call F;
 end;
 if (cc ∈ { ) + ⊣} )
   return:
  else error;
 endif
end T;
```

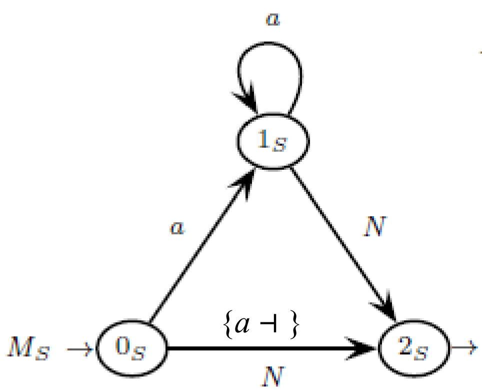
```
procedure F
 if (cc = ‘a’)
   cc=next;
  elsif (cc=‘(‘)
    cc:=next;
    call E;
    if (cc = ‘)’)
      cc:=next;
     else error;
    endif;
  else error;
 endif;
 if (cc ∈ { ) + × ⊣ } )
       return;
      else error;
 endif;
end F;
```

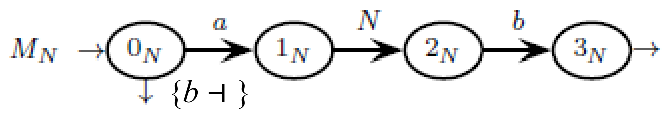
NB: this method **does not work** if

the sets of possible symbols on arcs outgoing from a single state (the «guide sets»)

are **not disjointed**

Example: $\boldsymbol{S \rightarrow a^* N}$ $\quad\quad$ $\boldsymbol{N \rightarrow aNb \mid \varepsilon}$ $\quad\quad$ $\boldsymbol{L=\{ a^n\ b^m \mid n \geq m \geq 0\}}$



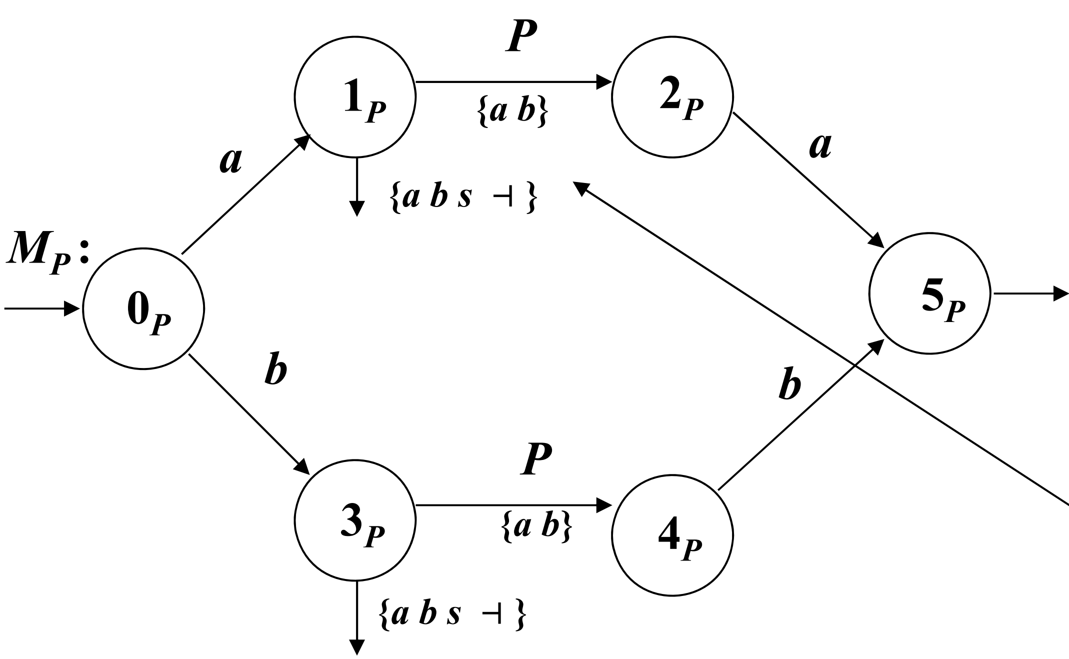



```
procedure S
 if (cc='a')
    ?? call N; ??
    ?? cc:=next; ??
   ...
end S;
```

Another Example – List of odd-length palindromes, separated by an ‘*s*’
(NB: it is a nondeterministic language: even the bottom-up method does not work)

$$S \rightarrow P\,(\,s\,P\,)^*$$
$$P \rightarrow aPa \mid bPb \mid a \mid b$$

```
procedure P
 if ( cc = ‘a’ )
   cc:=next;
   if ( cc ∈ {a b} )
     ?? call P; ??
     ?? return; ??
   ...
end P;
```



the recursive procedure cannot decide based on the next symbol