SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

FOUNDATIONS OF COMPUTER SCIENCE
LECTURE 6: Context-free languages

Prof. Daniele Gorla

How can we generate non-regular languages?

Let’s start from $L = \{0^n 1^n \mid n \geq 0\}$ that is not regular (see last class).

Hence, no regular grammar can generate this language.

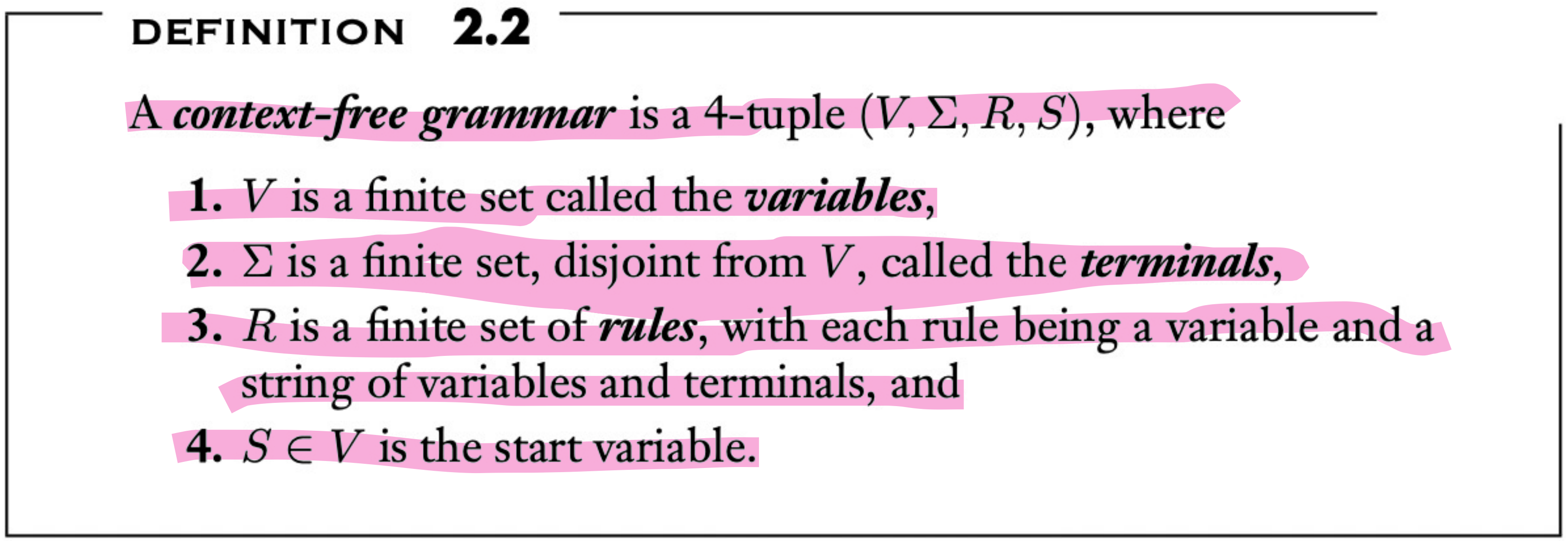
By contrast, it is easy to see that L is generated by the following (non-regular) grammar:

```
S ::= ε | 0S1
```

Notice that the grammar is not (left-/right-) linear because of the RHS $0S1$ (where the variable S is neither the left-most nor the right-most symbol).

So, we need a new kind of grammar that we call *context-free* (the origin of this name will be clear when we shall present the so called «context-sensitive» grammars).

Context-free Grammars



DEFINITION 2.2

A ***context-free grammar*** is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called the ***variables***,
2. Σ is a finite set, disjoint from V, called the ***terminals***,
3. R is a finite set of ***rules***, with each rule being a variable and a string of variables and terminals, and
4. $S \in V$ is the start variable.

Said in a more coincise way: $R \subseteq V \times (\Sigma \cup V)^*$

This generalizes regular grammars, where $R \subseteq V \times (\Sigma^* \cup V\Sigma^* \cup \Sigma^* V)$

We call ***context-free*** any language L for which there exists a context-free grammar G such that $L = L(G)$.

Since regular grammars are context-free, every regular language is context-free.

Algebric Expressions on Natural numbers

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

EXPR	::=	EXPR+EXPR | EXPR×EXPR | (EXPR) | NUMB
NUMB	::=	DIGIT | NONZERODIGIT DIGITSEQ
DIGITSEQ	::=	DIGIT | DIGIT DIGITSEQ
DIGIT	::=	0 | NONZERODIGIT
NONZERODIGIT	::=	1 | 2 | … | 9

This grammar (already seen) is not regular, but it is context-free.

There are many interesting problems related to context-free grammars, that mostly appear when using such grammars for specifying the syntax (and semantics) of a programming language.

→ *However, for the aims of this course, these aspects are not very relevant;*
if you're interested, please read the parts that we skip on the text books.

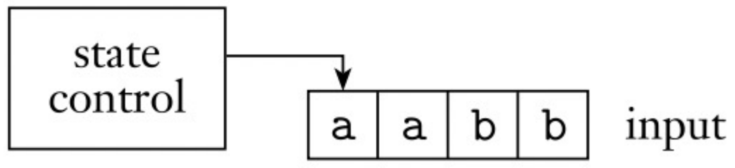
For some C.F. languages, we can easily find a C.F. grammar that generates them; but for other ones, this task can be less easy (e.g., try with the language $\{w \mid w$ has an equal number of 0s and 1s$\}$) → we now introduce a new kind of automata (that may simplify the task)

Pushdown Automata

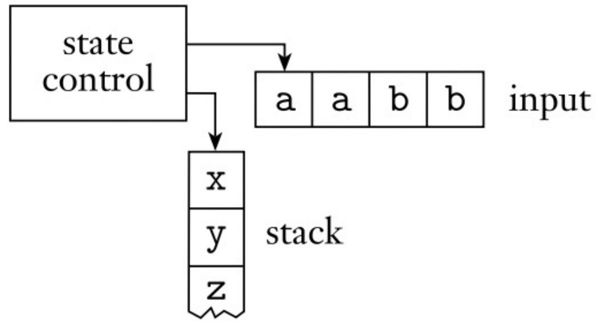
Like NFA but with an extra component called a ***stack***

→ additional memory beyond the finite amount available in the control

DFA/NFA:



PDA:



A PDA can write symbols on the stack and read them later

Writing a symbol "pushes down" all the other symbols on the stack

At any time the symbol on the top of the stack can be read and removed (and the remaining symbols then move back up).

Writing a symbol on the stack is ***pushing*** the symbol, and removing a symbol is ***popping*** it.

Access to the stack, for both reading and writing, may be done only at the top

→ a stack is a "last in, first out" (LIFO) storage device

The unlimited nature of a stack allows the PDA to store numbers of unbounded size

Intuitive use of a PDA

Consider the language $\{w \mid w \text{ has an equal number of 0s and 1s}\}$

As we already said, devising a C.F. grammar for it is not easy

By contrast, it is quite intuitive to imagine how a PDA can recognize it:

- If input = b ($\in \{0,1\}$) and the stack is empty → push b
- If input = b and top = b → push b
- If input = b and top $\neq b$ → pop
- If no input is left and the stack is empty → ACCEPT

  Otherwise REJECT

EXAMPLE: let's try with 00111100

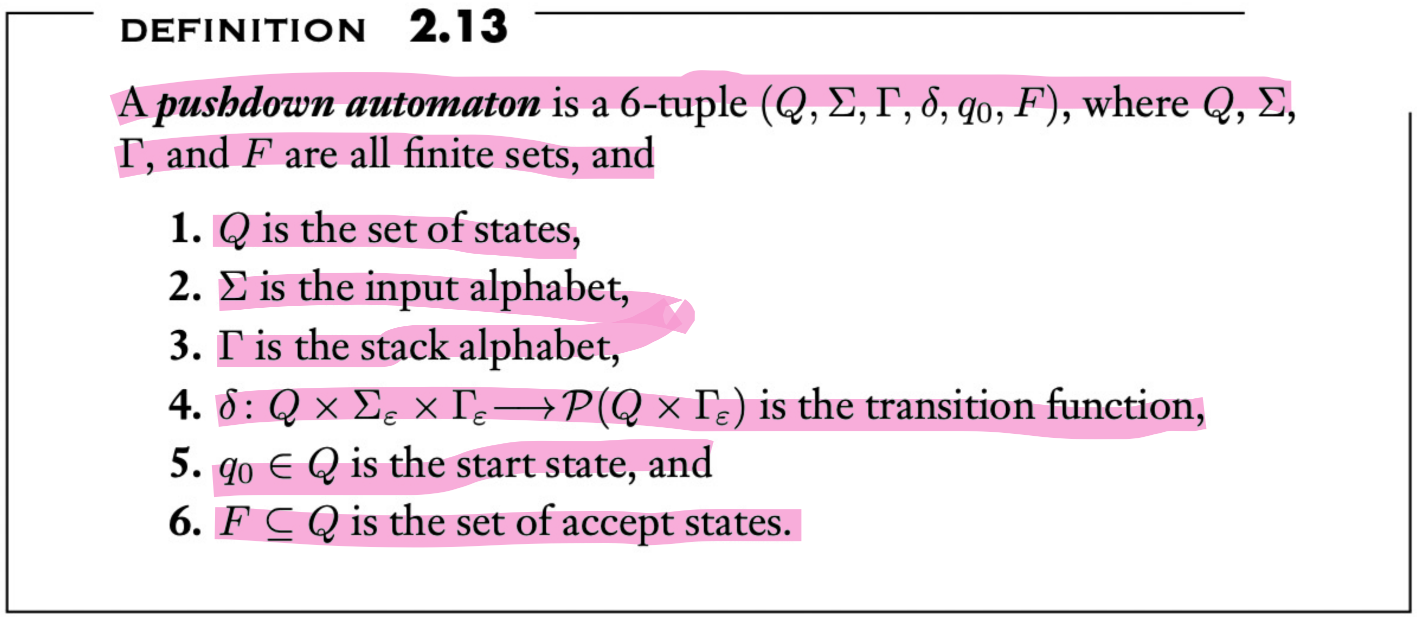
a) input 0, stack → 0 b) input 0, stack → 00 c) input 1, stack → 0

d) input 1, stack → ε e) input 1, stack → 1 f) input 1, stack → 11

g) input 0, stack → 1 h) input 0, stack → ε

$\Longrightarrow$ end of input, empty stack $\Longrightarrow$ ACCEPT

Formal Definition of PDA



DEFINITION 2.13

A ***pushdown automaton*** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ, and F are all finite sets, and

1. Q is the set of states,
2. Σ is the input alphabet,
3. Γ is the stack alphabet,
4. $\delta \colon Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.

The PDA may use different alphabets for inputs and stack (Σ and Γ)
Denote with $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ and $\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$

The domain of the transition function is $Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon$.
→ the next move is determined by the current state, next input and top of the stack
(either symbol may be ε: the PDA can move without reading from input and/or stack)

The range of the transition function is $P(Q \times \Gamma_\varepsilon)$
→ it enters some new state and possibly writes a symbol on the top of the stack
(many such moves, because of non-determinism)

Computation in a PDA

A PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ accepts input w if

$w = w_1 w_2 \dots w_m$, where each $w_i \in \Sigma_\varepsilon$

and there exists $r_0, r_1, \dots, r_m \in Q$ and $s_0, s_1, \dots, s_m \in \Gamma^*$ such that

- $r_0 = q_0$ and $s_0 = \varepsilon$
- $r_m \in F$
- For $i = 0, \dots, m - 1$,

  if $s_i = at$ for some $a \in \Gamma_\varepsilon$ and $t \in \Gamma^*$,

  then $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$ and $s_{i+1} = bt$ for some $b \in \Gamma_\varepsilon$

The language recognized by M, written $L(M)$, is the set of all inputs accepted by M.

Example

Consider the language $\{0^n 1^n \mid n \geq 0\}$. Let M_1 be $(Q, \Sigma, \Gamma, \delta, q_1, F)$, where

$Q = \{q_1, q_2, q_3, q_4\}$,

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

$\Gamma = \{0, \$\}$,

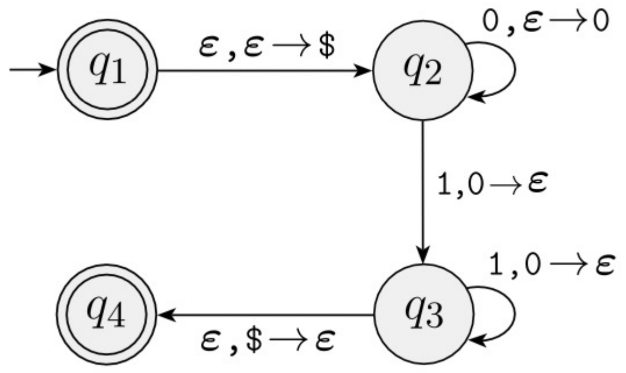
$F = \{q_1, q_4\}$, and

δ is given by the following table, wherein blank entries signify $\emptyset$.

Input:	0			1			ε		
Stack:	0	\$	ε	0	\$	ε	0	\$	ε
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$					
q_3				$\{(q_3, \varepsilon)\}$				$\{(q_4, \varepsilon)\}$	
q_4									

This can be pictorially represented as follows, where we write "$a, b \rightarrow c$" to mean that when the machine is reading an a from the input, it may replace the symbol b on the top of the stack with a c.

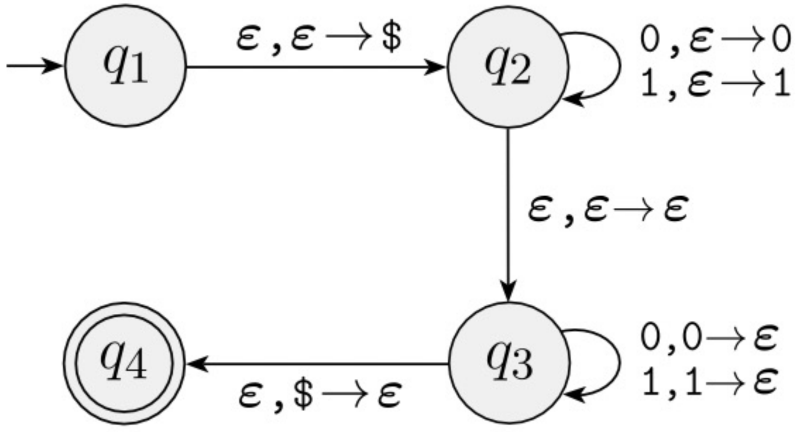
→ Any of a, b and c may be ε !!


q_1
$\varepsilon, \varepsilon \rightarrow \$$
q_2
$0, \varepsilon \rightarrow 0$
$1, 0 \rightarrow \varepsilon$
q_3
$1, 0 \rightarrow \varepsilon$
q_4
$\varepsilon, \$ \rightarrow \varepsilon$

Another example

Give a PDA that recognizes the language $\{ww^R \mid w \in \{0,1\}^*\}$

(recall that w^R means w written backwards)



q_1
$\varepsilon, \varepsilon \rightarrow \$$
q_2
$0, \varepsilon \rightarrow 0$
$1, \varepsilon \rightarrow 1$
$\varepsilon, \varepsilon \rightarrow \varepsilon$
q_3
$0, 0 \rightarrow \varepsilon$
$1, 1 \rightarrow \varepsilon$
$\varepsilon, \$ \rightarrow \varepsilon$
q_4

EXERCISE: Give the PDA for the language $\{w \mid w$ has an equal number of 0s and 1s$\}$

(Hint: it should behave like we said a few slides before)

PDAs and CF-Grammars (1)

Thm.: If L is context-free, then there exists a PDA that recognizes it

Proof

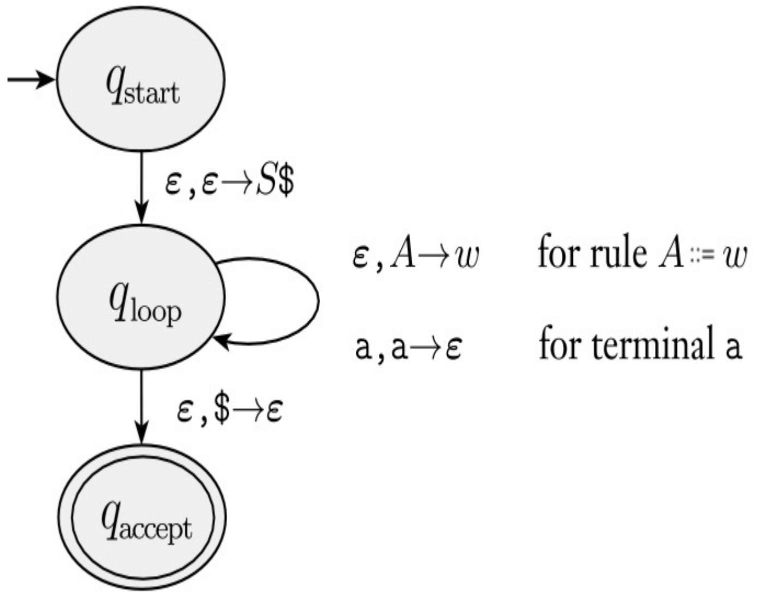
By definition there exists a CFG $G = (V, \Sigma, R, S)$ s.t. $L = L(G)$.

We show how to convert G into an equivalent PDA $M = (Q, \Sigma, \Sigma \cup V \cup \{\$\}, \delta, q_{start}, \{q_{accept}\})$

Intuitively:

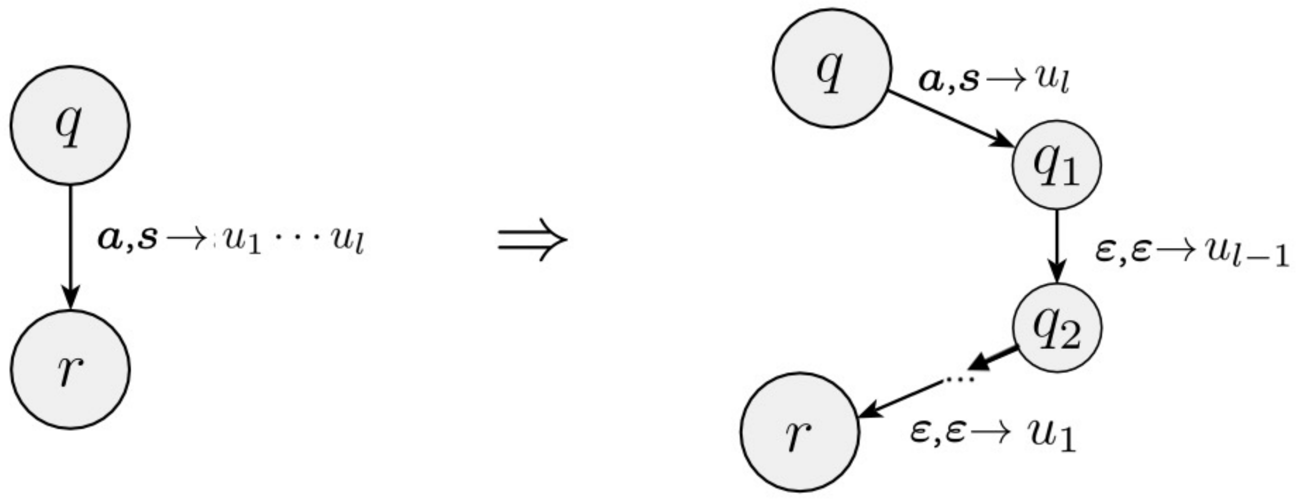
1. Place the marker symbol \$ and the start variable on the stack.
2. Repeat the following steps forever.
   - **a.** If the top of stack is a variable symbol A, nondeterministically select one of the rules for A and substitute A by the string on the right-hand side of the rule.
   - **b.** If the top of stack is a terminal symbol a, read the next symbol from the input and compare it to a. If they match, repeat. If they do not match, reject on this branch of the nondeterminism.
   - **c.** If the top of stack is the symbol \$, enter the accept state. Doing so accepts the input if it has all been read.

More formally:


q_{start}
$\varepsilon, \varepsilon \to S\$$
q_{loop}
$\varepsilon, A \to w$ for rule $A ::= w$
a, a $\to \varepsilon$ for terminal a
$\varepsilon, \$ \to \varepsilon$
q_{accept}

PDAs and CF-Grammars (2)

So, $Q = \{q_{\text{start}}, q_{\text{loop}}, q_{\text{accept}}\} \cup E$, where E is a set of new states that implement the writing of a sequence of stack symbols into the stack:



q
$a,s \rightarrow u_1 \cdots u_l$
r
$\Rightarrow$
q
$a,s \rightarrow u_l$
q_1
$\varepsilon,\varepsilon \rightarrow u_{l-1}$
q_2
...
$\varepsilon,\varepsilon \rightarrow u_1$
r

The transition function is defined accordingly (see the book for the details).

Q.E.D.

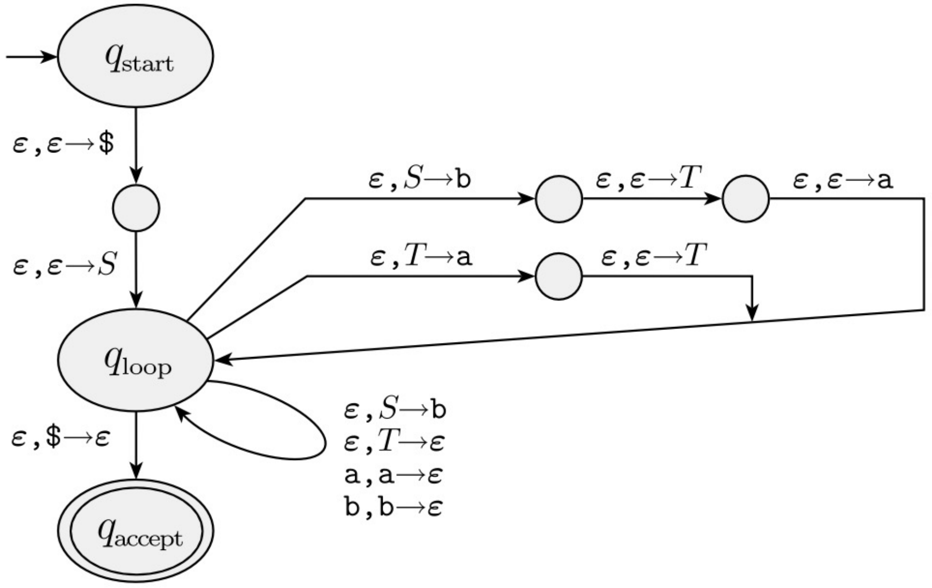
PDAs and CF-Grammars (3)

As an example, consider the grammar:

$$S ::= aTb \mid b$$
$$T ::= Ta \mid \varepsilon$$

This grammar generates $\{a^n b \mid n \geq 0\}$

The PDA associated to it is:



q_{start}
$\varepsilon,\varepsilon\to\$$
$\varepsilon,\varepsilon\to S$
$\varepsilon,S\to$ b
$\varepsilon,\varepsilon\to T$
$\varepsilon,\varepsilon\to$ a
$\varepsilon,T\to$ a
$\varepsilon,\varepsilon\to T$
q_{loop}
$\varepsilon,S\to$ b
$\varepsilon,T\to\varepsilon$
a,a$\to\varepsilon$
b,b$\to\varepsilon$
$\varepsilon,\$\to\varepsilon$
q_{accept}

This is how the stack and the input evolve while accepting *aab*:

			a		T				
			T	T	a	a			
		S	b	b	b	b	b		
STACK	ε	$	$	$	$	$	$	$	ε
INPUT			a			a	b		

PDAs and CF-Grammars (4)

<u>**Thm.:**</u> If L is recognized by a PDA, then L is context-free.

Proof

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a PDA that accepts L.

To simplify the proof, we modify M to give it the following three features:

1. It has a single accept state, q_{accept} ;
2. It empties its stack before accepting;
3. Each transition either pushes a symbol onto the stack or pops one off the stack, but it does not do both at the same time nor none of them.

Ensuring features 1 and 2 is easy:

- start the computation by putting \$ in the stack
- add an $(\varepsilon,\varepsilon,x)$-move (for some x) from every state of F to a new state q', where the stack is emptied;
- add an $(\varepsilon,\$,\varepsilon)$-move from q' to q_{accept} .

For giving feature 3, we replace

- each transition that simultaneously pops and pushes with a two transition sequence that goes through a new state;
- each transition that neither pops nor pushes with a two transition sequence that pushes then pops an arbitrary stack symbol by passing through a new state.

Let $M' = (Q', \Sigma, \Gamma, \delta', q_0, \{q_{\text{accept}}\})$ be the resulting PDA; trivially, $L(M) = L(M')$.

PDAs and CF-Grammars (5)

We now construct an equivalent CFG $G = (V, \Sigma, R, S)$ s.t. $L = L(G)$, where

- $V = \{ A_{pq} \mid p, q \in Q' \}$

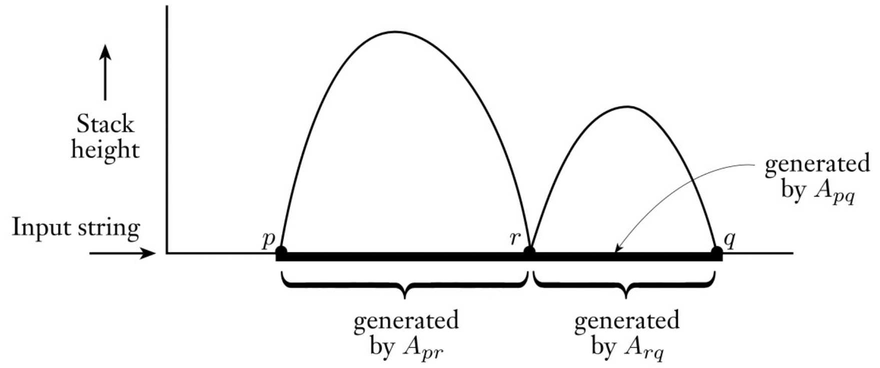
  → A_{pq} generates all words that bring M' from p with empty stack to q with empty stack
- $S = A_{q_0 q_{accept}}$
- R is defined as follows:
  - For each $p \in Q'$, put the rule $A_{pp} ::= \varepsilon$
  - For each $p,q,r \in Q'$ s.t. the stack is empty in r,

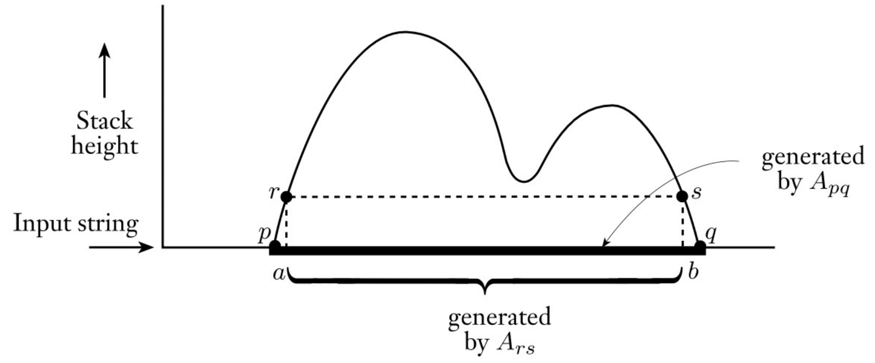
    put the rule $A_{pq} ::= A_{pr} A_{rq}$
  - For each $p,q,r,s \in Q'$, $u \in \Gamma$, and $a,b \in \Sigma_\varepsilon$,

    if $\delta'(p,a,\varepsilon)$ contains (r,u) and

    $\delta'(s,b,u)$ contains (q, ε),

    then put the rule $A_{pq} ::= a\, A_{rs}\, b$





To complete the proof, we need to prove that:

A_{pq} generates w *IFF* w can bring M'

from p with empty stack to q with empty stack

Both directions of this proof are quite technical; see the book if you want to see the details.

By taking $p = q_{\text{start}}$ and $q = q_{\text{accept}}$, the equivalence between the PDA and the grammar holds.

Q.E.D.

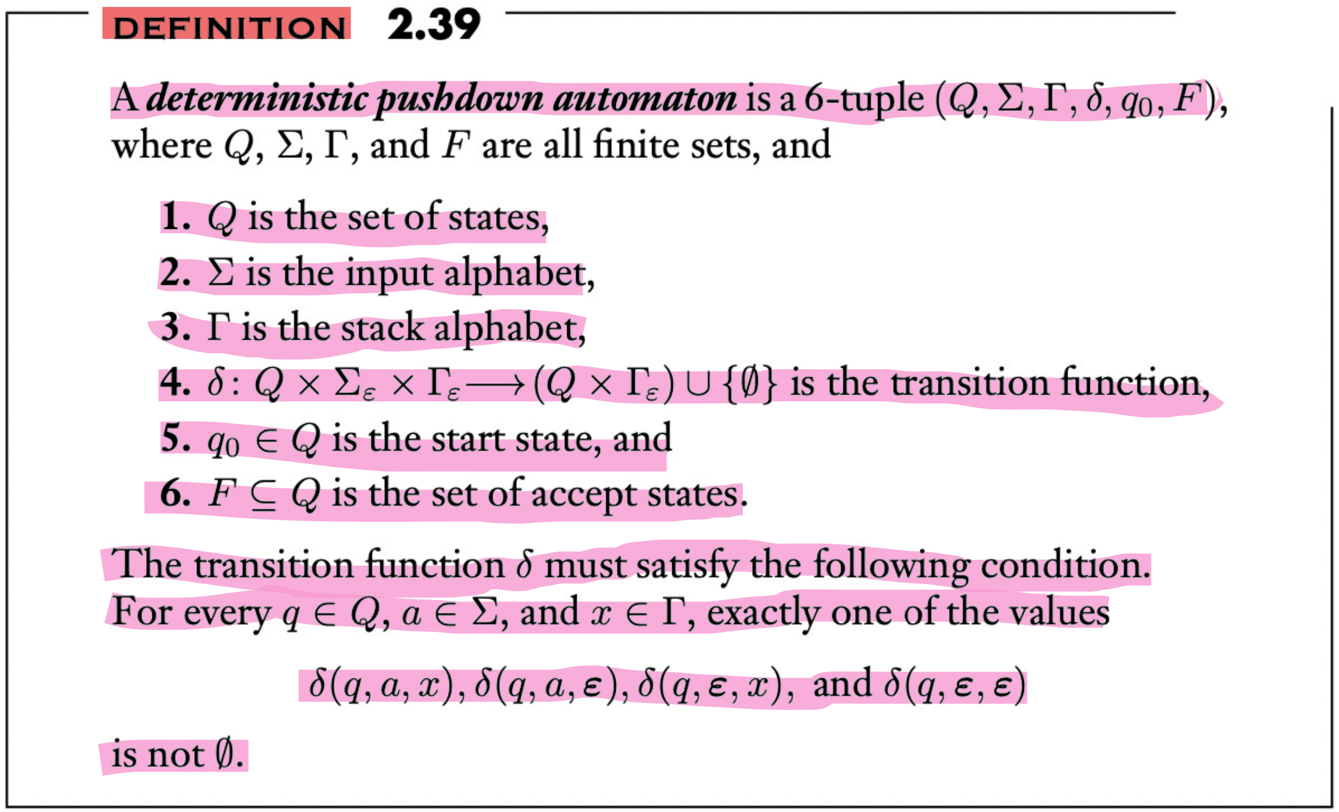
Deterministic PDA

The basic principle of determinism is: at each step, a DPDA has at most one way to proceed (according to its transition function).

Defining DPDAs is more complicated than defining DFAs

→ DPDAs may read an input symbol without popping a stack symbol, and vice versa.

Accordingly, we allow ε-moves in the DPDA's transition function (whereas ε-moves were prohibited in DFAs).



DEFINITION 2.39

A ***deterministic pushdown automaton*** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ, and F are all finite sets, and

1. Q is the set of states,
2. Σ is the input alphabet,
3. Γ is the stack alphabet,
4. $\delta\colon Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow (Q \times \Gamma_\varepsilon) \cup \{\emptyset\}$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.

The transition function δ must satisfy the following condition. For every $q \in Q$, $a \in \Sigma$, and $x \in \Gamma$, exactly one of the values

$$\delta(q, a, x), \delta(q, a, \varepsilon), \delta(q, \varepsilon, x), \text{ and } \delta(q, \varepsilon, \varepsilon)$$

is not $\emptyset$.

letter U epsilon

The determinism condition

For every $q \in Q$, $a \in \Sigma$, and $x \in \Gamma$, exactly one of the values

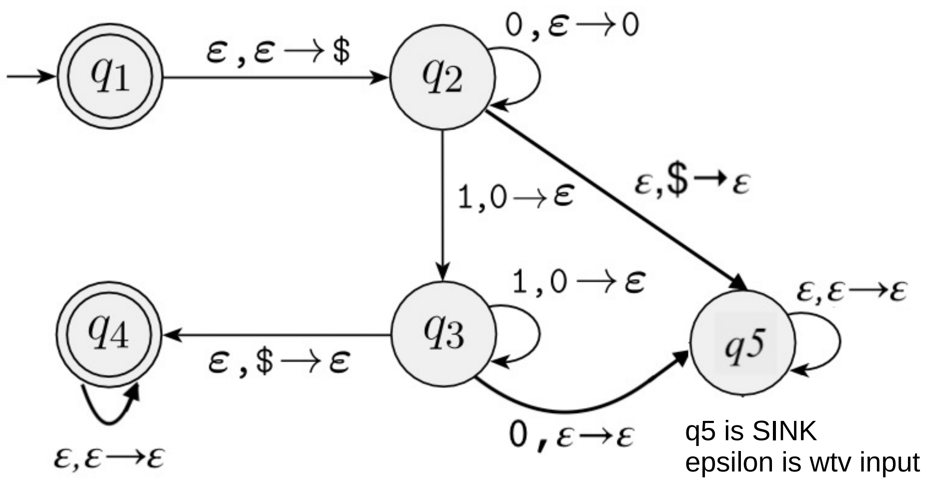
$$\delta(q,a,x), \delta(q,a,\varepsilon), \delta(q,\varepsilon,x), \text{ and } \delta(q,\varepsilon,\varepsilon)$$

is not $\emptyset$.

This means that, for every q

- Either we have only one outgoing arrow, that is labeled with $\varepsilon,\varepsilon \longrightarrow \ldots$
- Or it may have more outgoing arrows, but
  - for every input character a, it has exactly one arrow labeled either with $a,\varepsilon \longrightarrow \ldots$ or with $a,x \longrightarrow \ldots$, for some $x \in \Gamma$
  - for every stack character x, it has exactly one arrow labeled either with $\varepsilon,x \longrightarrow \ldots$ or with $a,x \longrightarrow \ldots$, for some $a \in \Sigma$

EXAMPLE: The PDA seen for $\{a^n b^n\}$ can be easily turned into a DPDA by adding the missing arrows:



q5 is SINK
epsilon is wtv input and wtv stack

Determinism vs Non-determinism in PDA

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

For regular languages, DFA = NFA; for context-free languages this is NOT the case (hence, PDAs are strictly more powerful than DPDAs)

Consider: $L = \{a^n b^n \mid n > 0\} \cup \{a^n b^{2n} \mid n > 0\}$

This is a C.F. language, as the following C.F. grammar testifies:

```
S ::= X | Y
X ::= ab | aXb
Y ::= abb | aYbb
```

By contradiction, assume that a DPDA M exists for L.

Since, for any n, we have that $a^n b^n \in L$, it must be that M arrives into some final states $q_1,\dots,q_k$ after reading such sequences.

Moreover, since $a^n b^{2n} \in L$ and M is deterministic, it must be that M from those precise states $q_1,\dots,q_k$ must read another n b's in input and accept again. because a^n b^2n = a^n b^n b^n

Let us now consider a new PDA M' with input $\{a,b,c\}$ obtained from M by reading c (instead of b) from states $q_1,\dots,q_k$

Then, $L(M') = \{a^n b^n c^n \mid n > 0\}$. Is this a C.F. language???

this is not CF, because it goes against the definition

proved in next slides