F. Languages and the Theory of Comp.

Topic 2. Context-free languages and pushdown automata

Han, Nguyen Dinh (han.nguyendinh@hust.edu.vn)

Faculty of Mathematics and Informatics
Hanoi University of Science and Technology

Outline

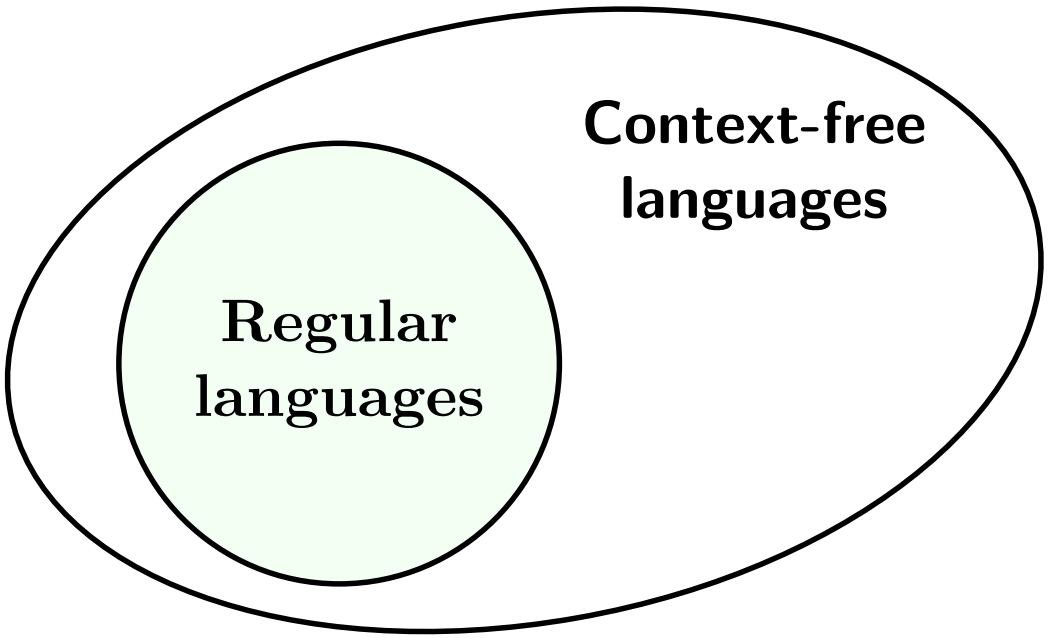
1

Context-free grammars

1.1

Basic definitions

We explore a more powerful method of describing languages and its applications. Our focus will be on **context-free grammars** that generate *context-free languages* as an extension of regular languages.



Context-free grammars play an important role in the specification and compilation of programming languages.

1.1. Basic definitions

A *context-free grammar* is a certain types of formal language generators. It consists of a collection of *substitution rules*, also called **productions**. Each rule appears as a line in the grammar, comprising a symbol and a string separated by an arrow. For example $S \to aMb$ is a rule, where the symbol is S and the string is aMb. The arrow $\to$ is read "can be". A context-free grammar operates as follows.

Start with the string consisting of the symbol S. Find a symbol in the current string that appears to the left of $\to$ in the one of its rules. Replace an occurrence of this symbol with the string that appears to the right of $\to$ in the same rule. Repeat this process until no such symbol can be found.

Example 2.1 We consider a context-free grammar that generates the string $000\sharp111$ after 5 steps as follows

$$\begin{aligned} S &\to 0S1 \quad (1)\\ S &\to B \quad (2)\\ B &\to \sharp \quad (3) \end{aligned}$$

These numbers are for reference

Step

1: Start with S, replace S by $0S1$ to receive $0S1$

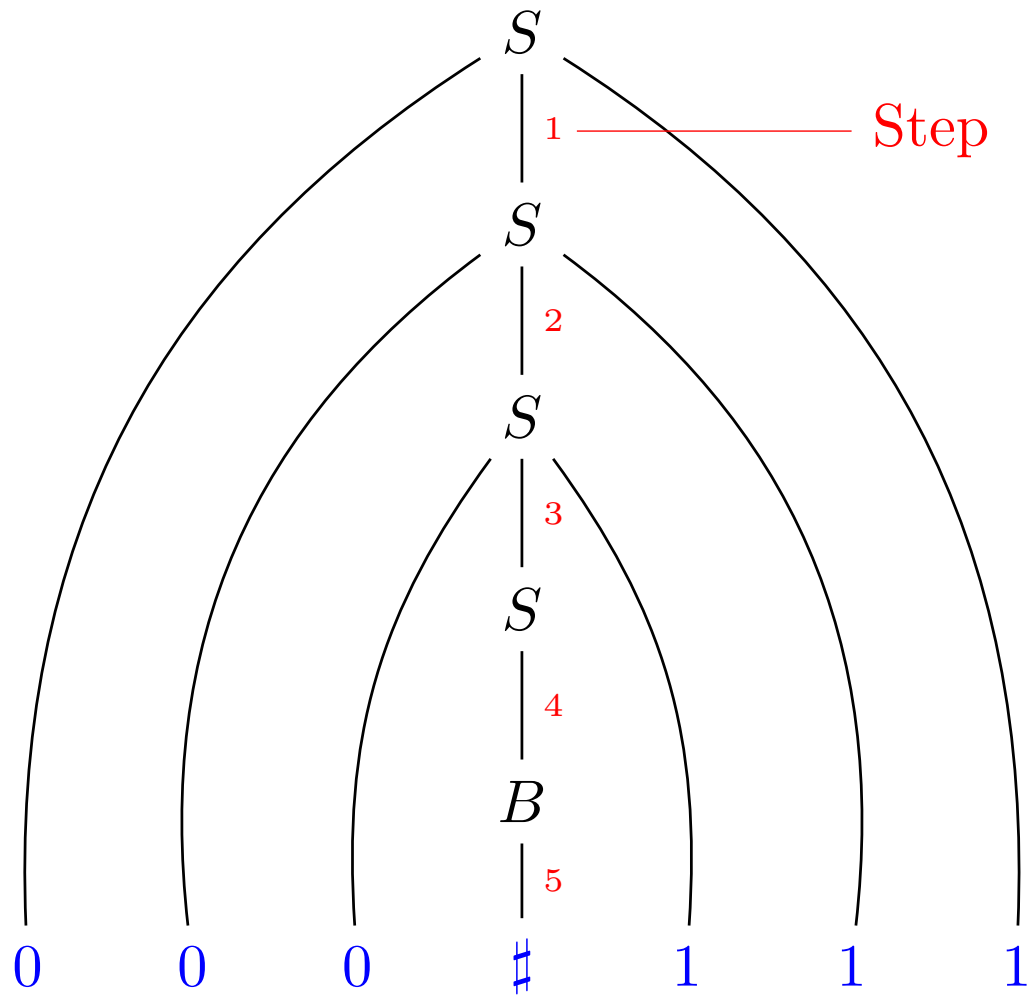
2: Start with $0S1$, replace S by $0S1$ to receive $00S11$

3: Start with $00S11$, replace S by $0S1$ to receive $000S111$

4: Start with $000S111$, replace S by B to receive $000B111$

5: Start with $000B111$, replace B by $\sharp$ to receive $000\sharp111$.

The sequence of substitutions to obtain a string is called a **derivation**. We often use **a parse tree** to represent pictorially a derivation. Here is the parse tree of Example 2.1 for 000♯111.



Definition 2.1 A **context-free grammar** G 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 is the *start variable*.

NOTES $\rightsquigarrow$ The members of V are sometimes called *nonterminals*. By convention, the start variable is the variable of the left-hand side of the first rule.

Definition 2.2 Let $G = (V, \Sigma, R, S)$ be a context-free grammar. Then

1. For any $A \in V$ and $u \in (V \cup \Sigma)^*$, we write $A \to_G u$ whenever $(A, u) \in R$.
2. For any $u, v \in (V \cup \Sigma)^*$, we write $u \Rightarrow_G v$ if and only if there are strings $x, y \in (V \cup \Sigma)^*$ and $A \in V$ such that $u = xAy, v = x\omega y$ and $A \to_G \omega$. The relation $\Rightarrow_G^*$ is the *reflexive, transitive closure* of $\Rightarrow_G$.
3. $L(G) = \{\omega \in \Sigma^* : S \Rightarrow_G^* \omega\}$ is the *language generated* by G. A language L is said to be a **context-free language** if $L = L(G)$ for some context-free grammar G.

Now, a derivation in G of ω_n from ω_0 can be written in the form

$$\omega_0 \Rightarrow_G \omega_1 \Rightarrow_G \cdots \Rightarrow_G \omega_n$$

Here, $\omega_0, \omega_1, \ldots, \omega_n$ may be any strings in $(V \cup \Sigma)^*$, and n, the length of the derivation, may be any natural number, including zero. We also say that the derivation has n steps.

NOTES ⇝ When the grammar to which we refer is obvious, we write $A \to \omega, u \Rightarrow v$ and $u \Rightarrow^* v$ instead of $A \to_G \omega, u \Rightarrow_G v$ and $u \Rightarrow_G^* v$. Furthermore $u \Rightarrow v$ is read "u **yields** v", and $u \Rightarrow^* v$ is read "u **derives** v".

Example 2.2 We consider the context-free grammar $G = (V, \Sigma, R, S)$, where $V = \{S\}$, $\Sigma = \{a, b\}$ and R consists of the rules $S \to aSb$ and $S \to \varepsilon$. A possible derivation is

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$$

Here the first three steps used the rule $S \to aSb$, and the last used the rule $S \to \varepsilon$. The language generated by G is

$$L(G) = \{a^n b^n : n \geq 0\}.$$

Note that this language is context-free, but it is not regular.

Example 2.3 We consider the context-free grammar $G = (V, \Sigma, R, S)$, where $V = \{S\}$, $\Sigma = \{a, b\}$ and R consists of the rules $S \to aSb$ and $S \to \varepsilon$. A possible derivation is

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$$

Here the first three steps used the rule $S \to aSb$, and the last used the rule $S \to \varepsilon$. The language generated by G is

$$L(G) = \{a^n b^n : n \geq 0\}.$$

Note that this language is context-free, but it is not regular.

Problem 2.1 Let $G = (V, \Sigma, R, S)$ be a context-free grammar, where $V = \{S, A\}$, $\Sigma = \{a, b\}$ and R consists of the rules $S \to aAa, S \to bAb, S \to \varepsilon$ and $A \to SS$.

(a) Give a derivation of the string $baabbb$ in G.

(b) Identify $L(G)$.

Problem 2.2 Design a context-free grammar that generates assignment statements of a programming language described in the form $id := expression$. Where id stands for any identifier, $expression$ is an arithmetic expression involving $+$ and $*$. Example of such statements are $id := id + id$ and $id := (id * id) + id$, but not $id := *id + (\,$ or $id := + * id$.

Your first assignment

Students work individually to carry out the following tasks:

1. Design a context-free grammar that generates a simple Python-like programming language. Your grammar can produce the following program:

   def add(x, y): z = x + y return z end
   if 10 > 5: a = 1 else: a = 2 end

2. Give a derivation of the above program

3. Report (in Word or Latex) your progress and results

NOTE ⇝ In the report you should present your design and experimental proof in detail. Please convert your report to PDF when you submit it

1.2

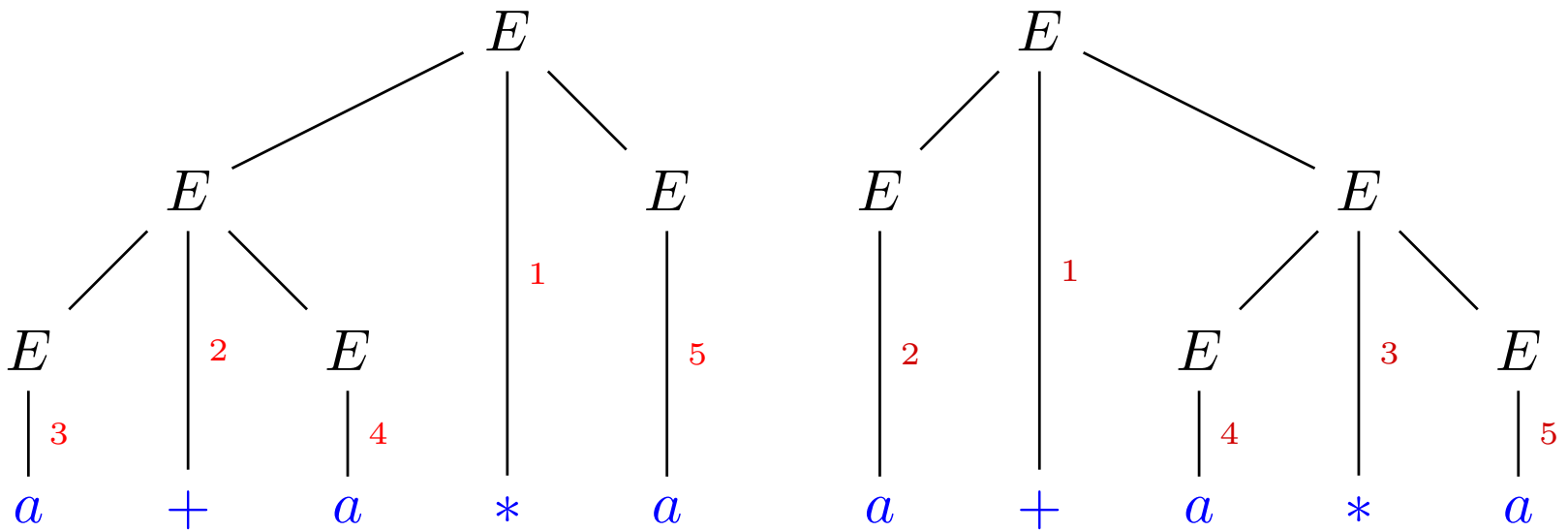
Ambiguity

Definition 2.3 Let $G = (V, \Sigma, R, S)$ be a context-free grammar. Then

1. A derivation of a string ω in G is a **leftmost derivation** (we write $S \overset{L}{\Rightarrow}^* \omega$) if at every step the leftmost remaining variable is the one replaced.
2. A string ω is derived **ambiguously** in context-free grammar G if it has two or more different leftmost derivations. Grammar G is **ambiguous** if it generates some string ambiguously.

NOTE $\rightsquigarrow$ Similarly, if the rightmost variable is replaced, we will have a **rightmost derivation** ($S \overset{R}{\Rightarrow}^* \omega$).

Example 2.4 Let $G = (V, \Sigma, R, E)$ be a context-free grammar, where $V = \{E\}$, $\Sigma = \{a, +, *\}$ and R consists of the rules $E \to E+E$, $E \to E*E$ and $E \to a$. This grammar generates the string $a + a * a$ ambiguously as follows.



Languages generated only by ambiguous context-free grammars are called **inherently ambiguous**. Programming languages, *however are never inherently ambiguous.*

Theorem 2.1 Let $G = (V, \Sigma, R, S)$ be a context-free grammar, and let $A \in V$ and $\omega \in (V \cup \Sigma)^*$. Then the following statements are equivalent.

1. $A \Rightarrow^* \omega$.
2. There is a parse tree with root A and yield ω.
3. There is a leftmost derivation $A \overset{L}{\Rightarrow}^* \omega$.
4. There is a rightmost derivation $A \overset{R}{\Rightarrow}^* \omega$.

1.3

Chomsky normal form

Chomsky normal form is useful in giving algorithms for working with context-free grammars. So we have

Definition 2.4 Let $G = (V, \Sigma, R, S)$ be a context-free grammar. Then G is in **Chomsky normal form** if every rule is of the form

$$A \to BC$$
$$A \to a$$

where a is any terminal and A, B and C are any variables - except that B and C may not be the start variable. In addition we permit the rule $S \to \varepsilon$, where S is the start variable.

Example 2.5 We consider two context-free grammars given below. By Definition 2.4, the grammar on the right-hand side is in Chomsky normal form.

$$
\begin{aligned}
S &\to ASA \,|\, aB \\
A &\to B \,|\, S \\
B &\to b \,|\, \varepsilon
\end{aligned}
\qquad
\begin{aligned}
S_0 &\to AA_1 \,|\, UB \,|\, a \,|\, SA \,|\, AS \\
S &\to AA_1 \,|\, UB \,|\, a \,|\, SA \,|\, AS \\
A &\to b \,|\, AA_1 \,|\, UB \,|\, a \,|\, SA \,|\, AS \\
A_1 &\to SA \\
U &\to a \\
B &\to b
\end{aligned}
$$

Theorem 2.2 For any context-free grammar G there is a context-free grammar G' in Chomsky normal form such that $L(G') = L(G) - (\Sigma \cup \{\varepsilon\})$. Furthermore, the construction of G' can be carried out in time polynomial in the size of G.

* *

*

Exercise: Make use of the procedure given by Theorem 2.9 in our textbook to convert the left-hand side context-free grammar of Example 2.5 into Chomsky normal form.

1.4

Non-context-free languages

Given a context-free grammar $G = (V, \Sigma, R, S)$, we define the **fanout** of G, denoted by $\phi(G)$, is the largest number of symbols on the right-hand side of any rule in R. The following theorem, a more powerful but also called the *pumping theorem*, enables us to show that certain languages are not context-free.

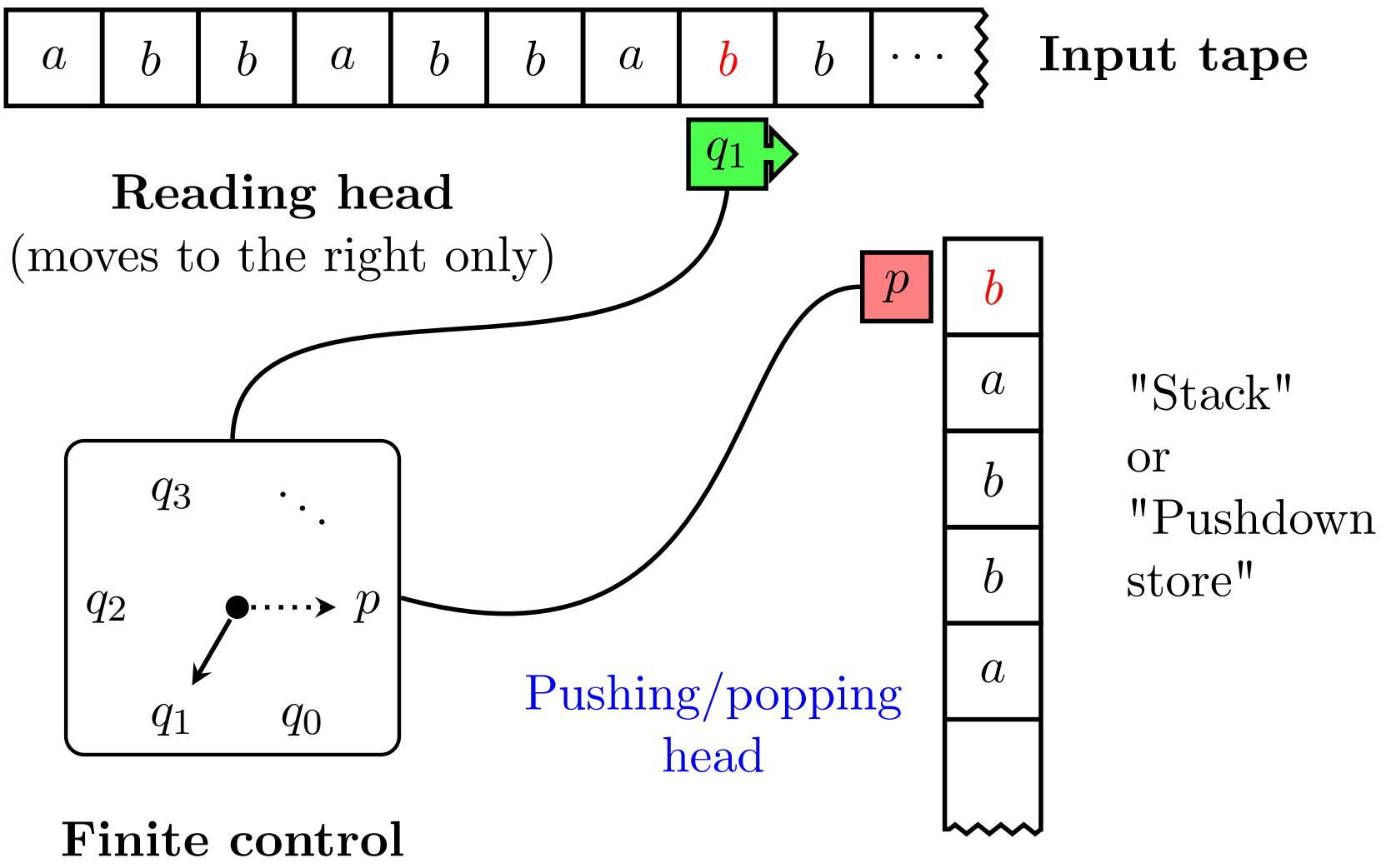
Theorem 2.4 Let $G = (V, \Sigma, R, S)$ be a context-free grammar. Then any string $\omega \in L(G)$ of length greater than $\phi(G)^{|V|}$ can be written as $\omega = uvxyz$ in such a way that either v or y is nonempty and uv^nxy^nz is in $L(G)$ for every $n \geq 0$.

Example 2.6 Show that the language $L = \{a^n b^n c^n : n \geq 0\}$ over the alphabet $\Sigma = \{a, b, c\}$ is not context-free.

Proof: Suppose that L is context-free. This implies $L = L(G)$ for some $G = (V, \Sigma, R, S)$. Let $n > \dfrac{\phi(G)^{|V|}}{3}$. Then we have $\omega = a^n b^n c^n \in L(G)$, $\omega = uvxyz$, $vy \neq \varepsilon$ and $uv^n xy^n z \in L(G), \forall n \geq 0$. There are two cases, both leading to a contradiction. If vy contains occurrences of all three symbols a, b, c, then at least one of v, y must contain occurrences of at least two of them. But then uv^2xy^2z contains two occurrences out of their correct order - a b before an a, or a c before an a or b. If vy contains occurrences of some but not all of the three symbols, then uv^2xy^2z has unequal numbers of a's, b's, and c's. □

2

Pushdown automata


a b b a b b a b b ···
Input tape
q1
Reading head
(moves to the right only)
p
b
a
b
b
a
"Stack"
or
"Pushdown
store"
q3
q2
p
q1
q0
Pushing/popping
head
Finite control

Definition 2.5 A *pushdown automaton* is a 6−tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where

1. Q is a finite set of *states*,
2. Σ is the *input alphabet*,
3. Γ is the *stack alphabet*,
4. $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \longrightarrow \mathcal{P}(Q \times (\Gamma \cup \{\varepsilon\}))$ is the *transition function*,
5. q_0 is the *start state*, and
6. $F \subseteq Q$ is the *set of accept states*.

Definition 2.6 Let $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a pushdown automaton. Then

1) A pair $((p, a, \beta), (q, \gamma)) \in \big(Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\})\big)\big(Q \times (\Gamma \cup \{\varepsilon\})\big)$ is called a **transition** of M. To **push** a symbol is to add it to the top of the stack; to **pop** a symbol is to remove it from the top of the stack. For example, the transaction $((p, u, \varepsilon), (q, a))$ pushes a, while $((p, u, a), (q, \varepsilon))$ pops a.

2) A **configuration** of M is a member of $Q \times \Sigma^* \times \Gamma^*$: the first component is the state of the machine, the second is the portion of the input yet to be read, and the third is the contents of the pushdown store, read *top-down*.

Definition 2.6 (cont.) Let $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a pushdown automaton. Then

3) If (p, x, α) and (q, y, ξ) are configurations of M, we say that (p, x, α) **yields in one step** (q, y, ξ) (notation: $(p, x, \alpha) \vdash_M (q, y, \xi)$) if there is a transition $((p, a, \beta), (q, \gamma))$ such that $x = ay, \alpha = \beta\eta$ and $\xi = \gamma\eta$ for some $\eta \in \Gamma^*$. We denote the *reflexive, transitive closure* of $\vdash_M$ by $\vdash_M^*$.

4) A string $\omega \in \Sigma^*$ is said to be **accepted** by M if and only if $(q_0, \omega, \varepsilon) \vdash_M^* (p, \varepsilon, \varepsilon)$ for some $p \in F$.

5) **The language accepted by** M, $L(M)$, is the set of all strings accepted by M.

Example 2.7 Let M be the pushdown automaton $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where $Q = \{q_0, q_1\}$, $\Sigma = \{a, b, c\}$, $\Gamma = \{a, b\}$, $F = \{q_1\}$, and δ contains the following five transitions.

$$
\begin{aligned}
&(1) \quad ((q_0, a, \varepsilon), (q_0, a)) \\
&(2) \quad ((q_0, b, \varepsilon), (q_0, b)) \\
&(3) \quad ((q_0, c, \varepsilon), (q_1, \varepsilon)) \\
&(4) \quad ((q_1, a, a), (q_1, \varepsilon)) \\
&(5) \quad ((q_1, b, b), (q_1, \varepsilon))
\end{aligned}
$$

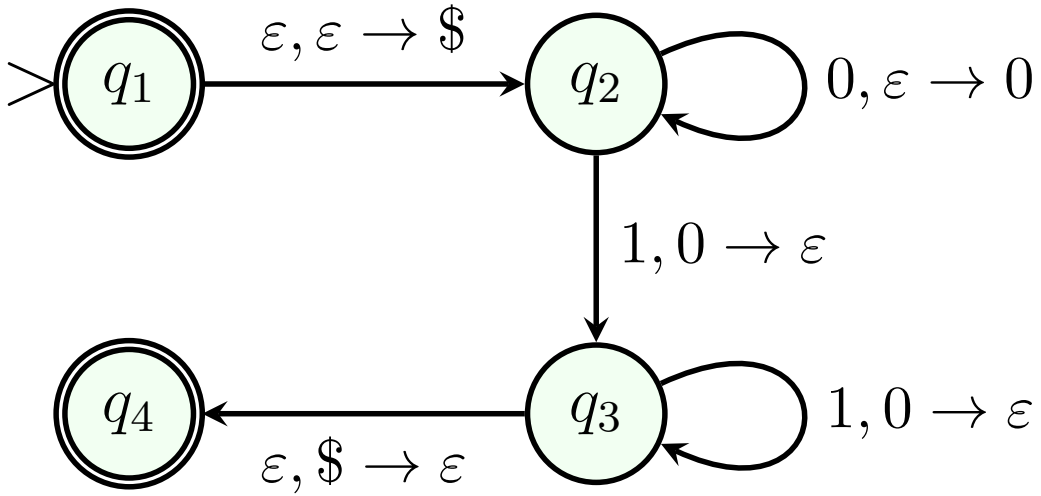
$L(M) = \{\omega c \omega^R : \omega \in \{a, b\}^*\}$. Now, let check how M accepts some strings in $L(M)$ such as $abbcbba$ and $ababcbaba$.

Example 2.7 (cont.) The sequence of transitions for $abbcbba$ is as follows.

State	Unread input	Stack	Transition used
q_0	$abbcbba$	ε	–
q_0	$bbcbba$	a	1
q_0	$bcbba$	ba	2
q_0	$cbba$	bba	2
q_1	bba	bba	3
q_1	ba	ba	5
q_1	a	a	5
q_1	ε	ε	4

We can also use a state diagram to describe a pushdown automaton (PDA). Here, we write $a, b \to c$ to signify 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.

The state diagram for the PDA that accepts $\{0^n 1^n : n \geq 0\}$



Both pushdown automata and context-free grammars are capable of describing the class of context-free languages. So they lay the foundation for the study of syntax analyzers for programming languages. The following theorem shows that context-free grammars and pushdown automata are equivalent in power. In addition, the proof of the theorem provides procedures to convert any context-free grammar into a pushdown automaton that accepts its language and vice versa.

Theorem 2.6 *A language is context-free if and only if some pushdown automaton accepts it.*

Problem 2.3 Construct pushdown automata that accept each of the following.

(a) The language $\{a^m b^n : m \leq n \leq 2m\}$.

(b) The language $\{\omega \in \{a, b\}^* : \omega$ has the same number of a's and b's$\}$.

(c) The language $\{\omega\omega^R : \omega \in \{a, b\}^*\}$.

(d) The language generated by the grammar $G = (V, \Sigma, R, S)$, where $V = \{S, A\}$, $\Sigma = \{a, b\}$ and R consists of the rules $S \to aAa, S \to bAb, S \to \varepsilon$ and $A \to SS$.

Your second assignment

Students work individually to carry out the following tasks:

1. Enrich the simple Python-like programming language that you have designed in your first assignment. Your language should support more operations (e.g. logic and comparison) and statements (e.g. conditional and loop statements)
2. Design and implement (in Python) a pushdown automaton that accepts your programming language
3. Report (in Word or Latex) your progress and results

NOTE ⇝ In the report you should present your design and implementation in detail. Please convert your report to PDF when you submit it

Thank you very much for your attention!