Automata and formal languages - 2IRR90

Contents

1. Regular Languages

1.1. Finite Automata

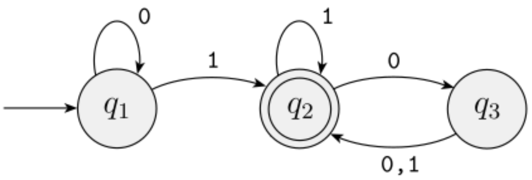

Figure 1: A finite automaton called M_1 that has three states

Figure 1 is called the **state diagram** of M_1. It has three **states**, labled q_1, q_2, q_3. The **start state**, q_1, is indicated by the arrow pointing at it from nowhere. The **accept state**, q_2, is the one with a double circle. The arrow going from one state to another are called **transitions**.

Definition 1.1.1 (Finite Automaton): A **finite automaton** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where
1. Q is a finite set called the **states**
2. Σ is a finite set called the **alphabet**,
3. $\delta : Q \times \Sigma \to Q$ is the **transition function**,
4. $q_0 \in Q$ is the **start state**, and
5. $F \subseteq Q$ is the **set of accept states**.

Example: We can describe M_1 in Figure 1 formally by writing $M_1 = (Q, \Sigma, \delta, q_1, F)$ where
1. $Q = \{q_1, q_2, q_3\}$
2. $\Sigma = \{0, 1\}$
3. δ is described as

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

4. q_1 is the start state, and
5. $F = \{q_2\}$

If A is the set of all strings tha machine M accepts, we say the A is the **language of machine M** and write $L(M) = A$. We say that **M recognizes A** or that **M accepts A**.

Formal Definition of Computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and let $w = w_1 w_2 \cdots w_n$ be a string where each $w_i \in \Sigma$. Then M **accepts** w if a sequence of states $r_0, r_1, \ldots, r_n \in Q$ exists with three conditions:

1. $r_0 = q_0$
2. $\delta(r_i, w_{i+1}) = r_{i+1}$ for $i = 0, \ldots, n-1$, and
3. $r_n \in F$

We say that M **recognizes language** A if $A = \{w \mid M \text{ accepts } w\}$.

Definition 1.1.2 (Regular language): A language is called a **regular language** if some finite automaton recognizes it.

Definition 1.1.3 (The regular operations): Le A and B be languages. We define the regular operations **union, concatenation,** and **star** as follows:

- **Union:** $A \cup B = \{x \mid x \in A \vee x \in B\}$.
- **Concatenation:** $A \circ B = \{xy \mid x \in A \wedge y \in B\}$.
- **Star:** $A^* = \{x_1 x_2 \ldots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$

Theorem 1.1.1: The class of regular languages is closed under the union operation.

$$A_1, A_2 \text{ are regular languages} \Longrightarrow A_1 \cup A_2 \text{ regular language}$$

Proof: Let M_1 recognize A_1, where $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$, and
M_2 recognize A_2, where $M_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2)$.

Construct $M = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \cup A_2$

1. $Q = Q_1 \times Q_2$
2. $\Sigma = \Sigma_1 \cup \Sigma_2$
3. $\delta : Q \times \Sigma \to Q$ is defined by

$$\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$$

4. $q_0 = (q_1, q_2)$
5. $F = \{(r_1, r_2) \mid r_1 \in F_1 \vee r_2 \in F_2\}$

By construction, M recognizes $A_1 \cup A_2$. □

1.2. Nondeterminism

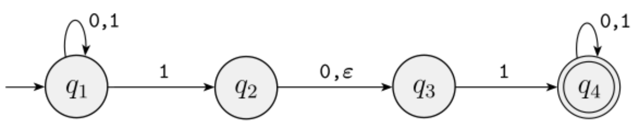

Figure 2: The nondeterministic finite automaton N_1

In a nondeterministic machine, several choices may exist for the next state at any point. Nondeterministic finite automata may have additional features.

- a state may have zero, one, or many exiting arrows for each alphabet symbol
- can have an arrow with the label ε.

Definition 1.2.1 (Nondeterministic Finite Automaton): A **nondeterministic finite automaton** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

1. Q is a finite set of states,
2. Σ is a finite alphabet
3. $\delta : Q \times \Sigma_\varepsilon \to \mathscr{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

Example: The formal definition of Figure 2 is $N_1 = (Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3, q_4\}$,
2. $\Sigma = \{0, 1\}$,
3. δ is described as

	0	1	2
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_3\}$	$\emptyset$

4. q_1 is the start state, and
5. $F = \{q_4\}$.

Formal Definition of Computation

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA and w a string over the alphabet Σ. Then we say that N **accepts** w if we can write w as $w = y_1 y_2 ... y_m$, where each $y_i \in \Sigma_\varepsilon$ and a sequence of states $r_0, r_1, ..., r_m \in Q$ exists with three conditions:

1. $r_0 = q_0$
2. $r_i \in \delta(r_{i-1}, y_i)$ for $i = 1, 2, ..., m$, and
3. $r_m \in F$.

Equivalence of NFA's and DFA's

Theorem 1.2.1: Every nondeterministic finite automaton has an equivalent deterministic finite automaton.

Corollary 1.2.1.1: A language is regular if and only if some nondeterministic finite automaton recognizes it.

Closure Under The Regular Operations

Theorem 1.2.2: The class of regular languages is closed under the concatenation operations.

$$A_1, A_2 \text{ regular languages} \Longrightarrow A_1 \circ A_2 \text{ regular language}$$

Proof: Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1, and
$N_2 = (Q_2, \Sigma, \delta, q_2, F_2)$ recognize A_2.

Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \circ A_2$.

1. $Q = Q_1 \cup Q_2$
2. $q_0 = q_1$
3. $F = F_2$
4. Define δ so that for any $q \in Q$ and $a \in \Sigma_\varepsilon$,

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q,a) & q \in F_1 \text{ and } a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_2\} & q \in F_1 \text{ and } a = \varepsilon \\ \delta_2(q,a) & q \in Q_2 \end{cases}$$

□

Theorem 1.2.3: The class of regular languages is closed under the star operation.

1.3. Regular Expressions

Definition 1.3.1 (Regular Expression): Say that R is a **regular expression** if R is

1. a for some $a \in \Sigma$,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is a regular expression.

Example: Let $\Sigma = \{0, 1\}$.

1. $0^*10^* = \{w \mid w \text{ contains a single } 1\}$.
2. $\Sigma^*1\Sigma^* = \{w \mid w \text{ has at least one } 1\}$.
3. $1^*(01^+)^* = \{w \mid \text{every } 0 \text{ in } w \text{ is followed by at least one } 1\}$.
4. $(\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is even}\}$.
5. $(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of } 3\}$.

1.4. Nonregular Languages

Let's take the language $B = \{0^n1^n \mid n \geq 0\}$. There is no NFA that recognizes this language, since the machine would need to remember unlimited number of 0's. We say that B is a nonregular language.

Theorem 1.4.1 (Pumping Lemma): If A is a regular language, then there is a number p (the pumping length) where if s is any string in A of length at least p, tthen s may be divided into three pieces, $s = xyz$, satisfying the following conditions:
1. for each $i \geq 0, xy^i z \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

Example: Let B be the language $\{0^n 1^n \mid n \geq 0\}$. We use the pumping lemma to prove that B is not regular. The proof is by contradiction.

Assume that B is regular. Let p be the pumping length given by the pumping lemma. Choose $s = 0^p 1^p$. Since $s \in B \land |s| > p$, the pumping lemma guarantees that s can be divided into three pieces, $s = xyz$, where $|y| > 0$, $|xy| \leq p$, and $xy^i z \in B$ for all $i \geq 0$. We now consider three cases.

1. The string y consists of only 0's. Then xy^2z has more 0's than 1's, so $xy^2z \notin B$.
2. The string y consists of only 1's. Then xy^2z has more 1's than 0's, so $xy^2z \notin B$.
3. The string y consists of both 0's and 1's. Then xy^2z may have the same number of 0's and 1's, but they will be out of order. Hence $xy^2z \notin B$

All three cases lead to a contradiction, thus B is not regular.

2. Context-Free Languages

2.1. Context-Free Grammars

$$A \to 0A1$$
$$A \to B$$
$$B \to \#$$

Figure 3: Context free grammar G_1

A grammar consits 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. The symbol is called a **variable**. The string consists of variables and other symbols called **terminals**. One variable is designated as the **start variable**.

A grammar describes a language by generating each string of the language in the following way:
1. Start with the string consisting of the start variable.
2. Replace each variable in the string by the right-hand side of one of its rules.
3. Repeat step 2 until no variables remain.

Example: Grammar G_1 in Figure 3 generates the string $000\#111$.The sequence of substitutions to obtain a string is called a **derivation**. A derivation $000\#111$ in grammar G_1 is

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111.$$

Figure 3 generates the language $L(G_1) = \{0^n \# 1^n \mid n \geq 0\}$.

Any language that can be generated by some context-free grammar is called a **context-free language** (CFL). We abbreviate several rules with the same left-hand side into a single line using the symbol $|$.

$$\begin{aligned}
\langle\text{SENTENCE}\rangle &\to \langle\text{NOUN-PHRASE}\rangle\langle\text{VERB-PHRASE}\rangle \\
\langle\text{NOUN-PHRASE}\rangle &\to \langle\text{CMPLX-NOUN}\rangle \mid \langle\text{CMPLX-NOUN}\rangle\langle\text{PREP-PHRASE}\rangle \\
\langle\text{VERB-PHRASE}\rangle &\to \langle\text{CMPLX-VERB}\rangle \mid \langle\text{CMPLX-VERB}\rangle\langle\text{PREP-PHRASE}\rangle \\
\langle\text{PREP-PHRASE}\rangle &\to \langle\text{PREP}\rangle\langle\text{NOUN-PHRASE}\rangle \\
\langle\text{CMPLX-NOUN}\rangle &\to \langle\text{ARTICLE}\rangle\langle\text{NOUN}\rangle \\
\langle\text{CMPLX-VERB}\rangle &\to \langle\text{VERB}\rangle \mid \langle\text{VERB}\rangle\langle\text{NOUN-PHRASE}\rangle \\
\langle\text{ARTICLE}\rangle &\to a \mid \text{the} \\
\langle\text{NOUN}\rangle &\to \text{boy} \mid \text{girl} \mid \text{flower} \\
\langle\text{VERB}\rangle &\to \text{touches} \mid \text{sees} \mid \text{likes} \\
\langle\text{PREP}\rangle &\to \text{with}
\end{aligned}$$

Figure 4: G_2 is another CFG

Grammar G_2 in Figure 4 has 10 variables (the capitalized grammatical terms inside brackets); 27 terminals (the standard English alphabet plus a space character); and 18 rules. Strings in $L(G_2)$ include:
- *a boy sees*
- *the boy sees a flower*
- *a girl with a flower like the boy*

Example: *a boy sees* is derived in the following way:

$$\begin{aligned}
\langle\text{SENTENCE}\rangle &\Rightarrow \langle\text{NOUN-PHRASE}\rangle\langle\text{VERB-PHRASE}\rangle \\
&\Rightarrow \langle\text{CMPLX-NOUN}\rangle\langle\text{VERB-PHRASE}\rangle \\
&\Rightarrow \langle\text{ARTICLE}\rangle\langle\text{NOUN}\rangle\langle\text{VERB-PHRASE}\rangle \\
&\Rightarrow a\langle\text{NOUN}\rangle\langle\text{VERB-PHRASE}\rangle \\
&\Rightarrow \text{a boy } \langle\text{VERB-PHRASE}\rangle \\
&\Rightarrow \text{a boy } \langle\text{CMPLX-VERB}\rangle \\
&\Rightarrow \text{a boy } \langle\text{VERB}\rangle \\
&\Rightarrow \text{a boy sees}
\end{aligned}$$

Definition 2.1.1 (Context-free grammer): A **context-free grammer** is a 4 -tuple (V, Σ, R, S), where
1. V is a fintie set called the **variables**,
2. Σ is a finite set, disjoint from V, called the **terminals**,
3. R is a finite set of **rules**, wirht each rule being a variable and a string of vairables and terminals, and
4. $S \in V$ is the start variable.

If u, v, and w are strings of variables and terminals, and $A \to w$ is a rule, we say that uAv **yields** uwv, written $uAv \Rightarrow uwv$, Say that u **derives** v, written $u \overset{*}{\Rightarrow} v$, if $u = v$ or if a sequence $u_1, u_2, ..., u_k$ exists for $k \geq 0$ and

$$u \Rightarrow u_1 \Rightarrow u_1 \cdots \Rightarrow u_k \Rightarrow v.$$

The **langauge of the grammer** is $\left\{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\right\}$.

Example: In grammar G_2 in Figure 4, $V = \{A, B\}, \Sigma = \{0, 1, \#\}, S = A$, and R is the collection of the three rules appearing in Figure 4

Example: Conisder the grammar $G_3 = (\{S\}, \{(,)\}, R, S)$. The set of rules, R, is

$$S \to (S) \mid SS \mid \varepsilon.$$

Then the language $L(G_3)$ is the set of strings of properly nested parentheses.

Note: the right-hand side of a rule may be the empty string.

Example: Consider grammar $G_4 = (V, \Sigma, R, \langle\text{EXPR}\rangle)$. Where $V = \{\langle\text{EXPR}\rangle, \langle\text{TERM}\rangle, \langle\text{FACTOR}\rangle\}$ and $\Sigma = \{a, +, \times, (,)\}$ The rules are

$$\begin{aligned} \langle\text{EXPR}\rangle &\to \langle\text{EXPR}\rangle + \langle\text{TERM}\rangle \mid \langle\text{TERM}\rangle \\ \langle\text{TERM}\rangle &\to \langle\text{TERM}\rangle \times \langle\text{FACTOR}\rangle \mid \langle\text{FACTOR}\rangle \\ \langle\text{FACTOR}\rangle &\to (\langle\text{EXPR}\rangle) \mid a \end{aligned}$$

The two strings $a + a \times a$ and $(a + a) \times a$ can be generated with grammar G_4. The parse trees are shown in the following figure:

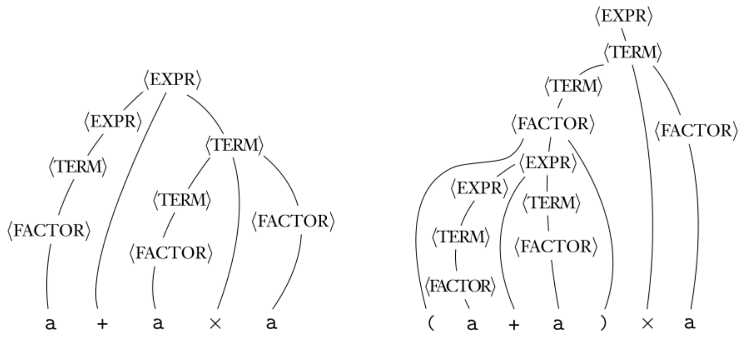

Figure 5: Parse trees for the strings $a + a \times a$ and $(a + a) \times a$

Proposition 2.1.1 (Constructing a CFG from a DFA):
1. Make a vairable R_i for each state q_i
2. Add the rule $R_i \to aR_j$ if $\delta(q_i, a) = q_j$
3. Add the rule $R_i \to \varepsilon$ if q_i is an accept state.
4. Make R_0 the start variable, where q_0 is the start state in the DFA.

If a grammar generates the same string in several different ways, we say that the string is derived **ambiguously** in that grammar. Such a grammar is called an **ambiguous grammer**.

Definition 2.1.2: A string w 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.

Example: Consider grammar G_5:

$$\langle\text{EXPR}\rangle \to \langle\text{EXPR}\rangle + \langle\text{EXPR}\rangle \mid \langle\text{EXPR}\rangle \times \langle\text{EXPR}\rangle \mid (\langle\text{EXPR}\rangle) \mid a$$

This grammar gernerates the string $a + a \times a$ ambiguously. the following figure shows the tow different parse trees.

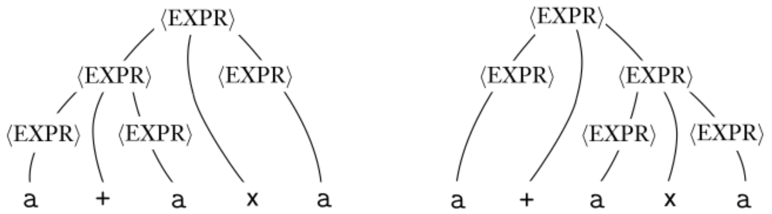

Figure 6: The two parse trees for the string $a + a \times a$ in grammar G_5

This grammar doesn't capture the usual precedence relations and so may group the $+$ before the $\times$ or vice versa. In contrast, grammar G_4 generates exactly the same language, but every generated string has a unique parse tree. Hence, G_4 is unambiguous, whereas G_5 is ambiguous.

Definition 2.1.3: A context-free grammar 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.

Theorem 2.1.1: Any context-free language is generated by a context-free grammar in Chomsky normal form.

Algorithm 2.1.1: First, we add a new start variable S_0 and the rule $S_0 \to S$, where S was the original start variable. This cahnge guarantees that the start variable doesn't occur on the right-hand side of a rule.

Second, we take care of all ε-rules. We remove an ε-rule $A \to \varepsilon$, where A is not the start variable. Then for each occurance of an A on the right-hand side of a rule, we add a new rule with that occurence deleted. In other words, if $R \to uAv$ is a rule in which u, v are strings of variables and terminals, we add rule $R \to uv$. We do so for each occurrence of A, so the rule $R \to uAvAw$ causes us to add $R \to uvAw, R \to uAvw, R \to uvw$. If we have the rule $R \to A$, we add $R \to \varepsilon$, unless we had previously removed the rule $R \to \varepsilon$.

Third, we handle all unit rules. We remove a unit rule $A \to B$. Then, whenever a rule $B \to u$ appears, we add the rule $A \to u$ unless this was a unit rule previously removed.

Finally, we convert all remaining rules into the proper form. We replace each rule $A \to u_1 u_2 \cdots u_k$, where $k \geq 3$ and each u_i is a variable or a terminal symbol, with the rules $A \to u_1 A_1, A_1 \to u_2 A_2, ..., A_{k-2} \to u_{k-1} u_k$. The A_i's are new variables. We replace any terminal u_i in the preceding rules with the new variable U_i and add the rule $U_i \to u_i$

Example: Let G_6 be the following CFG and convert it to Chomsky normal form. Rules in bold have just been added, while rules in gray have just been removed.

1. left-hand side: original CFG G_6.

$$\begin{aligned} S &\to ASA \mid aB \\ A &\to B \mid S \\ B &\to b \mid \varepsilon \end{aligned} \qquad \begin{aligned} \boldsymbol{S_0} &\boldsymbol{\to S} \\ S &\to ASA \mid aB \\ A &\to B \mid S \\ B &\to b \mid \varepsilon \end{aligned}$$

2. left-handside: remove $B \to \varepsilon$, right-hand side: remove $A \to \varepsilon$.

$$\begin{aligned} S_0 &\to S \\ S &\to ASA \mid aB \mid \boldsymbol{a} \\ A &\to B \mid S \mid \boldsymbol{\varepsilon} \\ B &\to b \mid \textcolor{gray}{\varepsilon} \end{aligned} \qquad \begin{aligned} S_0 &\to S \\ S &\to ASA \mid aB \mid a \mid \boldsymbol{SA} \mid \boldsymbol{AS} \mid \boldsymbol{S} \\ A &\to B \mid S \mid \textcolor{gray}{\varepsilon} \\ B &\to b \end{aligned}$$

3. a. left-hand side: remove unit rules $S \to S$, right-hand side: remove $S_0 \to S$

$$\begin{aligned} S_0 &\to S \\ S &\to ASA \mid aB \mid a \mid SA \mid AS \mid \textcolor{gray}{S} \\ A &\to B \mid S \\ B &\to b \end{aligned} \qquad \begin{aligned} S_0 &\to \textcolor{gray}{S} \mid \boldsymbol{ASA} \mid \boldsymbol{aB} \mid \boldsymbol{a} \mid \boldsymbol{SA} \mid \boldsymbol{AS} \\ S &\to ASA \mid aB \mid a \mid SA \mid AS \\ A &\to B \mid S \\ B &\to b \end{aligned}$$

   b. Remove unit rules $A \to B$ and $A \to S$

$$\begin{aligned} S_0 &\to S \\ S &\to ASA \mid aB \mid a \mid SA \mid AS \\ A &\to \textcolor{gray}{B} \mid S \mid \boldsymbol{b} \\ B &\to b \end{aligned} \qquad \begin{aligned} S_0 &\to \textcolor{gray}{S} \mid \boldsymbol{ASA} \mid \boldsymbol{aB} \mid \boldsymbol{a} \mid \boldsymbol{SA} \mid \boldsymbol{AS} \\ S &\to ASA \mid aB \mid a \mid SA \mid AS \\ A &\to \textcolor{gray}{S} \mid b \mid \boldsymbol{ASA} \mid \boldsymbol{aB} \mid \boldsymbol{a} \mid \boldsymbol{SA} \mid \boldsymbol{AS} \\ B &\to b \end{aligned}$$

4. Convert the remaining rules into the propert form by adding additional variables and rules

$$\begin{aligned} S_0 &\to AA_1 \mid UB \mid a \mid SA \mid AS \\ S &\to AA_1 \mid UB \mid a \mid SA \mid AS \\ A &\to b \mid AA_1 \mid UB \mid a \mid SA \mid AS \\ A_1 &\to SA \\ U &\to a \\ B &\to b \end{aligned}$$

2.2. Pushdown Automata

These automata are like nondeterministic finite automata but have an extra component called a **stack**.

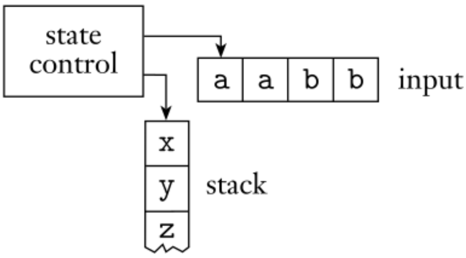

Figure 7: Schematic of a pushdown automaton

Definition 2.2.1 (Pushdown automaton): A **pushdown automaton** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where
1. Q is a finite set of states,
2. Σ is a finite set - **input alphabet**,
3. Γ is a finite set - **stack alphabet**,
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathscr{P}(Q \times \Gamma_\varepsilon)$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.

A pushdown automaton $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ computes as follows. It **accepts** input w if $w = w_1 w_2 \cdots w_m$, where $m_i \in \Sigma_\varepsilon$ and sequences of states $r_0, r_1, ..., r_m \in Q$ and strings $s_0, s_1, ..., s_m \in \Gamma^*$ exist that satisfy the following three conditions.
1. $r_0 = q_0, s_0 = \varepsilon$
2. For $i = 0, ..., m - 1$, we have $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$, where

$s_i = at$ and $s_{i+1} = bt$ for some $a, b \in \Gamma_\varepsilon, t \in \Gamma^*$.
3. $r_m \in F$

The strings s_i represent the sequence of stack contents that M has on the accepting branch of the computation.

Example: The following is the formal description of the PDA that recognizes the language $\{0^n 1^n \mid n \geq 0\}$. Let $M_1 = (Q, \Sigma, \Gamma, \delta, q_1, F)$, where
1. $Q = \{q_1, q_2, q_3, q_4\}$,
2. $\Sigma = \{0, 1\}$,
3. $\Gamma = \{0, \$\}$,
4. $F = \{q_1, q_4\}$, and
5. δ is given by the following table

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									

We can use a state diagram to describe a PDA as in Figure 8. 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. Any of a, b, and c may be ε. If a is ε, the machine may make this transition without reading any symbol from the input. If b is ε, the machine may make this transition without reading and popping any symbol from the stack. If c is ε, the machine does not write any symbol on the stack when going along this transition.

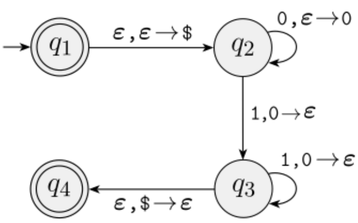

Figure 8: State diagram for the PDA M_1 that recognizes $\{0^n 1^n \mid n \geq 0\}$

This PDA uses the symbol \$ to check whether the stack is empty.

Question: page 115

Theorem 2.2.1: A language is context-free if and only if some pushdown automaton recognizes it.

Proof: View in book: page 117 □

2.3. Non-context-free languages

Theorem 2.3.1 (Pumping lemma for context-free languages): If A is a context free language, then there is a number p (the pumping length) where, if s is any string in A of length at least p, then s may be divided into five pieces, $s = uvxyz$, satisfying the following conditions:
1. for each $i \geq 0$, $uv^i xy^i z \in A$,
2. $|vy| > 0$, and
3. $| \ vxy \ | \leq p$

Example: USe the pumping lemma to show that the lagnauge $B = \{a^n b^n c^n \mid n \geq 0\}$ is not context free.

We assume that B is a CFL and obtain a contradiction. Let p be the pumping length for B that is guaranteed to exist by the pumping lemma. Select the string $s = a^p b^p c^p$. Clearly $s \in B$ and $|s| > p$ The pumping lemma states that s can be pumped, but we show that it cannot. In other words, we show that no matter how we devide s into $uvxyz$, one of the three conditions of the lemma is violated.

First, condition 2 stipulates that either v or y is nonempty. Then we consider one of two cases, depending on whether substrings v and y contain more than one type of alphabet symbol.