AUTOMATA THEORY

Exam materials

Author
Automata guy
Riga
2024

Contents

1 Basics

1.1 String

A string is a finite sequence, possibly empty, of symbols drawn from some alphabet Σ. Given any alphabet Σ, the shortest string that can be formed from Σ is the empty string, which we will write as ϵ. The set of all possible strings over an alphabet Σ is written Σ^*. This notation exploits the Kleene star operator, which we will define more generally below.

1.2 Language

A language is a (finite or infinite) set of strings over a finite alphabet Σ. When we are talking about more than one language, we will use the notation Σ_L to mean the alphabet from which the strings in the language L are formed.

2 Finite state machines (FSM)

Formally, a deterministic FSM (or DFSM) M is a quintuple $(K, \Sigma, , s, A)$, where: • K is a finite set of states,

- Σ is the input alphabet,
- $s \in K$ is the start state,
- $A \subseteq K$ is the set of accepting states, and
- σ is the transition function. It maps from:

$$K \times \Sigma \to K$$

A configuration of a DFSM M is an element of K Σ^*. Think of it as a snapshot of M. It captures the two things that can make a difference to M's future behavior:

• its current state • the input that is still left to read.

M halts whenever it enters either an accepting or a rejecting configuration. It will do so immediately after reading the last character of its input.

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

2.1 Equivalence of states

1. Acceptor states q and q' are not equivalent if exactly one of they have an accepting;
2. Transformer states q and q' are not equivalent if any inputs for letter x print different letters $y \neq y'$;
3. The states q and q' of any automaton are not equivalent if any for x leads to nonequivalent states $f(q, x) \neq f(q', x)$;
4. Otherwise, states q and q' are equivalent: $q \neq q'$.

2.2 Regex

Elements of regex

- * asterisk indicates zero or more occurrences of the preceding element. For example, ab*c matches "ac", "abc", "abbc", "abbbc", and so on.
- + the plus sign indicates one or more occurrences of the preceding element. For example, ab+c matches "abc", "abbc", "abbbc", and so on, but not "ac".

2.3 Example: FSM to Regex

2.4 Example: Regex to FSM

2.5 Transducer (Transformator)

This definition for a deterministic finite state transducer permits each machine to output any finite sequence of symbols as it makes each transition (in other words, as it reads each symbol of its input). FSMs that associate outputs with transitions are called Mealy machines, after their inventor George Mealy. A Mealy machine M is a six-tuple $(K, \Sigma, O, , s, A)$, where:

- K is a finite set of states,
- Σ is an input alphabet,
- O is an output alphabet,
- sK is the start state,
- AK is the set of accepting states (although for some applications this designation is not important),
- is the transition function. It is function $(K\Sigma)to(K)$, and
- D is the display or output function. It is a function from $(K)to(O*)$.

A Mealy (transducer defined above) machine M computes a function $f(w)$ iff, when it reads the input string w, its output sequence is $f(w)$.

2.6 Pumping lemma for Regular Languages

Theorem : If L is a regular language, then:

$$\begin{aligned}\exists k \geq 1(\forall \text{ strings } w \in L, where|w| \geq k(x, y, z(w =& xyz,\\ & |xy| \leq k,\\ & y \neq \epsilon, and\\ & \forall q \geq 0(xy^q z \in L)))).\end{aligned}$$

The Pumping Theorem tells us something that is true of every regular language. Generally, if we already know that a language is regular, we won't particularly care about what the

Pumping Theorem tells us about it. But suppose that we are interested in some language L and we want to know whether or not it is regular. If we could show that the claims made in the Pumping Theorem are not true of L, then we would know that L is not regular. It is in arguments such as this that we will find the Pumping Theorem very useful. In particular, we will use it to construct proofs by contradiction. We will say, "If L were regular, then it would possess certain properties. But it does not possess those properties. Therefore, it is not regular."

2.6.1 Example 8.8

AnBn is not Regular.

Let L be $AnBn = a^n b^n : n! = 0$. We can use the Pumping Theorem to show that L is not regular. If it were, then there would exist some k such that any string w, where |w| k, must satisfy the conditions of the theorem. We show one sof the Pumping Theorem. So there must exist x, y, and z, such that w = xyz, |xy| k, y , and q 0 (xyqz L). But we show that no such x, y, and z exist. Since we must guarantee that |xy| k, y must occur within the first k characters and so y = ap for some p. Since we must guarantee that y , p must be greater than 0. Let q = 2. (In other words, we pump in one extra copy of y.) The resulting string is ak+pbk. The last condition of the Pumping Theorem states that this string must be in L, but it is not since it has more a's than b's. Thus there exists at least one long string in L that fails to satisfy the conditions of the Pumping Theorem. So L = AnBn is not regular. tring w that does not. Let w = akbk. Since |w| = 2k, w is long enough and it is in L, so it must satisfy the conditions

3 Context-free languages and PDA

3.1 PDA

3.1.1 Definition of a (Nondeterministic) PDA

A pushdown automaton, or PDA, is a finite state machine that has been augmented by a single stack. In a minute, we will present the formal definition of the PDA model that we will use. But, before we do that, one caveat to readers of other books is in order. There are several competing PDA definitions, from which we have chosen one to present here. All are provably equivalent, in the sense that, for all i and j, if there exists a versioni PDA that accepts some language L then there also exists a versionj PDA that accepts L. We'll return to this issue in Section 12.5, where we will mention a few of the other models and sketch an equivalence proof. For now, simply beware of the fact that other definitions are also in widespread use.

We will use the following definition: A pushdown automaton (or PDA) M is a sixtuple (K, Σ, , , s, A), where:

• K is a finite set of states, • Σ is the input alphabet, • is the stack alphabet, • s K is the start state, • A K is the set of accepting states, and • is the transition relation. It is a finite subset of

(1)

state input or string of symbols to pop state string of symbols to

from top of stack push on top of stack We will use the following notational convention for describing M's stack as a string: The top of the stack is to the left of the string. So:

c

a will be written as:

cab

b

If a sequence c1c2…cn of characters is pushed onto the stack, they will be pushed rightmost first, so if the value of the stack before the push was s, the value after the push will be c1c2…cns.

A configuration of a PDA M is an element of K Σ* *. It captures the three things that can make a difference to M's future behavior:

• its current state, • the input that is still left to read, and • the contents of its stack.

The initial configuration of a PDA M, on input w, is (s, w,).

Note two things about what a transition ((q1, c, 1), (q2, 2)) says about how M manipulates its stack:

• M may only take the transition if the string 1 matches the current top of the stack. If it does, and the transition is taken, then M pops 1 and then pushes 2. M cannot "peek" at the top of its stack without popping off the values that it examines.

• If 1 = , then M must match against the top of the stack. But matches everywhere. So letting 1 be is equivalent to saying "without bothering to check the current value of the stack". It is not equivalent to saying, "if the stack is empty." In our definition, there is no way to say that directly, although we will see that we can create a way by letting M, before it does anything else, push a special marker onto the stack. Then, whenever that marker is on the top of the stack, the stack is otherwise empty.

3.2 Probabalistic automata

Q – stāvokļu kopa X – ieejas alfabēts p – pārejas funkcija: $Q \times X \times Q[0;1]$ vai $X \to [0;1]^{|Q|\times|Q|}$ $q_0 \in Q$ $Q_A \subseteq Q$ q_0 – sākumstāvoklis (dažreiz q0 [0;1]|Q|) q_A – akceptējošie stāvokļi (dažreiz QA [0;1]|Q|) $\lambda \in [0;1]$ – akceptēšanas slieksnis (dažreiz lambdas nav)