

5. AUTOMATA AND GRAMMARS

Finite automata accepting rational languages may be viewed as grammars and also as systems of equations. Consider for example the language $(1 + a)bc^*b$ accepted by the automaton

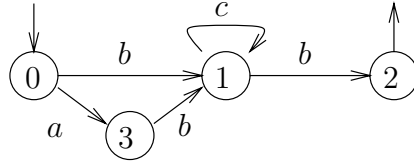


FIGURE 1. A finite automaton \mathcal{A} accepting $(1 + a)bc^*b$

5.1. Grammars. We can turn each edge $p_i \xrightarrow{a} p_j$ into a rewrite rule $A_i \rightarrow aA_j$. From the automaton \mathcal{A} we obtain the rules

$$A_0 \rightarrow aA_3 \text{ or } bA_1 \quad A_1 \rightarrow bA_2 \text{ or } cA_1 \quad A_3 \rightarrow bA_1,$$

and using these rules to rewrite A_0 as wA_2 yields labels w of paths from vertex 0 to vertex 2. For example

$$A_0 \rightarrow bA_1 \rightarrow bcA_1 \rightarrow bcbA_2.$$

Usually we add more rules. For each edge $p_i \xrightarrow{a} p_j$ such that p_j is a terminal vertex, add the rule $A_p \rightarrow a$; and if the initial vertex p_0 is also a terminal vertex, add $A_0 \rightarrow \lambda$. Now the labels of successful paths are just the words which can be obtained by derivations of the form

$$A_0 \rightarrow bA_1 \rightarrow bcA_1 \rightarrow bcb.$$

What we have is a regular grammar. Regular grammars consist of a terminal alphabet Σ , a nonterminal alphabet N , a start symbol in N , and a set of productions of the form $A \rightarrow aB$ where $A \in N$, $a \in \Sigma + \lambda$ and $B \in N + \lambda$. In the present case $\Sigma = \{a, b, c\}$, $N = \{A_0, A_1, A_2, A_3\}$, A_0 is the start symbol, and the productions are the rewrite rules introduced above.

The language generated by a regular grammar is the set of words in the terminal alphabet which can be derived from the start symbol. Regular grammars generate rational languages, or regular languages as they are also called. Different kinds of productions give other grammars.

1. Grammars with productions of the form $A \rightarrow \alpha \in \Sigma^*N\Sigma^*$ are called linear grammars and generate linear languages;
2. Allowing the right-hand side of a production to be any word in $(\Sigma + N)^*$ yields context-free grammars, which generate the context-free languages;
3. If in addition the left-hand side can be any word which contains at least one element of N , the resulting grammars, called type 0 grammars, generate all recursively enumerable sets.

Example 5.1. *The context-free grammar with terminals $\{a, a^{-1}, b, b^{-1}\}$, start symbol S and productions*

$$(1) \quad S \rightarrow \lambda \mid SS \mid aSa^{-1} \mid a^{-1}Sa \mid bSb^{-1} \mid b^{-1}Sb$$

generates the language of all words in a and b which are freely equal to the empty word, λ . Formula 1 stands for the six productions $S \rightarrow \lambda, S \rightarrow SS$, etc. These productions reflect the way a word freely equal to λ may be rewritten in terms of one or two shorter words with the same property. Here is a derivation.

$$S \rightarrow SS \rightarrow aSa^{-1}S \rightarrow aa^{-1}S \rightarrow aa^{-1}bSb^{-1} \rightarrow aa^{-1}ba^{-1}Sab^{-1} \rightarrow aa^{-1}ba^{-1}ab^{-1}$$

Exercise 5.2. Prove the assertion of Example 5.1.

5.2. Equations. Each edge $p_i \xrightarrow{a} p_j$ from Figure 1 with p_j not a terminal vertex determines an equation $X_i = aX_j$ in p_i ; if p_j is terminal, the equation is $X_i = aX_j + \lambda$. Add the right-hand sides corresponding to each variable to obtain a system of equations. The automaton \mathcal{A} yields

$$\begin{aligned} X_0 &= bX_1 + aX_3 \\ X_1 &= cX_1 + bX_2 \\ X_2 &= \lambda \\ X_3 &= bX_1. \end{aligned}$$

Clearly this system is just another way of recording the information in Figure 1. On the other hand the system is also a set of linear equations over the semiring of subsets of Σ^* . The semi-ring operations are union and product.

It is not hard to formulate conditions which insure that a linear system like the one above has a unique minimal solution. A solution is of course the assignment of a subset $S_i \subset \Sigma^*$ to each X_i in such a way that the equations hold when S_i is substituted for X_i . A solution is minimal if for any other solution $\{T_i\}$, $S_i \subset T_i$. In the minimal solution for the system above S_0 is just the rational language accepted by the automaton in Figure 1. S_1 is the rational language accepted by that automaton if the initial state is changed to vertex 1, etc.

Just as regular grammars give linear equations, context-free grammars yield polynomial equations. This observation can be taken as the beginning of an algebraic treatment of formal languages which makes use of semirings, matrices, and power series in non-commuting variables.

5.3. Automata. For each of type of grammar discussed above there is a corresponding kind of automaton which accept the languages the grammars generate. Regular grammars are essentially the same object as finite automata, and the situation is similar for linear grammars. Figure 2 shows an automaton for a linear language. In that

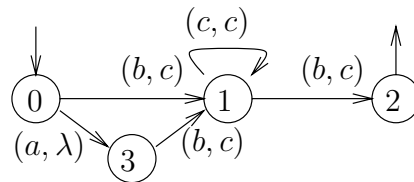


FIGURE 2. An automaton for a linear language.

figure the edge from vertex 0 to vertex 1 corresponds to a production $A_0 \rightarrow bA_1C$ etc. Successful paths begin at vertex 0 and end at vertex 2; their labels are read forward from initial to terminal vertex for the first coordinate and then back for the second coordinate. For example the successful path corresponding to the sequence of vertices 0, 1, 1, 2 has label $bccbcccc$.

Exercise 5.3. Find a pumping lemma for linear languages.

For context-free and type 0 grammars the connection between grammar and automaton is deeper. Type 0 grammars correspond to Turing machines, and context-free grammars to pushdown automata. Turing machines accept exactly the recursively enumerable sets, and pushdown automata accept context-free languages. Figure 3 shows a pushdown automaton \mathcal{P} . \mathcal{P} has a read-only input tape, a pushdown

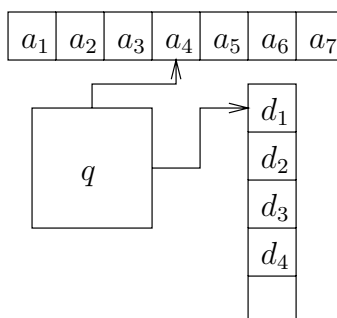


FIGURE 3. A pushdown automaton \mathcal{P} .

stack, and a finite number of internal states q . The input tape is horizontal and the stack is vertical. At any instant \mathcal{P} is in some state q and is scanning a square on its input tape. Based on q , the contents of the square being scanned, and the top element of the stack, \mathcal{P} makes a move. During a move \mathcal{P} may pop, i.e., remove, the top element of its stack, push a finite string of symbols onto the stack, move on to the next square of the input tape, and enter a new internal state. It need not do all these things during each move. When a symbol is popped from the stack, the symbol below it becomes the new top of the stack. At different points in its computation \mathcal{P} may have just one possible move, a choice of moves, or no moves at all.

At the beginning of a computation the input word is written left justified on the input tape, and the stack is empty except for a special end of stack symbol. \mathcal{P} is placed in a designated initial state and set to scan the first square of its input tape. If there is a sequence of moves which ends with empty stack and \mathcal{P} in one of a number of distinguished final states, and if during this sequence \mathcal{P} reads all its input, then \mathcal{P} accepts the input word. There may be other sequences of moves which end differently or not at all, but that does not matter.

A precise definition of pushdown automata can be given in terms of programming languages. From this point of view a pushdown automaton is a program with steps like

- k:** If the current input letter is a , and the top of the stack is d , then pop d , push bc , move to the next input letter, and go to step j . If the next input letter is $b \dots$

Finite automata and Turing machines may also be viewed as programs.