Automata Theory

7

Key Topics

Finite State Automata
State transition
Deterministic FSA
Non-deterministic FSA
Pushdown automata
Turing Machine

7.1 Introduction

Automata Theory is the branch of computer science that is concerned with the study of abstract machines and automata. These include finite-state machines, pushdown automata, and Turing machines. Finite-state machines are abstract machines that may be in one of a finite number of states. These machines are in only one state at a time (current state), and the input symbol causes a transition from the current state to the next state. Finite state machines have limited computational power due to memory and state constraints, but they have been applied to a number of fields including communication protocols, neurological systems and linguistics.

Pushdown automata have greater computational power than finite-state machines, and they contain extra memory in the form of a stack from which symbols may be pushed or popped. The state transition is determined from the current state of the machine, the input symbol and the element on the top of the stack. The action may be to change the state and/or push/pop an element from the stack.

G. O'Regan, *Guide to Discrete Mathematics*, Texts in Computer Science,
DOI 10.1007/978-3-319-44561-8_7

The Turing machine is the most powerful model for computation, and this theoretical machine is equivalent to an actual computer in the sense that it can compute exactly the same set of functions. The memory of the Turing machine is a tape that consists of a potentially infinite number of one-dimensional cells. The Turing machine provides a mathematical abstraction of computer execution and storage, as well as providing a mathematical definition of an algorithm. However, Turing machines are not suitable for programming, and therefore they do not provide a good basis for studying programming and programming languages.

7.2 Finite-State Machines

The neurophysiologists Warren McCulloch and Walter Pitts published early work on finite state automata in 1943. They were interested in modelling the thought process for humans and machines. Moore and Mealy developed this work further, and their finite-state machines are referred to as the 'Mealy machine' and the 'Moore machine'. The Mealy machine determines its outputs through the current state and the input, whereas the output of Moore's machine is based upon the current state alone.

Definition 7.1 (*Finite State Machine*) A finite state machine (FSM) is an abstract mathematical machine that consists of a finite number of states. It includes a start state q_0 in which the machine is in initially; a finite set of states Q; an input alphabet Σ; a state transition function δ; and a set of final accepting states F (where $F \subseteq Q$).

The state transition function δ takes the current state and an input symbol, and returns the next state. That is, the transition function is of the form

$$\delta : Q \times \Sigma \rightarrow Q$$

The transition function provides rules that define the action of the machine for each input symbol, and its definition may be extended to provide output as well as a transition of the state. State diagrams are used to represent finite state machines, and each state accepts a finite number of inputs. A finite-state machine (Fig. 7.1) may be deterministic or non-deterministic, and a *deterministic machine* changes to exactly (or at most)[1] one state for each input transition, whereas a *non-deterministic machine* may have a choice of states to move to for a particular input symbol.

Finite state automata can compute only very primitive functions, and so they are not adequate as a model for computing. There are more powerful automata such as the Turing machine that is essentially a finite automaton with a potentially infinite storage (memory). Anything that is computable is computable by a Turing machine.

[1]The transition function may be undefined for a particular input symbol and state.

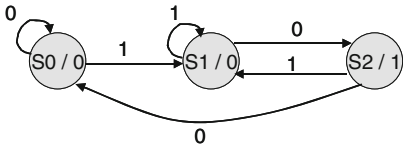

Fig. 7.1 Finite state machine

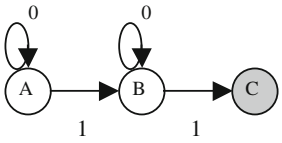

Fig. 7.2 Deterministic FSM

A finite-state machine can model a system that has a finite number of states, and a finite number of inputs/events that can trigger transitions between states. The behaviour of the system at a point in time is determined from the current state and input, with behaviour defined for the possible input to that state. The system starts in a particular initial state.

A finite-state machine (also known as finite-state automata) is a quintuple (Σ, Q, δ, q_0, F). The alphabet of the FSM is given by Σ; the set of states is given by Q; the transition function is defined by δ: $Q \times \Sigma \rightarrow Q$; the initial state is given by q_0; and the set of accepting states is given by F where F is a subset of Q. A string is given by a sequence of alphabet symbols: i.e. $s \in \Sigma^*$, and the transition function δ can be extended to δ^*: $Q \times \Sigma^* \rightarrow Q$.

A string $s \in \Sigma^*$ is accepted by the finite-state machine if $\delta^*(q_0, s) = q_f$ where $q_f \in F$, and the set of all strings accepted by a finite-state machine is the language generated by the machine. A finite-state machine is termed *deterministic* (Fig. 7.2) if the transition function δ is a function,[2] and otherwise (where it is a relation) it is said to be *non-deterministic*. A non-deterministic automata is one for which the next state is not uniquely determined from the present state and input symbol, and the transition may be to a set of states rather than a single state.

For the example above the input alphabet is given by $\Sigma = \{0, 1\}$; the set of states by {A, B, C}; the start state by A; the final state by {C}; and the transition function is given by the state transition table below (Table 7.1). The language accepted by the automata is the set of all binary strings that end with a one that contain exactly two ones.

[2]It may be a total or a partial function (as discussed in Chap. 2).

Table 7.1 State transition table

State	0	1
A	A	B
B	B	C
C	–	–

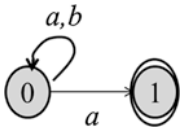

Fig. 7.3 Non-deterministic finite state machine

A *non-deterministic* automaton (NFA) or non-deterministic finite state machine is a finite state machine where from each state of the machine and any given input, the machine may jump to several possible next states. However, a non-deterministic automaton (Fig. 7.3) is equivalent to a deterministic automaton, in that they both recognize the same formal language (i.e. regular languages as defined in Chomsky's classification). For any non-deterministic automaton, it is possible to construct the equivalent deterministic automaton using power set construction.

NFAs were introduced by Scott and Rabin in 1959, and a NFA is defined formally as a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ as in the definition of a deterministic automaton, and the only difference is in the transition function δ.

$$\delta : Q \times \Sigma \to \mathbb{P}Q$$

The non-deterministic finite state machine $M_1 = (Q, \Sigma, \delta, q_0, F)$ may be converted to the equivalent deterministic machine $M_2 = (Q', \Sigma, \delta', q_0', F')$ where

$Q' = \mathbb{P}Q$ (the set of all subsets of Q)
$q_0' = \{q_0\}$
$F' = \{q \in Q' \text{ and } q \cap F \neq \varnothing\}$
$\delta'(q, \sigma) = \cup_{p \in q} \delta(p, \sigma)$ for each state $q \in Q'$ and $\sigma \in \Sigma$.

The set of strings (or language) accepted by an automaton M is denoted $L(M)$. That is, $L(M) = \{s : |\delta^*(q_0, s) = q_f \text{ for some } q_f \in F\}$. A language is termed regular if it is accepted by some finite-state machine. Regular sets are closed under union, intersection, concatenation, complement, and transitive closure. That is, for regular sets A, B $\subseteq \Sigma^*$ then

- A $\cup$ B and A $\cap$ B are regular.
- $\Sigma^* \backslash$A (i.e. A^c) is regular.
- AB and A^* is regular.

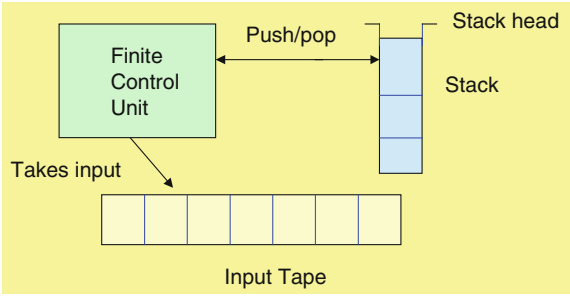

Fig. 7.4 Components of pushdown automata

The proof of these properties is demonstrated by constructing finite-state machines to accept these languages. The proof for A ∩ B is to construct a machine $M_{A \cap B}$ that mimics the execution of M_A and M_B and is in a final state if and only if both M_A and M_B are in a final state. Finite-state machines are useful in designing systems that process sequences of data.

7.3 Pushdown Automata

A pushdown automaton (PDA) is essentially a finite-state machine with a stack, and its three components (Fig. 7.4) are an input tape; a control unit; and a potentially infinite stack. The stack head scans the top symbol of the stack, and two operations (push or pop) may be performed on the stack. The *push* operation adds a new symbol to the top of the stack, whereas the *pop* operation reads and removes an element from the top of the stack.

A pushdown automaton may remember a potentially infinite amount of information, whereas a finite state automaton remembers only a finite amount of information. A PDA also differs from a FSM in that it may use the top of the stack to decide on which transition to take, and it may manipulate the stack as part of performing a transition. The input and current state determine the transition in a finite-state machine, and a FSM has no stack to work with.

A pushdown automaton is defined formally as a 7-tuple $(\Sigma, Q, \Gamma, \delta, q_0, Z, F)$. The set Σ is a finite set which is called the input alphabet; the set Q is a finite set of states; Γ is the set of stack symbols; δ, is the transition function which maps $Q \times \{\Sigma \cup \{\varepsilon\}\}$[3] $\times \Gamma$ into finite subsets of $Q \times \Gamma^*$[4]; q_0 is the initial state; Z is the initial stack top symbol on the stack (i.e. $Z \in \Gamma$); and F is the set of accepting states (i.e. $F \subseteq Q$).

[3]The use of$\{\Sigma \cup \{\varepsilon\}\}$is to formalize that the PDA can either read a letter from the input, or proceed leaving the input untouched.

[4]This could also be written as $\delta: Q \times \{\Sigma \cup \{\varepsilon\}\} \times \Gamma \rightarrow \mathbb{P}(Q \times \Gamma^*)$. It may also be described as a transition relation.

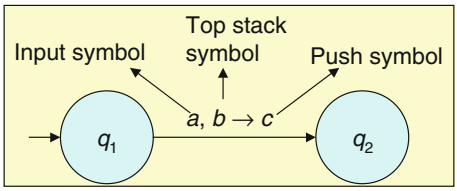

Fig. 7.5 Transition in pushdown automata

Figure 7.5 shows a transition from state q_1 to q_2, which is labelled as a, $b \rightarrow c$. This means that at if the input symbol a occurs in state q_1, and the symbol on the top of the stack is b, then b is popped from the stack and c is pushed onto the stack. The new state is q_2.

In general, a pushdown automaton has several transitions for a given input symbol, and so pushdown automata are mainly *non-deterministic*. If a pushdown automaton has at most one transition for the same combination of state, input symbol, and top of stack symbol it is said to be a *deterministic PDA* (DPDA). The set of strings (or language) accepted by a pushdown automaton M is denoted $L(M)$.

The class of languages accepted by pushdown automata is the context free languages, and every context free grammar can be transformed into an equivalent non-deterministic pushdown automaton. Chapter 12 has more detailed information on the classification of languages,

Example (Pushdown Automata)
Construct a non-deterministic pushdown automaton which recognizes the language $\{0^n\ 1^n | \ n \geq 0\}$.

Solution
We construct a pushdown automaton $M = (\Sigma, Q, \Gamma, \delta, q_0, Z, F)$ where $\Sigma = \{0,1\}$; $Q = \{q_0, q_1, q_f\}$; $\Gamma = \{A, Z\}$; q_0 is the start state; the start stack symbol is Z; and the set of accepting states is given by $\{q_f\}$:. The transition function (relation) δ is defined by

1. $(q_0, 0, Z) \rightarrow (q_0, AZ)$
2. $(q_0,\ 0, A) \rightarrow (q_0, AA)$
3. $(q_0, \varepsilon, Z) \rightarrow (q_1, Z)$
4. $(q_0, \varepsilon, A) \rightarrow (q_1, A)$
5. $(q_1, 1, A) \rightarrow (q_1, \varepsilon)$
6. $(q_1, \varepsilon, Z) \rightarrow (q_f, Z)$

The transition function (Fig. 7.6) essentially says that whenever the value 0 occurs in state q_0 then A is pushed onto the stack. Parts (3) and (4) of the transition function essentially states that the automaton may move from state q_0 to state q_1 at any moment. Part (5) states when the input symbol is 1 in state q_1 then one symbol

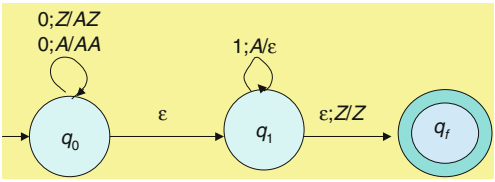

Fig. 7.6 Transition function for pushdown automata M

A is popped from the stack. Finally, part (6) states the automaton may move from state q_1 to the accepting state q_f only when the stack consists of the single stack symbol Z.

For example, it is easy to see that the string 0011 is accepted by the automaton, and the sequence of transitions is given by

$$(q_0,\ 0011, Z) \vdash (q_0,\ 011, AZ) \vdash (q_0,\ 11, AAZ) \vdash (q_1,\ 11, AAZ)$$
$$\vdash (q_1,\ 1, AZ) \vdash (q_1, \varepsilon, Z) \vdash (q_f, Z).$$

7.4 Turing Machines

Turing introduced the theoretical Turing Machine in 1936, and this abstract mathematical machine consists of a head and a potentially infinite tape that is divided into frames (Fig. 7.7). Each frame may be either blank or printed with a symbol from a finite alphabet of symbols. The input tape may initially be blank or have a finite number of frames containing symbols. At any step, the head can read the contents of a frame; the head may erase a symbol on the tape, leave it unchanged, or replace it with another symbol. It may then move one position to the right, one position to the left, or not at all. If the frame is blank, the head can either leave the frame blank or print one of the symbols.

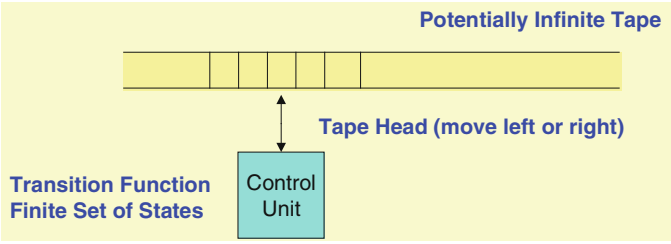

Fig. 7.7 Turing machine

Turing believed that a human with finite equipment and with an unlimited supply of paper to write on could do every calculation. The unlimited supply of paper is formalized in the Turing machine by a paper tape marked off in squares, and the tape is potentially infinite in both directions. The tape may be used for intermediate calculations as well as input and output. The finite number of configurations of the Turing machine was intended to represent the finite states of mind of a human calculator.

The transition function determines for each state and the tape symbol what the next state to move to and what should be written on the tape, and where to move the tape head.

Definition 7.2 (*Turing Machine*) A Turing machine $M = (Q, \Gamma, b, \Sigma, \delta, q_0, F)$ is a 7-tuple as defined formally in [1] as:

- Q is a finite set of *states*
- Γ is a finite set of the *tape alphabet/symbols*
- $b \in \Gamma$ is the *blank symbol* (This is the only symbol that is allowed to occur infinitely often on the tape during each step of the computation)
- Σ is the set of input symbols and is a subset of Γ (i.e. $\Gamma = \Sigma \cup \{b\}$).
- δ: $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$[5] is the transition function. This is a partial function where L is left shift and R is right shift
- $q_0 \in Q$ is the initial state.
- $F \subseteq Q$ is the set of final or accepting states.

The Turing machine is a simple machine that is equivalent to an actual physical computer in the sense that it can compute exactly the same set of functions. It is much easier to analyse and prove things about than a real computer, but it is not suitable for programming and therefore does not provide a good basis for studying programming and programming languages.

Figure 7.8 illustrates the behaviour when the machine is in state q_1 and the symbol under the tape head is a, where b is written to the tape and the tape head moves to the left and the state changes to q_2.

A Turing machine is essentially a finite-state machine (FSM) with an unbounded tape. The tape is potentially infinite and unbounded, whereas real computers have a large but finite store. The machine may read from and write to the tape. The FSM is essentially the control unit of the machine, and the tape is essentially the store. However, the store in a real computer may be extended with backing tapes and disks, and in a sense may be regarded as unbounded. However, the maximum amount of tape that may be read or written within n steps is n.

[5]We may also allow no movement of the tape head to be represented by adding the symbol 'N' to the set.

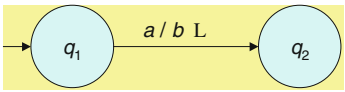

Fig. 7.8 Transition on turing machine

A Turing machine has an associated set of rules that defines its behaviour. Its actions are defined by the transition function. It may be programmed to solve any problem for which there is an algorithm. However, if the problem is unsolvable then the machine will either stop or compute forever. The solvability of a problem may not be determined beforehand. There is, of course, some answer (i.e. either the machine halts or it computes forever). The applications of the Turing machine to computability and decidability are discussed in Chap. 13.

Turing also introduced the concept of a Universal Turing Machine and this machine is able to simulate any other Turing machine.

7.5 Review Questions

1. What is a finite state machine?
2. Explain the difference between a deterministic and non-deterministic finite state machine.
3. Show how to convert the non-deterministic finite state automaton in Fig. 7.3 to a deterministic automaton.
4. What is a pushdown automaton?
5. What is a Turing machine?
6. Explain what is meant by the language accepted by an automaton.
7. Give an example of a language accepted by a pushdown automaton but not by a finite state machine.
8. Describe the applications of the Turing machine to computability and decidability.

7.6 Summary

Automata Theory is concerned with the study of abstract machines and automata. These include finite-state machines, pushdown automata and Turing machines. Finite-state machines are abstract machines that may be in one of a finite number of

states. These machines are in only one state at a time (current state), and the state transition function determines the new state from the current state and the input symbol. Finite-state machines have limited computational power due to memory and state constraints, but they have been applied to a number of fields including communication protocols and linguistics.

Pushdown automata have greater computational power than finite-state machines, and they contain extra memory in the form of a stack from which symbols may be pushed or popped. The state transition is determined from the current state of the machine, the input symbol and the element on the top of the stack. The action may be to change the state and/or push/pop an element from the stack.

The Turing machine is the most powerful model for computation, and it is equivalent to an actual computer in the sense that it can compute exactly the same set of functions. The Turing machine provides a mathematical abstraction of computer execution and storage, as well as providing a mathematical definition of an algorithm

Reference

1. Introduction to Automata Theory, Languages and Computation. Hopcroft, J.E., Ullman, J.D.: Addison-Wesley, Boston (1979).