1 Two-Way Deterministic Finite Automata

Hing Leung[1]

1.1 Introduction

In 1943, McCulloch and Pitts [4] published a pioneering work on a model for studying the behavior of the nervous systems. Following up on the ideas of McCulloch and Pitts, Kleene [2] wrote the first paper on finite automata, which proved a theorem that we now call Kleene's theorem. A finite automaton can be considered as the simplest machine model in that the machine has a finite memory; that is, the memory size is independent of the input length. In a 1959 paper [5], Michael Rabin and Dana Scott presented a comprehensive study on the theory of finite automata, for which they received the Turing award in 1976, the highest award in computer science. The citation for the Turing Award states that the award was granted:

> For their joint paper "Finite Automata and Their Decision Problem," which introduced the idea of nondeterministic machines, which has proved to be an enormously valuable concept. Their (Scott & Rabin) classic paper has been a continuous source of inspiration for subsequent work in this field.

In this project, we will not discuss nondeterministic machines. We consider two-way finite automata which is another concept that was introduced in the seminal paper by Rabin and Scott [5].

In an early stage, the theory of finite automata has been developed as a mathematical theory for sequential circuits. A sequential circuit maintains a current state from a finite number of possible states. The circuit logic (which is a finite state control) decides the new state based on the current state of the circuit and the given input symbol. Once an input symbol is processed, the circuit will not be able to read it again.

A main application of finite automata is text processing. In compiler design, finite automata are used to capture the logic of lexical analysis. Other applications include string matching, natural language processing, text compression, etc.

A one-way deterministic finite automata (DFA) is defined as the mathematical model of a machine with a finite amount of memory where the input is processed once from left to right. After an input has been read, the DFA decides whether the input is accepted or rejected.

1.1.1 Two-Way Finite Automata

A Turing machine is an abstract mathematical model of a computer. Recall that a Turing machine can move back and forth in the working tape while reading and/or writing.

There are several variants of Turing machines in the literature. Our variant of a Turing machine consists of a finite-state control (also called the program), a read-only input tape where the input is given and a working tape (also called memory) where computations are performed.

In computing an answer, intermediate results are computed and kept in the working tape so that it can be referenced later for further computations.

Unlike a real computer, a Turing machine has no limit to the amount of memory that it can use. As the input becomes more complicated (say longer or of larger value), a Turing machine may use more memory to compute.

Consider the problem of primality testing. That is, given a positive integer, we want to test if it is a prime. As the given integer becomes larger, we need to use more memory to test for primality. We say that the primality testing problem requires an unbounded amount of memory to solve.

[1]Department of Computer Science, New Mexico State University, Las Cruces, NM 88003; hleung@cs.nmsu.edu.

However, there are other problems that may require only a finite amount of memory to solve. Consider the problem of computing the parity of a binary sequence. The parity is the number of occurrences of 1's in the binary sequence modulo 2. (Note: The concept defined is not the same as whether the binary sequence, when considered as a binary number, is odd or even.) One can keep reading the binary sequence bit by bit, and maintain the intermediate result of the current parity in the working tape. That is, the memory usage is one bit (thus, finite) in the working tape no matter how long the binary sequence is. Note that the length of the input is not counted in the memory use. We assume that the input tape allows only read operations. The input tape cannot be over-written.

One may wonder what computational problems can be solved using finite memory Turing machines. Another interesting question is whether we can simplify the model of Turing machines using finite memory.

Since the memory usage is finite (independent of the size of the input), we can incorporate[2] the finite memory into the finite state control so that the Turing machine no longer uses a working tape. The resulting model is called the two-way deterministic finite automaton[3] (2DFA) in that it consists of a finite state control and a read-only input tape that allows an input to be read back and forth. As in the case of DFA, the 2DFA decides whether a given input is accepted or rejected.

One can see that a 2DFA can be equivalently defined as a read only Turing machine (without the ability to write symbols on the tape).

1.1.2 DFA and 2DFA

In comparison, DFA and 2DFA differ in that an input can be read only once from left to right by a DFA, whereas a 2DFA can read the input back and forth with no limit on how many times an input symbol can be read.

In computer science, DFA has been studied heavily, since many problems are found to be solvable by DFA. Many textbooks in discrete mathematics discuss the DFA model. However, no textbooks in discrete mathematics discuss the model of 2DFA.

Are 2DFA unrealistic? As we have discussed before, 2DFA is a very interesting model in that it captures/solves problems that are solvable by a computer using finite memory. In fact, in most real life computing tasks performed by a computer, the input has been saved into the computer's hard disk, or is kept in a CD-ROM or DVD-ROM. Thus, there is no reason why a program cannot read the input more than once using two-way processing. So, 2DFA is indeed a very meaningful and arguably more realistic model than DFA.

How does DFA compare to 2DFA? Clearly, DFA is a restricted version of 2DFA. Therefore, 2DFA can solve any problems that are solvable by DFA. Next, are there problems that can be solved by 2DFA but cannot be solved by DFA?

This question is one of many fundamental questions answered in Rabin and Scott's paper [5] on the theory of finite automata. It is proved that 2DFA can be simulated by DFA. That is, whatever problems can be solved by 2DFA can also be solved by DFA.

One may wonder why we should study 2DFA given that DFA, being a simpler model, can do the same job. Are there advantages of 2DFA over DFA? It turns out that 2DFA can be significantly[4] simpler in design for solving the same problem than DFA. In this project, we are going to illustrate the advantages of 2DFA using a number of examples.

[2]The technique involved, which we omit, is not really direct or immediate. But it is not difficult either.

[3]There is another machine model called two-way nondeterministic finite automata. But we are not going to discuss nondeterministic automata in this project.

[4]Technically (and, more accurately), we say that 2DFA can be *exponentially* more succinct in descriptional size than DFA for solving the same problems.

It is difficult to explain why most textbooks[5] in automata theory are not covering 2DFA. One possible reason is that the equivalence proof (that 2DFA can be simulated by the simpler model DFA) given by Rabin and Scott is too difficult.

John C. Shepherdson [6] was able to offer another proof of this important result. It is a very clean proof that we want to present in this project. Instead of going through the proof steps, we emphasize the technique for constructing[6] a DFA from a 2DFA.

Shepherdson is a retired professor in mathematics at the University of Bristol, Great Britain. He published many papers in symbolic logic, computability, logic programming and fuzzy logic.

In fact, Rabin and Scott referred the readers to Shepherdson's proof in their pioneering paper, and decided to give only a sketch of their proof of the equivalence result. Following is an excerpt from Rabin and Scott's paper about Shepherdson's proof:

> The result, with its original proof, was presented to the Summer Institute of Symbolic Logic in 1957 at Cornell University. Subsequently J. C. Shepherdson communicated to us a very elegant proof which also appears in this Journal. In view of this we confine ourselves here to sketching the main ideas of our proof.

We hope that the students using this project will not find Shepherdson's construction tricky, but instead will find that it makes a lot of sense and is the logical way to go for proving the equivalence result. Students will be required to read from the verbal explanations given by Shepherdson, and derive from it computer programs for solving problems in this project.

1.2 Project

We assume that students are familiar with the concept and formal definition of DFA.

Following are the definitions of one-way and two-way deterministic finite automata (adapted) from the paper by Shepherdson [6].

> Definition 1. A one-way finite automaton (DFA) over a finite alphabet[7] Σ is a system $A = (Q, \delta, q_0, F)$, where Q is a finite non-empty set (the internal states of A), δ is a function from $Q \times \Sigma$ into Q (the table of moves of A), q_0 is an element of Q (the initial state of A), and F is a subset of Q (the designated final states of A). The class $T(A)$ of tapes accepted[8] by A is the class of all finite sequences $\sigma_1, \ldots, \sigma_n$ of symbols of Σ for which the sequence q_0 (initial state), $q_1, \ldots, q_n$ defined by $q_{i+1} = \delta(q_i, \sigma_{i+1})$ $(i = 0, \ldots, n-1)$ satisfies $q_n \in F$. A set of tapes is said to be definable by a one-way automaton if it is equal to $T(A)$ for some A.

> Definition 2. A two-way finite automaton (2DFA) over Σ is a system $A = (Q, \delta, q_0, F)$ as in Definition 1 with the difference that now δ is a function from $Q \times \Sigma$ into $Q \times D$ where $D = \{L, S, R\}$. A operates as follows: It starts on the leftmost square of the given tape in state q_0. When its internal state is q and it scans the symbol σ, then if $\delta(q, \sigma) = (q', d)$ it goes into the new state q' and moves one square to the left, stays where it is, or moves one square to the right according as $d = L, S$, or R. The class $T(A)$ of tapes accepted by A

[5] Two textbooks ([1], [3]) cover 2DFA. One [1] follows the approach by Rabin and Scott, and the other [3] follows Shepherdson's ideas.

[6] In contrast, it is difficult to construct mechanically a DFA from a 2DFA based on the proof of Rabin and Scott.

[7] Σ is the set of input symbols. In the examples considered in this project, the input is always a binary sequence with $\Sigma = \{0, 1\}$.

[8] The word 'accepted' is used as the DFA is considered a machine. One can replace the word 'accepted' by the word 'denoted' or 'defined' in the definition of $T(A)$.

is the class of those tapes t such that A eventually moves off the right-hand edge of t in a state belonging to F.

We want to design a 2DFA over $\Sigma = \{0, 1\}$ that accepts tapes containing two 1's separated by 4 symbols in between them. That is, the 2DFA accepts finite sequences $\sigma_1 \ldots \sigma_n$ of symbols of Σ such that $\sigma_i = \sigma_{i+5} = 1$ for some $i \in \{1, \ldots, n-5\}$. For example, the sequence 001010101100 should be accepted as the 5th and 10th symbols are both 1.

The following 10-state 2DFA A_1, where q_0 is the starting state and q_9 is the only final state, systematically checks every possible 6-symbol subsequence to see if it begins and ends with 1's.

current state	symbol	new state	go to
q_0	0	q_0	R
q_0	1	q_1	R
q_1	0 or 1	q_2	R
q_2	0 or 1	q_3	R
q_3	0 or 1	q_4	R
q_4	0 or 1	q_5	R
q_5	0	q_6	L
q_5	1	q_9	R
q_6	0 or 1	q_7	L
q_7	0 or 1	q_8	L
q_8	0 or 1	q_0	L
q_9	0 or 1	q_9	R

1.2.1 Demonstrate the steps performed by A_1 in accepting the tape 11001010.

1.2.2 Demonstrate the steps performed by A_1 in processing the tape 11001001. Conclude that the tape is not accepted by A_1.

Another example 2DFA A_2 (taken from Example 2.14 of the textbook by Hopcroft and Ullman [1]) is given as follows, where q_0 is the starting state and $F = \{q_0, q_1, q_2\}$:

current state	symbol	new state	go to
q_0	0	q_0	R
q_0	1	q_1	R
q_1	0	q_1	R
q_1	1	q_2	L
q_2	0	q_0	R
q_2	1	q_2	L

1.2.3 Demonstrate that the input 101001 is accepted by A_2.

1.2.4 Demonstrate that the input 10111 is not accepted by A_2. Indeed, A_2 will run into an infinite loop.

Given the description of a 2DFA, Shepherdson explained how to derive the description of an equivalent DFA that accepts the same set of inputs. The following is an excerpt (modified slightly) from Shepherdson [6] describing the construction:

> The only way an initial portion t of the input tape can influence the future behaviour of the two-way machine A when A is not actually scanning this portion of the tape is via the state transitions of A which it causes. The external effect of t is thus completely determined by the transition function, or "table", τ_t which gives (in addition to the state in which the machine originally exits from t), for each state q of A in which A might re-enter t, the

> corresponding state q' which A would be in when it left t again. This is all the information the machine can ever get about t however many times it comes back to refer to t; so it is all the machine needs to remember about t. But there are only a finite number of different such transition tables (since the number of states of A is finite), so the machine has no need to use the input tape to supplement its own internal memory; a one-way machine $\bar{A}$ with a sufficiently large number of internal states could store the whole transition table τ_t of t as it moved forward, and would then have no need to reverse and refer back to t later. If we think of the different states which A could be in when it re-entered t as the different questions A could ask about t, and the corresponding states A would be in when it subsequently left A again, as the answers, then we can state the result more crudely and succinctly thus: A machine can spare itself the necessity of coming back to refer to a piece of tape t again, if, before it leaves t, it thinks of all the possible questions it might later come back and ask about t, answers these questions now and carries the table of question-answer combinations forward along the tape with it, altering the answers where necessary as it goes along.

To summarize Shepherdson's idea, we need to maintain for each prefix t two pieces of information: (1) the state in which the machine exits from t (when starting at q_0 in the leftmost square of the input), and (2) the external effect τ_t.

Let us refer to the processing of the input tape 101001 by A_2. Consider the first symbol of the input which is a 1. That is, let $t = 1$.

To answer the first question, we observe that when A_2 begins with the initial state q_0 at the symbol 1, it will exit t to its right at state q_1.

Next, we summarize the external effect τ_t where $t = 1$ by answering the following questions:

1. Suppose later in the processing of the input, the first symbol 1 is revisited by a left move from the right. What will be the effect if it is revisited with a state q_0?

2. Suppose later in the processing of the input, the first symbol 1 is revisited by a left move from the right. What will be the effect if it is visited with a state q_1?

3. Suppose later in the processing of the input, the first symbol 1 is revisited by a left move from the right. What will be the effect if it is visited with a state q_2?

Verify that the answers to the 3 questions are: (1) move right at state q_1, (2) move left at state q_2 and (3) move left at state q_2. In short the external effect can be succinctly summarized as a 3-tuple [(q_1, R), (q_2, L), (q_2, L)].

We combine the two pieces of information computed in a 4-tuple [(q_1, R), (q_1, R), (q_2, L), (q_2, L)]. Let us call this 4-tuple the effect of t where $t = 1$. Note that the first entry is the answer to the first question, whereas the next 3 entries are the external effect of t. That is, the effect of t is the total effect, which is more than just the external effect.

Next, given the effect computed for the first symbol 1, we want to compute the effect of the first two symbols 10 of the input 101001.

¿From the effect [(q_1, R), (q_1, R), (q_2, L), (q_2, L)] of the symbol 1, we know that the machine exits 1 to its right with the state q_1. Thus, the machine is at state q_1 when visiting the second symbol 0. According to the transition table, the machine will again move to the right of the second symbol at state q_1.

To compute the external effect of $t = 10$, we have to answer the following questions:

1. Suppose later in the processing of the input, the second symbol 0 is revisited by a left move from the right. What will be the effect if it is revisited with a state q_0?

2. Suppose later in the processing of the input, the second symbol 0 is revisited by a left move from the right. What will be the effect if it is visited with a state q_1?

3. Suppose later in the processing of the input, the second symbol 0 is revisited by a left move from the right. What will be the effect if it is visited with a state q_2?

The answers are as follows:

1. When the second symbol 0 is revisited with a state q_0, the machine according to the transition table will move to the right with the state q_0.

2. When the second symbol 0 is revisited with a state q_1, the machine according to the transition table will move to the right with the state q_1.

3. When the second symbol 0 is revisited with a state q_2, the machine according to the transition table will move to the right with the state q_0.

Therefore, the effect of $t = 10$ is summarized in the 4-tuple [(q_1, R), (q_0, R), (q_1, R), (q_0, R)].

Note that in the above computations, we do not make use of the external effect computed for the first prefix 1 to compute the effect for the prefix 10. This is not usually the case. In general, the external effect for t is needed in computing the new effect when t is extended by one more symbol.

1.2.5 Next, with the answers [(q_1, R), (q_0, R), (q_1, R), (q_0, R)] for the effect for 10, compute the effect when the input is extended by the third symbol 1.

¿From the effect for 10, we know that the machine arrives at the third symbol 1 at state q_1. According to the transition table, the machine will move to the left to the 2nd symbol at state q_2. Next, consulting the last entry of the effect for 10, we know the machine will leave 10 to its right at state q_0. Thus, the machine revisits the third symbol 1 at state q_0. Again, from the transition table, the machine moves to the right at state q_1. Verify that the external effect of 101 is [(q_1, R), (q_1, R), (q_1, R)]. That is, the effect for 101 is [(q_1, R), (q_1, R), (q_1, R), (q_1, R)].

In answering question 1.2.5, you should provide the steps involved in computing the external effect for 101.

Note that given the effect for t and a new symbol 1, we can compute the effect for $t' = t1$ by referring to the transition table for the machine. There is no need to know what t is. Only the effect for t is needed in computing the new effect for t'.

1.2.6 Repeatedly, compute the effects by extending the current input 101 with symbols 0, 0, 1. From the effect computed for 101001, conclude that the input is accepted. Recall that 101001 was shown in 1.2.3 to be accepted by A_2.

1.2.7 Repeat the whole exercise with the input 10111 as in 1.2.4. Conclude that the input 10111 is not accepted.

To summarize, in simulating A_2 by a DFA, we need only remember the effect of the sequence of input symbols that has been processed so far.

1.2.8 How many different effects can there be in simulating A_2 by a DFA? How many states are needed for a DFA to accept the same set of inputs as A_2? Note that there are only finitely many different effects even though we have an infinite number of inputs of finite lengths.

1.2.9 Applying Shepherdson's method to A_1, how many states are there in the DFA constructed?

The work involved is very tedious. It is impossible to do it by hand. You should write a program to perform the computation.

Note that the smallest DFA[9] accepting the same set of inputs as A_1 has 33 states.

[9]There is a mechanical method for computing the number of states of a smallest DFA.

In computer programming, we can detect if the end-of-input has been reached. For example, in C programming, we write

```
while ((input=getChar()) != EOF)
```

Thus, it is very reasonable to extend the 2DFA model so that the machine can detect the two ends of an input tape.

Read the following excerpt (modified slightly) from Shepherdson's paper [6] regarding extending the 2DFA model by providing each input tape with special marker symbols b, e at the beginning and end of the input. Let us call this new model 2DFA-with-endmarkers.

> At first sight it would appear that with a two-way automaton more general sets of tapes could be defined if all tapes were provided with special marker symbols b, e (not in Σ) at the beginning and end, respectively. For then it would be possible, e.g., to design a two-way machine which, under certain conditions, would go back to the beginning of a tape, checking for a certain property, and then return again. This would appear to be impossible for an unmarked tape because of the danger of inadvertently going off the left-hand edge of the tape in the middle of the computation. In fact, the machine has no way of telling when it has returned to the first symbol of the tape. However, the previous construction result implies that this is not so; that the addition of markers does not make any further classes of tapes definable by two-way automata. For if the set $\{b\}U\{e\}$ (of all tapes of the form bte for $t \in U$) is definable by a two-way automaton then it is definable by a one-way automaton; and it is easy to prove that $\{b\}U\{e\}$ is definable by a one-way automaton if and only if U is.

We assume that a 2DFA-with-endmarkers starts on the left endmarker b at the initial state.

Suppose we want to design a 2DFA-with-endmarkers over $\Sigma = \{0, 1\}$ to accept input tapes that has a symbol 1 in the sixth position from the right end. For example, the input 10101011 has 8 symbols with the third symbol being a 1, which is the sixth position from the last (eighth) position. The following 2DFA-with-endmarkers A_3, where q_0 is the starting state and q_9 is the accepting state, accepts the input tapes described. Observe that an accepted input must begin with b and end with e. Furthermore, b and e do not appear in other positions of the string accepted.

current state	symbol	new state	go to
q_0	b	q_1	R
q_1	0 or 1	q_1	R
q_1	e	q_2	L
q_2	0 or 1	q_3	L
q_3	0 or 1	q_4	L
q_4	0 or 1	q_5	L
q_5	0 or 1	q_6	L
q_6	0 or 1	q_7	L
q_7	1	q_8	R
q_8	0 or 1	q_8	R
q_8	e	q_9	R

1.2.10 Following the discussion by Shepherdson and based on the definition of A_3, explain the construction of a DFA equivalent to A_3 accepting the same set of input tapes where the inputs are delimited by b and e. How many states does the DFA have? Next, modify the DFA constructed to

accept inputs with the endmarkers b and e removed. What is the change in the number of states of the DFA?

As in 1.2.9, you should write a program to perform the computation.

Note that it can be shown that the smallest DFA accepting the same set of inputs (without the endmarkers b and e) as A_3 has 64 states.

Consider another 2DFA-with-endmarkers A_4, where q_0 is the starting state and q_9 is the accepting state, as defined below.

current state	symbol	new state	go to	
q_0	b	q_1	R	
q_1	0 or 1	q_1	R	
q_1	e	q_2	L	
q_i	0	q_{i+1}	L	$i = 2, 3, 4$
q_i	1	q_i	L	$i = 2, 3, 4$
q_5	0	q_7	L	
q_5	1	q_6	L	
q_6	0	q_6	L	
q_6	1	q_7	L	
q_7	0	q_7	L	
q_7	1	q_5	L	
q_7	b	q_8	R	
q_8	0 or 1	q_8	R	
q_8	e	q_9	R	

1.2.11 Re-do part 1.2.10 for A_4.

Note that it can be shown that the smallest DFA accepting the same set of inputs (without the endmarkers b and e) as A_4 has 64 states.

Notes to the Instructor

The project is suitable to be used in a senior undergraduate theory of computation course in computer science. While the programming skills needed to solve some of the project questions are not particularly challenging, they are not trivial either. It is suggested that a student needs two years of programming training before attempting the programming tasks given in this project. As extra credit problem, one can ask the students to deduce (using Myhill-Nerode theorem) the sizes of smallest DFAs that are equivalent to A_1, A_2, A_3 and A_4 respectively.

References

[1] Hopcroft, J. E., and Ullman, J. D., *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979.

[2] Kleene, S. C., "Representation of Events in Nerve Nets and Finite Automata", in *Automata Studies*, Shannon, S. C., McCarthy, J. (editors) Princeton University Press, NJ, 1956, 3–41.

[3] Kozen, D. C., *Automata and Computability*, Springer-Verlag, New York, 1997.

[4] McCulloch, W. S., and Pitts, W., "A Logical Calculus of Ideas Immanent in Nervous Activity", *Bull. Math. BioPhys.*, **5** (1943), 115–133.

[5] Rabin, M. O., Scott, D., "Finite Automata and Their Decision Problems", *IBM Journal of Research and Development*, **3** (1959), 114–125.

[6] Shepherdson, J., "The Reduction of Two-Way Automata to One-Way Automata," *IBM Journal of Research and Development*, **3** (1959), 198–200.