Regular Languages and Finite Automata

Hing Leung
Department of Computer Science
New Mexico State University

1. Introduction

In 1943, McCulloch and Pitts [4] published a pioneering work on a model for studying the behavior of nervous systems. Following on the ideas of McCulloch and Pitts, Kleene [3] wrote the first paper on finite automata and regular expressions. 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 of the theory of finite automata, for which they received the Turing Award, the highest award in computer science.

Rabin and Scott present finite automata in the simplest mathematical model. On the other hand, Kleene's more complex model of finite automata allows the user to express ideas easier, even though formally both models are equivalent in their expressiveness. A similar situation happens with programming languages. While a lower level assembly-styled language with simpler syntax is sufficient to program the computer to do whatever a high level language can do, the programming community embraces higher level programming languages like Java, C++ and Python for higher productivity.

In this project for students, we study Kleene's concept of finite automata and regular expressions [3]. In particular, we learn Kleene's own proof of the theorem (which is now called Kleene's theorem) that shows finite automata and regular expressions are equivalent in their expressiveness for denoting languages.

1.1. S. C. Kleene

Stephen Cole Kleene [1] (born on January 5, 1909 in Hartford, Connecticut; died on January 25, 1994 in Madison, Wisconsin) was an American mathematician who helped lay the foundations for theoretical computer science. He was awarded a Ph.D. in mathematics, under the supervision of Alonzo Church, from Princeton University in 1934.

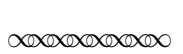
[1] For a detailed biography of Kleene, see http://en.wikipedia.org/wiki/Stephen_Cole_Kleene and http://www-history.mcs.st-andrews.ac.uk/Biographies/Kleene.html.

Kleene's research was on the theory of algorithms and recursive function theory. He developed the field of recursion theory with Church, Gödel, Turing and others.

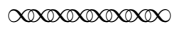
By providing methods of determining which problems are solvable, Kleene's work led to the study of which functions can be computed. Kleene developed a diverse array of topics in computability: the arithmetical hierarchy, degrees of computability, computable ordinals and hyperarithmetic theory, finite automata and regular sets, computability on higher types, recursive realizability for intuitionistic arithmetic with consequences for philosphy and for program correctness in computer science.

2. Kleene's Finite Automata Model

In [3], Kleene investigates McCulloch-Pitts nerve nets. As explained in the excerpt below, his study of nerve nets serves as an illustration of the general theory of finite automata.



Finally, we repeat that we are investigating McCulloch-Pitts nerve nets only partly for their own sake as providing a simplified model of nervous activity, but also as an illustration of the general theory of automata, including robots, computing machines and the like. What a finite automaton can and cannot do is thought to be of some mathematical interest intrinsically, and may also contribute to better understanding of problems which arise on the practical level.



Below is Kleene's description of the finite automata model:

8.1 CELLS. Time shall consist of a succession of discrete moments numbered by the positive integers, except in Section 10 where all the integers will be used.

We shall consider automata constructed of a finite number of parts called *cells*, each being at each moment in one of a finite number of ≥ 2 of states.

We shall distinguish two kinds of cells, *input cells* and *inner cells*.

An input cell admits two states, 0 and 1 (or "quiet" and "firing"), which one is assumed at a given moment being determined by the environment.

The restriction to 2 states for input cells makes the notion of input to the automaton coincide with the notion of an input to a nerve net as formulated in

Sections 4 and 6.5. But the present theory would work equally well with more than 2 states. Nothing would be gained, however, as p cells each admitting 2 states could be used to replace one cell admitting any number q ($2 \leq q \leq 2^p$) of states $0, 1, \ldots, q-1$, where if $q < 2^p$ we could either consider only inputs in which states $q, \ldots, 2^p - 1$ do not occur or identify those states with the state $q-1$ in all the operations of the automaton.

The number of states of an inner cell is not restricted to 2, and different inner cells may have different numbers of states.

The state of each inner cell at any time $t > 1$ is determined by the states of all the cells at time $t-1$. Of course it may happen that we do not need to know the states of all the cells at time $t-1$ to infer the state of a given inner cell at time t. Our formulation merely leaves it unspecified what kind of a law of determination we use, except to say that nothing else than the states of the cells at $t-1$ can matter.

For time beginning with 1, the state of each of the inner cells at that time is to be specified.

8.2 STATE. With k input cells $N_1, \ldots, N_k$ ($k \geq 0$), and m inner cells $M_1, \ldots, M_m$ ($m \geq 1$) with respective number of states $s_1, \ldots, s_m$, there are exactly $2^k s_1 \cdot \ldots \cdot s_m$ possible (*complete*) *states* of the automaton. We can consider each state as a combination of an *external state*, of which there are 2^k possible, and an *internal state*, of which there are $s_1 \cdot \ldots \cdot s_m$ possible.

The law by which the states of the inner cells at time $t > 1$ are determined by the states of all the cells at time $t-1$ can be given by specifying to each of the complete states at time $t-1$ which one of the internal states at time t shall succeed it.

∞∞∞∞∞∞∞∞∞

We notice that Kleene's model is a lot more complex than most of the versions adopted in the textbooks ([1], [2], [6]). In fact, Kleene was well aware of the possibility for defining the finite automata model in the simpler way as in the modern textbooks. He gave the following arguments to defend his preference:

∞∞∞∞∞∞∞∞∞

We could indeed consider the entire aggregate of m internal cells as replaced by a single one admitting $s_1 \cdot \ldots \cdot s_m$ states. We shall not take advantage of this possibility, because we have in view applications of the theory of finite automata in which the cells have certain simple properties and are connected in certain simple ways.

We could also (but shall not) get along with a single input cell, by scheduling the inputs on the k original input cells to come in successively in some order on the new one, which would alter the time scale so that k moments of the new scale correspond to 1 of the original. Events referring to the new time scale could then be interpreted in terms of the original.

∞∞∞∞∞∞∞∞∞

With k input cells in the finite automaton, to describe a past history of the input for t time units, we need a table of t rows by k columns. Thus, an "input" is described by a table.

Next, Kleene defines "event" as follows:

∞∞∞∞∞∞∞∞∞

By an **event** we shall mean any property of the input. In other words, any subclass of the class of all the possible tables describing the input over all past time constitutes an event, which **occurs** when the table describing the actual input belongs to this subclass.

∞∞∞∞∞∞∞∞∞

In the terminology of modern textbooks ([1], [2], [6]), an event is called a "language". That is, an event (or, language) is a subset of the set of all possible inputs.

In what way does a finite automaton denote an event? When Kleene defines the event that a finite automaton denotes, for the sake of convenience, he makes use of what he calls *aggregate states*. This should be not confused with his claim that he "shall not take advantage of this possibility" [of replacing every aggregate state by a single state] as a finite automaton is still allowed to have multiple internal cells.

∞∞∞∞∞∞∞∞∞

Now let us call the states $a_1, \ldots, a_r$ where $r = 2^k \cdot s_1 \cdot \ldots \cdot s_m$ and the internal states $b_1, \ldots, b_q$ where $q = s_1 \cdot \ldots \cdot s_m$. Let the notation be arranged so that the internal state at time 1 is b_1.

With the internal state at time 1 fixed, the state at time p is a function of the input over the time $1, \ldots, p$ (including the value of p, or when $k = 0$ only this).

So each of the states $a_1, \ldots, a_r$ represents an event, which occurs ending with time p, if and only if the input over the time $1, \ldots, p$ is one which results in that one of $a_1, \ldots, a_r$ being the state at time p. Thus the automaton can know about its past experience (inclusive of the present) only that it falls into one of r mutually exclusive classes (possibly some of them empty).

Similarly an internal state at time $p+1$, or a property of the complete state at time p, or a property of the internal state at time $p+1$, or a property of the internal state at time $p+s$ for an $s > 1$ represents an event. Thus to say that the state at time p has a certain property is to say that the state then is one of a certain subclass of the r possible states, so that the past experience falls into the set sum (or disjunction) of the respective classes of past experiences which are separately represented by the states of the subclass.

Exercise 2.1. Compare Kleene's finite automata model with the finite automata model that you find in your textbook. Point out the similarity and differences between the two. Argue that Kleene's model is no more powerful in denoting languages than the textbook's model.

For Exercises 2.2 and 2.3, we consider the following problem. Suppose we have to maintain the value of a 16-bit binary number. Initially, the binary number has the value 0. At every time unit, we are given a new 16-bit binary number as input, which we add to the 16-bit binary number that we are maintaining. In adding two 16-bit binary numbers, overflow may occur. The higher ordered bits that cannot be represented within 16 bits are lost. For example,

```
   1100100100001101
+  0110010011011000
--------------------
   0010110111100101
```

The following is a summary of the logic for adding two 16-bit binary numbers:

```
initialize s = 0000000000000000;
for each time t,
    let b_t (a 16-bit binary number) be the new input
    s = s + b_t;  // binary addition
```

Exercise 2.2. Give the design of a finite automaton, according to Kleene's model, with 16 input cells and 16 inner cells, for the event that the current value of the 16-bit binary number consists of all 0's.

Exercise 2.3. Discuss the design of a finite automaton, according to your textbook's model, that captures the same event as defined in Exercise 2.2.

For Exercises 2.4 and 2.5, we consider the chess game. Initially, all the chess pieces are positioned in their rightful places. Each input corresponds to a move, which can be described as an ordered pair of positions $((x, i), (y, j))$ where $x, y \in \{a, b, c, d, e, f, g, h\}$ and $i, j \in \{1, 2, 3, 4, 5, 6, 7, 8\}$. All the game rules have to be observed. Once the game is over, no more moves are allowed.

Exercise 2.4. Sketch the design of a finite automaton, according to Kleene's model, with 65 inner cells (one for each of the 64 squares on the chessboard and one cell for remembering whose turn, black or white, it is for the next move) for the event that the sequence of moves are valid.[2]

Exercise 2.5. Discuss the design of a finite automaton, according to your textbook's model, that captures the same event as defined in Exercise 2.4.

3. Regular Events

The concept of regular events was introduced by Kleene via the definition of regular expressions. In the terminology of modern textbooks ([1], [2], [6]), regular events are called regular languages.

The following is Kleene's definition of regular events:

∞∞∞∞∞∞∞∞∞

First we define three operations on sets of tables. If E and F are sets of tables, $E \vee F$ (their **sum** or **disjunction**) shall be the set of tables to which a table belongs exactly if it belongs to E or belongs to F.

If E and F are sets of tables, EF (their **product**) shall be the set of tables to which a table belongs exactly if it is the result of writing any table of F next below any table of E. `[This statement is slightly adapted from the original source. Also, some lines are omitted.]`

Obviously $E \vee F$ and EF are associative operations. We may write E^0F for F, E^1 for E, E^2 for EE, E^3 for EEE, etc.

[2] It is not required to give the full design details.

If E and F are sets of tables, E^*F (the **iterate** of E on F, or briefly E **iterate** F) shall be the infinite sum of the sets F, EF, EEF, ..., or in self-explanatory symbolism $F \vee EF \vee EEF \vee \ldots$ or $\sum_{n=0}^{\infty} E^n F$.

The **regular sets (of tables)** shall be the least class of sets of tables which includes the unit sets (i.e., the sets containing one table each) and the empty set and which is closed under the operations of passing from E and F to $E \vee F$, to EF and to E^*F.

An event shall be **regular**, if there is a regular set of tables which describes it in the sense that the event occurs or not according as the input is described by one of the tables of the set or by none of them.

∞∞∞∞∞∞∞∞∞

Kleene defines a related concept of regular expressions as follows[3]:

∞∞∞∞∞∞∞∞∞

By a **regular expression**, we shall mean a particular way of expressing a regular set of tables starting with single-table sets and applying zero or more times the three operations (passing from E and F to $E \vee F$, EF or E^*F).

∞∞∞∞∞∞∞∞∞

Exercise 3.1. Compare Kleene's definitions of regular events and expressions to the corresponding definitions of regular sets (equivalently, regular languages) and regular expressions given in your textbook. The definitions differ slightly. In what way do the definitions differ? Are the differences significant?

Why are regular expressions and regular events introduced? Kleene explains the reasons in the following excerpt:

∞∞∞∞∞∞∞∞∞

Our reason for introducing the regular events, as given by regular sets of tables described by regular expressions, is Theorem 5, which we discovered before Theorem 3. By using the notion of regular events, we thus demonstrate that a McCulloch-Pitts nerve net can represent any event which any other kind of finite digital automaton (in the sense to be developed in detail in Section

[3]The excerpt is slightly adapted from the original source without introducing new terminologies that are present in the original text.

8) can represent. This of course includes a number of special results which McCulloch and Pitts obtained for alternative kinds of nerve nets, but is more general. The way is open to attempt similarly to verify the like for other kinds of "cells" in place of neurons, or to seek some characterization of the properties of the cells in order that aggregates of them have the capacity for representing all representable (i.e., all regular) events.

∞∞∞∞∞∞∞∞∞

That is, regular expressions and regular events are introduced so that we can prove the equivalence with "other finite digital automata" by showing that the events that they can represent are exactly the regular events.

In Theorem 5, Kleene shows that finite automata denote regular events, which are defined in terms of regular expressions.

∞∞∞∞∞∞∞∞∞

What sorts of events can be represented? As the concept of input is the same as in Part I, we can use the notion of "regular event" which was introduced in Section 7. The following theorem, together with Theorem 3 referring to a special kind of finite automaton, answer the question.

Theorem 5. In any finite automaton (in particular, in a McCulloch-Pitts nerve net), started at time 1 in a given internal state b_1, the event represented by a given state existing at time p is regular.

∞∞∞∞∞∞∞∞∞

Next, we give Kleene's proof of Theorem 5.

∞∞∞∞∞∞∞∞∞

PROOF. Since the initial internal state is specified, there are 2^k possible initial states (the results of combining the given initial internal state b_1 with each of the 2^k possible external states at time 1).

So if we can show that the automaton starting from a given state at time 1 will reach a given state at time p, if and only if a certain regular event occurs ending with time p, then the theorem will follow by taking the disjunction of 2^k respective regular events, which is itself a regular event.

Given any state a at time $t-1$ ($t \geq 2$), exactly 2^k states are possible at time t, since the internal part of the state at time t is determined by a, and the external part can happen in 2^k ways. Let us say each of these 2^k states is *in relation* R to a.

∞∞∞∞∞∞∞∞∞

At this point, Kleene decided to give a generalized treatment for the main proof technique behind the proof of Theorem 5. This is in contrast with the treatments that we find in modern textbooks ([1], [2], [6]). Besides, the general treatment is interesting in its own right.

∞∞∞∞∞∞∞∞∞

The next part of our analysis will apply to any binary relation R defined on a given set of $r \geq 1$ objects $a_1, \ldots, a_r$ (called "states"), whether or not it arises in the manner just described.

Consider any two a and $\bar{a}$ of the states, not necessarily distinct. We shall study the strings of states $d_p d_{p-1} \ldots d_1$ $(p \geq 1)$ for which d_p is a, d_1 is $\bar{a}$, and for each t $(t = 2, \ldots, p)$ d_t is in relation R to d_{t-1} (in symbols, $d_t R d_{t-1}$); say such strings *connect* a to $\bar{a}$.

We say a set of strings is "regular" under the following definition (chosen analogously to the definition of "regular" sets of tables in 7.1).

The empty set and for each i $(i = 1, \ldots, r)$ the unit set $\{a_i\}$ having as only member a_i considered as a string of length 1 are *regular*. If A and B are regular, so is their sum, written $A \vee B$. If A and B are regular, so is the set, written AB, of the strings obtainable by writing a string belonging to A just left of a string belonging to B. If A and B are regular, so is the sum, written A^*B, for $n = 0, 1, 2, \ldots$ of the sets $A \ldots AB$ with n A's preceding the B.

Lemma 7. The strings $d_p \ldots d_1$ connecting a to $\bar{a}$ constitute a regular set.

PROOF OF LEMMA, by induction on r.

BASIS: $r = 1$. Then $\bar{a}$ is a. If $\overline{aRa}$ (i.e., if R is an irreflexive relation), the set of the strings connecting a to a is the unit set $\{a\}$, which is regular. If aRa, then the set is $\{a, aa, aaa, \ldots\}$, which is regular, since it can be written A^*A where $A = \{a\}$.

INDUCTION STEP: $r > 1$.

CASE 1: $a = \bar{a}$. In this case any string connecting a to $\bar{a}$ is of the form

$$a \to a \to a \to \ldots a \to a,$$

where the number of $a \to$'s is ≥ 0 and each $\to$ represents independently the empty string (this being possible only if aRa) or a non-empty string without any a in it. Let $e_1, \ldots, e_g$ $(g \geq 0)$ be the states e such that aRe but $e \neq a$, and $f_1, \ldots, f_h$ $(h \geq 0)$ the states f such that fRa but $f \neq a$. Now any non-empty string represented by an $\to$ must start with one of $e_1, \ldots, e_g$ and end with one of $f_1, \ldots, f_h$. For each pair $e_i f_j$, by the hypothesis of the induction the set of

the strings connecting e_i to f_j without a in it is regular. Say $B_1, \ldots, B_{gh}$ are these regular sets; and let A be $\{a\}$. Now if aRa, the set of the possible strings $a \rightarrow$ is $A \vee A(B_1 \vee \ldots \vee B_{gh})$ (which reduces to A if $gh = 0$ or all the B's are empty); and if $\overline{aRa}$, it is $A(B_1 \vee \ldots \vee B_{gh})$ (which is empty if $gh = 0$ or all the B's are empty). Let this set be C. Then the set of the strings leading from a to a is C^*A (which reduces to A if C is empty).

CASE 2: $a \neq \bar{a}$. Now we have instead

$$a \rightarrow a \rightarrow a \rightarrow \ldots a \rightarrow a \dashrightarrow \bar{a}$$

where the number of $a \rightarrow$'s is ≥ 0 and each $\rightarrow$ and the $\dashrightarrow$ represents independently the empty string or a non-empty string without any a in it. If D is the set of the possible strings $a \dashrightarrow$, and $E = \{\bar{a}\}$, the set of the strings connecting a to $\bar{a}$ is C^*DE, which is regular.

With the proof of Lemma 7 done, Kleene is ready to complete the proof of Theorem 5.

PROOF OF THEOREM (completed). We need to show that, for a given state a and each of 2^k states $\bar{a}$, the state is a at time p and $\bar{a}$ at time 1, if and only if a certain regular event occurs over the time $1, \ldots, p$.

By the lemma, the set of the strings which can connect a to $\bar{a}$ is regular. Consider an expression for this regular set in terms of the empty set and the sets $\{a_i\}$ as the units (cf. 7.2). In this expression let us replace each unit $\{a_i\}$ by the unit set consisting of the $k \times 1$ table which (if $k > 0$) describes the external part of the state a_i, labeled initial or non-initial according to whether that unit $\{a_i\}$ was earliest or not. Each empty set as unit we replace by itself (but write it $\bar{I}$). There results a regular expression. The state changes from $\bar{a}$ at time 1 to a at time p, exactly if the event described by this regular expression occurs over the time $1, \ldots, p$.

Exercise 3.2. Consider Case 2 of the induction proof for Lemma 7. Is it justified to claim that C^*DE is regular? Carefully explain your answer.

Exercise 3.3. Compare critically Kleene's proof that "the events [languages] denoted by finite automata are regular" with the proof that you find

in your textbook. In your opinion, which proof is easier to follow? Which proof offers a more rigorous correctness argument? Explain your answers.

Exercise 3.4. Kleene shows that any finite automaton, as defined according to Kleene's definition, denotes a regular event. Re-prove the result for the modern day definition of finite automata by making use of Kleene's Lemma 7.

References

[1] J. E. Hopcroft, R. Motwani and J. D. Ullman, Introduction to automata theory, languages, and computation, 2nd Edition, Addison-Wesley, 2001.

[2] J. E. Hopcroft and J. D. Ullman, Introduction to automata theory, languages, and computation, Addison-Wesley, 1979.

[3] S. C. Kleene, "Representation of events in nerve nets and finite automata", in: C. Shannon and J. McCarthy, (eds.) *Automata Studies*, Princeton University Press, NJ, 1956, 3–41.

[4] W. S. McCulloch and W. Pitts, "A logical calculus of ideas immanent in nervous activity", *Bull. Math. BioPhys.*, **5** (1943), 115–133.

[5] M. O. Rabin and D. Scott, "Finite automata and their decision problems", *IBM Journal of Research and Development*, **3** (1959), 114–125.

[6] M. Sipser, Introduction to the theory of computation, 2nd Edition, Thomson Course Technology, 2006.

Notes to the Instructor

The project can be used in a senior undergraduate or beginning graduate level course on the theory of computation. Before working on the project, students are expected to have studied the concepts of finite automata and regular languages from a modern textbook. Students will learn that Kleene's original definition of a finite automaton is more liberal than the textbook's definition given by Rabin and Scott [5]. Also presented in the project is Kleene's proof of the equivalence between the expressiveness of regular expressions and finite automata.

The completion of the project requires two weeks. If the students are strong in mathematical thinking and proofs, an instructor may wish to assign the project as homework. Otherwise, the instructor can use some class time to go over materials in the project that the students find challenging. Students can work on the project individually or in groups of two or three. Exercises 2.1-2.5 can be completed in one week, while the rest of the exercises are completed in the second week.

Given that the exercises are open-ended questions, it is advised that an instructor should carefully work through the project before assigning it to the students.

Notes on Exercise 2.2:

The usual logic for addition requires the carry bit to be passed sequentially from the lower ordered digits to the higher ordered digits as the addition process is performed. Recall that, with Kleene's model, the logic for determining the i-th bit (an inner cell) at time t is said to be "determined by the states of all the cells [that include the input cells and inner cells] at time $t-1$". However, Kleene's model leaves it wide open how the states of all the cells at time $t-1$ decide the state of an inner cell at time t. So, the mechanism for determining the state of an inner cell can indeed be summarized by a sequential procedure (as employed in the addition logic).

Notes on Exercise 3.3:

The textbooks in [1] and [6] use the state elimination method in constructing a regular expression from a given finite automaton. The construction requires the introduction of the concept of generalized nondeterministic finite automata. In [2] a dynamic programming approach, that resembles Floyd-Warshall's algorithm for computing transitive closure of a graph, is

used. In contrast, Kleene's approach seems to be more in line with the usual mathematical induction technique in that the assertion is proved for a small number of states, and an induction step is used to prove the claim as the number of states increases. So, it is quite possible that the students will find Kleene's approach more natural assuming that they have had a good training in proofs by mathematical induction.