CSE 322 - Introduction to Formal Methods in Computer Science
Formal Definition of a Deterministic Finite Automata

Dave Bacon
Department of Computer Science & Engineering, University of Washington

I. EXAMPLE FROM MY YOUTH AND MY OLD AGE

Back in the old days, you know, before the Earth cooled, there were these games which we used to play called Text Adventure games. In these games, you would be confronted with a text prompt and could type in command like "GO NORTH" and the computer would reply "YOU CAN'T GO NORTH BECAUSE THERE IS A WALL THERE." So you might type "GO SOUTH" and the computer would reply "YOU ENTER A DARK DUNGEON WHERE PRISONERS HAVE RECENTLY BEEN TORTURED. THERE IS A KEY ON THE GROUND." So then you might type "GET KEY" and the computer would tell you "YOU PICKUP THE KEY, BUT DISCOVER TOO LATE THAT THE KEY IS ATTACHED TO A STRING WHICH PULLS A LEVER AND RELEASES A BOULDER WHICH SMASHES YOU FLAT. YOU ARE DEAD. PLAY AGAIN Y/N?" Now you make think that such a silly game would not suck you in and make you addicted to it, but I can tell you quite personally, that this is not true and these games can suck away large portions of your life.

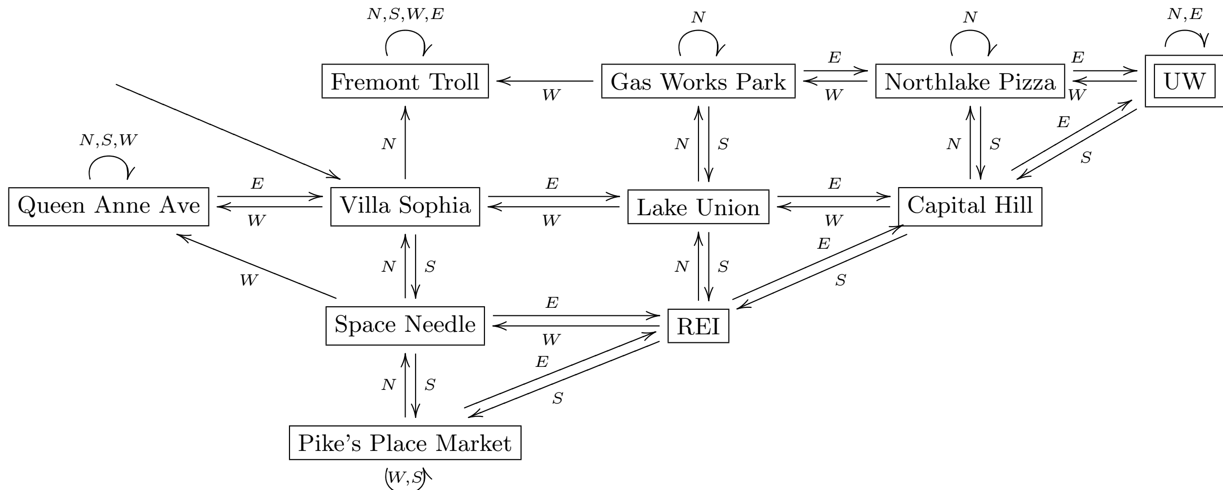

FIG. 1: Adventure: Seattle version. Ah, the good old days of texted based adventure games. Long live Zork!

Now for simple versions of these games you would often make up a little map of the dungeon or world you were exploring. The simplest such map might be a diagram like the one if Figure 1. This is for the game called "Get Dave to UW in the Morning" (alpha version.) The goal of this game, as the name implies, is to get from Dave's home "Villa Sophia" to that mysterious mystical place of higher education, "UW." Of course there are dreadful dangers in this trip. For example, if Dave encounters the "Fremont Troll" he gets stuck there staring at the tourists passing by and now matter what you do you cannot get Dave away from the troll!

Now we all know, informally, how to use the map in Figure 1. If we are at a given node, then depending on what we type into the computer, it will tell us where to go next: simply follow the arrow. Thus if we are at "Villa Sophia" and type N we go to the "Fremont Troll" (oh noes!) For any sequence of inputs to the computer, like for instance, *ESSN*, along with knowing where we currently are, we will can figure out where this will take us. For example if we input *ESSN* from the location "Gas Works," we follow the path "Gas Works"⇒"Northlake Pizza"⇒"Capital Hill"⇒"REI"⇒"Lake Union." Notice that we have listed the string we input *ESSN* from left to right in the order in which we input this string. Thus this means follow E first, then S, then S, and finally N. Finally note that on this map we have two special things appearing. One is an arrow coming from no-where. This is an arrow to indicate for our game where we are to begin the game. And finally we note that "UW" is enclosed in a double frame. This is to indicate that if we reach "UW" we win (yipee!)

Let's now get a little form about this game and begin by introducing the notions of alphabets and strings. Notice that the inputs you can enter into this game are drawn only from the set $\{N, E, W, S\}$. This set is the *alphabet* for our game. We often denote alphabets by the symbol Σ, so in this case we would say our alphabet is $\Sigma = \{N, E, W, S\}$. What can you do with an alphabet? Well write strings, of course! A *string* from an alphabet is a deterministic finite sequence of symbols from that alphabet. Thus for instance $NNEEWWSS$ is a string from our Σ. Given a string w, we denote the length of this string by $|w|$. Thus $|NNEEWWSS| = 8$. The empty string, i.e. the string made up of no elements of the alphabet, is denoted by ε. If w is of length n we write $w = w_1w_2w_3 \dots w_n$, where $w_i \in \Sigma$. String z is a *substring* of w if z appears consecutively within w. Thus EW is a substring of $NNEEWWSS$, but EWS is not a substring. Another concept which will be important for us will be the concatenation of two strings. The concatenation of x and y, written xy is the string by appending y on to the end of x. In other words if $x = x_1x_2 \dots x_m$ and $y = y_1y_2 \dots y_n$, then $xy = x_1x_2 \dots x_my_1y_2 \dots y_n$ and is a string of length $n + m$. A final concept which we will need is that of a *language*. A *language*, simply, is a set of strings. Thus a language over our alphabet might be $\{NES, SNES\}$ (that's the Nintendo language folks.)

So now we have described the possible inputs which we feed into the "Get Dave to UW in the Morning" game, we can move on to the reason for introducing this game. In particular the reason we have introduced this game is that this game can be played by a very simple controller which instantiates a deterministic finite automaton. The controller for this game, as we can guess from the above diagram only needs to keep a few bits in memory to store where you are on the map. In particular since there are 11 locations in the above map, we can use 4 bits ($2^4 = 16$ which is greater than 11) to store the location. The control thus keeps your current location in these four bits of memory, and then, depending on your next input, an element from the alphabet $\Sigma = \{N, E, W, S\}$, possibly changes these four bits to indicate your new location. Thus the controller is an example of a deterministic finite automaton, a machine with a limited memory, that can make transitions dependent on input and the current *state* of the limited memory.

Okay, so now lets talk a little bit about the map we've drawn above. This map is an example of a *state diagram* for a deterministic finite automaton (for now ignore that "determnistic" qualifier.) A state diagram is roughly defined as follows. It is a direct graph, a digraph (thus edges have directions.) Each of the nodes is labeled. These labels are called the *states* of the deterministic finite automaton. For our map game, the state is the location we are currently at. Further each directed edge, which are called *transitions*, is labeled. These labels are taken from an element of some alphabet. In our case the edges are labeled by an element of the set $\Sigma = \{N, E, W, S\}$. Further a state diagram has a special state, called the *start state*, which is indicated by having a directed edge pointing at in it from nowhere. Finally, a state diagram has *accept states*. These are denoted by doubling the frame of the box (or circle) for the state. For example, the "UW" state is an accept state in our above map.

So given a state diagram, we can now see how it can represent the action of a simple machine, a deterministic finite automaton, which processes some input and produces and output. The input is a string, like say, $EENE$, and the output is either *accept* or *reject*. (For now we will work with just this simple binary output: generalizations are, of course possible.) The automaton receives input from the string right to left one at a time. Being in a particular state, the deterministic finite automaton transitions along the edge labeled by the next input being received. If, after processing the entire string, the automaton is in a state which is an accept state, then we say that the automaton accepts this string. If, after processing the entire string, the automaton is in a state which is not an accept state, then we say the automaton rejects this string.

Just to be pedantic, lets work a simple example for the state diagram in Figure 1. Suppose that the input to the deterministic finite automaton represented by this state diagram is $EENE$. The automaton then proceeds as follows.

1. The start state is denoted by the state with an arrow coming from nowhere. In this case the start state is "Villa Sophia."
2. Read E, follow the transition from "Villa Sophia" to "Lake Union."
3. Read E, follow the transition from "Lake Union" to "Capital Hill."
4. Read N, follow the transition from "Capital Hill" to "Northlake Pizza." (mmm)
5. Read E, follow the transition from "Northlake Pizza" to "UW."
6. *Accept* because "UW" is an an accept state and we are at the end of the computation.

Finally you can see that for the above state diagram, for any string, the deterministic finite automaton will make transitions among states and end up in either an accept or reject state. We can then consider the set of strings for which the above machine ends up in the accept state of the deterministic finite automaton. We will call this the *language* of this particular deterministic finite automaton.

II. FORMAL DEFINITION OF A DETERMINISTIC FINITE AUTOMATON

State diagrams are easy to grasp intuitively, but this is a theory course, so we want to be more precise and give a formal definition of a deterministic finite automaton. This will remove any ambiguity we might have about deterministic finite automata and, even better, provide a nice clean language for talking about deterministic finite automata. A good notation will help clarify your thinking and lead you to much much less frustrated in life (believe me.) (On the other hand, rigor to the point of mortis is a trap you should also avoid falling into.)

The formal definition of a deterministic finite automaton has five parts. These five parts are the *states*, the *alphabet*, a *transition function*, the *start state* and the *set of accept states*. Why don't we just dive in and give the definition

A *deterministic finite automaton* is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a deterministic finite set called the *states*.
2. Σ is a deterministic finite set called the *alphabet*.
3. δ is a function from $Q \times \Sigma$ to Q, called a *transition function*.
4. $q_0 \in Q$ is the *start state*
5. $F \subseteq Q$ is the *set of accept states*.

Everything in this formal definition should make sense to you, with the possible exception, of the *transition function*. First of all you need to recall what the *Cartesian product* of two sets is. If S_1 and S_2 are deterministic finite sets then the Cartesian product of these two sets, denoted $S_1 \times S_2$ is the set of all pairs where the first element is from S_1 and the second element is from S_2. I.e. $S_1 \times S_2 = \{(x, y)|x \in S_1, y \in S_2\}$. For example if $S_1 = \{A, B\}$ and $S_2 = \{0, 1, 2\}$, then $S_1 \times S_2 = \{(A, 0), (A, 1), (A, 2), (B, 0), (B, 1), (B, 2)\}$. Back to that transition function. As you can read from the definition the transition function is a function from $Q \times \Sigma$ to Q. In other words it is a function which takes two inputs, the first from Q and the second from Σ and which outputs an element of Q.

Given the formal definition of a deterministic finite automaton, we already sort of know, from our intuitive picture, what each of the things in the definition is, with, again, perhaps, the transition function. The transition function encapsulates the transitions arrows in the state diagram. In particular we recall that a transition was a directed labeled edge, with the label being an element of the alphabet, Σ. In other words, given a node in the state diagram, i.e. a state, and an element of the alphabet Σ, a transition will give you a new state, which is the state and end of the appropriate transition edge for the given state. Thus, for example, if we look back at Figure 1, we will find that for the transition function for this deterministic finite automaton, $\delta(\text{“Villa Sophia''}, N) = \text{“Fremont Troll''}$.

The formal definition of a deterministic finite automaton puts to rest a few questions which you might have had about deterministic finite automata. For example, you might wonder if it is allowed to have a deterministic finite automaton which has no accept states. We see from the above definition, this is possible, because we could set $F = \emptyset$. You might also wonder if for every state, there must be at least one transition for every element of the alphabet. Indeed this is true because we have specified that δ is a function defined for every state and alphabet element.

A. Some simple examples of deterministic finite automaton to cut your teeth on

Okay lets give some simple deterministic finite automata to cut our teeth on. Consider the deterministic finite automaton with the following state diagram: To specify M formally we need to write down those five things which

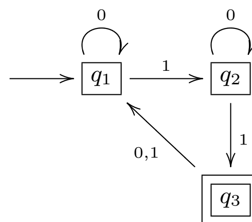

FIG. 2: deterministic finite automaton M

define the deterministic finite automaton: $M = (Q, \Sigma, \delta, q_0, F)$. So, lets do that

1. $Q = \{q_1, q_2, q_3\}$, the set of three states of our deterministic finite automaton.
2. $\Sigma = \{0, 1\}$, our alphabet, which is just the two binary values 0 and 1.
3. δ is a bit more difficult to specify since it is a function of two inputs and one output. We could do this by specifying each $d(q, x)$ but it is often more convenient to do this in the form of a table:

	0	1
q_1	q_1	q_2
q_2	q_2	q_3
q_3	q_1	q_1

4. The start state is $q_0 = q_1$.
5. There is only one accept state thus $F = \{q_3\}$.

Okay so it should be fairly straightforward to write down a deterministic finite automaton formal specification from a state diagram. But one question is what, exactly, does this deterministic finite automaton do? In particular we are mostly interested in the *language of machine M*. The language of a machine M is the set of all strings that machine M accepts. We write this language as $L(M)$ and say that M recognizes $L(M)$. While a machine may accept many strings, but it recognizes only one language. What is the language for the above machine? A little bit of thought will tell you that strings like those in the following set

$$E = \{w|w \text{ ends with a string of the form } 0^l 10^k 1, k, l \in \{0, 1, 2, \dots\}\}$$

are important. Here we introduced a useful notation for a string: x^k. The expression x^k is shorthand for x concatenated together $k \in \{0, 1, 2, \dots\}$ times:

$$x^k = \underbrace{xx \cdots x}_{k \text{ times}}$$

and where we define $x^0 = \varepsilon$. Indeed a little more thought will tell you that the strings in E are of the form:

$$A = L(M) = \{w|w = z_1 a_1 z_2 a_2 \dots a_{m-1} z_m, z_i \in E, a \in \Sigma\}$$

That's a crazy language. But there you have it.

III. A BRIEF TRIVIA INTERLUDE

The word *automaton* derives from the Greek word *automatas* ($\alpha\upsilon\tau o\mu\alpha\tau o\varsigma$) meaning "acting of ones own will."

IV. FORMAL DEFINITION OF COMPUTATION

We've described a deterministic finite automaton formally, via a 5-tuple. We can now proceed to make a formal definition for a deterministic finite automatons computation.

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a deterministic finite automata and let $w = w_1 w_2 \dots w_n$ be a string where each w_i is in Σ. Then we say that M *accepts* w if a sequence of states $r_0, r_1, \dots, r_n$ exists with the three conditions

1. $r_0 = q_0$
2. $\delta(r_i, w_{i+1}) = r_{i+1}$ for $i = 0, 1, \dots, n-1$
3. $r_n \in F$.

The first of these conditions says that r_0 is the start state. The second condition says that the state r_{i+1} is obtained from r_i after reading the character w_{i+1}. The final condition says that at the end of the computation the state of the deterministic finite automaton is an accept state. We say that M recognizes language A if $A = \{w|w \text{ accepts } w\}$. Languages which are recognized by a deterministic finite automaton are called *regular languages.*

A language is called a *regular language* if some deterministic finite automaton recognizes it.

We will be spending a lot of time with regular languages, but you might just ponder this for a second: are all languages regular?