Automata Theory

Jorge A. Samayoa
Industrial Engineering Department
Purdue University

David Reese
School of Aeronautics and Astronautics
Purdue University

1 Introduction & History

The theory of automata provides a basis for a tremendously useful branch of modeling and analysis: that of computational complexity. Automata theory describes the ability of abstract machines to evaluate mathematical problems given a set of inputs. Such machines surround us in the modern world; from traffic lights at intersections to test stand controllers for aerospace applications, various forms of automata observe, react to, and control the environment around us.

Automata theory possesses a noble history which spans several branches of mathematics and science. The mathematician Alan Turing developed a logical theory at Princeton in 1936 [1], describing a machine which consisted of:

> an infinite memory capacity obtained in the form of an infinite tape marked out into squares, on each of which a symbol could be printed. At any moment there is one symbol in the machine... called the scanned symbol. The machine can alter the scanned symbol and its behavior is in part determined by that symbol, but the symbols on the tape elsewhere do not affect the behaviour of the machine. However, the tape can be moved back and forth through the machine, this being one of the elementary operations of the machine. Any symbol on the tape may therefore eventually have an innings [2].

This theory is an example of the most abstract and capable automaton, the usefulness of which was not realized until much later. Two other men widely regarded as pioneers in this area were Warren McCulloch and Walter Pitts, a pair of neurophysiologists from the Massachusetts Institute of Technology. Their seminal 1943 paper, "A Logical Calculus Immanent in Nervous Activity", was the first to present a description of finite automata [3]. This theory was later broadened to include other types of automata by Bell Labs researchers George Mealy and Edward Moore in the mid 1950s.

In this paper, we will outline the most important types of automata as applicable to observing and controlling a large-scale system-of-systems, including those developed by the historical figures described above. An example of such a system will be presented using the framework of various automata, and the success of this modeling attempt evaluated. The results of this example indicate that SoS modeling is useful for initial evaluation of high-level and low-level elements of SoS. However, to truly capture emergent behavior and unplanned interactions between the systems, more advanced modeling techniques are required.

2 Automata

An *automaton* is an abstract object which recognizes or accepts, in a discrete time steps, a string of characters from a particular set (Σ^*), where $\Sigma \supset \Sigma^*$ is a finite alphabet. Given that the elements accepted by the automaton are a subset of Σ^*, we say that these elements, called words, form a language. Therefore, an automaton, M, recognizes or accepts a language, $L(M)$, contained in Σ^*. The following definition of an automaton is similar to that presented by Jirí Adámek and Vera Trnková [4].

Definition 1. An automaton is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where
Q is a set, called the *set of states*;
Σ is a non-empty set, called the *input alphabet*;
$\delta : Q \times \Sigma \to Q$ is the *transition function*;
$q_0 \in Q$ is the *initial state*;
$F \subseteq Q$ is the set of *accept states.*

There are several different types of automata, varying in their types of transitions, the number of states which they contain, the type of inputs they require, and the type of acceptance conditions they allow. Here, we present a few which are considered crucial types in the study of automata theory.

2.1 Types of Automata

2.1.1 Deterministic Finite Automata (DFA)

The main difference between the definition outlined above and the formal definition of a Deterministic Finite Automata is that the cardinality of Q is finite, and the transition function δ is one-to-one, i.e., for each input symbol, the machine's next state may be one, and only one, of its other states. Therefore, the definition of a DFA is as follows:

Definition 2. A Deterministic Finite Automaton M is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states; Σ is a non-empty finite set, called the input alphabet; $\delta : Q \times \Sigma \to Q$ is the transition function; $q_0 \in Q$ is the *initial state*; $F \subseteq Q$ is the set of accept states.

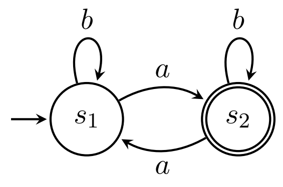
The input of δ is a letter of Σ and a state belonging to Q. The output is a state of Q (possibly the same one). If the automaton is in state q and reads the letter r, then (q, r) is the input for δ and $\delta(q, r)$ is the next state. Given a non-empty string in $\Sigma^* \subset \Sigma = \{a, b\}$ the automaton reads the string or word as follows:

1. It begins in the initial state q_0, and **reads** the first letter in the string. If the first letter is $a \in \Sigma$, then it **moves** to state $s_1 = \delta(q_0, a)$.
2. The automaton reads the the second letter of the string. If the second letter is b, then the automaton moves to state $s_2 = \delta(s_1, b)$.
3. As the automaton continues to read the given string of letters from the alphabet, it moves from one state to another. Eventually, the automaton reads every letter in the string and then **stops**.
4. After reading the last letter of the string, if the current state belongs to the set of acceptance states, then the automaton **accepts** the string. Otherwise, it rejects it.

Example 1. Let $M = (Q, \Sigma, \delta, q_0, F)$, where
$Q = \{s_1, s_2\}$, $\Sigma = \{a, b\}$, $q_0 = s_1$, $F = \{s_2\}$, and
δ is defined by the following transition table:

	$\boldsymbol{a}$	$\boldsymbol{b}$
$\boldsymbol{s_1}$	s_2	s_1
$\boldsymbol{s_2}$	s_1	s_2

The **state diagram** is given by



Note that we identify q_0 with an **arrow**, and the elements of F with a **double circle**.
Some of the strings recognized by this automaton are *a, ab, aaabb, bbaaa, baaab, bbbabb* and *baababb*. In general, the language recognized by this automaton is a regular language [11] given by $b^*a(ab^*ab^*)^*b^*$, where * is the *Kleene star*[*] (a^* denotes any non-negative number (including zero) of symbols a).

Theorem 1. A language is regular if and only if it is accepted by an automaton.

Proof can be found in [5].

2.1.2 Nondeterministic Finite Automata (NFA)

A nondeterministic Finite Automaton is a finite state machine very similar to a DFA. The main difference is that the transition function δ is not one-to-one, i.e., for each input symbol, its next state may be any one of several possible states. Thus, the next state is an element of 2^S, where S is the number of states. With this in mind, the formal definition of a NFA is as follows:

Definition 3. Let λ be the empty string. A Deterministic Finite Automaton M is a quintuple $M = (Q, \Sigma, \Delta, q_0, F)$, where Q is a finite set of states; Σ is a non-empty finite set, called the input alphabet; $\Delta : Q \times (\Sigma \cup \{\lambda\}) \to 2^Q$ is the transition function; $q_0 \in Q$ is the *initial state*; $F \subseteq Q$ is the set of accept states.

NFAs and DFAs work in a similar fashion, with the following main differences:

- The number of possible candidate states to which a move can occur after each step can be greater than one.
- The automaton can move from one state to another using the empty string.
- The automaton accepts a string if the machines stops in any accept state, or can move to another accept state by the empty string λ.

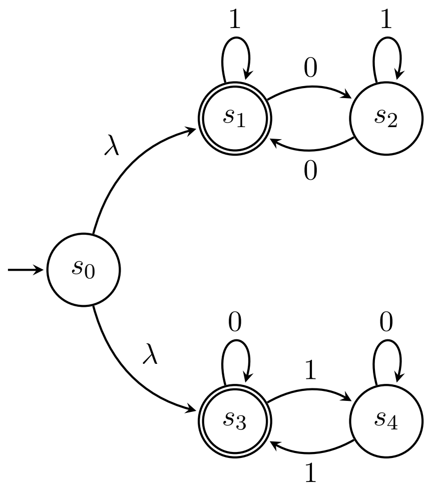
In fact, the automaton M accepts the string (word) if a sequence of states, $r_0, r_1, \ldots, r_n$ exists in Q with the following conditions [10]:
$r_0 = q_0$
$f_{i+1} \in \Delta(r_i, a_{i+1})$, for $i = 0, \ldots, n-1$
$r_n \in F$.
This conditions are suggesting that the automaton has to start in q_0. Transitions are ruled by the transition relation Δ, and the automaton will accept an string w if the last input of w causes the machine to stop in one of the accepting states. Otherwise, the string is rejected by the automaton.

Example 2. [†] Let $M = (\{s_0, s_1, s_2, s_3, s_4\}, \{0, 1\}, \Delta, s_0, \{s_1, s_3\})$, where the transition relation is given by the following table:

[*]ref. `http://nayuki.eigenstate.org/page/countable-sets-and-kleene-star`
[†]This example is a modification (in notation) of an example found in `http://en.wikipedia.org/wiki/Nondeterministic_finite_automata`

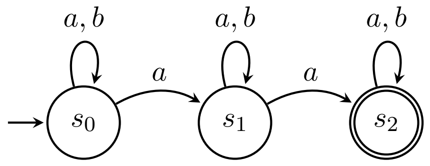
	0	**1**	**λ**
s_0	$\{\}$	$\{\}$	$\{s_1, s_3\}$
s_1	$\{s_2\}$	$\{s_1\}$	$\{\}$
s_2	$\{s_1\}$	$\{s_2\}$	$\{\}$
s_3	$\{s_3\}$	$\{s_4\}$	$\{\}$
s_4	$\{s_4\}$	$\{s_3\}$	$\{\}$

The **state diagram** is given by



Some of the strings recognized by this automaton are *λ,11,000,1010,0101* and *001100*. In general, the language recognized by this automaton is a regular language given by (1*(01*01*)*) $\cup$ (0*(10*10*)*).

Example 3. This example is just to show another way of representing the state diagrams. The characteristics of the automaton can be drawn from the diagram.



The language recognized by this automaton is $(a \vee b)^* a(a \vee b)^* a(a \vee b)^*$.

2.1.3 Turing Machine

A Turing Machine, named after the mathematician Alan Turing [‡], is an abstract machine capable of recognizing and manipulating a language. In the context of Turing Machines, we assume that the input strings are on a strip of tape, and the machine can erase, print and re-print symbols on the strip [12].

Definition 4. A **Deterministic** Turing Machine is a quintuple $(Q, \Sigma, \Gamma, \delta, s_0, h)$, where where Q is the set of states, Σ is a finite set of tape symbols, which includes the alphabet and #, s_0 is the starting state, h is the halt state, and δ is a function from $Q \times \Gamma$ to $Q \times \Gamma \times N$ where N consists of either L, which indicates a movement on the tape one position to the left, R, which indicates a movement on the tape one position to the right, or #, which indicates that no movement takes place.

Example 4. An example of a rule is (s_1, a, s_2, b, L), which means that if the machine is in state s_1 and reads the letter a, it is to change to state s_2, print the letter b in place of the letter a and move one square to the left.

[‡] `http://en.wikipedia.org/wiki/Alan_Turing`

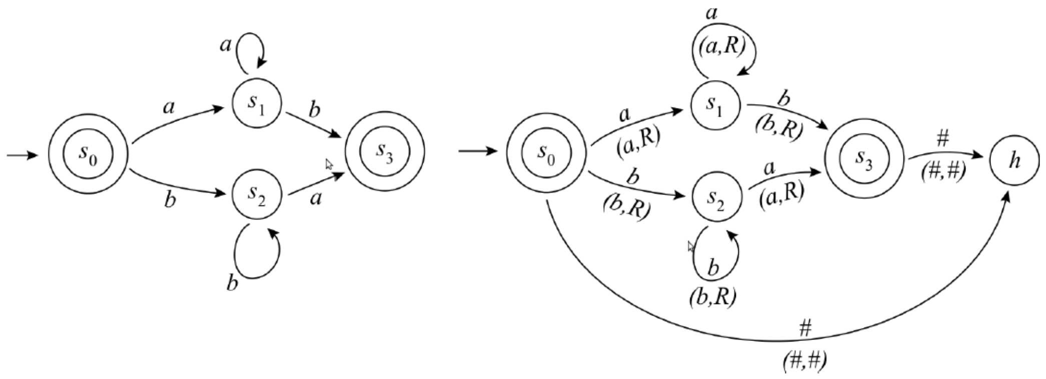


Figure 1: (left) DFA; (right) Turing Machine modeled as an automata.

Example 5. The rule $(s_1, a, s_2, \#, R)$ means that if the machine is in state s_1 and reads the letter a, it changes to state s_2 , erases the a and moves one square to the right.

Example 6. The rule $(s_1, \#, h, \#, \#)$ means that if the machine is in state s_1 and reads a *blank* then it **halts**, and thus does not print anything or move the position on the tape.

For more examples of Turing Machines, we encourage the reader to visit `http://www.cs.odu.edu/~toida/nerzic/390teched/tm/definitions.html`.

Theorem 2. Every regular language is recognized by a Turing Machine.

Proof can be found in [5].

Assuming the previous theorem, and using the previous result that an automaton recognizes a regular language, Fig. 1 illustrates a DFA and its equivalent Turing Machine representation.

2.1.4 Other Types of Automata

There are several other types of automata having different characteristics and uses, some of which include Pushdown Automata, Cellular Automata, Timed Automata, Linear Bounded Automata, Rabin Automata (deterministic and nondeterministic), Moore Automata (deterministic and nondeterministic), and Muller Automata (deterministic and nondeterministic).

2.2 Applications

2.2.1 Biology

Biology is one of the areas where Automata theory has had more impact. This is due to the capability of self-replication posessed by some automata (e.g., cellular automata). Even though no actual biological mechanism has an infinite structure, the huge complexity of these mechanisms has driven scientists to apply results of infinite-state automata to areas such as genome structure, biochemical reactions and evolution [6]. Figure 2 shows an excitable automata, presented in the paper "*Cellular Automata Approaches to Biological Modeling*," by Ermentrout and Edelstein-Keshet [7].

One of the most well-known applications of automata theory widely used not only in biology, but also in several other sciences, is John Conway's Game of Life [8]. In fact, Paul Rendell[§] designed a Turing Machine in Conway's Game of Life.

[§]ref. `http://rendell-attic.org/gol/tm.htm`

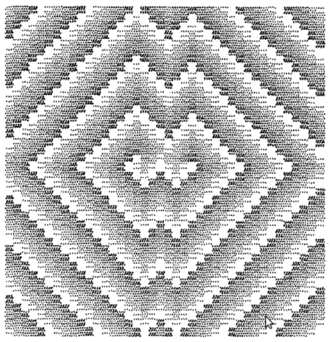


Figure 2: “Spiral” wave pair in an excitable automata.

2.2.2 Embedded Hardware Design

Perhaps the most obvious application of deterministic finite automata show up in hardware design for embedded computing systems. Such automata have tremendous control over many aspects of our day-to-day lives, as they show up in a variety of devices. Deterministic finite automata are capable of accepting inputs from the surrounding world and acting upon them in a pre-ordained fashion, enabling their use in applications ranging from home appliances to missile guidance systems. An example of such an automaton for a traffic light is shown in Fig. 3. This example could be expanded to include additional external inputs, e.g., to detect whether a backup of cars exists at a red light.

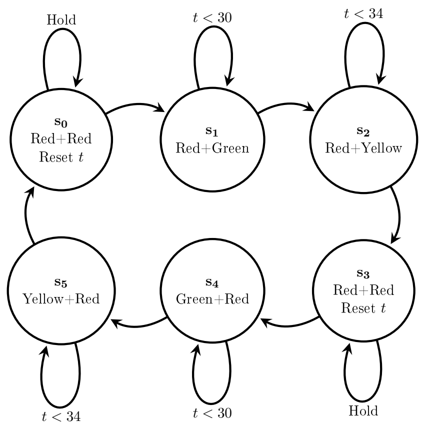


Figure 3: Sample DFA state machine for a 30 second green/4 second yellow timed traffic light.

2.3 Limitations of Automata

Automata were developed to model finite state machines as an approach to facilitate the study of properties of real machines. But, in the particular case of a DFA or NFA, these automata have limited power in the languages they can recognize. One simple example is the language $a^n b^n$, having the same number of a as b; this language cannot be recognized by either a DFA or NFA. In general, automata have played a major role in theory of computation and compiler design, but the limitation of the computational power have made scientists come up with modifications of them, such as cellular automata, to make them more applicable to real world situations. Automata theory presents a poor toolset for capturing emergent behavior. While it is possible for a Turing Machine to learn from its past experience (by, e.g., writing to its stack), the timescale for such development and the knowledge to ultimately act on this experience is potentially longer than the model may be able to capture. Additionally, automata may become cumbersome and computationally intensive for large models where many moving parts exist; in parallel complex systems, care must be taken to avoid programming a so-called "race condition", where faster nodes of the machine continue execution while slower nodes attempt to catch up, potentially causing pseudo-stochastic behavior.

3 Application to Systems of Systems: Carrier Flight Operations

3.1 Problem Scope & Taxonomy

For this example, we will consider a naval carrier strike group (CSG) as the system-of-systems, with the taxonomy laid out as in Table 1.

Table 1: Taxonomy for CSG System-of-Systems Analysis

	Resources	Operations	Policies	Economies
α elements	Aircraft Ordinance	Pilot Capabilities		
β elements	Shipboard Aircraft		Service Capabilities	
γ elements	Aircraft Carriers	Ship Captain		
δ elements	Carrier Fleet		Commandant	Authorization

This problem fits Maier's key traits of Systems of Systems [9]. Each level of hierarchy can exhibit operational independence; aircraft can function without a ship, for example. The system also exhibits managerial independence via the captain (a γ-level participant). All intermediate forms are stable; should aircraft or even entire carriers become disabled, the fleet contains enough redundancy to continue on. Additionally, should communication between the various levels of the SoS be disrupted, enough sentience is present at each level to allow the system to continue to operate. The CSG is a strongly centrally directed system, with highly leveraged interfaces – information transfer is critical between the system levels in order for emergent behavior to be captured fully. Finally, collaboration is highly incentivized to prevent loss of assets.

3.2 Inputs & Outputs

As automata are effectively intelligent state machines, it is useful to examine the modeling problem using a top-down approach to determine the necessary transition logic and state functions necessary to describe the system-of-systems. Due to the complex nature of this large SoS, Turing Machine logic is likely the most efficient means of analysis, allowing the system to act upon, modify, and store various sets of inputs.

At the δ **level**, the CSG must be deployed according to the rules established by the commandant, subject to the authorization provided to it by the government. These two input variables are both stochastic mechanisms which may generate a wide range of conditions. The result of these inputs would command number of ships to be deployed, deployment locations, and capabilities of the ships deployed.

γ**-level components** include the carrier ships themselves and their captains. The long timescale behavior of the carriers can be modeled as a deterministic finite automaton – given a heading and speed, its behavior is fairly predictable. The captain is responsible for ensuring that this behavior occurs, and is hopefully capable of maintaining the on-design case. Additionally, the captain serves as a critical point of feedback from the β level to the δ level, and vice versa; he is responsible for order and results communication across the SoS, on timescales ranging from short (β-level) to long (γ and δ level).

β**-level components** include manned aircraft, unmanned aircraft, and the agents which control these. Ideally, the β-level agents can operate as a deterministic finite state machine, with a known progression of states for a given set of inputs. However, representing activity at this layer as such may limit the ability of the model to capture emergent behavior, which is key to the flexibility of the entire SoS.

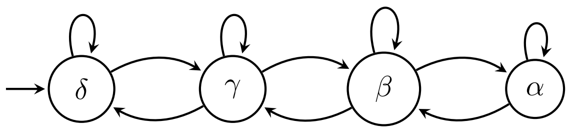


Figure 4: Long-timescale automata for CSG modeling.

α-level components exhibit the shortest timescales, but are ultimately responsible for achieving the goals of the SoS. The primary resource at this level is the aircraft ordinance itself, functioning in tightly controlled, short timescale deterministic finite automata, designed to respond in a variety of situations predicted during flight (drop, ignite, guidance, burnout, separation, kill). These aspects of the level are applied using the limitations of a human pilot.

3.3 Effectiveness of Automata Modeling

Of note is that this theory applies best at very low (i.e., α) or very high (i.e., δ) levels of the SoS, where the timescales are finite and little interaction is required between the SoS and the elements themselves. However, at the central layers (β, γ), a wide variety of information transfer between the levels is critical at timescales varying from seconds to months. The model automata should be able to capture such behavior, but ensuring optimal model performance across such wide timescales may make the problem difficult.

The theory of automata does manage to capture a good portion of Maier's essential characteristics for an SoS [9]. Most critically, each individual automaton in the system must exhibit managerial and operational independence in order to operate. By extension, they also exhibit stable intermediate forms (given individual inputs, of course). Collaboration and leveraged interfaces are also encouraged, as each automaton is able to capture input from the previous level, and is able to function more optimally given such inputs. The only drawbacks to the application of automata are the inability for each element to perform policy triage on its surroundings, and to capture emergent behavior fully.

4 Conclusions

Automata Theory provides a good famework to work several abstract problems that have led to the study and analysis of complex structures and systems. Even though automata can capture behavior at consistent timescales, they are not as efficient at managing lots of information transfer at varying timescales.

Ideally, the entire SoS operates as a "well-oiled machine", which is the nominal case for modeling with automata theory. However, the human factors and emergent behavior present in such a large CSG are bound to confound even the most well-intentioned programmers with possibilities, leaving some depths of SoS behavior unplumbed.

References

[1] Kowalik, J. M. "Alan Mathison Turing". Virginia Tech: 1995. http://ei.cs.vt.edu/ history/Turing.html, last accessed 6 Feb 2012.

[2] Turing, A. M. "Intelligent Machinery." Reprinted in "Cybernetics: Key Papers." Ed. C.R. Evans and A.D.J. Robertson. Baltimore: University Park Press, 1968. p. 31.

[3] Roberts, E. "Basics of Automata Theory". Stanford University: 2005. http://www-cs-faculty.stanford.edu/ eroberts/courses/soco/projects/2004-05/automata-theory/basics.html, last accessed 6 Feb 2012.

[4] Adámek, J; Trnková, V;, "*Automata and Algebras in Categories*," Mathematics and its applications. East European series; 37.

[5] Anderson, J.A.;, Cambridge University Press, New York, 2006.

[6] Baer, R. M. and H. M. Martinez; "*Automata and biology,*" Ann, Rev. Biophys. 3,255. 1974.

[7] Ermentrout, G. B., and L. Edelstein-Keshet. "*Cellular Automata Approaches to biological modeling*". Journal of Theoretical Biology 160 (1): 97-113. 1993.

[8] Gardner, Martin. "*Mathematical Games - The fantastic combinations of John Conway's new solitaire game "life"*". pp. 120-123. ISBN 0894540017. 1970.

[9] Maier, M. W.;, "*Architecting Principles for Systems-of-Systems*," John Wiley & Sons, Inc., Syst Eng 1: 267-284, 1998.

[10] Rabin, M. O.; Scott, D.;, "*Finite Automata and Their Decision Problems*," IBM Journal of Research and Development , vol.3, no.2, pp.114-125, April 1959

[11] Sipser, M.;, "*Introduction to the Theory of Computation*," PWS Publishing. ISBN 0-534-94728-X. pp.31-90, 1997.

[12] Turing, A.M.;, "*Computing Machinery and Intelligence*," Oxford University Press, Mind, New Series, Vol. 59, No. 236 (Oct., 1950), pp. 433-460.