# Applications of Deterministic Finite Automata

Eric Gribkoff
ECS 120
UC Davis

Spring 2013

## 1 Deterministic Finite Automata

Deterministic Finite Automata, or DFAs, have a rich background in terms of the mathematical theory underlying their development and use. This theoretical foundation is the main emphasis of ECS 120's coverage of DFAs. However, this handout will focus on examining real-world applications of DFAs to gain an appreciation of the usefulness of this theoretical concept. DFA uses include protocol analysis, text parsing, video game character behavior, security analysis, CPU control units, natural language processing, and speech recognition. Additionally, many simple (and not so simple) mechanical devices are frequently designed and implemented using DFAs, such as elevators, vending machines, and traffic-sensitive traffic lights.

As the examples below will demonstrate, DFAs naturally lend themselves to concisely representing any system which must maintain an internal definition of state. Our examples begin with vending machines, which need to remember how much money the user has input, and continue to more complicated examples of video game agent AI and communication protocols. As our final example, we will consider the incorporation of finite state machines into the Apache Lucene open-source search engine, where they are used to implement search term auto-completion.

Formally, a deterministic finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ such that:

1. $Q$ is a finite set called the **states**
2. $\Sigma$ is a finite set called the **alphabet**
3. $\delta : Q \times \Sigma \to Q$ is the **transition function**

4. $q_0 \in Q$ is the **start state**

5. $F \subseteq Q$ is the **set of accept states**

We also discuss Mealy machines, which add to the expressiveness of DFAs by producing output values at each transition, where the output depends on the current state and input. Rather than accepting or rejecting strings, Mealy machines map an input string to an output string. They can be thought of as functions that receive a string and produce a string in response.

Formally, a Mealey machine is a 6-tuple $(Q, \Sigma, \Gamma, \delta, \omega, q_0)$ such that:

1. $Q, \Sigma, \delta$, and $q_0$ are defined as in a DFA.

2. $\Gamma$ is a finite set called the **output alphabet**

3. $\omega : Q \times \Sigma \to \Gamma$ is the **output function**

# 2 A Non-Exhaustive List of DFA Applications

- Vending Machines
- Traffic Lights
- Video Games
- Text Parsing
- Regular Expression Matching
- CPU Controllers
- Video Games
- Protocol Analysis
- Natural Language Processing
- Speech Recognition

# 3 Vending Machines

Figure 1 presents a DFA that describes the behavior of a vending machine which accepts dollars and quarters, and charges \$1.25 per soda. Once the machine receives at least \$1.25, corresponding to the blue-colored states in the diagram, it will allow the user to select a soda. Self-loops represent ignored input: the machine will not dispense a soda until at least \$1.25 has been deposited, and it will not accept more money once it has already received greater than or equal to \$1.25.

To express the DFA as a 5-tuple, the components are defined as follows:

1. $Q = \{\$0.00, \$0.25, \$0.50, \$0.75, \$1.00, \$1.25, \$1.50, \$1.75, \$2.00\}$ are the **states**
2. $\Sigma = \{\$0.25, \$1.00, \mathit{select}\}$ is the **alphabet**
3. $\delta$, the **transition function**, is described by the state diagram.
4. $q_0 = \$0.00$ is the **start state**
5. $F = \emptyset$ is the **set of accept states**

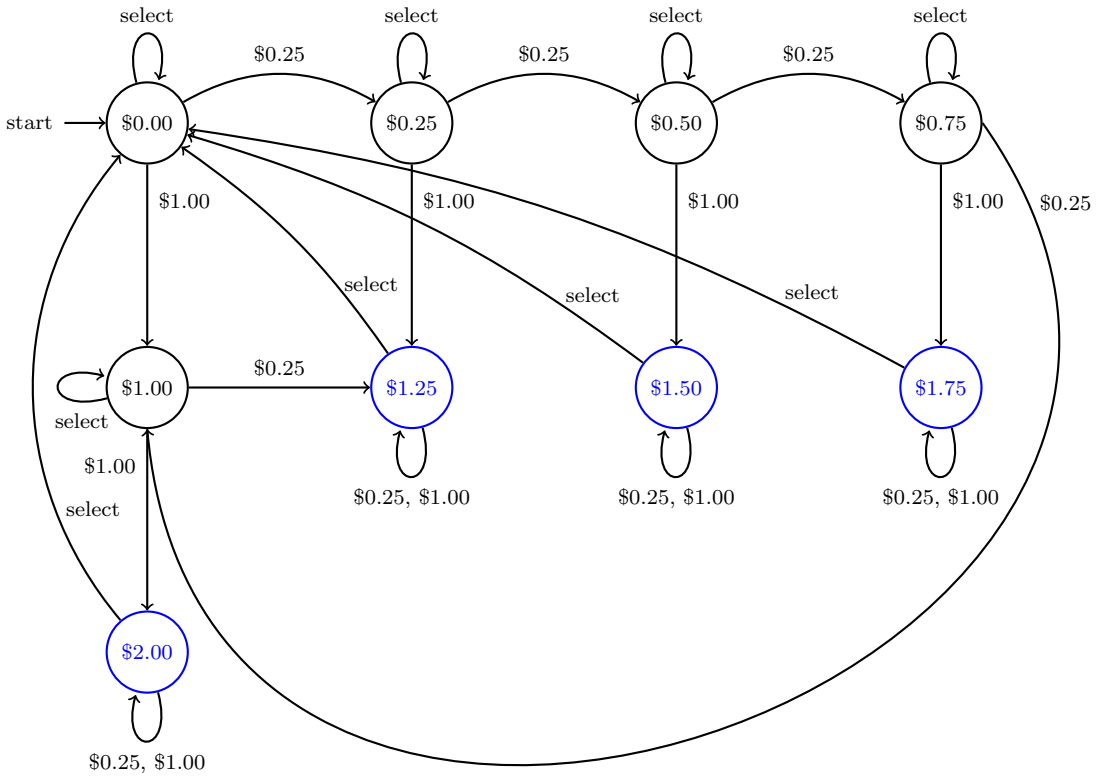


Figure 1: Vending Machine State Diagram

# 4 AI in Video Games: Pac-Man's Ghosts

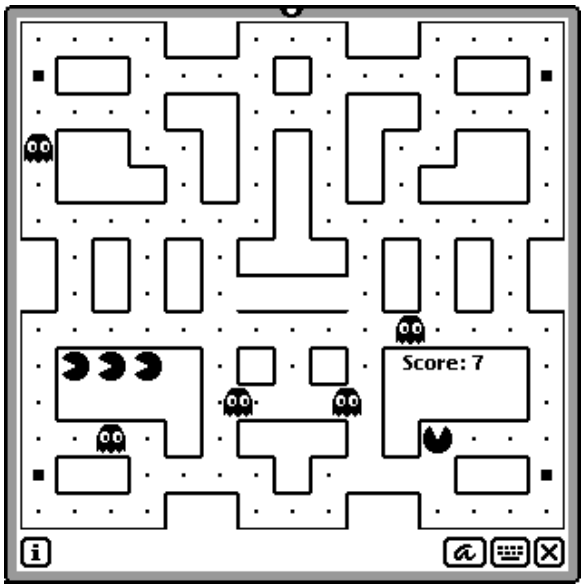


Figure 2: Screenshot of a Pacman Clone

Finite state machines lend themselves to representing the behavior of computer-controller characters in video games. The states of the machine correspond to the character's behaviors, which change according to various events. These changes are modeled by transitions in the state diagram. State machines are certainly not the most sophisticated means of implementing artificially intelligent agents in games, but many games include characters with simple, state-based behaviors that are easily and effectively modeled using state machines.

Here we consider the classic game, Pac-Man. For those unfamiliar with the game-play, Pac-Man requires the player to navigate through a maze, eating pellets and avoiding the ghosts who chase him through the maze. Occasionally, Pac-Man can turn the tables on his pursuers by eating a power pellet, which temporarily grants him the power to eat the ghosts. When this occurs, the ghosts' behavior changes, and instead of chasing Pac-Man they try to avoid him.

The ghosts in Pac-Man have four behaviors:

1. Randomly wander the maze

2. Chase Pac-Man, when he is within line of sight

3. Flee Pac-Man, after Pac-Man has consumed a power pellet

4. Return to the central base to regenerate

These four behaviors correspond directly to a four-state DFA. Transitions are dictated by the situation in the game. For instance, a ghost DFA in state 2 (Chase Pac-Man) will transition to state 3 (Flee) when Pac-Man consumes a power pellet.

For a further discussion of state machines for game AI, see `http://research.ncl.ac.uk/game/mastersdegree/gametechnologies/aifinitestatemachines/`.

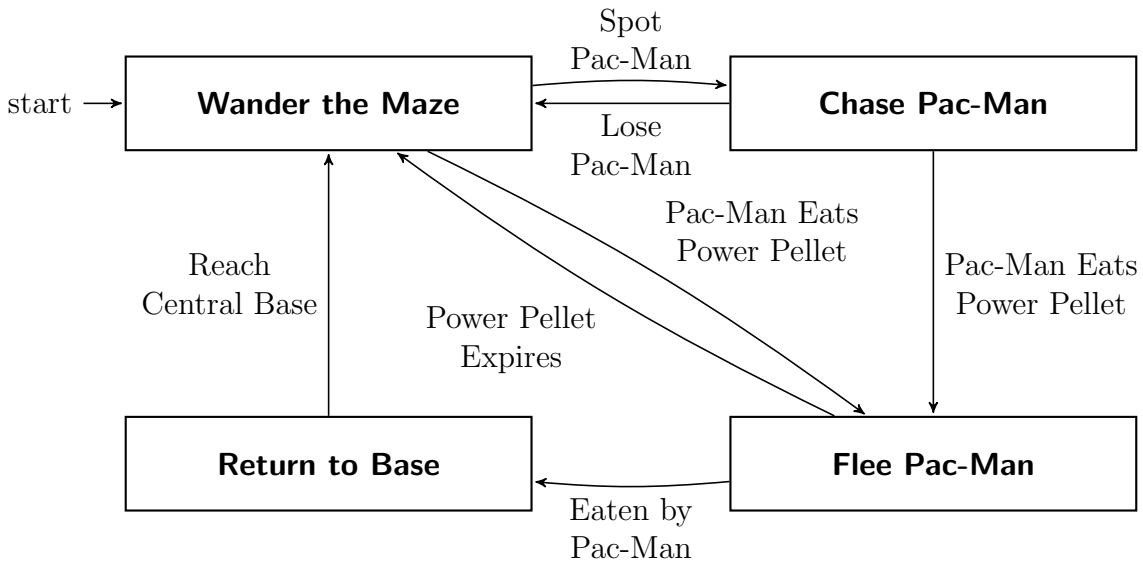


Figure 3: Behavior of a Pac-Man Ghost

# 5 Internet Protocols: TCP as a DFA

Internet protocols also lend themselves to descriptions as DFAs. The state diagram below represents a simplified version of the Transmission Control Protocol (TCP).

The state machine in Figure 4 describes the "life stages" of a TCP connection. A connection between two computers begins in the closed state. Following a series of signals, described by the transition arcs of the diagram, the two machines reach the established state where communication can proceed. Once completed, the transition arcs describe the process whereby the connection returns to the closed state.

See `http://www.tcpipguide.com/free/` for further discussion of the TCP protocol, and `http://www.texample.net/tikz/examples/tcp-state-machine/` for the source used to generate the state machine diagram.

Representation of complex protocols, such as TCP or Transport Layer Security (TLS), as DFAs helps in understanding the protocols and aids implementations, as the state diagrams clearly illustrate allowed and disallowed transitions between states. In addition, finite state security analysis has been used to examine the security of protocols such as SSL and TLS.

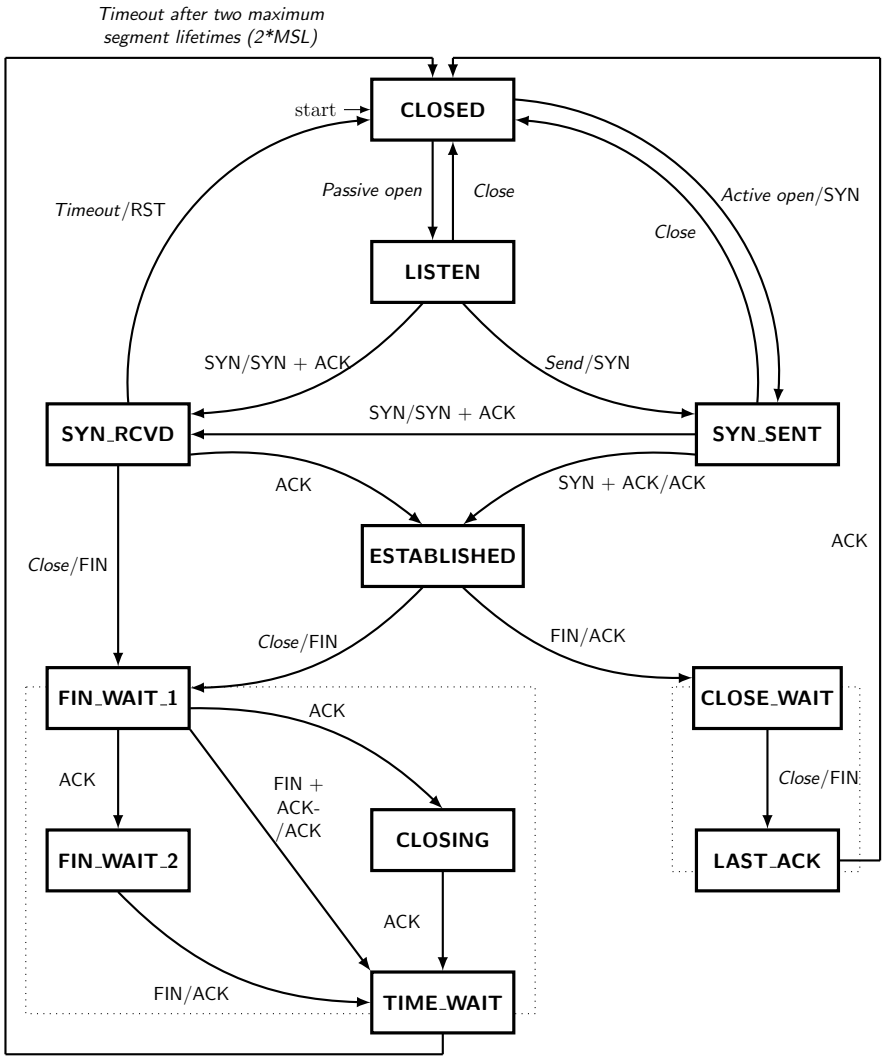


Figure 4: TCP State Diagram

# 6 Auto-Complete in Apache Lucene

Simple extensions to the DFA definition can greatly expand their power while preserving their ability to concisely encapsulate ideas and processes. One such extension is the finite state transducer (FST), which enhances the DFA definition by adding an output capability. Mealy machines, defined above, are a type of FST.

The figure below illustrates the use of a Mealy machine to index strings in a sorted array. Words are matched to their index in the search term data structure by processing the words through the Mealy machine and summing the output values encountered. (Empty outputs are omitted from the diagram.)

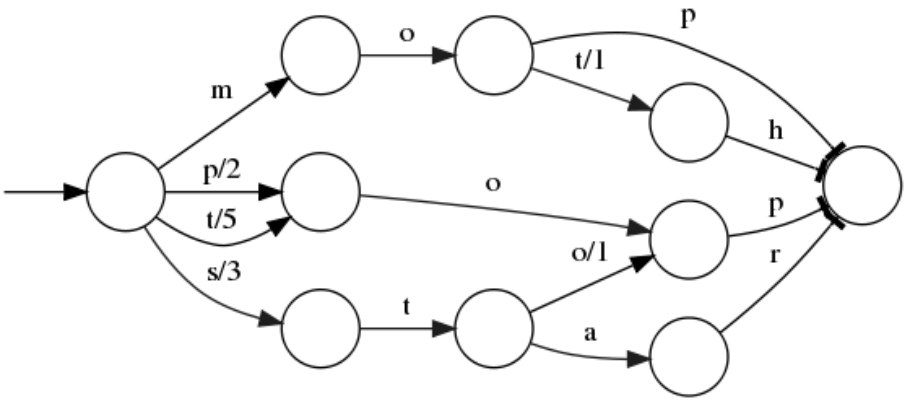


Figure 5: FST for Text Lookups

This Mealy machine proceses the words in the ordered set $\{mop, moth, pop, star, stop, top\}$ to produce the numbers corresponding to their alphabetical order within the set. Consider *stop*, the fifth (index = 4) element of the set. Beginning with the start state, we follow the *s* arc. This arc has associated output 3. Then we follow the *t* arc, with no output. Then we follow the *o* arc, with output 1. Our sum currently stands at four. Then we follow the *p* arc, with no output, and reach the end of the word. Our sum gives us the index of *stop*: 4.

The advantage of the FST approach is that the common prefixes and suffixes of stored terms are shared, greatly reducing the memory footprint required for text lookups.

The Apache Lucene project used a Mealy machine to dramatically decrease the memory footprint of their auto-complete search term feature. By using an FST to conduct lookups in its search term index, the Apache Lucene project is able to store the entire index in memory, resulting in much faster lookups. More information on the approach adopted by the Lucene project can be found at `http://blog.mikemccandless.com/2010/12/using-finite-state-transducers-in.html`.