EXAMPLE 5.29 A Typical United States Traffic Light

Consider the following controller for a single direction of a very simple U.S. traffic light (which ignores time of day, traffic, the need to let emergency vehicles through, etc.). We will also ignore the fact that a practical controller has to manage all directions for a particular intersection. In Exercise 5.30, we will explore removing some of these limitations.

The states in this simple controller correspond to the light's colors: green, yellow and red. Note that the definition of the start state is arbitrary. There are three inputs, all of which are elapsed time.

EXAMPLE 5.30 Generating Parity Bits

The following Mealy machine adds an odd parity bit after every four binary digits that it reads. We will use the notation a/b on an arc to mean that the transition may be followed if the input character is a. If it is followed, then the string b will be generated.

EXAMPLE 5.31 A Bar Code Reader

Bar codes are ubiquitous. We consider here a simplification: a bar code system that encodes just binary numbers. Imagine a bar code such as:

![Image of a bar code]

It is composed of columns, each of the same width. A column can be either white or black. If two black columns occur next to each other, it will look to us like a single, wide, black column, but the reader will see two adjacent black columns of the standard width. The job of the white columns is to delimit the black ones. A single black column encodes 0. A double black column encodes 1.

We can build a finite state transducer to read such a bar code and output a string of binary digits. We'll represent a black bar with the symbol B and a white bar with the symbol W. The input to the transducer will be a sequence of those symbols, corresponding to reading the bar code left to right. We'll assume that every correct bar code starts with a black column, so white space ahead of the first black column is ignored. We'll also assume that after every complete bar code there are at least two white columns. So the reader should, at that point, reset to be ready to read the next code. If the reader sees three or more black columns in a row, it must indicate an error and stay in its error state until it is reset by seeing two white columns.

EXAMPLE 5.32 Letter Substitution

When we define a regular language, it doesn't matter what alphabet we use. Anything that is true of a language L defined over the alphabet \( \{a, b\} \) will also be true of the language \( L' \) that contains exactly the strings in \( L \) except that every a has been replaced by a 0 and every b has been replaced by a 1. We can build a simple bidirectional transducer that can convert strings in \( L \) to strings in \( L' \) and vice versa.
This technique seems quite different from the first. It can easily be implemented, even by a child, for small values of $n$. But it actually requires exactly the same number of moves as does the recursive technique. And no better technique exists.

So the shortest solution to the 64-disk problem is very long. Nevertheless, the system of poles and disks can easily be modeled as a non-deterministic finite state machine. The start state is the one in which all 64 disks are stacked on the first pole. The accepting state is the one in which all 64 disks are stacked properly on the goal pole. Because there is a finite number of disks and the position of each disk can be uniquely described by naming one of the three poles, the number of distinct states of the system, and thus of the machine we'll build to model it, is finite. Finite but not tractable: This system has $3^{64}$ states (because each of the 64 disks can be on any one of the three poles). The transitions of the machine correspond to the legal moves (i.e., those that satisfy the rule that all disks must be on a pole and that no disk may be on top of a smaller one). Each transition can be labeled with one of six symbols: 12 (meaning that the top disk from pole 1 is removed and placed on pole 2), 13, 21, 23, 31, and 32. To make the machine as simple as possible, we have left out transitions that pick up a disk and put it right back in the same place.

We can define the Towers of Hanoi language to be the set of strings that correspond to move sequences that take the system from its start state to its accepting state. The complexity of the Hanoi language is regular because it is accepted by the Towers of Hanoi FSM as just described. And it is infinite, since there is no limit to the number of times a disk can be moved between poles. But the shortest string in the language has length $2^{64} - 1$ (namely the length of the optimal sequence of moves that solves the problem).

P.3 The Arithmetic Logic Unit (ALU)

In most computer chip designs, the ALU performs the fundamental operations of integer arithmetic, Boolean logic, and shifting. The ALU’s operation can be modeled as a finite state transducer, using either a Moore machine (in which outputs are associated with states) or a Mealy machine (in which outputs are associated with transitions).

P.3.1 An Adder

As an example, consider a simple binary adder, shown in Figure P.7. Two numbers can be added by adding their digits right to left. So we can describe an adder as a Mealy machine whose input is a stream of pairs of binary digits (one digit from each of the two numbers to be added). The machine has two states, one of which corresponds to a carry-in bit of 0 and the other of which corresponds to a carry-in bit of 1. When the machine is reset, it enters the state corresponding to no carry (i.e., a carry-in bit of 0). This simple one-bit adder can be embedded into a larger system that adds numbers of any fixed number of bits.

P.3.2 A Multiplier

Binary adders can also be used as building blocks for binary multipliers. Figure P.8 shows a schematic diagram that describes the behavior of an 8-bit multiplier. The multiplier can be implemented as the finite state transducer shown in Figure P.9.

![A schematic diagram for an 8-bit multiplier.](image)

![The finite state transducer that implements the multiplier.](image)