A Finite State Machine for the Towers of Hanoi
CS 330 Discrete Structures
Fall Semester, 2017

The following beautiful example and its exposition are shamelessly plagiarized without permission from section 6.4 of the wonderful book Automatic Sequences by Jean-Paul Allouche and Jeffery Shallit, Cambridge University Press, 2003.

The Towers of Hanoi puzzle shown above consists of $N$ disks of radii $1, 2, \ldots, N$ placed on peg 1 in decreasing order by size—the largest on the bottom, the second largest on top of that, \ldots, the smallest on top. A “move” consists of taking the topmost disk from one peg and moving it to another peg, subject to the restriction that a disk may not be placed on top of a smaller disk. The goal is to move the pile of disks from one peg to another, using the third peg. We use the notation “$x \rightarrow y$” to mean that we move the top disk from peg $x$ to peg $y$. For example, when $N = 3$, the seven moves $1 \rightarrow 2, 1 \rightarrow 3, 2 \rightarrow 3, 1 \rightarrow 2, 3 \rightarrow 1, 3 \rightarrow 2, 1 \rightarrow 2$ successfully move the 3 disks from peg 1 to peg 2.

Theorem 1. $2^N - 1$ moves are necessary and sufficient to move $N$ disks from one peg to another.

Proof. That $2^N - 1$ moves are sufficient follows from a simple recursive algorithm: to transfer 1 disk, just move it. To transfer $N > 1$ disks from peg $x$ to peg $y$ using peg $z$, recursively transfer the top $N - 1$ disks from peg $x$ to peg $z$ using peg $y$; move the top disk from peg $x$ to peg $y$; recursively transfer the $N - 1$ disks from peg $z$ to peg $y$ using peg $x$. The number of moves is $(2^{N-1} - 1) + 1 + (2^{N-1} - 1) = 2^N - 1$.

Necessity is proved by induction. For a single disk, clearly one move is required. To move $N > 1$ disks, we must move the largest (bottommost) disk—this requires that the $N - 1$ disks above it be moved. By induction, this requires at least $2^{N-1} - 1$ moves; now we can move the largest disk to the target peg, but then the remaining $N - 1$ disks must be moved to the target peg, again requiring at least $2^{N-1} - 1$ moves. The total number of moves is thus at least $(2^{N-1} - 1) + 1 + (2^{N-1} - 1) = 2^N - 1$. \hfill \Box

Algorithm 1 thus gives the numbered sequence of moves for $N$ disks. For example,

1: $i \leftarrow 0$
2: Tower(3, 1, 2, 3)

produces the output
Move 0: 1 → 2
Move 1: 1 → 3
Move 2: 2 → 3
Move 3: 1 → 2
Move 4: 3 → 1
Move 5: 3 → 2
Move 6: 1 → 2

Note that the moves are indexed from 0; we want to determine the move from the index; that is, we want ask, say, “What is Move 4?” and get the answer 3 → 1. To accomplish this, it is convenient to modify the Towers of Hanoi problem to insist that if the number of disks $N$ is odd, we must move the disks from peg 1 to peg 2, while if $N$ is even, we must move the disks from peg 1 to peg 3. This seems unnatural, but it makes the sequence of $2^N - 1$ moves for $N$ disks begin with the sequence of $2^{N-1} - 1$ moves for $N - 1$ disks (why?). Hence there is an infinite sequence of moves

\[1 \to 2, 1 \to 3, 2 \to 3, 1 \to 2, 3 \to 1, 3 \to 2, 1 \to 2, 1 \to 3, 2 \to 3, 2 \to 1, 3 \to 1, 2 \to 3, 1 \to 2, 1 \to 3, 2 \to 3, 1 \to 2, 3 \to 1, 3 \to 2, \ldots \]  

(1)

which gives the solution for $N = 1, 2, 3, \ldots$ disks; this can be interpreted as solving the Towers of Hanoi puzzle for infinitely many disks!

**Theorem 2.** The finite state machine in Figure 1, when given as input the bits of $i$, written in binary, scanned from high order to low order, starting at state 1 → 2, leaves the machine in the state that gives the $i$th move of the above infinite sequence of moves.

For example, to find Move 4, we write $4 = 100_2$, start at state 1 → 2 and follow the state transitions caused by the input string 100: from 1 → 2 input 1 takes us to 1 → 3 and then input 0 takes us to 2 → 3 and finally input 0 takes us to 3 → 1. Notice that the machine ignores leading zeros, so 0000100_2 takes the machine to the same state as 100_2.

The remainder of these notes proves this theorem by exploring properties of the infinite sequence of moves. It will be convenient to have single character abbreviations for the six states of the finite state machine in Figure 1. We use

**Algorithm 1** Recursive algorithm to generate the sequence of moves that solve the $N$-disk Towers of Hanoi problem.

function $\text{Tower}(N, x, y, z)$
1: if $N = 1$ then
2: \hspace{1em} OUTPUT($\text{Move}_{i++}: x \to y$)
3: else
4: \hspace{1em} $\text{Tower}(N - 1, x, z, y)$
5: \hspace{1em} OUTPUT($\text{Move}_{i++}: x \to y$)
6: \hspace{1em} $\text{Tower}(N - 1, z, y, x)$
7: end if

*Both the ++ operator and global variables can lead to difficult bugs because of side effects! They are used here out of parsimony.*
Figure 1: The finite state machine for the Towers of Hanoi: When the bits of the number $n$, written in binary, are scanned from high order to low order, starting at $1 \to 2$ (abbreviated $a$), the machine is left in the state giving the $n$th move of the Towers of Hanoi with infinitely many disks.
1 → 2  a  2 → 1  ā
2 → 3  b  3 → 2  ē
3 → 1  c  1 → 3  ē

So the infinite sequence of moves given above becomes the infinite string:

\[ H = ačbačbačbačbačbačb \cdots \]

We now describe this infinite string more formally. The recursive nature of the sequence of moves given by Algorithm 1 means we should be able to describe the string \( H \) recursively; the difficulty is that the roles of the pegs change with the recursive calls. To mimic this change in roles, we alter the **moves** by a function \( σ \); the altered moves are

\[
\begin{align*}
σ(a) &= b \\
σ(ā) &= ă \\
σ(b) &= c \\
σ(ē) &= Ź \\
σ(c) &= a \\
σ(ē) &= Ź
\end{align*}
\]

That is,

- 1 → 2 becomes 2 → 3
- 2 → 3 becomes 3 → 1
- 3 → 1 becomes 1 → 2

so that peg 2 plays the role of the original peg 1, peg 3 plays the role of the original peg 2, and peg 1 plays the role of the original peg 3. We also need the inverse of this transformation,

\[
\begin{align*}
σ^{-1}(a) &= c \\
σ^{-1}(ā) &= ć \\
σ^{-1}(b) &= a \\
σ^{-1}(ē) &= Ź
\end{align*}
\]

We extend the definitions of \( σ \) and \( σ^{-1} \) from characters to strings by saying that \( σ(uv) = σ(u)σ(v) \) for strings \( u \) and \( v \), and \( σ(ε) = ε \), where \( ε \) is the empty string. Similarly, \( σ^{-1}(uv) = σ^{-1}(u)σ^{-1}(v) \). Using the recursive structure of the solution to the Towers of Hanoi in Algorithm 1 we can define

\[
\begin{align*}
H_{2i+1} &= H_{2i}σ^{-1}(H_{2i}), & i & ≥ 0 \\
H_{2i} &= H_{2i−1}ēσ(2i−1), & i & ≥ 1,
\end{align*}
\]

where \( H_0 = ε \), the empty string. These definitions say, respectively, that

- For \( i ≥ 0 \), \( H_{2i+1} \) first moves disks 1, 2, \ldots, 2i from peg 1 to peg 3, then moves disk 2i + 1 from peg 1 to peg 2, and then moves disks 1, 2, \ldots, 2i from peg 3 to peg 2,
- For \( i ≥ 1 \), \( H_{2i} \) first moves disks 1, 2, \ldots, 2i − 1 from peg 1 to peg 2, then moves disk 2i from peg 1 to peg 3, and then moves disks 1, 2, \ldots, 2i − 1 from peg 2 to peg 3.

Thus, \( H_N \) is the sequence of moves solving the \( N \)-disk modified Towers of Hanoi problem. The above formulas give us
$H_1 = a$

$H_2 = a\bar{c}b$

$H_3 = a\bar{c}ba\bar{c}ba\bar{c}$

$H_4 = a\bar{c}ba\bar{c}ba\bar{c}ba\bar{c}ba\bar{c}$

\vdots

Of course, $H_N$ is a prefix of $H_{N+1}$, so we can take the limit,

$$H = \lim_{N \to \infty} H_N,$$

and we get the infinite sequence of moves (1).

To derive the finite state machine of Figure 1, we need an important property of $H$. Define the alphabet

$$\Sigma = \{a, \bar{a}, b, \bar{b}, c, \bar{c}\},$$

and consider the following mapping of single characters in $\Sigma$ to pairs of characters in $\Sigma$:

$$\varphi(a) = a\bar{c} \quad \varphi(\bar{a}) = ac$$

$$\varphi(b) = cb \quad \varphi(\bar{b}) = cb$$

$$\varphi(c) = b\bar{a} \quad \varphi(\bar{c}) = ba$$

Extending the function $\varphi$ to strings in $\Sigma^*$ (as we did with $\sigma$ and $\sigma^{-1}$) we find that

$$\varphi(H) = H.$$  \hspace{1cm} (2)

$H$ is thus a fixed point of $\varphi$. We leave the verification of this remarkable property to Exercises 3 and 4.

Now we can understand the state transitions in Figure 1. Let $x$ be the $n$th character, $n \geq 0$, in $H$; then, because $\varphi$ doubles the length of its argument, $\varphi(x)$ gives the $(2n)$th and $(2n+1)$st characters of $\varphi(H) = H$, say $\varphi(x) = yz$. If we have read the number $n$ in binary and we are in the state corresponding to the $n$th move (state $x$), then when we read the next character—a 0 or a 1—we will have read the number $2n$ or $2n + 1$, respectively, so the finite state machine transition must take us to the corresponding state, $y$ for a 0 or $z$ for a 1. For example, if we are in state $x = \bar{a}$ (that is, we are in state $2 \to 1$), $\varphi(x) = ac$; thus 0 takes the finite state machine to state $a$ (that is, $1 \to 2$) and a 1 takes it to state $c$ (that is, $3 \to 1$), as shown. The remaining 10 state transitions are verified similarly (Exercise 5).
Exercises

1. (a) Prove by induction for $i \geq 1$ that

$$\varphi^{2i}(a) = H_{2i}a,$$
$$\varphi^{2i+1}(a) = H_{2i+1}\bar{c}.$$  
(b) Use part (a) to prove that $H = \lim_{n \to \infty} \varphi^n(a)$

2. Prove that $H = (aCbAcB)^\infty$ where $A$ is either $a$ or $\bar{a}$ and similarly for symbols $B$ and $C$.

3. Prove that for every string $w \in \Sigma^*$,

$$\varphi(\sigma(w)) = \sigma^{-1}(\varphi(w)),$$
$$\varphi(\sigma^{-1}(w)) = \sigma(\varphi(w)).$$

4. (a) Using Exercise 3 prove by induction that for $i \geq 0$,

$$\varphi(H_{2i})a = H_{2i+1},$$
$$\varphi(H_{2i+1})b = H_{2i+2}.$$  
(b) Show that there is an infinite sequence of prefixes of $H$ that are mapped by $\varphi$ into longer prefixes of $H$, thus proving (2), that $H$ is a fixed point of $\varphi$.

5. Verify the remaining 10 state transitions of Figure 1.