Algorithms and State Machines

We build a mathematical model for algorithms using the notion of a state machine.

Definitions and Examples

In this section we give basic definitions and illustrate our definitions on a finite state machine.

Definition. A *state machine* M is a triple $M = (S, T, s_0)$ where

1. The *states* of M are the elements of the set S.
2. The *transition relation* of M is the set $T \subseteq S \times S$.
3. The *initial state* of M is the state $s_0 \in S$.

Note that mathematically speaking a state machine is a *pointed directed graph.* We will often write $s \to t$ to denote that $(s, t) \in T$ is a possible transition of M.

Example. The triple $M = (S, T, s_0)$ is a state machine where

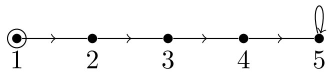
1. $S = \{1, 2, 3, 4, 5\}$
2. $T = \{(1, 2), (2, 3), (3, 4), (4, 5), (5, 5)\}$
3. $s_0 = 1$

We call this machine the "four-step overflow machine" below.

It is often be useful to draw pictures of state machines. We use the following conventions to make our pictures.

- States are represented by labeled points.
- Transitions are represented by arrows pointing from the old state to the new state.
- The initial state is circled.

Using these conventions, the four-step overflow machine from the previous example is pictured below.



Definition. An *evaluation* of a state machine $M = (S, T, s_0)$ is a sequence $(s_0, s_1, \cdots, s_n)$ where each $s_i \in S$ is a state, s_0 is the initial state, and $s_{i-1} \to s_i$ for all $i \in [n]$. The *length* of such an evaluation is n.

We use $s_0 \to s_1 \to \cdots \to s_n$ to denote the evaluation $(s_0, s_1, \cdots, s_n)$. The only evaluation of length 0 is (s_0).

Example. In the four-step overflow machine, the evaluations are simple; there is exactly one evaluation for each length n. The evaluations of length 0, 1, 2, 3, and 4 are (1), $(1, 2)$, $(1, 2, 3)$, and $(1, 2, 3, 4)$ respectively. For length $n \geq 4$, the unique evaluation of length n is $(1, 2, 3, 4, 5, 5, \cdots, 5)$, with $n - 3$ repeats of the state 5.

The set of all states which can be reached from our initial state is of fundamental importance in our study of algorithms; this set describes the possible "middle steps" an algorithm might pass through before termination.

Definition. A state t in $M = (S, T, s_0)$ is *reachable* when there is an evaluation $s_0 \to s_1 \to \cdots \to s_n$ with $t = s_n$.

Example. Every state is reachable in the four-step overflow machine.

Fundamentally, we want algorithms which terminate by outputting a state with a desired property.

Definition. A *terminal state* or *terminus* of $M = (S, T, s_0)$ is a reachable state $t \in S$ so that $(t, s) \notin T$ for all $s \in S$.

Example. The four-step overflow machine has no terminal states.

Because we think of terminal states as outputs, it will later be useful to know that a state machine has an terminus; showing a terminus exists can be difficult. One method (described later) uses functions defined on states.

Definition. A *derived variable* on a state machine $M = (S, T, s_0)$ is a function $f : S \to \mathbb{R}$.

Note that there is no restriction on derived variables. On the other hand, derived variables are often only useful if they have some nice properties with respect to the transition relation on M.

Definition. A derived variable $f : S \to \mathbb{R}$ on state machine $M = (S, T, s_0)$ is

1. *strictly increasing* if $f(s) < f(t)$ for all $s, t \in S$ with $s \to t$.
2. *weakly increasing* if $f(s) \leq f(t)$ for all $s, t \in S$ with $s \to t$.
3. *strictly decreasing* if $f(s) > f(t)$ for all $s, t \in S$ with $s \to t$.
4. *weakly decreasing* if $f(s) \geq f(t)$ for all $s, t \in S$ with $s \to t$.

Example. The derived variable $f : S \to \mathbb{R}$

When studying state machines, it is often useful to consider predicates defined on the states themselves; doing so can (later in the document) help prove an algorithm computes what we want (i.e. for "proof of correctness").

Definition. A *preserved invariant* of a state machine $M = (S, T, s_0)$ is a predicate formula P with domain S such that if $P(s)$ is true and $s \to t$ is a possible transition, then $P(t)$ is true.

State machines often have many preserved invariants.

Example. For the four-step overflow machine, the formulas $P(s) : s = 5$ and $Q(s) : s \geq 3$ are preserved invariants.

General Remarks

We now make a few general remarks on the structures of state machines.

Modeling Algorithms with State Machines

We think of algorithms using state machines as follows. The states are a set of objects which the algorithm manipulates. We include a transition $s \to t$ whenever the algorithm can change s to t via its operation. The initial state is the desired input to the algorithm. Finally, the terminal states are the states which lead us to output; the output is a function of the set of terminal states. In this scheme, the set of meaningful inputs (i.e. states which can be used as initial states) may be a strict subset of the full set of states. We will see an example later.

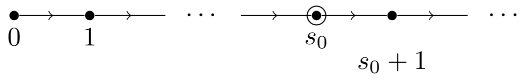
The Halting Problem

The "best" algorithms take an input and yield a uniquely determined output. In the context of state machines, this is to say that the "best" algorithms actually give a pair (S, T) of states and transitions with a function $\alpha : \operatorname{init}(S, T) \to \operatorname{term}(S, T)$ so that $\operatorname{init}(S, T)$ is the set of meaningful inputs, $\operatorname{term}(S, T)$ is the set of states t which have $(t, s) \notin T$ for all $s \in S$, and $\alpha(s_0)$ is the only terminus for the state machine (S, T, s_0).

One naturally wonders the following; given an input s_0 for an algorithm modeled by (S, T, s_0), can we ensure that the state machine has a terminus? Even if it has a terminus, can we ensure that the machine must reach a terminus in finitely many steps? In other words, can we guarantee that for every meaningful input s_0, all evaluations of $M = (S, T, s_0)$ are of finite length?

One possible failure can arise from the existence of an infinite evaluation without any repeats.

Example. The state machine $M = (S, T, s_0)$ with states $S = \mathbb{N}_0$, transitions $T = \{n \to (n + 1) : n \in \mathbb{N}_0\}$, and any initial state $s_0 \in \mathbb{N}_0$ has an infinite evaluation!



Another possible failure that can arise is the existence of infinite loops; the four-step overflow machine has an infinite loop (in fact, any machine with any transition (s, s) has an infinite loop). Another failure is given as follows.

Example. The state machine $M = (S, T, s_0)$ with states $S = \{1, 2, 3\}$, transitions $T = \{(1, 2), (2, 3), (3, 1)\}$, and any initial state $s_0 \in S$ has an infinite loop (draw a picture to see the loop)!

The existence of these "failures" is related to the *halting problem*; given a state machine $M = (S, T, s_0)$, can we guarantee that there is a terminus, or is there an infinite evaluation? In fact, one can prove (relatively easily, in fact) that there is no algorithm to answer this question in all cases!

Preserved Invariants

The following is a useful tool when working with state machines.

Proposition (Preserved Invariant Lemma)**.** *Let $M = (S, T, s_0)$ be a state machine with preserved invariant P. If $P(s_0)$ is true, then $P(s)$ is true for all reachable states of M.*

Sketch. Induct on the length of an evaluation. □

Notice that this proposition is essentially a jazzed-up version of induction! In fact, induction is a special case of this result where we work on the following state machine.

$$\begin{tikzcd} \underset{0}{\odot} \arrow[r] & \underset{1}{\bullet} \arrow[r] & \cdots \end{tikzcd}$$

Examples of State Machines

This section will provide several examples of state machines. Proofs are left to the reader.

Remainders Machine

Let $S = \mathbb{Z} \times \mathbb{Z}_{>0}$ and let T be given by the following for all $(n, d) \in S$.

$$(n, d) \to \begin{cases} (n - d, d) & \text{if } n \geq d \\ (n + d, d) & \text{if } n < 0 \end{cases}$$

Then for all $s_0 \in S$ there is a unique terminal state in $M(s_0) = (S, T, s_0)$; in particular the unique terminal state of $M(n, d)$ is $(\operatorname{rem}(n, d), d)$ where $\operatorname{rem}(n, d)$ is the remainder of n under division by d. Hence this machine describes an algorithm for computing remainders under integer division.

Division Algorithm Machine

Let $S = \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}_{>0}$ and let T be given by the following for all $(r, q, d) \in S$.

$$(n, d) \to \begin{cases} (n - d, q + 1, d) & \text{if } n \geq d \\ (n + d, q - 1, d) & \text{if } n < 0 \end{cases}$$

For all $s_0 \in S$ there is a unique terminal state in $M(s_0) = (S, T, s_0)$. Moreover, for initial states of the form $s_0 = (n, 0, d)$ the unique terminal state has the form $(\operatorname{rem}(n, d), \operatorname{quot}(n, d), d)$ where $\operatorname{quot}(n, d)$ is the quotient of n under division by d. This is easy to verify using the derived variable $f(x, y, z) = x + yz$ and the Quotient-Remainder Theorem. Hence this machine describes an algorithm to compute the quotient and remainder under integer division.

Euclidean Algorithm Machine

Let $S = \mathbb{Z} \times \mathbb{Z}$ and let T be the set of transitions described below for all $(m, n) \in S$.

$$(m, n) \to \begin{cases} (|m|, |n|) & \text{if either } m < 0 \text{ or } n < 0 \\ (n, m) & \text{if } n < m \\ (\operatorname{rem}(n, m), m) & \text{if } 0 < m \leq n \end{cases}$$

For all $(m, n) \in S$ with $(m, n) \neq (0, 0)$, there is a unique terminal state of the form $(0, d)$; this can be shown using the derived variable $f(x, y) = x$, which is strictly decreasing on transitions whenever $x, y \geq 0$. On the other hand, $P(x, y) : \gcd(x, y) | \gcd(m, n)$ and $Q(x, y) : \gcd(m, n) | \gcd(x, y)$ are preserved invariants of the machine $M(m, n)$ for all $(0, 0) \neq (m, n) \in S$. Hence by the Preserved Invariant Lemma (noting trivially that $P(m, n)$ and $Q(m, n)$ are both true) we have that the unique terminal state of $M(m, n)$ is precisely $(0, \gcd(m, n))$.

The state machine above is one machine which encodes the Euclidean Algorithm.

Extended Euclidean Algorithm Machine

Let $S = \mathbb{Z}^8$ and let T be the set of transitions described below for all $(m, n, x, y, z, a, b, c) \in S$ with $a \neq 0$.

$$(m, n, x, y, z, a, b, c) \to (m, n, a - \operatorname{quot}(a, x)x, b - \operatorname{quot}(b, x)y, c - \operatorname{quot}(c, x)z, x, y, z)$$

We consider the machine $M = M(m, n, n, 0, 1, m, 1, 0)$ for given $(0, 0) \neq (m, n) \in \mathbb{Z}^2$.

The derived variable $f(m, n, x, y, z, a, b, c) = x$ is strictly decreasing across all transitions provided $x > 0$; moreover, every length 1 evaluation of M has the reached state $f(m, n, r, s, t, r', s', t') \geq 0$. Moreover, the derived variables $g(m, n, x, y, z, a, b, c) = mb + nc$ and $h(m, n, x, y, z, a, b, c) = a$ yield the preserved invariant $P(s) : g(s) = h(s)$. Moreover, the Euclidean Algorithm Machine (from the previous section) lives inside of this machine; indeed, the action of the machine on (x, a) is almost the same (why is it different?). A slight modification of the work in that section can be used to conclude that M has a unique terminal state of the form $(m, n, \gcd(m, n), s, t, 0, b, c)$; finally $\gcd(m, n) = ms + nt$ by our argument above. Hence M encodes the Extended Euclidean Algorithm on (m, n).

Fast Exponentiation Machine

Consider the Fast Exponentiation Algorithm described below.

Let $a \in \mathbb{R}$ and $b \in \mathbb{N}_0$; set $x_0 = a$, $y_0 = 1$, and $z_0 = b$.

```
1. Given $(x_i, y_i, z_i)$.
    (a) If $z_i = 0$, output $y_i$ and stop.
    (b) Otherwise, let $x_{i+1} = x_i^2$, $z_{i+1} = \operatorname{quot}(z_i, 2)$, and $r = \operatorname{rem}(z_i, 2)$.
        i. If $r = 1$, let $y_{i+1} = x_i y_i$.
        ii. If $r = 0$, let $y_{i+1} = y_i$.
2. Return to step 1 with $(x_{i+1}, y_{i+1}, z_{i+1})$.
```

We can model this algorithm with a state machine as follows. Let $S = \mathbb{R} \times \mathbb{R} \times \mathbb{N}_0$, and let the set T of transitions be given by all transitions of the following form.

$$(x, y, z) \to \begin{cases} (x^2, y, \operatorname{quot}(z, 2)) & \text{if } z \neq 0 \text{ and } \operatorname{rem}(z, 2) = 0 \\ (x^2, xy, \operatorname{quot}(z, 2)) & \text{if } \operatorname{rem}(z, 2) = 1 \end{cases}$$

This models the algorithm above; the transitions are derived precisely from the algorithm itself. Moreover, the machine $M(a, 1, b) = (S, T, (a, 1, b))$ enacts this algorithm on the given $a \in \mathbb{R}$ and $b \in \mathbb{N}_0$. Finally the derived variable $f(x, y, z) = yx^z$ is constant and $g(x, y, z) = z$ is strictly decreasing on transitions of the machine. Using these it is easy to show that $M(a, 1, b)$ has a unique terminal state, which has the form $(x, a^b, 0)$.