1 Monday, February 23

1.1 LLM 5.4: State Machines

Reading: LLM 5.4.

1.1.1 State Machine Definition

Def: A *state machine* is a triple (S, T, s_0) where S is a set of *states*, T is a set of *transitions* between states (so T is a subset of $S \times S$), and $s_0 \in S$ is an *initial state*. A state machine *executes* by constructing a sequence $s_0, s_1, \ldots, s_n, \ldots$ where s_0 is the initial state, and for all $n \geq 0$, when s_n is a state in S, the sequence value s_{n+1} is a state in S such that the pair (s_n, s_{n+1}) is in T (that is, we can transition from s_n to s_{n+1}).

When talking about state machines, we will make use of the following concept.

> Let $M = (S, T, s_0)$ be a state machine.
>
> **Preserved Invariant:** A property $P : S \to \{\mathbf{T}, \mathbf{F}\}$ is a *preserved invariant* for M if "$\forall (s, t) \in T$, $P(s) \to P(t)$." That is, if a state has property P, then all states that are reachable from that state by one transition also have that property.
>
> **Invariant Principle:** If a preserved invariant of a state machine is true for the start state, then it is true for all reachable states.

Observe that the above concept is equivalent to induction. If $s_0, s_1, s_2, \ldots, s_n, \ldots$ is a sequence of states given by a state machine, then let $Q(n) = P(s_n)$. The base case is that $Q(0)$ is true, which means the property is held by the initial state s_0. The preserved invariant property means that if $Q(n)$ is true, then $Q(n+1)$ is true. So, you can use induction explicitly, or you can prove that a property is a preserved invariant.

1.1.2 GCD Algorithm

Recall Euclid's algorithm for computing the *greatest common divisor*.

> **Input:** Integers a and b where $a \geq b \geq 0$.
>
> If $b \equiv 0$, then return a.
>
> Otherwise, let q, r be integers such that $a = qb + r$ and $0 \leq r < b$.
>
> Assign $a \leftarrow b$ and $b \leftarrow r$, then repeat the algorithm.

We may prove that this algorithm is correct later, but instead let's prove that this algorithm will terminate in a finite number of steps.

Thm: If a and b are integers with $a \geq b \geq 0$, the above algorithm will halt in a finite number of steps.

Proof. Observe that every time a and b are reassigned, b decreases by at least one (as the value r is guaranteed to be in the range $0 \leq r < b$). Therefore, the values a and b are reassigned at most b times (and the if statement is tested at most $b + 1$ times). □

Let $E = (S, T, s_0)$ be the machine where S is the set of pairs $(a, b) \in \mathbb{N} \times \mathbb{N}$ where $a \geq b$, $s_0 = (a_0, b_0)$ for some pair $(a_0, b_0) \in \mathbb{N} \times \mathbb{N}$, and the transition set T is given by

$$T = \big\{\big((a, b), (b, a\%b)\big) : a, b \in \mathbb{N}, a \geq b > 0\}.$$

(Recall that $a\%b$ returns the remainder after integer division of a by b.)

The theorem that the GCD Algorithm halts is equivalent to saying this machine will reach a state $(a, 0)$ (where there are no transitions out).

Let $M = (S, T, s_0)$ be a state machine.
Decreasing Functions: Let $f : S \to \mathbb{N}$ be a function[1] The function f is *decreasing* if $\forall (s, t) \in T(f(s) > f(t)$. That is, for every transition $s \to t$, the value of f decreases from $f(s)$ to $f(t)$ with $f(t) < f(s)$.

Monotonicity Principle: If $f : S \to \mathbb{N}$ is a decreasing function, then the time it takes for a machine to terminate starting at a state s is at most $f(s)$.

Thm: The GCD Algorithm will terminate in a finite number of steps.

Proof. Define $f(a, b) = a + b$. Note that if $(a, b) \to (b, r)$ is a transition, then $0 \leq r < b \leq a$. Therefore, $f(b, r) = b + r < a + b = f(a, b)$ and hence f is a decreasing function.

By the monotonicity principle, starting at (a, b) will result in a halt in at most $a + b$ steps. (This is a gross over-count!) □

Thm: Let $d \geq 1$ be a positive integer. Let $P_d(a, b)$ be the property "a and b are both multiples of d." $P_d(a, b)$ is a preserved invariant for the machine E.

Proof. Suppose that $P_d(a, b)$ is true and $(a, b) \to (b, r)$ is a transition (implying $b > 0$ and $r = a\%b$). Then there exist nonnegative integers i, j such that $a = di$ and $b = dj$.

By integer division with remainder, there exists an integer q such that $a = qb + r$. So, $di = qdj + r$, which implies that $r = d(i - qj)$. Therefore, r is a multiple of d, and hence $P_d(b, r)$ is true. □

Thm: The GCD algorithm outputs the correct value.

In order to prove this statement, we need to make a different state machine, one that is *reversible*.

Let $M = (S, T, s_0)$ be a state machine.
Reversible Invariant: A property $P : S \to \{\mathbf{T}, \mathbf{F}\}$ is a *reversibile invariant* if for all transitions $s \to t$ we have $P(s) = P(t)$. (Equivalently, $(s \to t)$ implies $P(s) \leftrightarrow P(t)$.)

Reversibility Principle: If P is a reversible property and s is a state reachable from the initial state s_0, then $P(s) = P(s_0)$.

Proof. Recall the property P_d is a preserved invariant for the transitions in T. We will show it is also a reversible invariant for the transitions in T. Suppose $(a, b) \to (b, a\%b)$ is a transition in T and $P_d(b, a\%b)$ is true. Then $b = d\ell$ and $a\%b = r = dk$ for some integers k and ℓ. By definition of integer division there is an integer q where $a = qb + r$. Then $a = qb + r = q(d\ell) + dk = d(q\ell + k)$, so d is a divisor of both a and b. Thus, $P_d(a, b)$ is true.

Let d be the greatest common divisor of a and b. Then $P_d(a, b)$ is true, and $P_{d'}(a, b)$ is false for all $d' > d$. If we follow transitions from T, we will terminate in a state $(r, 0)$. By the Invariant Principal, $P_d(r, 0)$ is true,

so d is a divisor of r, and $d \leq r$. However, r is a divisor of both r and 0, so by the Reversibility Principle $P_r(a, b)$ is true. Therefore, $r \leq d$ (since d is the greatest common divisor of a and b) and therefore $r = d$. $\square$

1.1.3 Suggested Homework

LLM Problems 5.10, 5.28–38.

2 Wednesday, February 25

2.1 LLM 5.4: State Machines — Rosen 5.3: Recursive Definitions and Structural Induction

Reading: LLM 5.4, 6.1. Rosen 5.3.

2.1.1 Recursively Defined Functions

If the definition of a function refers to its own output (on different, usually smaller values), then the function is defined *recursively*.

For example, the *Fibonacci sequence* $\{F_n\}_{n=0}^{\infty}$ is really a function from the natural numbers to the natural numbers, defined as $F(0) = 0$, $F(1) = 1$, and $F(n) = F(n-1) + F(n-2)$ for all $n \geq 2$. This is well-defined, but you should be very careful when constructing a recursive function that it is well defined!

We will use the following property of the Fibonacci numbers.

Thm: Let $\phi = \frac{1+\sqrt{5}}{2}$. For all $n \geq 3$, $F_n \geq \phi^{n-2}$.

There is a proof of this fact in the book, using strong induction.

Lamé's Theorem. Let a and b be positive integers with $a \geq b$. The number of divisions used by the Euclidean algorithm to find GCD(a, b) is less than or equal to five times the number of decimal digits in b (i.e. at most $5\lceil \log_{10} b \rceil$).

Note: We will prove something slightly weaker: The GCD state machine follows at most $5(\log_{10} b + 1)$ transitions.

Proof. If we follow the algorithm, and consider all of the pairs we use during the while loop, we will have a sequence $(a_0, b_0), (a_1, b_1), \ldots (a_n, b_n), (a_{n+1}, b_{n+1})$ where for all $i \in \{0, \ldots, n\}$, $a_{i+1} = b_i$ and $b_{i+1} = a_i \% b_i$. (Note that now we can replace a_i with b_{i-1}.) Finally, since we halted, $b_{n+1} = 0$. By the definition of integer division, for every $i \in \{0, \ldots, n\}$, there is an integer q_i where $b_{i-1} = q_i b_i + b_{i+1}$. Also, since $a_i \geq b_i$ for all i (check!) we have that $q_i \geq 1$, so $b_{i-1} \geq b_i + b_{i+1}$.

Specifically in the last equation, we have $b_{n-1} = q_n b_n + b_{n+1}$, but since $b_{n-1} > b_n$ and $b_{n+1} = 0$, we have that $q_n \geq 2$.

Claim: For all $k \geq 0$, $b_{n-k} \geq F_{k+2}$.

We prove this claim by induction on k.

Case $k = 0$: $b_n \geq 1 = F_2$.

Case $k = 1$: $b_{n-1} \geq 2b_n \geq 2 = F_3$.

(Strong Inductive Hypothesis) Suppose that for some K where $0 < K \leq n$ we have $b_{n-k} \geq F_{k+2}$ for all k where $0 \leq k < K$.

Case K: $b_{n-K} = q_{n-(K-1)} b_{n-(K-1)} + b_{n-(K-2)} \geq b_{n-(K-1)} + b_{n-(K-2)} \geq F_{K+1} + F_{K+2} = F_{K+3}$.

Thus, $b_{n-k} \geq F_{k-2}$, proving the claim. Therefore $b_0 = b_{n-n} \geq F_{n-2} \geq \phi^{n-4}$.

We now have an exponential relationship between the value of b_0 and the number of steps until halting $(n+1)$. That is $\log_{10} b_0 \geq \log_{10} \phi^{n-4} = (n-4) \log_{10} \phi$. Since $\phi \approx 1.61$, and $\log_{10} \phi \approx 0.209 > \frac{1}{5}$, we have that $\log_{10} b_0 \geq \frac{1}{5}(n-4)$. Thus, $5(\log_{10} b_0 + 1) \geq n + 1$. □

2.1.2 Alphabets and Strings

Let Σ be a finite set (here we say Σ is the name of a set, not the summation notation). We will use Σ to denote a special set, called an *alphabet*. The elements of Σ will be "letters" of "words" but in a very abstract sense. Typically, $\Sigma = \{0, 1\}$, but Σ could be any finite set.

For $k \geq 0$, the set Σ^k is the set of k-tuples where every entry comes from Σ. The set Σ^* is equal to $\cup_{k=0}^{\infty} \Sigma^k$, the set of all finite strings (note that for every finite string $\mathbf{x}$ of length k, $\mathbf{x} \in \Sigma^k \subset \Sigma^*$). We will denote a string $\mathbf{x} = (x_1, x_2, \ldots, x_k)$ as $x_1 x_2 \ldots x_k$.

If $\Sigma = \{0, 1\}$, then Σ^* contains all finite-length *binary strings*. This includes the *empty string* w which has length 0.

We can define state machines using Σ^* as the state space! A natural class of transitions is to move between strings by adding or removing letters from a string. Let $\Sigma = \{0, 1\}$ and consider the following possible transitions:

1. If $\mathbf{x} = x_1 x_2 \ldots x_k$ is a string and k is even, then $x_1 x_2 \ldots x_k \to x_1 x_2 \ldots x_k 1$ is an allowed transition. (Append a 1 to the end of the string.)
2. If $\mathbf{x} = x_1 x_2 \ldots x_k$ is a string and k is odd, then $x_1 x_2 \ldots x_k \to x_1 x_2 \ldots x_k 0$ is an allowed transition. (Append a 0 to the end of the string.)
3. If $\mathbf{x} = x_1 x_2 \ldots x_k$ is a string and $k \geq 1$, then $x_1 x_2 \ldots x_k \to x_2 \ldots x_k$ is an allowed transition. (Delete the first position.)

Thm: If $\mathbf{x} = x_1 x_2 \ldots x_k$ is any string in Σ^*, then $\mathbf{x}$ is reachable from the empty string by the above transitions.

Proof. Starting from the empty word w, follow k total transitions of type (1) and (2), alternating as necessary to create the string $101010 \ldots 10$ or $101010 \ldots 01$ of length k. For each $i \in \{1, 2, \ldots, k\}$, we will make the following choices:

- If $x_i = 1$ and the current string has even length, then use (1) to append a 1.
- If $x_i = 0$ and the current string has odd length, then use (2) to append a 0.
- If $x_i = 1$ and the current string has odd length, then use (3) to delete the first position, then (1) to append a 1.
- If $x_i = 0$ and the current string has even length, then use (3) to delete the first position, then (2) to append a 0.

During this process, we may have deleted some leading positions, but at most one position was deleted per letter that was added. Therefore, we deleted no more than k elements, which is the length we had before adding the letters $x_1 \ldots x_k$. Thus, there are at least k letters in the string at this point, and the final k letters correspond to the string $\mathbf{x}$.

Use (3) to delete the first position until the length of the string is k. The resulting string is $\mathbf{x}$. □

Def: Let Σ be a finite alphabet. Define W as follows: (Basis) the empty word is in W, and (Recursive Step) if $x_1 \ldots x_n \in W$, and $y \in \Sigma$, then $x_1 \ldots x_n y \in W$.

Thm: $W = \Sigma^*$.

Proof. We must show $W \subseteq \Sigma^*$ and $\Sigma^* \subseteq W$.

($W \subseteq \Sigma^*$) Here we must claim that all elements of W are finite-length strings in the alphabet Σ. We create a state machine M with initial state w, the empty word. If $\mathbf{x}$ is a reachable state, then we let $\mathbf{x} \to \mathbf{x}y$ be a possible transition for every $y \in \Sigma$. Thus, the elements of W are exactly the reachable states in this state machine! Our preserved invariant is this: "$\mathbf{x}$ is a word in Σ^*." If $\mathbf{x}$ is a word in Σ^* (so $\mathbf{x} = x_1 \dots x_n \in \Sigma^n$ for some n), then the concatenation $\mathbf{x}y = x_1 \dots x_n y \in \Sigma^{n+1}$ is also a word in Σ^*. By the invariance principle, every element of W is also an element of Σ^*.

($\Sigma^* \subseteq W$) For this case, we need to demonstrate that for any word $x_1 \dots x_n \in \Sigma^*$, we have that $x_1 \dots x_n \in W$. Here we use regular induction to prove "$\Sigma^n \subseteq W$" for all $n \geq 0$.

Case $n = 0$: $\Sigma^0 = \{w\}$ where w is the empty word. $w \in W$.

(Induction Hypothesis) Let $n \geq 0$ and suppose that $\Sigma^n \subseteq W$.

Case $n + 1$: Let $x_1 \dots x_n x_{n+1} \in \Sigma^{n+1}$. The word $x_1 \dots x_n$ is in Σ^n, and $\Sigma^n \subseteq W$ by the Induction Hypothesis. By the recursive step, since $x_{n+1} \in \Sigma$ and $x_1 \dots x_n \in W$, we have that $x_1 \dots x_n x_{n+1} \in W$. Therefore, $\Sigma^{n+1} \in W$.

Since every element in Σ^* is in some set Σ^n, we have that $\Sigma^* \subseteq W$. □

We can define an invariant, called *length*, on the set Σ^*. Define the function $\ell : \Sigma^* \to \mathbb{N}$ as (Basis) $\ell(w) = 0$ where w is the empty word, and (Recursive Step) if $\mathbf{x} \in W$, then $\ell(\mathbf{x}1) = \ell(\mathbf{x}) + 1$ and $\ell(\mathbf{x}0) = \ell(\mathbf{x}) + 1$. The length of a string $\mathbf{x}$ is usually denoted by $|\mathbf{x}| = \ell(\mathbf{x})$.

We can define a set S recursively by using a Basis: $x_1, \dots, x_n \in S$ and a Recursive Step "if $x \in S$ then $f(x) \in S$" (or sometimes, "if $x \in S$, then $A_x \subseteq S$" where A_x is a set defined by the element x).

Note. In the previous example, we used one element for the Basis: the empty word is in W. In the Recursive Step, we use the set form: for an alphabet Σ and a word $\mathbf{x} = x_1 x_2 \dots x_n$, let $A_{\mathbf{x}} = \{x_1 \dots x_n y : y \in \Sigma\}$. Then our Recursive Step is that for all $\mathbf{x} \in W$, $A_{\mathbf{x}} \subseteq W$.

Exclusion Rule. A recursively defined set contains *only* the elements that are *required* to be in the set by the implications.

An equivalent definition of the exclusion rule is that we have a state machine M whose states are *all possible set elements*, and we have a transition $x \to y$ if and only if the implication step has $y = f(x)$ or $y \in A_x$. We can then define the set S to be all states reachable from an initial state (defined by the base case). This allows us to use the Invariance Principle when proving things about elements of our recursively-defined set!

Def: A proof by *structural induction* consists of two parts. These parts are

BASIS STEP: Show that the result holds for all elements specified in the basis step of the recursive definition to be in the set.

RECURSIVE STEP: Show that if the statement is true for each of the elements used to construct new elements in the recursive step of the definition, the result holds for these new elements.

The Recursive Step of this proof is essentially showing that in our state-machine description, the property "$x \in S$" is a preserved invariant.

Ex: Define $S \subseteq \mathbb{R}$ as (Basis) $0 \in S$, and (Recursive Step) If $x \in S$, then $x + 1 \in S$. Prove that $S = \mathbb{N}$.

Ex: Define $S \subseteq \mathbb{R}$ as (Basis) $2 \in S$, and (Recursive Step) If $x \in S$ and $y \in S$, then $xy \in S$. Prove that $S = \{2^i : i \geq 1\}$.

Ex: Define $S \subseteq \mathbb{R}$ as (Basis) $1 \in S$, and (Recursive Step) If $x \in S$, then $x + x \in S$ and $\frac{1}{x} \in S$. Prove that $S = \{2^i : i \in \mathbb{Z}\}$.

Ex: Define $S \subseteq \mathbb{R}$ as (Basis) $0 \in S$, and (Recursive Step) If $x \in S$, then $x + 1 \in S$, $-x \in S$, and if $x > 0$ then $\frac{1}{x} \in S$. Prove that $S = \mathbb{Q}$. (Hint: To show that $\mathbb{Q} \subseteq S$, let $\frac{p}{q}$ be a fraction where p and q have no

common divisors other than 1 and $p \geq 0$ and $q > 0$. Use strong induction on $p + q \geq 1$ to show that each such fraction $\frac{p}{q}$ is in S; use either $\frac{p}{q} - 1$ or $\frac{q}{p} - 1$ in the induction step.)

Ex: Let $\Sigma = \{0, 1\}$. Define $S \subseteq \Sigma^*$ as (Basis) $w \in S$ where w is the empty word, and (Recursive Step) If $x_1 \dots x_n \in S$ then $x_1 \dots x_n 0 \in S$ and for every $k \geq 0$, the string $x_1 \dots x_n 1 \underbrace{0 \dots 0}_{k \text{ zeroes}} 1 \in S$. Prove that $S = \{\mathbf{x} \in \Sigma^* : \text{there are an even number of 1's in } \mathbf{x}\}$.

Ex: Let $\Sigma = \{0, 1\}$. Define $S \subseteq \Sigma^*$ as (Basis) $0 \in S$, $1 \in S$ and $w \in S$ where w is the empty word, and (Recursive Step) If $\mathbf{x} \in S$, then $0\mathbf{x}0 \in S$ and $1\mathbf{x}1 \in S$. Prove that S is the set of *palindromes*: strings $x_1 \dots x_n$ where $x_1 \dots x_n = x_n \dots x_1$.

Ex*: Let $\Sigma = \{0, 1\}$. Define $S \subseteq \Sigma^*$ and $T \subseteq \Sigma^*$ as (Basis) $w \in S$ where w is the empty word and $1 \in T$, and (Recursive Step) If $x_1 \dots x_n \in S$ then $x_1 \dots x_n 0 \in S$ and $x_1 \dots x_n 1 \in T$; if $x_1 \dots x_n \in T$ then $x_1 \dots x_n 0 \in T$ and $x_1 \dots x_n 1 \in T$. Prove that $S = \{\mathbf{x} \in \Sigma^* : \text{there are an even number of 1's in } \mathbf{x}\}$ and $T = \{\mathbf{x} \in \Sigma^* : \text{there are an odd number of 1's in } \mathbf{x}\}$.

Ex*: (See LLM 6.2) Let $\Sigma = \{[,]\}$. Define $M \subseteq \Sigma^*$ as (Basis) $w \in M$ where w is the empty word, and (Recursive Step) If $\mathbf{x} \in M$ and $\mathbf{y} \in M$, then $\mathbf{x} \cdot \mathbf{y} \in M$ ($\mathbf{x} \cdot \mathbf{y}$ is the concatenation of the strings) and $[\mathbf{x}] \in M$. Prove that the set M is equal to the set of *correctly-nested brackets*, that is in a string $x_1 \dots x_n$ the opening brackets [are paired with closing brackets] such that if (x_i, x_j) is such a pair ($[,]$) then $i < j$, and if (x_i, x_j) and (x_a, x_b) are two distinct pairs where $i < a$ then $b < j$.

Recommended Homework: Rosen 5.3: 1–4, 5–6, 7–8, 12–19, 20–21, 23–25, 26–27, 28*, 32, 36*, 40*, 47*, 48–52, 55., 60–62, 63–65.

3 Friday, February 25, 2015

3.1 Rosen 5.3: Recursive Definitions and Structural Induction

ReadingL Rosen 5.3, Ducks 10.5, LLM 10.1,

Today, we will define *rooted binary trees* recursively. We will also define some invariants of binary trees using this recursive construction.

A *rooted tree* T is given by a set $V(T)$ of *vertices*, where every vertex has at most two *children*, if u is a child of v, then v is the *parent* of u, every vertex has at most one parent, and there is exactly one vertex with no parent; that vertex is called the *root*.

Define the set T of *rooted trees* using the following recursive definition:

Basis Step: The tree on one vertex is in T.

Recursive Step: If $T_1, \ldots, T_k$ are in T, then let $V = \{r\} \cup \{(i, v) : i \in \{1, \ldots, k\}, v \in V(T_i)\}$ and for $i \in \{1, \ldots, k\}$ let (i, u) be a child of (i, v) exactly when u is a child of v in T_i. Let r be the parent of (i, r_i) for each $i \in \{1, \ldots, k\}$ where r_i is the root of T_i.

A *rooted binary tree* is a rooted tree where every vertex has at most two children, and each is labeled as a *left* or *right* child (we can have just one child, but not two of the same type).

The *extended binary trees* is the set B defined as:

Basis Step: The empty set $\varnothing$ is in B.

Recursive Step: If T_1, T_2 are in B, then let $V = \{r\} \cup \{(i, v) : i \in \{1, 2\}, v \in V(T_i)\}$ and for $i \in \{1, 2\}$ let (i, u) be a child of (i, v) exactly when u is a child of v in T_i. If $T_1 \neq \varnothing$, then the left child of r is $(1, r_1)$ where r_1 is the root of T_1. If $T_2 \neq \varnothing$, then the right child of r is $(2, r_2)$ where r_2 is the root of T_2.

Recommended Homework: Rosen 5.3: 44.