Whence it is manifest that if we could find characters or signs appropriate for expressing all our thoughts. . . , we could in all subjects—in so far as they are amenable to reasoning—*accomplish what is done in Arithmetic and Geometry. For all inquiries which depend on reasoning would be performed by the transposition of characters and by a kind of calculus, which would immediately facilitate the discovery of beautiful results. . . . And if someone would doubt my results, I should say to him: "Let us calculate, Sir," and thus by taking to pen and ink, we should soon settle the question.*

— Gottfried Wilhelm Leibniz, *Preface to the General Science* (1677), translated by Philip Wiener (1951)

I hope the reader sees that the alphabet can be understood by any intelligent being who has any one of the five senses left him,—by all rational men, that is, excepting the few eyeless deaf persons who have lost both taste and smell in some complete paralysis. . . . Whales in the sea can telegraph as well as senators on land, if they will only note the difference between long spoutings and short ones. . . . A tired listener at church, by properly varying his long yawns and his short ones, may express his opinion of the sermon to the opposite gallery before the sermon is done.

— Edward Everett Hale, "The Dot and Line Alphabet", Altlantic Monthy (October 1858)

If indeed, as Hilbert asserted, mathematics is a meaningless game played with meaningless marks on paper, the only mathematical experience to which we can refer is the making of marks on paper.

— Eric Temple Bell, *The Queen of the Sciences* (1931)

1 Strings

Throughout this course, we will discuss dozens of algorithms and computational models that manipulate sequences: one-dimensional arrays, linked lists, blocks of text, walks in graphs, sequences of executed instructions, and so on. Ultimately the input and output of any algorithm must be representable as a finite string of symbols—the raw contents of some contiguous portion of the computer's memory. Reasoning about computation requires reasoning about strings.

This note lists several formal definitions and formal induction proofs related to strings. These definitions and proofs are *intentionally* much more detailed than normally used in practice—most people's intuition about strings is fairly accurate—but the extra precision is necessary for any sort of formal proof. It may be helpful to think of this material as part of the "assembly language" of theoretical computer science. We normally think about computation at a *much* higher level of abstraction, but ultimately every argument must "compile" down to these (and similar) definitions.

1.1 Definitions

Fix an arbitrary finite set $\boldsymbol{\Sigma}$ called the ***alphabet***; the elements of Σ are called ***symbols*** or ***characters***. As a notational convention, I will always use lower-case letters near the start of the English alphabet ($a, b, c, \ldots$) as symbol variables, and *never* as explicit symbols. For explicit symbols, I will always use fixed-width upper-case letters (`A`, `B`, `C`, . . .), digits (`0`, `1`, `2`, . . .), or other symbols (`◇`, `$`, `#`, `•`, . . .) that are clearly distinguishable from variables.

A ***string*** (or ***word***) over Σ is a finite sequence of zero or more symbols from Σ. Formally, a string w over Σ is defined recursively as either

- the empty string, denoted by the Greek letter $\boldsymbol{\varepsilon}$ (epsilon),or
- an ordered pair $\boldsymbol{(a, x)}$, where a is a symbol in Σ and x is a string over Σ.

We normally write either $\boldsymbol{a \cdot x}$ or simply $\boldsymbol{ax}$ to denote the ordered pair (a, x). Similarly, we normally write explicit strings as sequences of symbols instead of nested ordered pairs; for example, STRING is convenient shorthand for the formal expression $(\mathtt{S},(\mathtt{T},(\mathtt{R},(\mathtt{I},(\mathtt{N},(\mathtt{G},\varepsilon))))))$. As a notational convention, I will always use lower-case letters near the end of the alphabet $(\ldots, w, x, y, z)$ to represent unknown strings, and SHOUTY◇MONOSPACED◇TEXT to represent explicit symbols and (non-empty) strings.

The set of all strings over Σ is denoted $\boldsymbol{\Sigma^*}$ (pronounced "sigma star"). It is very important to remember that every element of Σ^* is a *finite* string, although Σ^* itself is an infinite set containing strings of every possible *finite* length.

The ***length*** $\boldsymbol{|w|}$ of a string w is the number of symbols in w, defined formally as follows:

$$|w| := \begin{cases} 0 & \text{if } w = \varepsilon, \\ 1 + |x| & \text{if } w = ax. \end{cases}$$

For example, the string SEVEN has length 5. Although they are formally different objects, we do not normally distinguish between symbols and strings of length 1.

The ***concatenation*** of two strings x and y, denoted either $\boldsymbol{x \bullet y}$ or simply $\boldsymbol{xy}$, is the unique string containing the characters of x in order, followed by the characters in y in order. For example, the string NOWHERE is the concatenation of the strings NOW and HERE; that is, NOW • HERE = NOWHERE. (On the other hand, HERE • NOW = HERENOW.) Formally, concatenation is defined recusively as follows:

$$w \bullet z := \begin{cases} z & \text{if } w = \varepsilon \\ a \cdot (x \bullet z) & \text{if } w = ax \end{cases}$$

(Here I'm using a larger dot • to formally distinguish the operator that concatenates two arbitrary strings from from the operator · that builds a string from a single character and a string.)

When we describe the concatenation of more than two strings, we normally omit all dots and parentheses, writing $wxyz$ instead of $(w \bullet (x \bullet y)) \bullet z$, for example. This simplification is justified by the fact (which we will prove shortly) that • is associative.

1.2 Induction on Strings

Induction is *the* standard technique for proving statements about recursively defined objects. Hopefully you are already comfortable proving statements about *natural numbers* via induction, but induction actually a far more general technique. Several different variants of induction can be used to prove statements about more general structures; here I describe the variant that I recommend (and actually use in practice). This variant follows two primary design considerations:

- ***The case structure of the proof should mirror the case structure of the recursive definition.*** For example, if you are proving something about all *strings*, your proof should have two cases: Either $w = \varepsilon$, or $w = ax$ for some symbol a and string x.

- ***The inductive hypothesis should be as strong as possible.*** The (strong) inductive hypothesis for statements about natural numbers is *always* "Assume there is no counterexample k such that $k < n$." I recommend adopting a similar inductive hypothesis for strings: "Assume there is no counterexample x such that $|x| < |w|$." Then for the case $w = ax$, we have $|x| = |w| - 1 < |w|$ by definition of $|w|$, so the inductive hypothesis applies to x.

Thus, string-induction proofs have the following boilerplate structure. Suppose we want to prove that every string is perfectly cromulent, whatever that means. The white boxes hide additional proof details that, among other things, depend on the precise definition of "perfectly cromulent".

Proof: Let w be an arbitrary string.
Assume, for every string x such that $|x| < |w|$, that x is perfectly cromulent.
There are two cases to consider.

- Suppose $w = \varepsilon$.

  $\boxed{\quad}$

  Therefore, w is perfectly cromulent.

- Suppose $w = ax$ for some symbol a and string x.
  The induction hypothesis implies that x is perfectly cromulent.

  $\boxed{\quad}$

  Therefore, w is perfectly cromulent.

In both cases, we conclude that w is perfectly cromulent. □

Here are three canonical examples of this proof structure. When developing proofs in this style, I strongly recommend first ***mindlessly*** writing the green text (the boilerplate) with lots of space for each case, then filling in the red text (the actual theorem and the induction hypothesis), and only then starting to actually think.

Lemma 1.1. *For every string w, we have $w \bullet \varepsilon = w$.*

Proof: Let w be an arbitrary string. Assume that $x \bullet \varepsilon = x$ for every string x such that $|x| < |w|$. There are two cases to consider:

- Suppose $w = \varepsilon$.

$$\begin{aligned} w \bullet \varepsilon &= \varepsilon \bullet \varepsilon && \text{because } w = \varepsilon, \\ &= \varepsilon && \text{by definition of concatenation,} \\ &= w && \text{because } w = \varepsilon. \end{aligned}$$

- Suppose $w = ax$ for some symbol a and string x.

$$\begin{aligned} w \bullet \varepsilon &= (a \cdot x) \bullet \varepsilon && \text{because } w = ax, \\ &= a \cdot (x \bullet \varepsilon) && \text{by definition of concatenation,} \\ &= a \cdot x && \text{by the inductive hypothesis,} \\ &= w && \text{because } w = ax. \end{aligned}$$

In both cases, we conclude that $w \bullet \varepsilon = w$. □

Lemma 1.2. *Concatenation adds length: $|w \bullet x| = |w| + |x|$ for all strings w and x.*

Proof: Let w and x be arbitrary strings. Assume that $|y \bullet x| = |y| + |x|$ for every string y such that $|y| < |w|$. (Notice that we are using induction only on w, not on x.) There are two cases to consider:

- Suppose $w = \varepsilon$.

$$\begin{aligned} |w \bullet x| &= |\varepsilon \bullet x| && \text{because } w = \varepsilon \\ &= |x| && \text{by definition of } |\ | \\ &= |\varepsilon| + |x| && |e| = 0 \text{ by definition of } |\ | \\ &= |w| + |x| && \text{because } w = \varepsilon \end{aligned}$$

- Suppose $w = ay$ for some symbol a and string y.

$$\begin{aligned}
|w \bullet x| &= |ay \bullet x| && \text{because } w = ay\\
&= |a \cdot (y \bullet x)| && \text{by definition of } \bullet\\
&= 1 + |y \bullet x| && \text{by definition of } |\,|\\
&= 1 + |y| + |x| && \text{by the inductive hypothesis}\\
&= |ay| + |x| && \text{by definition of } |\,|\\
&= |w| + |x| && \text{because } w = ay
\end{aligned}$$

In both cases, we conclude that $|w \bullet x| = |w| + |x|$. □

Lemma 1.3. *Concatenation is associative:* $(w \bullet x) \bullet y = w \bullet (x \bullet y)$ *for all strings* w, x, *and* y.

Proof: Let w, x, and y be arbitrary strings. Assume that $(z \bullet x) \bullet y = w \bullet (x \bullet y)$ for every string z such that $|z| < |w|$. (Again, we are using induction only on w.) There are two cases to consider.

- Suppose $w = \varepsilon$.

$$\begin{aligned}
(w \bullet x) \bullet y &= (\varepsilon \bullet x) \bullet y && \text{because } w = \varepsilon\\
&= x \bullet y && \text{by definition of } \bullet\\
&= \varepsilon \bullet (x \bullet y) && \text{by definition of } \bullet\\
&= w \bullet (x \bullet y) && \text{because } w = \varepsilon
\end{aligned}$$

- Suppose $w = az$ for some symbol a and some string z.

$$\begin{aligned}
(w \bullet x) \bullet y &= (az \bullet x) \bullet y && \text{because } w = az\\
&= (a \cdot (z \bullet x)) \bullet y && \text{by definition of } \bullet\\
&= a \cdot ((z \bullet x) \bullet y) && \text{by definition of } \bullet\\
&= a \cdot (z \bullet (x \bullet y)) && \text{by the inductive hypothesis}\\
&= az \bullet (x \bullet y) && \text{by definition of } \bullet\\
&= w \bullet (x \bullet y) && \text{because } w = az
\end{aligned}$$

In both cases, we conclude that $(w \bullet x) \bullet y = w \bullet (x \bullet y)$. □

This is not the only boilerplate that one can use for induction proofs on strings. For example, we can modify the inductive case analysis using the following observation: A *non-empty* string w is either a single symbol or the concatenation of two non-empty strings, which (by Lemma 1.2) must be shorter than w. Here is an alternate proof of Lemma 1.3 that uses this alternative recursive structure:

Proof: Let w, x, and y be arbitrary strings. Assume that $(z \bullet x) \bullet y = w \bullet (x \bullet y)$ for every string z such that $|z| < |w|$. There are ***three*** cases to consider.

- Suppose $w = \varepsilon$.

$$\begin{aligned}
(w \bullet x) \bullet y &= (\varepsilon \bullet x) \bullet y && \text{because } w = \varepsilon\\
&= x \bullet y && \text{by definition of } \bullet\\
&= \varepsilon \bullet (x \bullet y) && \text{by definition of } \bullet\\
&= w \bullet (x \bullet y) && \text{because } w = \varepsilon
\end{aligned}$$

- Suppose w is equal to some symbol a.

$$\begin{aligned}
(w \bullet x) \bullet y &= (a \bullet x) \bullet y && \text{because } w = a \\
&= (a \cdot x) \bullet y && \text{because } a \bullet z = a \cdot z \text{ by definition of } \bullet \\
&= a \cdot (x \bullet y) && \text{by definition of } \bullet \\
&= a \bullet (x \bullet y) && \text{because } a \bullet z = a \cdot z \text{ by definition of } \bullet \\
&= w \bullet (x \bullet y) && \text{because } w = a
\end{aligned}$$

- Suppose $w = uv$ for some nonempty strings u and v.

$$\begin{aligned}
(w \bullet x) \bullet y &= ((u \bullet v) \bullet x) \bullet y && \text{because } w = uv \\
&= (u \bullet (v \bullet x)) \bullet y && \text{by the inductive hypothesis, because } |u| < |w| \\
&= u \bullet ((v \bullet x) \bullet y) && \text{by the inductive hypothesis, because } |u| < |w| \\
&= u \bullet (v \bullet (x \bullet y)) && \text{by the inductive hypothesis, because } |v| < |w| \\
&= (u \bullet v) \bullet (x \bullet y) && \text{by the inductive hypothesis, because } |u| < |w| \\
&= w \bullet (x \bullet y) && \text{because } w = uv
\end{aligned}$$

In both cases, we conclude that $(w \bullet x) \bullet y = w \bullet (x \bullet y)$. □

1.3 Indices, Substrings, and Subsequences

For any string w and any integer $1 \le i \le |w|$, the expression $\boldsymbol{w_i}$ denotes the ith symbol in w, counting from left to right. More formally, w_i is recursively defined as follows:

$$w_i := \begin{cases} a & \text{if } w = ax \text{ and } i = 1 \\ x_{i-1} & \text{if } w = ax \text{ and } i > 1 \end{cases}$$

As one might reasonably expect, w_i is formally undefined if $i < 1$ or $w = \varepsilon$, and therefore (by induction) if $i > |w|$. The integer i is called the ***index*** of w_i.

We sometimes write strings as a concatenation of their constituent symbols using this subscript notation: $w = w_1 w_2 \cdots w_{|w|}$. While standard, this notation is slightly misleading, since it *incorrectly* suggests that the string w contains at least three symbols, when in fact w could be a single symbol or even the empty string.

In actual code, subscripts are usually expressed using the bracket notation $\boldsymbol{w[i]}$. Brackets were introduced as a typographical convention over a hundred years ago because subscripts and superscripts[1] were difficult or impossible to type.[2] We sometimes write strings as explicit arrays $w[1\,..\,n]$, with the

[1]The same bracket notation is also used for bibliographic references, instead of the traditional footnote/endnote superscripts, for exactly the same reasons.

[2]A ***typewriter*** is an obsolete mechanical device loosely resembling a computer keyboard. Pressing a key on a typewriter moves a lever (called a "typebar") that strikes a cloth ribbon full of ink against a piece of paper, leaving the image of a single character. Many historians believe that the ordering of letters on modern keyboards (`QWERTYUIOP`) evolved in the late 1800s, reaching its modern form on the 1874 Sholes & Glidden Type-Writer™, in part to separate many common letter pairs, to prevent typebars from jamming against each other; this is also why the keys on most modern keyboards are arranged in a slanted grid. (The common folk theory that the ordering was deliberately intended to slow down typists doesn't withstand careful scrutiny.) A more recent theory suggests that the ordering was influenced by telegraph[3] operators, who found older alphabetic arrangements confusing, in part because of ambiguities in American Morse Code.

understanding that $n = |w|$. Again, this notation is potentially misleading; always remember that n might be zero; the string/array could be empty.

A ***substring*** of a string w is another string obtained from w by deleting zero or more symbols from the beginning and from the end. Formally, a string y is a substring of w if and only if there are strings x and z such that $w = xyz$. Extending the array notation for strings, we write $\boldsymbol{w[i\,..\,j]}$ to denote the substring of w starting at w_i and ending at w_j. More formally, we define

$$w[i\,..\,j] := \begin{cases} \varepsilon & \text{if } j < i, \\ w_i \cdot w[i+1\,..\,j] & \text{otherwise.} \end{cases}$$

A ***proper substring*** of w is any substring other than w itself. For example, LAUGH is a proper substring of SLAUGHTER. Whenever y is a (proper) substring of w, we also call w a (proper) ***superstring*** of y.

A ***prefix*** of $w[1\,..\,n]$ is any substring of the form $w[1\,..\,j]$. Equivalently, a string p is a ***prefix*** of another string w if and only if there is a string x such that $px = w$. A ***proper prefix*** of w is any prefix except w itself. For example, DIE is a proper prefix of DIET.

Similarly, a suffix of $w[1\,..\,n]$ is any substring of the form $w[i\,..\,n]$. Equivalently, a string s is a ***suffix*** of a string w if and only if there is a string x such that $xs = w$. A ***proper suffix*** of w is any suffix except w itself. For example, YES is a proper suffix of EYES, and HE is both a proper prefix and a proper suffix of HEADACHE.

A ***subsequence*** of a string w is a strong obtained by deleting zero or more symbols from *anywhere* in w. More formally, z is a subsequence of w if and only if

- $z = \varepsilon$, or
- $w = ax$ for some symbol a and some string x such that z is a subsequence of x.
- $w = ax$ and $z = ay$ for some symbol a and some strings x and y, and y is a subsequence of x.

A ***proper subsequence*** of w is any subsequence of w other than w itself. Whenever z is a (proper) subsequence of w, we also call w a (proper) ***supersequence*** of z.

Substrings and subsequences are not the same objects; don't confuse them! Every substring of w is also a subsequence of w, but not every subsequence is a substring. For example, METAL is a subsequence, but not a substring, of MEATBALL. To emphasize the distinction, we sometimes redundantly refer to substrings of w as ***contiguous*** substrings, meaning all their symbols appear together in w.

[3]A ***telegraph*** is an obsolete electromechanical communication device consisting of an electrical circuit with a switch at one end and an electromagnet at the other. The sending operator would press and release a key, closing and opening the circuit, originally causing the electromagnet to push a stylus onto a moving paper tape, leaving marks that could be decoded by the receiving operator. (Operators quickly discovered that they could directly decode the clicking sounds made by the electromagnet, and so the paper tape became obsolete almost immediately.) The most common scheme within the US to encode symbols, developed by Alfred Vail and Samuel Morse in 1837, used (mostly) short (·) and long (−) marks—now called "dots" and "dashes", or "dits" and "dahs"—separated by gaps of various lengths. American Morse code (as it became known) was ambiguous; for example, the letter Z and the string SE were both encoded by the sequence $\cdots\ \cdot$ ("di-di-dit, dit"). This ambiguity has been blamed for the S key's position on the typewriter keyboard near E and Z.

Vail and Morse were of course not the first people to propose encoding symbols as strings of bits. That honor apparently falls to Francis Bacon, who devised a five-bit binary encoding of the alphabet (except for the letters J and U) in 1605 as the basis for a steganographic code—a method or hiding secret message in otherwise normal text.

Exercises

Most of the following exercises ask for proofs of various claims about strings. For each claim, give a complete, self-contained, formal proof by inductive definition-chasing, using the boilerplate structure recommended in Section 1.2. You can use Lemmas 1.1, 1.2, and 1.3, but don't assume any other facts about strings that you have not proved. Do not use the words "obvious" or "clearly" or "just". Most of these claims **are** in fact obvious; the real exercise is understanding **why** they're obvious.

1. For any symbol a and any string w, let $\boldsymbol{\#(a, w)}$ denote the number of occurrences of a in w. For example, $\#(\texttt{A}, \texttt{BANANA}) = 3$ and $\#(\texttt{X}, \texttt{FLIBBERTIGIBBET}) = 0$.

   (a) Give a formal recursive definition of the function $\# \colon \Sigma \times \Sigma^* \to \mathbb{N}$.

   (b) Prove that $\#(a, xy) = \#(a, x) + \#(a, y)$ for every symbol a and all strings x and y. Your proof must rely on both your answer to part (a) and the formal recursive definition of string concatenation.

2. Recursively define a set L of strings over the alphabet $\{\texttt{0}, \texttt{1}\}$ as follows:

   - The empty string ε is in L.
   - For any two strings x and y in L, the string $\texttt{0}x\texttt{1}y\texttt{0}$ is also in L.
   - These are the only strings in L.

   (a) Prove that the string $\texttt{000010101010010100}$ is in L.

   (b) Prove by induction that every string in L has exactly twice as many $\texttt{0}$s as $\texttt{1}$s. (You may assume the identity $\#(a, xy) = \#(a, x) + \#(a, y)$ for any symbol a and any strings x and y; see Exercise 1(b).)

   (c) Give an example of a string with exactly twice as many $\texttt{0}$s as $\texttt{1}$s that is *not* in L.

3. For any string w and any non-negative integer n, let $\boldsymbol{w^n}$ denote the string obtained by concatenating n copies of w; more formally, we define

$$w^n := \begin{cases} \varepsilon & \text{if } n = 0 \\ w \bullet w^{n-1} & \text{otherwise} \end{cases}$$

   For example, $(\texttt{BLAH})^5 = \texttt{BLAHBLAHBLAHBLAHBLAH}$ and $\varepsilon^{374} = \varepsilon$.

   Prove that $w^m \bullet w^n = w^{m+n}$ for every string w and all integers non-negative integer n and m.

4. Let w be an arbitrary string, and let $n = |w|$. Prove each of the following statements.

   (a) w has exactly $n + 1$ prefixes.

   (b) w has exactly n proper suffixes.

   (c) w has at most $n(n + 1)/2$ distinct substrings.

   (d) w has at most $2^n - 1$ proper subsequences.

5. The ***reversal*** $\boldsymbol{w^R}$ of a string w is defined recursively as follows:

$$w^R := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ x^R \bullet a & \text{if } w = a \cdot x \end{cases}$$

   (a) Prove that $|w^R| = |w|$ for every string w.
   (b) Prove that $(wx)^R = x^R w^R$ for all strings w and x.
   (c) Prove that $(w^R)^n = (w^n)^R$ for every string w and every integer $n \geq 0$. (See Exercise 1.)
   (d) Prove that $(w^R)^R = w$ for every string w.

6. Let w be an arbitrary string, and let $n = |w|$. Prove the following statements for all indices $1 \leq i \leq j \leq k \leq n$.

   (a) $\big|w[i\,..\,j]\big| = j - i + 1$
   (b) $w[i\,..\,j] \bullet w[j+1\,..\,k] = w[i\,..\,k]$
   (c) $w^R[i\,..\,j] = (w[i'\,..\,j'])^R$ where $i' = |w| + 1 - j$ and $j' = |w| + 1 - i$.

7. A ***palindrome*** is a string that is equal to its reversal.

   (a) Give a recursive definition of a palindrome over the alphabet Σ.
   (b) Prove that any string p meets your recursive definition of a palindrome if and only if $p = p^R$.

8. A string $w \in \Sigma^*$ is called a ***shuffle*** of two strings $x, y \in \Sigma^*$ if at least one of the following recursive conditions is satisfied:

   - $w = x = y = \varepsilon$.
   - $w = aw'$ and $x = ax'$ and w' is a shuffle of x' and y, for some $a \in \Sigma$ and some $w', x' \in \Sigma^*$.
   - $w = aw'$ and $y = ay'$ and w' is a shuffle of x and y', for some $a \in \Sigma$ and some $w', y' \in \Sigma^*$.

   For example, the string BANANANANASA is a shuffle of the strings BANANA and ANANAS.

   (a) Prove that if w is a shuffle of x and y, then $|w| = |x| + |y|$.
   (b) Prove that if w is a shuffle of x and y, then w^R is a shuffle of x^R and y^R.

9. Consider the following pair of mutually recursive functions on strings:

$$evens(w) := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ odds(x) & \text{if } w = ax \end{cases} \qquad odds(w) := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ a \cdot evens(x) & \text{if } w = ax \end{cases}$$

   (a) Prove the following identity for all strings w and x:

$$evens(w \bullet x) = \begin{cases} evens(w) \bullet evens(x) & \text{if } |w| \text{ is even,} \\ evens(w) \bullet odds(x) & \text{if } |w| \text{ is odd.} \end{cases}$$

(b) State and prove a similar identity for $\mathit{odds}(w \bullet x)$.

10. For any positive integer n, the ***Fibonacci string F_n*** is defined recursively as follows:

$$F_n = \begin{cases} \texttt{0} & \text{if } n = 1, \\ \texttt{1} & \text{if } n = 2, \\ F_{n-2} \bullet F_{n-1} & \text{otherwise.} \end{cases}$$

For example, $F_6 = \texttt{10101101}$ and $F_7 = \texttt{0110110101101}$.

(a) Prove that for every integer $n \geq 2$, the string F_n can also be obtained from F_{n-1} by replacing every occurrence of `0` with `1` and replacing every occurrence of `1` with `01`. More formally, prove that $F_n = \mathit{Finc}(F_{n-1})$, where

$$\mathit{Finc}(w) = \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ \texttt{1} \cdot \mathit{Finc}(x) & \text{if } w = \texttt{0}x \\ \texttt{01} \bullet \mathit{Finc}(x) & \text{if } w = \texttt{1}x \end{cases}$$

[Hint: First prove that $\mathit{Finc}(x \bullet y) = \mathit{Finc}(x) \bullet \mathit{Finc}(y)$.]

(b) Prove that `00` and `111` are not substrings of any Fibonacci string F_n.

11. Prove that the following three properties of strings are in fact identical.

    - A string $w \in \{\mathtt{0}, \mathtt{1}\}^*$ is ***balanced*** if it satisfies one of the following conditions:
      - $w = \varepsilon$,
      - $w = \mathtt{0}x\mathtt{1}$ for some balanced string x, or
      - $w = xy$ for some balanced strings x and y.
    - A string $w \in \{\mathtt{0}, \mathtt{1}\}^*$ is ***erasable*** if it satisfies one of the following conditions:
      - $w = \varepsilon$, or
      - $w = x\mathtt{01}y$ for some strings x and y such that xy is erasable. (The strings x and y are not necessarily erasable.)
    - A string $w \in \{\mathtt{0}, \mathtt{1}\}^*$ is ***conservative*** if it satisfies ***both*** of the following conditions:
      - w has an equal number of `0`s and `1`s, and
      - no prefix of w has more `0`s than `1`s.

    (a) Prove that every balanced string is erasable.

    (b) Prove that every erasable string is conservative.

    (c) Prove that every conservative string is balanced.

    *[Hint: To develop intuition, it may be helpful to think of `0`s as left brackets and `1`s as right brackets, but **don't** invoke this intuition in your proofs.]*

12. A string $w \in \{\mathtt{0}, \mathtt{1}\}^*$ ***equitable*** if it has an equal number of `0`s and `1`s.

    (a) Prove that a string w is equitable if and only if it satisfies one of the following conditions:
    - $w = \varepsilon$,
    - $w = \mathtt{0}x\mathtt{1}$ for some equitable string x,
    - $w = \mathtt{1}x\mathtt{0}$ for some equitable string x, or
    - $w = xy$ for some equitable strings x and y.

    (b) Prove that a string w is equitable if and only if it satisfies one of the following conditions:
    - $w = \varepsilon$,
    - $w = x\mathtt{01}y$ for some strings x and y such that xy is equitable, or
    - $w = x\mathtt{10}y$ for some strings x and y such that xy is equitable.

    In the last two cases, the individual strings x and y are not necessarily equitable.

    (c) Prove that a string w is equitable if and only if it satisfies one of the following conditions:
    - $w = \varepsilon$,
    - $w = xy$ for some balanced string x and some equitable string y, or
    - $w = x^R y$ for some for some balanced string x and some equitable string y.

    (See the previous exercise for the definition of "balanced".)