> *Vario, inquit [Epicurus], ordine ac positione conveniunt atomi sicut literae, quae cum sint paucae, varie tamen collocatae innumerabilia verba conficiunt.*
> *[Atoms, like letters, says Epicurus, come together in various orders and positions; there are few of them, but different combinations produce countless words.]*
>
> — Gottfried Leibniz, *Dissertatio de Arte Combinatoria* (1666)

> *THOMAS GODFREY, a self-taught mathematician, great in his way, and afterward inventor of what is now called Hadley's Quadrant. But he knew little out of his way, and was not a pleasing companion; as, like most great mathematicians I have met with, he expected universal precision in everything said, or was forever denying or distinguishing upon trifles, to the disturbance of all conversation. He soon left us.*
>
> — Benjamin Franklin, *Memoirs, Part 1* (1771)
> describing one of the founding members of the Junto

> *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.

But the actual definitions and theorems are *not* the point. The point of playing with this material is to get some experience working with formal/mechanical definitions and proofs, especially inductive definitions and recursive proofs. Or should I say recursive definitions and inductive proofs? Whatever, they're the same thing. Strings are a particularly simple and convenient playground for $\left\{{\text{induct} \atop \text{recurs}}\right\}$ion; we'll see *many* more examples throughout the course. When you read this note, don't just look at the *content* of the definitions and proofs; pay close attention to their *structure* and the *process* for creating them.

1.1 Strings

Fix an arbitrary finite set Σ called the ***alphabet***; the individual 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. For further emphasis, I will almost always typeset explicit symbols in RED.

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 English alphabet $(\ldots, w, x, y, z)$ for string variables, and SHOUTY⋄RED⋄MONOSPACED⋄TEXT to typeset explicit (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.

1.2 Recursive Functions

Our first several proofs about strings will involve two natural functions, one giving the length of a string, the other gluing two strings together into a larger string. These functions behave exactly the way you think they do, but if we actually want to *prove* anything about them, we first have to *define* them in a way that supports formal proofs. Because the objects on which these functions act—strings–are defined recursively, the functions must also be defined recursively.

1.2.1 Length

The ***length*** $\boldsymbol{|w|}$ of a string w is the number of symbols in w. For example, the string FIFTEEN has length 7, the string SEVEN has length 5, and the string 5 has length 1. (Although they are formally different objects, we rarely distinguish between symbols and strings of length 1.) The length function is defined recursively as follows:

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

For example, we can compute $|\mathtt{Two}|$ by repeatedly expanding this definition as follows:

$$\begin{aligned} |\mathtt{TWO}| &= 1 + |\mathtt{WO}| \\ &= 1 + (1 + |\mathtt{O}|) \\ &= 1 + (1 + (1 + |\varepsilon|)) \\ &= 1 + (1 + (1 + 0)) \\ &= 1 + (1 + 1) \\ &= 1 + 2 \\ &= 3 \end{aligned}$$

The first four equalities are from the definition of length; the last three are just arithmetic. But if we truly believe in the Recursion Fairy, we can write this derivation much more simply as follows!

$$\begin{aligned} |\texttt{SEVEN}| &= 1 + |\texttt{EVEN}| \\ &= 1 + 4 \\ &= 5 \end{aligned} \quad \Big) \; *** \textit{ Recursion! } ***$$

1.2.2 Concatenation

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 recursively 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 syntactic sugar · that builds a string from a single character and a string.) For example, we can compute NOW • HERE as follows:

$$\begin{aligned} \texttt{NOW} \bullet \texttt{HERE} &= \texttt{N} \cdot (\texttt{OW} \bullet \texttt{HERE}) \\ &= \texttt{N} \cdot (\texttt{O} \cdot (\texttt{W} \bullet \texttt{HERE})) \\ &= \texttt{N} \cdot (\texttt{O} \cdot (\texttt{W} \cdot (\varepsilon \bullet \texttt{HERE}))) \\ &= \texttt{N} \cdot (\texttt{O} \cdot (\texttt{W} \cdot \texttt{HERE})) \\ &= \texttt{N} \cdot (\texttt{O} \cdot \texttt{WHERE}) \\ &= \texttt{N} \cdot \texttt{OWHERE} \\ &= \texttt{NOWHERE} \end{aligned}$$

Or more simply, with the help of the Recursion Fairy:

$$\begin{aligned} \texttt{NOW} \bullet \texttt{HERE} &= \texttt{N} \cdot (\texttt{OW} \bullet \texttt{HERE}) \\ &= \texttt{N} \cdot \texttt{OWHERE} \\ &= \texttt{NOWHERE} \end{aligned} \quad \Big) \; *** \textit{ Recursion! } ***$$

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 the function • is associative.

1.3 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 is 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. Some proofs may require breaking the second case into even finer subcases.
- ***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 $w = \varepsilon$.

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

  Therefore, w is perfectly cromulent.

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

The strategy I strongly recommend for developing proofs in this style is to start by ***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. Here is a canonical examples of this proof structure, with boilerplate in green and induction hypothesis and result in red.

Lemma 1.1. *Adding nothing does nothing: 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 $w \bullet \varepsilon = w$. □

Many students are confused (or at least bored and distracted) by the fact that we are proving *mind-bogglingly* obvious facts. If you're one of these students, try to remember that ***the lemmas themselves are not the point.*** Pay close attention to the *structure* of the proofs. Notice how each proof follows the boilerplate described above. Notice how every sentence of the proof follows *mechanically* from earlier sentences, definitions, and the rules of standard logic and arithmetic.

1.4 More than One String

But what if the result that we want to prove involves more than one string? Do we need to do induction on every string in the theorem? How would we do that? Or do we only need induction on one of the string? Okay, then which one? Let's consider the following example:

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

It's easy to write down the original boilerplate, but what should we write down for the induction hypothesis? Are we inducting on w, or on x, or on both? The trick to answering this question is to refer back to the definitions of the terms of the theorem, and in particular in the *recursive* cases within those definitions. Faced with this theorem, I might scribble something like this on a piece of scratch paper:

$$\begin{tikzcd} {|w \bullet x|} \arrow[d, Rightarrow, "w \text{ long}"] \\ {|ay \bullet x|} \arrow[d, Rightarrow, "\text{def. } \bullet"] \\ {|a \cdot (y \bullet x)|} \arrow[d, Rightarrow, "\text{def. } |\,|"] \\ {1 + |y \bullet x|} \end{tikzcd}$$

y shorter than w

$\Longrightarrow$ induct on w

Meanwhile, here is roughly what would run through my head when I wrote that down.

> *How is the expression $|w \bullet x|$ actually defined?*
> *But first, how is the subexpression $w \bullet x$ defined?*
> *These definitions have cases. Ew. For now let's assume w is long.*
> *So we can write $w = ay$.*
> *Now the definition of $\bullet$ says $(ay) \bullet x = a \cdot (y \bullet x)$.*
> *So $|(ay) \bullet x| = |a \cdot (y \bullet x)|$*
> *And now the definition of $|\cdot|$ says $|a \cdot (y \bullet x)| = 1 + |y \bullet x|$.*
> *Look, there's an expression $|y \bullet x|$ just like we started with.*

> *Can we give that to the Induction Fairy?*
> *The first string y is shorter than w, but the second string x is the same.*
> ***Yes! When we recurse, the first string gets shorter!***
> *Cool. So let's induct on the first string.*

One nice side effect of this thought process is that it already gives us most of the inductive case of the proof!

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! I'll say more about this after the proof.) There are two cases to consider:

- Suppose $w = \varepsilon$.

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

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

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

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

Exactly the same approach works for claims about more than strings. For example, if I want to prove that concatenation is associative ($(w \bullet x) \bullet y = w \bullet (x \bullet y)$), I would expand the definition of $(w \bullet x) \bullet y$ as follows:

$$\begin{tikzcd}
(w \bullet x) \bullet y \arrow[d, Rightarrow, "w \text{ long}"] \\
(az \bullet x) \bullet y \arrow[d, Rightarrow, "\text{def. } \bullet"] \\
(a \cdot (z \bullet x)) \bullet y \arrow[d, Rightarrow, "\text{def. } \bullet"] \\
a \cdot \underline{((z \bullet x)) \bullet y)}
\end{tikzcd}$$

z shorter than w
$\Longrightarrow$ induct on w!

At each step, we are mechanically applying the recursive case of the definition of concatenation to drag the first symbol a to the front, eventually leaving a smaller instance of the theorem we're trying to prove. Okay, now we're ready to write down the proof.

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 $(z \bullet x) \bullet y = z \bullet (x \bullet y)$ 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 $(w \bullet x) \bullet y = w \bullet (x \bullet y)$. □

1.5 More Than One Path up the Mountain

This is not the *only* boilerplate that one can use for induction proofs on strings. For example, we can model our case analysis on the following observation, whose easy proof we leave as an exercise (hint, hint): A string $w \in \Sigma^*$ is ***non-empty*** if and only if either

- $w = a$ for some symbol $a \in \Sigma$, or
- $w = x \bullet y$ for some *non-empty* strings x and y.

In the latter case, Lemma 1.2 implies that $|x| < |w|$ and $|y| < |w|$, so in an inductive proof, we can apply the inductive hypothesis to either x or y (or even both).

Here is a proof of Lemma 1.3 that uses this alternative recursive structure:

Proof: Let w, x, and y be arbitrary strings. Assume that $(z \bullet x') \bullet y' = z \bullet (x' \bullet y')$ for all strings x', y', and z such that $|z| < |w|$. (We need a stronger induction hypothesis here than in the previous proofs!) 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 $w = u \bullet v$ for some nonempty strings u and v.

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

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

1.6 Indices, Substrings, and Subsequences

Finally, I'll conclude this note by formally defining several other common functions and terms related to strings.

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 completely standard, this notation is slightly misleading, because 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

[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.

$w[1\,..\,n]$, with the 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 string 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 ··· · ("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. Here "prove" means give a complete, self-contained, formal proof by inductive definition-chasing, using the boilerplate structure recommended in Section 1.3. Feel free to use Lemmas 1.1, 1.2, and 1.3 without proof, but don't assume any other facts about strings that you have not actually proved. (Some later exercises rely on results proved in earlier exercises.) Do not appeal to intuition, and do not use the words "obvious" or "clearly" or "just". Most of these claims *are* in fact obvious; the real exercise is understanding and formally expressing *why* they're obvious.

> **Note to instructors:** ***Do not assign any of these problems—especially on exams—before solving them yourself.*** It's *very* easy to underestimate the difficulty of these problems, or at least the lengths of their solutions, which for exams is a reasonable proxy for difficulty. Also, several later exercises rely implicitly on identities like $\#(a, x \bullet y) = \#(a, x) + \#(a, y)$ that are only proved in earlier exercises. It's unfair to assign these problems to students without telling them which earlier results they can use.

Useful Facts About Strings

1. 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. (Why "at most"?)

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

2. Prove the following useful identities for all strings w, x, y, and z directly from the definition of $\bullet$, *without* referring to the length of any string.

   (a) If $x \bullet y = x$, then $y = \varepsilon$.

   (b) If $x \bullet y = y$, then $x = \varepsilon$.

   (c) If $x \bullet z = y \bullet z$, then $x = y$.

   (d) If $x \bullet y = x \bullet z$, then $y = z$.

3. Prove the following useful fact about substrings. An arbitrary string x is a substring of another arbitrary string $w = u \bullet v$ if and only if at least one of the following conditions holds:

   - x is a substring of u.
   - x is a substring of v.
   - $x = yz$ where y is a suffix of u and z is a prefix of v.

4. 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) $|w[i \mathinner{..} j]| = j - i + 1$

   (b) $w[i \mathinner{..} j] \bullet w[j + 1 \mathinner{..} k] = w[i \mathinner{..} k]$

   (c) $w^R[i \mathinner{..} j] = (w[i' \mathinner{..} j'])^R$ where $i' + j = j' + i = |w| + 1$.

Recursive Functions

5. 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.

   (c) Prove the following identity for all alphabets Σ and all strings $w \in \Sigma^*$:

   $$|w| = \sum_{a\in\Sigma} \#(a,w)$$

   [Hint: Don't try to use induction on Σ.]

6. 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 $\#(a, w^R) = \#(a,w)$ for every string w and every symbol a. (See Exercise 5.)

   (c) Prove that $(w \bullet x)^R = x^R \bullet w^R$ for all strings w and x.

   (d) Prove that $(w^R)^R = w$ for every string w. *[Hint: Use part (c).]*

7. 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 $\varepsilon^{374} = \varepsilon$.

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

   (b) Prove that $\#(a, w^n) = n \cdot \#(a,w)$ for every string w, every symbol a, and every non-negative integer n. (See Exercise 5.)

   (c) Prove that $(w^R)^n = (w^n)^R$ for every string w and every non-negative integer n.

   (d) Prove that for all strings x and y that if $x \bullet y = y \bullet x$, then $x = w^m$ and $y = w^n$ for some string w and some non-negative integers m and n. *[Hint: Careful with ε!]*

8. The ***complement*** $\boldsymbol{w^c}$ of a string $w \in \{\texttt{0},\texttt{1}\}^*$ is obtained from w by replacing every $\texttt{0}$ in w with a $\texttt{1}$ and vice versa. The complement function can be defined recursively as follows:

   $$w^c := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ \texttt{1} \cdot x^c & \text{if } w = \texttt{0}x \\ \texttt{0} \cdot x^c & \text{if } w = \texttt{1}x \end{cases}$$

(a) Prove that $|w| = |w^c|$ for every string w.

(b) Prove that $(x \bullet y)^c = x^c \bullet y^c$ for all strings x and y.

(c) Prove that $\#(\texttt{1}, w) = \#(\texttt{0}, w^c)$ for every string w.

(d) Prove that $(w^R)^c = (w^c)^R$ for every string w.

(e) Prove that $(w^n)^c = (w^c)^n$ for every string w and every non-negative integer n.

9. For any string $w \in \{\texttt{0}, \texttt{1}, \texttt{2}\}^*$, let $\boldsymbol{w^+}$ denote the string obtained from w by replacing each symbol a in w by the symbol corresponding to $(a+1) \bmod 3$. for example, $\texttt{0102101}^+ = \texttt{1210212}$. This function can be defined more formally as follows:

$$w^+ := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ \texttt{1} \cdot x^+ & \text{if } w = \texttt{0}x \\ \texttt{2} \cdot x^+ & \text{if } w = \texttt{1}x \\ \texttt{0} \cdot x^+ & \text{if } w = \texttt{2}x \end{cases}$$

(a) Prove that $|w| = |w^+|$ for every string $w \in \{\texttt{0}, \texttt{1}, \texttt{2}\}^*$.

(b) Prove that $(x \bullet y)^+ = x^+ \bullet y^+$ for all strings $x, y \in \{\texttt{0}, \texttt{1}, \texttt{2}\}^*$.

(c) Prove that $\#(\texttt{1}, w^+) = \#(\texttt{0}, w)$ for every string $w \in \{\texttt{0}, \texttt{1}, \texttt{2}\}^*$.

(d) Prove that $(w^+)^R = (w^R)^+$ for every string $w \in \{\texttt{0}, \texttt{1}, \texttt{2}\}^*$.

10. For any string $w \in \{\texttt{0}, \texttt{1}\}^*$, let $\mathit{swap}(w)$ denote the string obtained from w by swapping the first and second symbols, the third and fourth symbols, and so on. For example:

$$\mathit{swap}(\texttt{101}) = \texttt{011}$$
$$\mathit{swap}(\texttt{100111}) = \texttt{011011}$$
$$\mathit{swap}(\texttt{10 11 00 01 10 1}) = \texttt{01 11 00 10 01 1}.$$

The *swap* function can be formally defined as follows:

$$\mathit{swap}(w) := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ w & \text{if } w = \texttt{0} \text{ or } w = \texttt{1} \\ ba \bullet \mathit{swap}(x) & \text{if } w = abx \text{ for some } a, b \in \{\texttt{0}, \texttt{1}\} \text{ and } x \in \{\texttt{0}, \texttt{1}\}^* \end{cases}$$

(a) Prove that $|\mathit{swap}(w)| = |w|$ for every string w.

(b) Prove that $\mathit{swap}(\mathit{swap}(w)) = w$ for every string w.

(c) Prove that $\mathit{swap}(w^R) = (\mathit{swap}(w))^R$ for every string w such that $|w|$ is even. *[Hint: Your proof must invoke* ***four*** *different recursive definitions: reversal w^R, concatenation $\bullet$, length $|w|$, and the swap function!]*

11. For any string $w \in \{\texttt{0}, \texttt{1}\}^*$, let $\mathit{sort}(w)$ denote the string obtained by sorting the characters in w. For example, $\mathit{sort}(\texttt{010101}) = \texttt{000111}$. The sort function can be defined recursively as follows:

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

(a) Prove that $\#(\mathtt{0}, sort(w)) = \#(\mathtt{0}, w)$ for every string $w \in \{\mathtt{0}, \mathtt{1}\}^*$.

(b) Prove that $sort(w \bullet 1) = sort(w) \bullet 1$ for every string $w \in \{\mathtt{0}, \mathtt{1}\}^*$.

(c) Prove that $\#(\mathtt{1}, sort(w)) = \#(\mathtt{1}, w)$ for every string $w \in \{\mathtt{0}, \mathtt{1}\}^*$.

(d) Prove that $|w| = |sort(w)|$, for every string $w \in \{\mathtt{0}, \mathtt{1}\}^*$.

(e) Prove that $sort(w) \neq x \bullet \mathtt{10} \bullet y$, for all strings $w, x, y \in \{\mathtt{0}, \mathtt{1}\}^*$.

(f) Prove that $sort(w) \in \mathtt{0}^*\mathtt{1}^*$ for every string $w \in \{\mathtt{0}, \mathtt{1}\}^*$.

(g) Prove that $sort(w) = \mathtt{0}^{\#(\mathtt{0},w)}\mathtt{1}^{\#(\mathtt{1},w)}$, for every string $w \in \{\mathtt{0}, \mathtt{1}\}^*$. (In other words, prove that our recursive definition is correct.)

(h) Prove that $sort(sort(w)) = sort(w)$, for all strings $w \in \{\mathtt{0}, \mathtt{1}\}^*$.

(i) Prove that $sort(w^R) = sort(w)$, for every string $w \in \{\mathtt{0}, \mathtt{1}\}^*$.

12. Consider the following recursively defined function:

$$merge(x, y) := \begin{cases} y & \text{if } x = \varepsilon \\ x & \text{if } y = \varepsilon \\ \mathtt{0} \cdot merge(w, y) & \text{if } x = \mathtt{0}w \\ \mathtt{0} \cdot merge(x, z) & \text{if } y = \mathtt{0}z \\ \mathtt{1} \cdot merge(w, y) & \text{if } x = \mathtt{1}w \text{ and } y = \mathtt{1}z \end{cases}$$

For example:

$$merge(\mathtt{10}, \mathtt{10}) = \mathtt{1010}$$
$$merge(\mathtt{10}, \mathtt{010}) = \mathtt{01010}$$
$$merge(\mathtt{010}, \mathtt{0001100}) = \mathtt{0000101100}$$

(a) Prove that $merge(x, y) \in \mathtt{0}^*\mathtt{1}^*$ for all strings $x, y \in \mathtt{0}^*\mathtt{1}^*$. (The regular expression $\mathtt{0}^*\mathtt{1}^*$ is shorthand for the language $\{\mathtt{0}^a\mathtt{1}^b \mid a, b \geq 0\}$.)

(b) Prove that $sort(x \bullet y) = merge(sort(x), sort(y))$ for all strings $x, y \in \{\mathtt{0}, \mathtt{1}\}^*$. (The *sort* function is defined in the previous exercise.)

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

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

For example, $evens(\mathtt{MISSISSIPPI}) = \mathtt{ISSIP}$ and $odds(\mathtt{MISSISSIPPI}) = \mathtt{MSISPI}$.

(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 $odds(w \bullet x)$.

(c) Prove that every string w is a shuffle of $evens(w)$ and $odds(w)$.

14. Consider the following recursively defined function:

$$stutter(w) := \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ aa \bullet stutter(x) & \text{if } w = ax \end{cases}$$

For example, $stutter(\texttt{MISSISSIPPI}) = \texttt{MMIISSSSIISSSSIIPPPPII}$.

(a) Prove that $|stutter(w)| = 2|w|$ for every string w.

(b) Prove that $evens(stutter(w)) = w$ for every string w.

(c) Prove that $odds(stutter(w)) = w$ for every string w.

(d) Prove that $stutter(w)$ is a shuffle of w and w, for every string w.

(e) Prove that w is a palindrome if and only if $stutter(w)$ is a palindrome, for every string w.

15. Consider the following recursive function:

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

For example, $faro(\texttt{0011}, \texttt{0101}) = \texttt{00011011}$. (A "faro shuffle" splits a deck of cards into two equal piles and then perfectly interleaves them.)

(a) Prove that $|faro(x, y)| = |x| + |y|$ for all strings x and y.

(b) Prove that $faro(w, w) = stutter(w)$ for every string w.

(c) Prove that $faro(odds(w), evens(w)) = w$ for every string w.

16. For any string w, let $declutter(w)$ denote the string obtained from w by deleting any symbol that equals its immediate successor. For example, $declutter(\texttt{MISSISSIPPI}) = \texttt{MISISIPI}$, and $declutter(\texttt{ABBCCCAAAACCCBBA}) = \texttt{ABCACBA}$.

(a) Given a recursive definition for the function $declutter$.

(b) Using your recursive definition, prove that $declutter(stutter(w)) = declutter(w)$ for every string w.

(c) Using your recursive definition, prove that $declutter(w^R) = (declutter(w))^R$ for every string w.

(d) Using your recursive definition, prove that w is a palindrome if and only if $declutter(w)$ is a palindrome, for every string w.

17. Consider the following recursively defined function

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

Prove that $|hanoi(w)| = 2^{|w|} - 1$ for every string w.

18. Consider the following recursively defined function

$$slog(w) = \begin{cases} \varepsilon & \text{if } w = \varepsilon \\ a \cdot slog(evens(w)) & \text{if } w = ax \end{cases}$$

Prove that $|slog(w)| = \lceil \log_2(|w|+1) \rceil$ for every string w.

19. Consider the following recursively defined function

$$bitrev(w) = \begin{cases} w & \text{if } |w| \leq 1 \\ bitrev(odds(w)) \bullet bitrev(evens(w)) & \text{otherwise} \end{cases}$$

(a) Prove that $|bitrev(w)| = |w|$ for every string w.

*(b) Prove that $bitrev(bitrev(w)) = w$ for every string w such that $|w|$ is a power of 2.

20. The ***binary value*** of any string $w \in \{0, 1\}^*$ is the integer whose binary representation (possibly with leading 0s) is w. The value function can be defined recursively as follows:

$$value(w) := \begin{cases} 0 & \text{if } w = \varepsilon \\ 2 \cdot value(x) & \text{if } w = x \bullet 0 \\ 2 \cdot value(x) + 1 & \text{if } w = x \bullet 1 \end{cases}$$

(a) Prove that $value(w) + value(w^c) = 2^{|w|} - 1$ for every string $w \in \{0, 1\}^*$.

(b) Prove that $value(x \bullet y) = value(x) \cdot 2^{|y|} + value(y)$ for all strings $x, y \in \{0, 1\}^*$.

*(c) Prove that $value(x)$ is divisible by 3 if and only if $value(x^R)$ is divisible by 3.

Recursively Defined Sets

20. Consider the set of strings $L \subseteq \{0, 1\}^*$ defined recursively as follows:

- The empty string ε is in L.
- For any string x in L, the string $0x$ is also in L.
- For any strings x and y in L, the string $1x1y$ is also in L.
- These are the only strings in L.

(a) Prove that the string 101110101101011 is in L.

(b) Prove that every string $w \in L$ contains an even number of 1s. (You may assume the identity $\#(a, xy) = \#(a, x) + \#(a, y)$ for any symbol a and any strings x and y; see Exercise 5(b).)

(c) Prove that every string $w \in \{0, 1\}^*$ with an even number of 1s is a member of L.

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

- The empty string ε is in L.

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

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

(b) Prove by induction that every string in L has exactly twice as many $\mathtt{0}$s as $\mathtt{1}$s. (You may assume the identity $\#(a, xy) = \#(a, x) + \#(a, y)$ for any symbol a and any strings x and y; see Exercise 5(b).)

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

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

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

(a) Prove that the string $\mathtt{01000110111001}$ is in L.

(b) Prove by induction that every string in L has exactly the same number of $\mathtt{0}$s and $\mathtt{1}$s. (You may assume the identity $\#(a, xy) = \#(a, x) + \#(a, y)$ for any symbol a and any strings x and y; see Exercise 5(b).)

(c) Prove by induction that L contains every string with the same number of $\mathtt{0}$s and $\mathtt{1}$s.

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

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

(a) Prove that the string $\mathtt{01000110111001}$ is in L.

(b) Prove by induction that every string in L has exactly the same number of $\mathtt{0}$s and $\mathtt{1}$s. (You may assume the identity $\#(a, xy) = \#(a, x) + \#(a, y)$ for any symbol a and any strings x and y; see Exercise 5(b).)

(c) Prove by induction that L contains every string with the same number of $\mathtt{0}$s and $\mathtt{1}$s.

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

- The empty string ε is in L.
- For any string x in L, the string $\mathtt{0}x$ is also in L.
- For any strings x and y in L, the strings $\mathtt{1}x\mathtt{2}y$ and $\mathtt{2}x\mathtt{1}y$ are also in L.
- These are the only strings in L.

(a) Prove that the string $\mathtt{001201110220121220}$ is in L.

(b) For any string $w \in \{\mathtt{0},\mathtt{1},\mathtt{2}\}^*$, let ***ModThree(w)*** denote the sum of the digits of w modulo 3. This function can be defined recursively as follows:

$$\mathit{ModThree}(w) = \begin{cases} 0 & \text{if } w = \varepsilon \\ \mathit{ModThree}(x) & \text{if } w = \mathtt{0}x \\ (\mathit{ModThree}(x) + 1) \bmod 3 & \text{if } w = \mathtt{1}x \\ (\mathit{ModThree}(x) + 2) \bmod 3 & \text{if } w = \mathtt{2}x \end{cases}$$

For example, $\mathit{ModThree}(\mathtt{2211}) = 0$ and $\mathit{ModThree}(\mathtt{001201110220121220}) = 0$. Prove that $\mathit{ModThree}(w) = 0$ for every string in $w \in L$.

(c) Find a string $w \in \{\mathtt{0},\mathtt{1},\mathtt{2}\}^*$ such that $\mathit{ModThree}(w) = 0$ but $w \notin L$. Prove that your answer is correct.

(d) Prove that $\#(\mathtt{1}, w) = \#(\mathtt{2}, w)$ for every string $w \in L$.

(e) Find a string $w \in \{\mathtt{0},\mathtt{1},\mathtt{2}\}^*$ such that $\#(\mathtt{1}, w) = \#(\mathtt{2}, w)$ but $w \notin L$. Prove that your answer is correct.

(f) Prove that $L = \big\{ w \in \{\mathtt{0},\mathtt{1},\mathtt{2}\}^* \;\big|\; \mathit{ModThree}(w) = 0 \text{ and } \#(\mathtt{1}, w) = \#(\mathtt{2}, w) \big\}$.

25. 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 $p = p^R$.

(c) Using your recursive definition, prove that the strings $w \bullet w^R$ and $w \bullet a \bullet w^R$ are palindromes, for every string w and symbol a.

(d) Using your recursive definition, prove that p^n is a palindrome for every palindrome p and every natural number n. (See Exercise 7.)

(e) Using your recursive definition, prove that for every palindrome p, there is at most one symbol a such that $\#(a, p)$ is odd. (See Exercise 5.)

26. 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 w is a shuffle of x and y if and only if w^R is a shuffle of x^R and y^R.

27. For any positive integer n, the ***Fibonacci string*** $\boldsymbol{F_n}$ 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 the Fibonacci string F_n begins with 1 if and only if n is even.

(c) Prove that 00 is not a substring of any Fibonacci string F_n.

(d) Prove that 111 is not a substring of any Fibonacci string F_n.

*(e) Prove that 01010 is not a substring of any Fibonacci string F_n.

*(f) Find another string w that is not a substring of any Fibonacci string F_n, such that 00 and 111 and 01010 are not substrings of w.

★(g) Find a set of strings F with the following properties:

- No string in F is a substring of any Fibonacci string F_n.
- No string in F is a proper substring of any other string in F.
- For all strings $x \in \{\texttt{0},\texttt{1}\}^*$, if x has no substrings in F, then x is a substring of some Fibonacci string F_n.

★(h) Prove that the reversal of each Fibonacci string is a substring of another Fibonacci string. More formally, prove that for every integer $n \geq 0$, the string F_n^R is a substring of F_m for some integer $m \geq n$.

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

- A string $w \in \{\texttt{0},\texttt{1}\}^*$ is ***balanced*** if it satisfies one of the following conditions:
  - $w = \varepsilon$,
  - $w = \texttt{0}x\texttt{1}$ for some balanced string x, or
  - $w = xy$ for some balanced strings x and y.
- A string $w \in \{\texttt{0},\texttt{1}\}^*$ is ***erasable*** if it satisfies one of the following conditions:
  - $w = \varepsilon$, or
  - $w = x\texttt{01}y$ for some strings x and y such that xy is erasable. (The strings x and y are not necessarily erasable.)
- A string $w \in \{\texttt{0},\texttt{1}\}^*$ is ***conservative*** if it satisfies **both** of the following conditions:

  - w has an equal number of 0s and 1s, and
  - no prefix of w has more 0s than 1s.

(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.]*

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

(a) Prove that a string w is equitable if and only if it satisfies one of the following conditions:

- $w = \varepsilon$,
- $w = 0x1$ for some equitable string x,
- $w = 1x0$ 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:

- $w = \varepsilon$,
- $w = x01y$ for some strings x and y such that xy is equitable, or
- $w = x10y$ 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".)