Sequences

A *sequence* is a function the domain of which is either the set of all natural numbers (an *infinite* sequence), or some initial segment thereof (a *finite* sequence), where by an initial segment we mean any set of the form $\{0, 1, \ldots, n\}$.

For example, the function f that maps each natural number n to a rational number, $f(n) = (-1)^n/(n+1)$, defines an infinite sequence

$$1, -\frac{1}{2}, \frac{1}{3}, -\frac{1}{4}, \frac{1}{5}, \ldots$$

The finite sequence

$$2, 3, 5, 7, 11, 13$$

is formally a function $f : \{0, 1, 2, 3, 4, 5\} \to \mathbf{N}$ such that

$$\begin{aligned} f(0) &= 2 \\ f(1) &= 3 \\ f(2) &= 5 \\ f(3) &= 7 \\ f(4) &= 11 \\ f(5) &= 13 \end{aligned}$$

Notations for Sequences

There is no standard notation for sequences, though finite sequences are often written by listing the elements in order, sometimes enclosed within parentheses

$$(s_1, s_2, \ldots, s_n)$$

or angle brackets

$$\langle s_1, s_2, \ldots, s_n \rangle.$$

Curly brackets are used to denote sets, never sequences.

In the case of infinite sequences one often lists the first few terms to indicate a pattern, though such descriptions are evidently ambiguous.

For example,

$$1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$$

refers to the *Fibonacci sequence*. The n-th term in this sequence, denoted by s_n or $s[n]$, is obtained by adding the two preceding values.

Sequences are similar to tuples, but different in terms of their set-theoretic definition.

Sequences of Sets

The domain of a sequence must be the set of natural numbers or a subset thereof. But the codomain can be any set; for instance, a set of sets.

For example, we may define a function D on the natural numbers by

$$D(n) = \{m \in \mathbf{Z} : m \text{ is a multiple of } n\}.$$

What is the codomain of this function?

The powerset of **Z**.

Generalized union and intersection operations can be applied to sequences with such codomains.

Generalized Union and Intersection

If s is a sequence of sets we define two operations as follows:

$$\bigcup_k s = \{x : x \in s_k \text{ for some } k \in \mathbf{N}\}$$

and

$$\bigcap_k s = \{x : x \in s_k \text{ for all } k \in \mathbf{N}\}.$$

For example, if D is the above sequence then $\bigcup_k D = \mathbf{Z}$ and $\bigcap_k D = \{0\}$.

More generally, if A is a set, we define the *union over* A by

$$\bigcup A = \{x : x \in a \text{ for some } a \in A\}$$

and the *intersection over* A by

$$\bigcap A = \{x : x \in a \text{ for all } a \in A\}.$$

Let S be any set.

What set is $\bigcup \mathcal{P}(S)$?

The set S.

What set is $\bigcap \mathcal{P}(S)$?

The empty set.

Strings

A *string* is a finite sequence of elements from some given set.

For instance, in formal language theory one studies sets of strings over a given finite set Σ, usually called an *alphabet*.

The alphabet may be as simple as a two-element set $\{0, 1\}$ or a slightly larger set containing all lower and upper-case letters and some special characters. The elements of a given alphabet are usually called *symbols*.

Strings are usually represented by juxtaposition of symbols, e.g., $penguin$ or 100100.

Formally, a string of length n over Σ is a function, the domain of which is the set $\{0, 1, \ldots, n-1\}$ and the codomain the set Σ.

Thus $penguin$, as a string, is a function on $\{0, 1, 2, 3, 4, 5, 6\}$ with

$$\begin{aligned} penguin(0) &= p \\ penguin(1) &= e \\ penguin(2) &= n \\ penguin(3) &= g \\ penguin(4) &= u \\ penguin(5) &= i \\ penguin(6) &= n \end{aligned}$$

Basic String Operations

The *length* of a string is formally defined to be the size of its function domain. (You recall that the natural number n may be defined as the set $\{0, \ldots, n-1\}$, so that the length of a string actually *is* its domain.)

If w is a string, we denote its length by $|w|$. For example $|rat| = 3$ and $|catcher| = 7$.

A string of length zero is called an *empty string*. Which function does an empty string correspond to?

The same symbol may occur more than once in string, such as c in $catcher$. We then speak of different *occurrences* of a symbol.

A common operation is that of *concatenation* of two strings v and w, usually denoted by juxtaposition vw or sometimes by $v\hat{}w$. Formally, the string $x = vw$ is defined by: $x(i) = v(i)$ if $0 \le i < |v|$, and $x(i) = w(i - |v|)$ if $|v| \le i < |v| + |w|$.

For example, if $v = rat$ and $w = catcher$, then $vw = ratcatcher$. We have $vw(2) = t = v(2)$ and $vw(4) = a = w(1)$.

Concatenation is an *associative* operation: $(uv)w = u(vw)$, for all strings u, v, and w.

Substrings

We say that v is a *substring* of w if there exist strings x and y such that $w = xvy$.

For example, $road$ and $runner$ are substrings of $roadrunner$.

How many substrings does a string of length n have?

> *At most* $\frac{1}{2}n(n+1) + 1$. (Proof on next slide.)

For example, cat has seven substrings, but $dodo$ has fewer than eleven.

If $w = xv$ for some string x, then v is called a *suffix* of w.

Similarly, if $w = vx$ for some string x, then v is called a *prefix* of w.

Thus $road$ is a prefix of $roadrunner$, whereas $runner$ is a suffix.

Lemma.

> *A string of length n has at most $\frac{1}{2}n(n+1) + 1$ substrings.*

Proof. We use induction to prove that the following property P is true for all natural numbers n.

> *$P(n)$: If w is a string of length n then it has no more than $(\sum_{i=1}^{n} i) + 1$ substrings.*

Induction basis. If $n = 0$ then w is the empty string and has one substring. The assertion is true because the empty string haas only one substring (namely, itself) and $\sum_{i=1}^{0} i = 0$.

Induction step. We have to prove that $P(k)$ implies $P(k+1)$ for all natural numbers k.

Induction hypothesis. Let k be an arbitrary natural number and suppose $P(k)$ is true.

We need to show that $P(k+1)$ is true. For that purpose let w be a string of length $k+1$. Since w is a nonempty string it can be written as aw', where $a \in \Sigma$ and w' is a string of length k.

By the induction hypothesis, w' has no more than $\sum_{i=1}^{k} i + 1$ substrings. Also, if v is a substring of w, but not of

w', then it must be a *prefix* of w. Since w is of length $k+1$ it has $k+1$ different prefixes.

Thus the total number of substrings of w is at most

$$(k+1)+(\sum_{i=1}^{k} i+1)=\sum_{i=1}^{k+1} i+1.$$

This completes the induction proof. The lemma follows from the well-known fact that $\sum_{i=1}^{n} i = \frac{1}{2}n(n+1)$.

Inductive Definition of Strings

Another way of defining strings is by induction using the basic operation of adding a symbol at the beginning of a string.

Let Σ be a finite set. The set Σ^* is defined by the following rules:

1. The empty string, denoted by Λ, is an element of Σ^*.

2. If w is an element of Σ^* and a is an element of Σ, then $a \cdot w$ is an element of Σ^*.

3. The expressions obtained by the above rules are the only elements of Σ^*.

The operation $\cdot : \Sigma \times \Sigma^* \to \Sigma^*$ maps a pair (a, w) to the string beginning with the symbol a and followed by w.

For example, if $\Sigma = \{a, b, c, \dots, x, y, z\}$ then

$$v \cdot i \cdot o \cdot l \cdot a \cdot \Lambda$$

produces the string $viola$.

Equality of Strings

The view of strings underlying the inductive definition is that the elements of Σ^* are *constructed* by the binary operator $\cdot$ and the constant Λ.

One can take this approach a step further and give an abstract definition of strings, where Λ and $a \cdot w$ are not explicitly defined, and hence not tied to any specific representation. For such an "abstract data type specification" it is essential to specify suitable requirements, including an equality relation as well as so-called "destructor" operations that allow one to decompose strings.

For example, equality can be defined recursively as follows: If v and w are elements of Σ^*, then

$$v \approx w \text{ if and only if}$$

1. v and w are both identical to Λ, or else

2. v and w are both different from Λ and can be written as $a \cdot v'$ and $b \cdot w'$, respectively, where $a = b$ and $v' \approx w'$.

This definition of coincides with the equality of strings as finite sequences.

Recursive Definitions of String Operations

Inductive definitions provide a natural basis for recursive definitions of functions defined on these domains.

For example, the length function can be defined by:

1. $|\Lambda| = 0$

2. $|a \cdot w| = |w| + 1$

whereas concatenation of strings is defined by:

1. If v is the string Λ and w is any string, then $vw = w$.

2. If v is a string $a \cdot v'$ and w is a string, then $vw = a \cdot w'$ where w' is $v'w$.

Further Recursive Definitions

The substring relation can be defined as follows. For all strings v and w,

1. if v is Λ, then v is a substring of w;
2. if v is a prefix of w, then v is a substring of w;
3. if w is of the form $a \cdot w'$ and v is a substring of w', then v is also a substring of w;

and in no other cases is v a substring of w.

This definition depends on the prefix relation, which can be defined as follows.

A string v is a prefix of a string w if and only if

1. v is Λ or else
2. v and w are of the form $a \cdot v'$ and $a \cdot w'$, respectively, where v' is a prefix of w'.

Inductive Definitions of Sets

Sets in general can often be defined in a constructive way via induction. An inductive definition of a set S consists of three parts:

Basis

Define one or more objects to be elements of S.

Induction

Give *rules* to construct new elements of S from existing elements of S.

Closure

Limit the elements of S to those that can be obtained by the two preceding steps.

For example, the set $\mathbf{O}$ of odd natural numbers can be defined by specifying that

(i) $1 \in \mathbf{O}$ and
(ii) if $x \in \mathbf{O}$ then $x + 2 \in \mathbf{O}$.

Note that usually only the basis and induction parts are explicitly stated, whereas the closure condition is assumed implicitly.

Inductive Sets

A set is called *inductive* if it can be specified by an inductive definition.

An example of an inductive set is the set $A = \{2^n - 1 : n \in \mathbf{N}\}$, which can be inductively specified by:

(i) $0 \in A$ and
(ii) if $x \in A$ then $2x + 1 \in A$.

The following set,

$$B = \{2, 3, 4, 7, 8, 11, 15, 16, ...\},$$

is also inductive.

Let us first give a formal (non-inductive) definition of this set:

$$B = \{x \in \mathbf{N} : \text{either } x = 2^k, \text{ for some } k \in \mathbf{N} \text{ with } k \geq 1, \text{ or else } x \bmod 4 = 3\}.$$

A possible inductive definition is:

(i) $2 \in B$ and $3 \in B$, and
(ii) if $x \in B$ and x is even, then $2x \in B$;
 if $x \in B$ and x is odd, then $x + 4 \in B$.

We shall see later on that not all sets are inductive.

Examples - Sets of Strings

We have already given an inductive definition of the set Σ^* of *all* strings built with symbols from the alphabet Σ.

Suppose $\Sigma = \{1, 0\}$ and S is a subset of Σ^* containing the strings 1, 10, 100, 1000, etc. (i.e., all strings of a 1 followed by any number of 0's). Here is an inductive definition of this set:

(i) $1 \in S$ and
(ii) if $x \in A$ then $x0 \in S$.

The set T of all strings of any number of a's followed by the same number of b's can be inductively defined by:

(i) $\Lambda \in T$ and
(ii) if $x \in T$ then $axb \in T$.

Determine which set (of strings over $\{a, b\}$) is specified by the following definition:

(i) $\Lambda \in U$ and
(ii) if $x \in U$ then $abx \in U$.

Inductive Definition of Lists

Let A be a set. The set $lists_A$ of lists over A can defined inductively in a similar way as strings:

Basis

$\langle\,\rangle \in lists_A$.

Induction

if $x \in A$ and $L \in lists_A$, then $x :: L \in lists_A$.

Keep in mind that we implicitly assume closure: only objects obtained by the above two steps are elements of $lists_A$.

The element $\langle\,\rangle$ is called the *empty list*, and also denoted by nil. We sometimes write $cons(x, L)$ instead of $x :: L$ and, in general, simplify the notation by writing

$$\langle a_1, a_2, \ldots, a_{n-1}, a_n \rangle$$

instead of

$$a_1 :: (a_2 :: (\cdots (a_{n-1} :: (a_n :: nil) \cdots).$$

Basic List Operations

We may view $\langle\,\rangle$ and $cons$ as operations that *construct* lists.

Other operations may be viewed as "decomposing" given lists, and are called *destructors*. These include, for lists, the *head* and *tail* functions, which are defined as follows:

If L is a non-empty list $x :: L'$, then $head(L) = x$ and $tail(L) = L'$.

Note that the two functions are not defined for the empty list, and hence are partial functions on the domain $lists_A$.

The above definition implies that the following identity is valid for all non-empty lists L:

$$L = head(L) :: tail(L).$$

Exercises.

Define a function that returns the length of a given list.

Give a recursive definition of the equality relation for lists over A.

Examples - Lists

We next give an inductive definition of the set S of all lists over A that are of even length:

(i) $nil \in S$ and

(ii) if $a \in A$, $b \in A$, and $L \in S$, then $a :: (b :: L) \in S$.

Characterize which elements comprise the following set T, inductively defined by:

(i) if $a \in A$ and $L \in lists_A$ then $(a, a :: L) \in T$, and

(ii) if $a \in A$ and $(b, L) \in T$, then $(b, a :: L) \in T$.

Finally, we inductively define the set U of all lists over $\{a, b\}$ with alternating occurrences of a's and b's:

(i) the lists nil, $\langle a \rangle$, and $\langle b \rangle$ are elements of U;

(ii) if $a :: L \in U$ then $b :: (a :: L) \in U$; and

(iii) if $b :: L \in U$ then $a :: (b :: L) \in U$.