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., $viola$ or 100100.

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

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

$$\begin{aligned} viola(0) &= v \\ viola(1) &= i \\ viola(2) &= o \\ viola(3) &= l \\ viola(4) &= a \end{aligned}$$

Basic String Operations

The *length* of a string is formally defined to be the size of its function domain. In fact, if you recall that the natural number n may be defined as the set $\{0, \ldots, n-1\}$, then the length of a string *is* its domain.

If w is a string, we denote its length by $|w|$. For example $|viola| = 5$ and $|violin| = 6$.

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 i in $violin$. Therefore one sometimes has to distinguish between different *occurrences* of a symbol, e.g., the occurrence of i in the second and fifth position, respectively.

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

For example, if $v = house$ and $w = cat$, then $vw = housecat$ and $vw(2) = u = v(2)$ and $vw(6) = a = w(1)$.

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 $\sum_{i=1}^{0} i = 0$.

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

Induction hypothesis. Let k be an arbitary 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 $\sum_{i=1}^{n} i = \frac{1}{2}n(n+1)$.

Formal Languages

The set of all strings over an alphabet Σ is denoted by Σ^*. In other words, the elements of Σ^* are all the strings built from symbols in Σ.

Any subset of Σ^* is called a *(formal) language* over Σ.

For instance, $\emptyset$ and Σ^* are languages in this sense. Programming languages are more interesting examples of formal languages and the specification of formal languages is an important issue for computer science.

One possible way of specifying languages is via comprehension:

$$L = \{w \in \Sigma^* \ : \ P(w)\}.$$

For example, we may define

$L_1 = \{w \in \{0,1\}^* \ : \ w$ has an equal number of 0's and 1's$\}$

or

$$L_2 = \{w \in \{0,1\}^* \ : \ w \text{ begins with a } 1\}.$$

But for computational purposes comprehension is too powerful a formalism in its full generality. For instance, the problem of determining whether a string is an element of a specified language can not be solved algorithmically for arbitrary languages specified in this way.

Set Operations for Formal Languages

If L is a language over Σ, then its *complement* is defined to be the language $\Sigma^* \setminus L$.

If L_1 and L_2 are languages over the same alphabet Σ, their concatenation, denoted by $L_1 \circ L_2$ or simply $L_1 L_2$, is the language

$$\{w \in \Sigma^* \ : \ w = xy \text{ for some } x \in L_1 \text{ and some } y \in L_2\}.$$

For example, let L_1 be the set of all strings over $\{0,1\}$ that begin with a 1 and L_2 be the set of all strings over $\{0,1\}$ that end with a 0. What is $L_1 L_2$?

> *The set of all strings over* $\{0,1\}$ *that begin with a* 1 and *end with a* 0.

Another set operation on formal languages is the *closure* or *Kleene star* of a language L, denoted by L^* and defined by:

$$\begin{aligned} L^* \ = \ & \{w \in \Sigma^* \ : \ w = w_1 w_2 \cdots w_k \\ & \text{for some } k \geq 0 \text{ and some strings } w_1, \ldots, w_k \text{ in } L\}. \end{aligned}$$

For example, if $L_3 = \{1, 10, 100\}$ then $1010 \in L_3^*$. Which languages are L_1^* and L_2^*?

> $L_1^* = L_1 \cup \{e\}$ *and* $L_2^* = L_2 \cup \{e\}$, *where* e *denotes the empty string.*

We also write L^+ to denote the language LL^*.