1.3 Ordered Structures, Tuples, Lists, Strings and Languages

Sets are useful for unordered, possibly infinite collections of elements.

A **tuple** is a finite, ordered collection of elements (aka *members*, *components*).

We’ll denote a tuple by writing its elements, in order, separated by commas, beginning with “(" and ending with ")".

For example, the tuple

\[(12, R, −9)\]

has three elements: the first is 12, the second \(R\) and the third \(−9\).

If a tuple has \(n\) elements, we say it has length \(n\), and call it an \(n\)-tuple.

There is a unique 0-tuple, which can be written \((\ )\), called the *empty* tuple.

Sometimes tuples are called “vectors” or (finite) “sequences”.

2-tuples are usually called *ordered pairs*.
Tuple equality, Cartesian product

Two \( n \)-tuples \((x_1, \ldots, x_n)\) and \((y_1, \ldots, y_n)\) are equal if

\[ x_i = y_i \quad \text{for all } i \ (1 \leq i \leq n). \]

For sets \( A, B \), the \textit{Cartesian product} of \( A \) and \( B \), written \( A \times B \), is defined as follows.

\[ A \times B = \{ (x, y) \mid x \in A \text{ and } y \in B \} \]

For example, if \( A = \{0, 1\} \) and \( B = \{1, 2\} \), then

\[ A \times B = \]
\[ \emptyset \times \mathcal{N} = \]

We extend the Cartesian product as follows. For sets \( A_1, \ldots, A_n \),

\[ A_1 \times \cdots \times A_n = \{ (x_1, \ldots, x_n) \mid \text{for all } i \ (1 \leq i \leq n), \ x_i \in A_i \}. \]

If all sets \( A_i \) are the same set \( A \), we also write \( A^n \) for their Cartesian product.

\[ A^1 = \]
\[ A^0 = \]
Each element of $A^n$ is an $n$-tuple, which is essentially an array of length $n$.

And we often use array-like notations to give “direct access” to the components of an $n$-tuple $x$:

$$(x_1, x_2, \ldots, x_n)$$

or

$$(x(1), x(2), \ldots, x(n))$$

or

$$(x[1], x[2], \ldots, x[n]) .$$

And when we “implement” $n$-tuples, we typically implement them as arrays.

Each element of $(A^m)^n$ is an $n$-tuple, each of whose members is an $m$-tuple.

We can think of an element of $(A^m)^n$ as a two-dimensional array, with $n$ rows and $m$ columns.

And similar remarks apply about notation and implementation.

Along the same lines, elements of $A \times B$ are analogous to records or structures with two fields.
Counting tuples

If $A$ and $B$ are finite sets, with $|A| = m$ and $|B| = n$, then

what is $|A \times B|$?

If $A$ is a finite set, how can we express $|A^n|$ in terms of $|A|$ and $n$?

For finite set $A$, what is the relationship between

$|\text{power}(A)|$

and

$|\{0, 1\}^{|A|}|$?
Relations

If $R$ is a subset of $A_1 \times \cdots \times A_n$, then $R$ is said to be an $n$-ary relation on (or over) $A_1 \times \cdots \times A_n$.

If $R$ is an $n$-ary relation on $A^n$, we also say, more simply, that $R$ is an $n$-ary relation on $A$.

Instead of 1-ary we typically say unary, instead of 2-ary binary and instead of 3-ary ternary.

Notice: For any set $A$, $A^n$ is the largest $n$-ary relation on $A$.

What is the smallest $n$-ary relation on $A$?

If $|A| = k$, how many binary relations on $A$?

How many $n$-ary relations on $A$?
Lists

A list is a finite sequence of elements.

So a list is essentially a tuple, except that with lists we do not typically assume that we are working with a tuple of a particular given length, and so we take a different view of how to access the elements of a list.

Instead, we access list elements by taking either the “head” of the list (its first element) or the “tail” of the list.

Accordingly, we use slightly different notation for lists: “⟨” and “⟩”, instead of “(“ and “)”.  

The empty list is written ⟨ ⟩.

The length of a list  

\[ L = \langle x_1, x_2, \ldots, x_n \rangle \]

is \( n \), with  

\[ \text{head}(L) = x_1 \]

and  

\[ \text{tail}(L) = \langle x_2, \ldots, x_n \rangle. \]

Notice that head(⟨ ⟩) and tail(⟨ ⟩) are undefined.
Constructing lists

We have a convenient notation for constructing a new list by adding a new element $h$ at the head of a list $L$ — we write

$$\text{cons}(h, L).$$

For example, if

$$L = \langle 1, 2 \rangle$$

then

$$\text{cons}(0, L) = \langle 0, 1, 2 \rangle$$

and

$$\text{cons}(1, \text{cons}(0, L)) = \langle 1, 0, 1, 2 \rangle.$$

As you know from prior study of list implementation via linked lists, the operations $\text{cons}$, $\text{head}$ and $\text{tail}$ have efficient computer implementations.

Observe that, for any nonempty list $L$,

$$\text{cons}(\text{head}(L), \text{tail}(L)) = L.$$
Of course lists can have lists as elements:

\[
\begin{align*}
\text{head}(\langle\langle a, b\rangle, \langle \rangle, c, d\rangle) &= \\
\text{tail}(\langle\langle a, b\rangle, \langle \rangle, c, d\rangle) &= 
\end{align*}
\]

We’ll eventually see how to represent trees as lists (whose elements may be lists…).

If all the elements of a list \( L \) belong to a set \( A \), we say \( L \) is a list \textit{over} \( A \).

It is convenient to let

\[
\text{lists}(A)
\]

denote the set of all lists over \( A \).
Counting lists

If $|A| = n$, how many lists over $A$ of length $m$?

How many lists over $A$ of length at most $m$?

How many lists of length $m$ over $A \times A$?
Alphabets and strings

*Strings* are a simpler version of lists, in which all list elements come from a finite set of symbols, called an *alphabet*.

Because of this simpler structure, there is no need to use " ⟨", "⟩" and " ," when representing strings.

For example, if the alphabet is \{0, 1\}, we write

```
010
```

to represent the string of length 3 that corresponds to the list ⟨0, 1, 0⟩.

But this simplified notation for strings imposes a slight cost. We need a notation for the *empty string* — the string of length 0. It is written

```
Λ.
```

Eventually we will work with strings a lot, and so we also introduce a convenient notation for the length of a string \(s\):

```
|s|.
```
Languages, string concatenation

Given an alphabet $A$, a language over $A$ is a set of strings over $A$.

For example, $\emptyset$, $\{\Lambda\}$, $\{a\}$, $\{\Lambda, a, aa\}$ are all languages over the alphabet $\{a\}$.

We write $A^*$ to denote the set of all strings over alphabet $A$.
Notice that $A^*$ itself is a language over $A$.

If $x$ and $y$ are strings, we denote their concatenation by writing

$$xy.$$  

For example, if $x = 01$ and $y = 10$, then

$$xy = 0110, yx = 1001, xx = 0101, yy = 1010, xyx = 01101.$$  

Notice that, for all strings $s$,

$$\Lambda s = s = s\Lambda.$$  

For any string $s$ and $n \in \mathbb{N}$, $s^n$ denotes the concatenation of $s$ with itself $n$ times:

$$s^0 = \Lambda, s^1 = s, s^2 = ss, s^3 = sss, \ldots$$
Here are some examples of the use of exponent notation for string concatenation:

\[
\{ a^n \mid n \in \mathcal{N} \} =
\]

\[
\{ ab^n \mid n \in \mathcal{N} \} =
\]

\[
\{ a^n b^n \mid n \in \mathcal{N} \} =
\]

\[
\{ (ab)^n \mid n \in \mathcal{N} \} =
\]

\[
\{ xx^n \mid n \in \mathcal{N}, x \in \{a, b\}^* \} = \{a, b\}^*?
\]
Products of languages

The *product* of languages $L$ and $M$ is the language

$$LM = \{ xy \mid x \in L, y \in M \}.$$  

For example, if $L = \{0, 1\}$ and $M = \{\Lambda, 0\}$, then

$$LM = \text{ and } ML = \text{.}$$  

Notice: For all languages $L$,

$$L\{\Lambda\} = \{\Lambda\}L = \text{ and } L\emptyset = \emptyset L = \text{.}$$  

We also have exponent notation for language products:

$$L^n = \{ x_1 \cdots x_n \mid \text{for all } i \ (1 \leq i \leq n), \ x_i \in L \}.$$  

The special case when $n = 0$ is given by

$$L^0 = \{\Lambda\}.$$  

What is $L^1$?

Notice that $L^m L^n = L^{m+n}$.  


(Kleene) closure of a language

The closure of a language $L$, written $L^*$, is defined as follows.

$$L^* = \bigcup_{i \in \mathbb{N}} L^i.$$

Consider some examples:

$\{0\}^* =$

$\{00\}^* =$

$\{0, 1\}^* =$

$\{0\}^* \{1\}^* =$

$\{\Lambda\}^* =$

$\emptyset^* =$
Positive closure of a language

The *positive closure* of a language $L$, written $L^+$, is defined as follows.

$$ L^+ = \bigcup_{i \in \mathbb{N}} L^{i+1}. $$

For all languages $L$, $L^+ \cup \{\Lambda\} =$

If $\Lambda \in L$, then $L^+ = L^*$?

$L^* L^* =$

$(L^*)^* =$

$L^+ L^+ =$

$(L^+)^+ =$

$(L^*)^+ =$

$(L^*)^+ =$
Counting strings

How many strings of length $k$ over alphabet $A$?

How many strings of length 5 over \{a, b, c, d\} that end with $a$ or $b$?

How many strings of length 5 over \{a, b, c, d\} that end with $a$ or $b$ and contain at least one $c$?
How many strings of length 5 over \( \{a, b, c, d\} \) contain at least one \( c \) and at least one \( d \)?

We can begin by subtracting from \( \{a, b, c, d\}^5 \) the strings that either lack \( c \) or lack \( d \).

\[
\left| \{a, b, c, d\}^5 \right| - \left( \left| \{a, b, d\}^5 \cup \{a, b, c\}^5 \right| \right) \\
= \left| \{a, b, c, d\}^5 \right| - \left| \{a, b, d\}^5 \cup \{a, b, c\}^5 \right| \\
= 4^5 - \left| \{a, b, d\}^5 \cup \{a, b, c\}^5 \right| \\
(\text{and because } |A \cup B| = |A| + |B| - |A \cap B|) \\
= 4^5 - (\left| \{a, b, d\}^5 \right| + \left| \{a, b, c\}^5 \right| - \left| \{a, b\}^5 \right|) \\
= 4^5 - (3^5 + 3^5 - 2^5) \\
= 4^5 - 2(3^5) + 2^5
\]