1.3

Formal Languages & Regular Expressions

P. Danziger

1 Cartesian Products

Definition 1 *Let $n \in \mathbf{Z}^+$, and let $x_1, x_2, \ldots, x_n$ be n (not necessarily distinct) elements of some set. The ordered n-tuple $(x_1, x_2, \ldots, x_n)$ consists of $x_1, x_2, \ldots, x_n$ together with the ordering.*

- *An ordered 2-tuple (x_1, x_2) is called an ordered pair.*
- *An ordered 3-tuple (x_1, x_2, x_3) is called an ordered triple.*
- *Two ordered n-tuples $(x_1, x_2, \ldots, x_n)$ and $(y_1, y_2, \ldots, y_n)$ are equal if and only if*

$$x_1 = y_1 \wedge x_2 = y_2 \wedge \ldots \wedge x_n = y_n$$

Thus $(a, b) = (c, d)$ iff $a = c$ and $b = d$.

Definition 2

1. *Given 2 sets A and B the Cartesian product of A and B, denoted $A \times B$ (A cross B) is the set of ordered pairs (a, b) with $a \in A$ and $b \in B$.*

$$\textit{i.e. } A \times B = \{(a, b) \mid a \in A \wedge b \in B\}.$$

2. *Given sets $A_1, A_2, \ldots, A_n$ the Cartesian product $A_1 \times A_2 \times \ldots \times A_n$ is the set of all ordered n-tuples $(a_1, a_2, \ldots, a_n)$.*

$$\textit{i.e. } A_1 \times A_2 \times \ldots \times A_n = \{(a_1, a_2, \ldots, a_n) \mid a_1 \in A_1 \wedge a_2 \in A_2 \wedge \ldots \wedge a_n \in A_n\}.$$

Example 3

1. $A = \{1, 2\}$, $B = \{3, 4, 5\}$, $A \times B = \{(1, 3), (1, 4), (1, 5), (2, 3), (2, 4), (2, 5)\}$
2. $\mathbf{R} \times \mathbf{R} = \mathbf{R}^2 = \{(x, y) \mid x, y \in \mathbf{R}\}$.
3. $\mathbf{R}^n = \underbrace{\mathbf{R} \times \mathbf{R} \times \ldots \times \mathbf{R}}_{n \text{ times}} = \{(x_1, x_2, \ldots, x_n) \mid x_1, x_2, \ldots, x_n \in \mathbf{R}\}$.
4. $\mathbf{R} \times \mathbf{N} = \{(x, a) \mid x \in \mathbf{R} \wedge a \in \mathbf{N}\}$.

2 Alphabets and Strings

Definition 4

1. An alphabet, Σ is a finite set. The elements of an alphabet are called symbols or characters.

   Example 5

   (a) $\Sigma_E = \{a, b, \ldots, Y, Z\}$ - The standard alphabet for English.
   (b) $\Sigma_A = ASCII = \Sigma_E \cup \{!, @, \ldots, ?\}$ - Standard alphabet for computer I/O.
   (c) $\Sigma_0 = \{0, 1\}$ - The natural alphabet of computers.

2. A string over an alphabet Σ is any ordered n-tuple of elements of Σ.
   We usually write strings with no commas or parantheses.
   We allow the empty string and denote it by the symbol ϵ.

   Example 6

   (a) If $\Sigma = \Sigma_0$ then $\epsilon, 0, 00, 01, 11, 01101100$ are all strings over Σ.
   (b) If $\Sigma = \Sigma_E$ then ϵ, "a", "set", "qwerty" are all strings over Σ.

3. The length of a string is the number of characters which make it up.
   The empty string ϵ always has length 0.

   Example 7

   (a) $\Sigma = \Sigma_0$, 0 and 1 have length 1. 00, 01 and 11 have length 2.
   01101100 has length 8.
   (b) $\Sigma = \Sigma_E$, "a" has length 1, "set" has length 3, "qwerty" has length 6.

4. Given an alphabet Σ
   Σ^n denotes the set of all strings of length n over Σ.
   Σ^* denotes the set of all strings of any finite length (including 0) over Σ.

Example 8

$\Sigma = \Sigma_0$.
$\Sigma^0 = \{\epsilon\}$,
$\Sigma^1 = \Sigma = \{0, 1\}$,
$\Sigma^2 = \{00, 01, 10, 11\}$ etc.

5. Given any two strings x and y over an alphabet Σ, the concatenation of x and y is the string xy.

Example 9

$$x = 01,\ y = 001, \qquad xy = 01001, \qquad yx = 00101.$$

Generally, we use lowercase letters from the beginning of the alphabet a, b, c to denote single characters from an alphabet, and lowercase letters from the end of the alphabet u, v, w, x, y, z to denote strings of characters from an alphabet.

3 Formal Languages

Definition 10
A Formal Language over an alphabet Σ is some fixed subset, L, of Σ^*.
Members of L are called words.

Example 11

1. $\Sigma_0,\ L = \{00, 01, 10, 11\} = \Sigma^2$ - Binary strings of length 2.
2. $\Sigma_0,\ L = \Sigma^8$ - Bytes.
3. $\Sigma_0,\ L_0 = \{x \in \Sigma^* \mid x \text{ starts with } 0\ \}$.
4. $\Sigma_0,\ L_1 = \{x \in \Sigma^* \mid x \text{ ends with } 1\ \}$.
5. $\Sigma_E,\ L = \{$ English Words $\}$.
6. $\Sigma = \{0, 1, +, -\}$,
   $L = \{x \in \Sigma^* \mid x$ contains exactly one of + or -, and it is not the first or last symbol $\}$
   011+110, 1-0, 11+01 are all words.

4 Operations on Languages

Definition 12 *Let L_1 and L_2 be two languages (not necessarily distinct). Then we define the following operations:*

1. The union of L_1 and L_2 consists of any string which is in either L_1 or L_2.
$$L_1 \cup L_2 = \{x \mid x \in L_1 \vee x \in L_2\}$$
2. The set concatenation of L_1 with L_2 is the set of string obtained by concatenating every word from L_1 with every word from L_2.
$$L_1 L_2 = \{xy \mid x \in L_1 \wedge y \in L_2\}$$
3. The Kleene closure of a language L, denoted L^* is the set of all strings formed by concatenating any finite number of strings from L.
$L^n =$\{ strings formed by concatenating n words from L \}.
$L^* = \{\epsilon\} \cup L^1 \cup L^2 \cup L^3 \cup \ldots$.

Note: The Kleene closure allows us to concatenate *any* number of strings, including none. Thus the empty string is always in the Kleene closure of any language L, i.e. $\forall$ languages L, $\epsilon \in L^*$.

Example 13 *Let* $L_1 = \{0, 01\}$, $L_2 = \{1\}$.

$L_1L_2 = \{01, 011\}$, $L_2L_1 = \{10, 101\}$,
$L_1 \cup L_2 = \{0, 01, 1\}$, $L_1^2 = \{00, 001, 010, 0101\}$
$L_2^* = \{\epsilon, 1, 11, 111, \ldots\}$
$L_1^* = \{\epsilon, 0, 01, 00, 001, 010, 0101, \ldots\}$

5 Regular Sets & Regular Expressions

Regular Sets are formal languages which can be formed over an alphabet in a special way using the above operations. They have a special syntax which allows them to be easily recognised, this syntax is known as a regular expression. We define both the language and the syntax together.

Definition 14 (Regular Sets, Regular Expression) *Given an alphabet Σ the following are regular expressions over Σ:*

1. *$\{\}$. The empty set. Denoted ϕ.*
2. *$\{\epsilon\}$. The empty string. Denoted ϵ.*
3. *$\{a\}$ for every $a \in \Sigma$. Denoted a.*
4. *If L_A and L_B are regular languages over Σ, denoted by A and B respectively, then the following are also regular:*
   (a) *$L_A \cup L_B$ Denoted $(A \vee B)$ or $(A + B)$.*
   (b) *$L_A L_B$ Denoted (AB).*
   (c) *L_A^* Denoted (A^*).*
5. *No set other than those generated by a finite number of applications of 1 - 4 above is regular.*

We denote that set of all regular sets by $\mathcal{R}$

5.1 Parenthetic Omission

In certain circumstances we may drop some of the brackets from a regular expression.

1. We alway may drop the outermost bracket from a completed expression.
2. In the absence of explicit brackets, the order of precedence is Kleene closure, concatenation, union. Thus Kleene closure is performed first, followed by concatenation then union. So
$$x \vee yz^* = x \vee (y(z^*))$$
3. Given an alphabet Σ the following hold $\forall x, y, z \in \Sigma^*$,
   - Associativity of $\vee$, $x \vee (y \vee z) = (x \vee y) \vee z$. Thus $x \vee y \vee z$ is well defined.
   - Associativity of concatenation, $x(yz) = (xy)z$. Thus xyz is well defined.
   - Distributivity of concatenation over $\vee$, $(x \vee y)z = xz \vee yz$ and $z(x \vee y) = zx \vee zy$.

Note: Neither union nor concatenation distributes over Kleene closure, nor vice versa.

Example 15

1. $(0 \vee 1)^* = \Sigma_0^* =$ All strings over $\{0, 1\}$ (binary strings).
2. $0^* \vee 1^* =$ Strings which either consist of all zeros, or all ones $= \{\epsilon, 0, 00, 000, \ldots, 1, 11, 111, \ldots\}$

5.2 Examples

When describing languages given by a regular expression we can use the following phrases for the various operations.

- Union: *or*
- Concatenation: *followed by*
- Kleene closure: *as many times as we like*

Thus $a \vee bc^*$ could be expressed as *'Either b followed by as many c's as we like, or a alone'.*
While this will always produce *an* answer, a truly correct solution should describe the language as succinctly as possible.
For example, $(0 \vee 1)^*$ would be described as *'As many of either 0 or 1 as we like'.* But a much better answer is *'Any string of 0s and 1s'.*

Example 16

1. Find regular expressions for the following languages.
   (a) $L = \{x \in \{0, 1\}^* \mid x \text{ begins with a } 0\ \}$
   $0\,(0 \vee 1)^*$

(b) $L = \{x \in \{0,1\}^* \mid x \text{ begins with a 0 and ends in a 1}\}$
$0\,(0 \vee 1)^*\,1$

(c) $L = \{x \in \{0,1\}^* \mid x \text{ begins with a 0 or ends in a 1}\}$
$0\,(0 \vee 1)^* \vee (0 \vee 1)^*\,1$

(d) All strings with at least one 0.
$(0 \vee 1)^*\,0\,(0 \vee 1)^*$.

(e) All strings of length two or three from the alphabet $\{a, b, c, d\}$
$(a \vee b \vee c \vee d)(a \vee b \vee c \vee d)(\epsilon \vee a \vee b \vee c \vee d)$.

(f) All strings over {0, 1} which have no repeated 1's
$(10 \vee 0)^*(\epsilon \vee 1)$

2. Describe the languages which correspond to the following regular expressions over the given alphabet Σ.

(a) $\Sigma = \Sigma_0$, $(0 \vee 1)^*\,1$.
$L = \{x \in \{0,1\}^* \mid x \text{ ends in a 1 }\}$ = All strings ending in a 1.

(b) $\Sigma = \Sigma_0$, $(00 \vee 1)^*$.
"00 or 1 as many times as we like".
All strings over $\{0,1\}$ where the 0's appear in runs of even length.

(c) $\Sigma = \Sigma_0$, $0^*1\,0^*1\,0^*$.
All strings with exactly two 1's.

(d) $\Sigma = \Sigma_0$, $(0^*1\,0^*1\,0^*)^*$.
All strings with an even number of 1's.

(e) $(00 \vee 000)^*$
This Language consists of all strings of the form 0^{2n+3m} for some n and m in $\mathbb{N}$.
It can be shown (by induction) that any $k \in \mathbb{N}$, with $k \geq 2$, can be written in the form $k = 2n + 3m$. $k = 0$ can also be written in this form, but $k = 1$ cannot.
Thus this Language consists of all strings of 0's except the string '0' itself.

3. Suppose we are working in an operating system which only allows filenames containing the symbols a, b, c, d or ".". Further there can be at most one ".".

Let Σ = { valid filename characters } = $\{a, b, c, d, .\}$. Write the regular expression which gives the set of all possible filenames on this system.

$(a \vee b \vee c \vee d)^*(\epsilon \vee .)(a \vee b \vee c \vee d)^*$.

Using * as short for $(a \vee b \vee c \vee d)^*$ and dropping the ϵ gives *.*, look familiar?

4. Unix allows much more complex pattern matching using regular expressions (with its own syntax).

Try
`ls [aAbB]*`
to see all the files beginning with either "a" or "b"

Check out the man pages for regexp - built in routine for matching regular expressions.

Also check out the man pages for `grep, ed, vi, awk, sed` and many more.

5. Compilers use regular expressions to check syntax.

   Let $\Sigma_D = \{0, 1, \ldots, 9\}$, then a regular expression for identifiers might look like $(\Sigma_E \vee \_)(\Sigma_E \vee \Sigma_D \vee \_)^*$.

6 Exercises

1. Let $\Sigma = \{0, 1\}$. Describe the language denoted by the following regular expressions.
   (a) 0^*1^*
   (b) $0^*(1^* \vee 0^*)$
   (c) $0 \vee 1$
   (d) 01
   (e) $(01)^*$
   (f) $(00)^*$
   (g) $0^*(0 \vee 1)^*$
   (h) $(0^*1)^*0^*$
   (i) $(00^*)((0 \vee 1)^*1$
   (j) $(01)^*((01) \vee 1^*)$

2. Let $\Sigma = \{1, 2, 3\}$. Give a regular expression for each of the following regular languages over Σ.
   (a) $\{1, 2, 3\}$
   (b) $\{1, 12, 122, 1222, \ldots\}$
   (c) $\{\epsilon, 12, 1212, 121212, \ldots\}$
   (d) $\{\epsilon, 2, 3, 33, 333, \ldots\}$
   (e) $\{\epsilon, 1, 2, 12, 1212, 121212, \ldots\}$
   (f) All strings over Σ which contain an odd number of 1's.

3. (Hard) Let $\Sigma = \{0, 1\}$. Find a regular expression for all strings over Σ which contain an odd number of 0's, and an even number of 1's.