Formal languages I: regular languages

LING83800

1 Introduction

A *formal language* is a specification of a (usually infinite) *set* of *strings*, defined by formal rules. Formal languages are abstract mathematical objects, but certain types of formal languages map onto various domains of human language. These types of formal languages are relevant both to the cognitive science of language—they allow us to formalize what it means to *learn* or *parse* a language—and are used to build speech and language technologies.

In this handout, we'll define several key notions in formal language theory: *sets* and operations over sets (section 2), *strings* and operations over strings (section 3), and *languages* and operations over languages (section 4). We'll then introduce a family of formal languages known as the *regular languages* (section 5) and their relation to *regular expressions* (section 6).

In later weeks, we will introduce computational devices known as *finite automata*, which "express" or implement subsets of the regular languages, and then show how these can also be used to express grammatical rules.

Bibliographic note

Some of the examples and notation here are adapted from Partee et al. 1993:§1, Gorman and Sproat 2021:§1, and Hopcroft et al. 2008:§1.5. The notation here also roughly conforms to the notation in Prof. Al Khatib's semantics lectures. Jurafsky and Martin (2008):§2–2.1 review regular expression syntax in some detail.

2 Sets

2.1 Definition

A *set* is an abstract, unordered collection of distinct objects, the *members* or *elements* of that set.

- They are an abstract, purely logical notion, and their definition does not presuppose any particular method of representing them in hardware or software.
- They are unordered in the sense that there need not be any natural ordering among the elements or members of any set.

Members of a set can be of any type, including other sets. Sets may either be finite (e.g., the set consisting of students in this class) or infinite (e.g., the set of grammatical sentences of English).

Set membership is indicated with the $\in$ symbol (`\in` in LaTeX). The expression $x \in X$ is read as "x is a member of X". We can also deny this relation using $\notin$ (`\notin` in LaTeX); the expression $x \notin X$ is read as "x is not a member of X".

Problem How is a set as defined here like a Python `set`? How is it different?

Solution Like `set`, sets are unordered and do not contain repeated elements. However, Python `set` objects may not be infinite and may not contain other `set` objects. Furthermore, objects stored in a Python set must be immutable and hashable.

2.2 Specification

By convention, we use capital Italic letters (X, Y, Z) to denote sets and lowercase Italic letters x, y, z to denote members of sets.

There are several ways to specify a set. For finite sets, we can simply list the members enclosed in curly braces. This is known as *extension notation* or *list notation*.

$$\{2, 3, 5, 7\}$$

Note that it is an accidental feature that the members of a set are listed in a particular order; there is no intrinsic ordering of the members of a set. Thus all the following are equivalent:

$$\{2, 3, 5, 7\}, \{7, 5, 3, 2\}, \{3, 2, 7, 5\}, \{2, 5, 3, 7\}, \ldots$$

Another method to specify a set—including infinite sets—is to refer to properties that uniquely identify the set's members. This is known as *set-builder notation* or *predicate notation*.

$$\{x \mid x \text{ is prime}\}$$

2.3 Subsets

The set X is said to be a *subset* of another set Y just in the case that every member of X is also a member of Y. We indicate this using $\subseteq$ (`\subseteq` in LaTeX). The expression $X \subseteq Y$ is read as "X is a subset of Y". We can also deny this relation using $\nsubseteq$ (`\nsubseteq` in LaTeX); the expression $X \nsubseteq Y$ is read as "X is not a subset of Y". There is also a special set known as the *empty set*, written as $\varnothing$ (`\emptyset` in LaTeX). For every set S, $\varnothing \subseteq S$.

Problem Let:

$$\begin{aligned} K &= \{\text{Mars}, \text{Saturn}, \text{Uranus}\} \\ L &= \{x \mid x \text{ is a planet in our solar system}\} \end{aligned}$$

Is K a subset of L? And, is L a subset of K?

Solution $K \subseteq L$; $L \nsubseteq K$ (for instance, Venus $\in L, \notin K$).

2.4 Operations

2.4.1 Union

The *union* of two sets $X \cup Y$ (`\cup` in LaTeX) is the set that contains just the elements which are members of X, of Y, or both X and Y. Thus it corresponds to *disjunction operator* $\vee$ in logic, and (loosely) to the conjunction *or* in English.

$$X \cup Y = \{x \mid x \in X \vee x \in Y\}$$

Problem Let:

$$\begin{aligned} K &= \{a, b\} \\ L &= \{c, d\} \\ M &= \{b, d\} \end{aligned}$$

$$\begin{aligned} K \cup L &= \_\_ \\ K \cup M &= \_\_ \\ L \cup M &= \_\_ \\ K \cup K &= \_\_ \end{aligned}$$

Solution

$$\begin{aligned} K \cup L &= \{a, b, c, d\} \\ K \cup M &= \{a, b, d\} \\ L \cup M &= \{b, c, d\} \\ K \cup K &= \{a, b\} = K \end{aligned}$$

2.4.2 Intersection

The *intersection* of two sets $X \cap Y$ (`\cap` in LaTeX) is the set that contains just the elements which are members of both X and Y. Thus it corresponds to the *conjunction operator* $\wedge$ in logic, and to the conjunction *and* in English.

$$X \cap Y = \{x \mid x \in X \wedge x \in Y\}$$

Problem Let:

$$\begin{aligned} K &= \{a, b\} \\ L &= \{c, d\} \\ M &= \{b, d\} \end{aligned}$$

$$
\begin{aligned}
K \cap L &= \_\_ \\
K \cap M &= \_\_ \\
L \cap M &= \_\_ \\
K \cap K &= \_\_
\end{aligned}
$$

Solution

$$
\begin{aligned}
K \cap L &= \varnothing \\
K \cap M &= \{b\} \\
L \cap M &= \{d\} \\
K \cap K &= \{a, b\} = K
\end{aligned}
$$

2.4.3 Difference

The *difference* of two sets $X - Y$ is the set that contains just the elements which are members of X but not members of Y. (Recall that $\wedge$ is the logical conjunction operator.)

$$X - Y = \{x \mid x \in X \wedge x \notin Y\}$$

Problem Let:

$$
\begin{aligned}
K &= \{a, b\} \\
L &= \{c, d\} \\
M &= \{b, d\}
\end{aligned}
$$

$$
\begin{aligned}
K - M &= \_\_ \\
M - L &= \_\_ \\
K - \varnothing &= \_\_ \\
K - K &= \_\_
\end{aligned}
$$

Solution

$$
\begin{aligned}
K - M &= \{a\} \\
M - L &= \{b\} \\
K - \varnothing &= \{a, b\} = K \\
K - K &= \varnothing
\end{aligned}
$$

2.5 Closure properties

A set is said to be *closed* with respect to (or to have *closure* over) a binary mathematical operator $\bullet$ if for all sets X, Y, the expression denoted by $X \bullet Y$ is itself a set.[1] Sets are closed with respect to union, intersection, and difference, among other operators.

3 Strings

3.1 Definition

Let Σ be the *alphabet*, a (non-empty) finite set of symbols. We make no assumption about the nature of these symbols; they may be numbers, characters, words, etc. A *string* (or *word*) is any finite ordered sequence of zero or more symbols where each symbol is an member of Σ. The length of a string s, the number of symbols in that string, is denoted $|s|$. By convention, the empty string, a string of length 0, is denoted by ϵ (`\epsilon` in LaTeX).

Problem How is a string as defined here like a Python `str`? How is it different?

Solution Like `str`, strings are finite and ordered. However, Python `str` objects may only contain Unicode codepoints whereas strings may contain arbitrary symbols.

Problem Let $\Sigma = \{0, 1\}$. Now, list all strings of length 3.

Solution $\{\texttt{000}, \texttt{001}, \texttt{010}, \texttt{100}, \texttt{011}, \texttt{101}, \texttt{110}, \texttt{111}\}$.

3.2 Specification

There is no single convention for specifying strings. In some cases, we use comma-separated values wrapped in angular brackets (`\langle` and `\rangle` in LaTeX) to specify strings, as in $\langle \texttt{x}, \texttt{y} \rangle$. When the alphabet is a subset of printable characters, we may write strings using typewriter text (`\texttt` in LaTeX) as in `xy`.

3.3 Operations

3.3.1 Concatenation

The *concatenation* of two strings s and t, written st, is the string defined by the sequence of symbols in s followed by the sequence of symbols in t. Strings are closed with respect to concatenation.

Problem Let $s = \texttt{aab}$ and $t = \texttt{cdf}$. What is st? What is ts?

[1] Similarly, a set is said to be closed with respect to a unary (prefix) mathematical operator $\bullet$ if for all sets X, the expression denoted by $\bullet X$ is itself a set.

Solution

$$st = \mathtt{aabcdf}$$
$$ts = \mathtt{cdfaab}$$

3.3.2 Reversal

The *reversal* of a string s, written s^R, is the string defined by the sequence of symbols in s in reverse order. Strings are closed with respect to reversal.

Problem Let $s = \mathtt{aab}$ and $t = \mathtt{cdf}$. What is $s^R t$? What is $(st)^R$?

Solution

$$s^R t = \mathtt{baacdf}$$
$$(st)^R = \mathtt{fdcbaa}$$

4 Languages

4.1 Definition

A set of strings is traditionally known as a *language*. This is not intended to supplant common-sense—or linguistic—notions of what a human language is, it's just a term of art.

4.2 Specification

Languages are specified in the same fashion as ordinary sets: using either type of the curly-brace notation introduced in subsection 2.2.

4.3 Operations

As languages are sets, union, intersection, and difference (subsection 2.4) are all well-defined and languages are closed with respect to these operations.

4.3.1 Concatenation

We can generalize concatenation from strings to languages. If X and Y are languages, then XY contains the concatenation of each string $x \in X$ with each string $y \in Y$.

$$XY = \{xy \mid x \in X \wedge y \in Y\}$$

The notation X^n, where n is a natural number, denotes a language consisting of n "self-concatenations" of X; e.g., $X^0 = \{\epsilon\}$ and $X^4 = XXXX$.

Problem Let $S = \{\mathrm{a}, \mathrm{bc}, \mathrm{d}\}$ and $T = \{\mathrm{ef}, \mathrm{g}\}$. Now list the elements of ST, TT, and T^3.

Solution

$$
\begin{aligned}
ST &= \{\mathrm{aef}, \mathrm{ag}, \mathrm{bcef}, \mathrm{bcg}, \mathrm{def}, \mathrm{dg}\} \\
TT &= \{\mathrm{efef}, \mathrm{efg}, \mathrm{gef}, \mathrm{gg}\} \\
T^3 &= \{\mathrm{efefef}, \mathrm{efefg}, \mathrm{efgef}, \mathrm{efgg}, \mathrm{gefef}, \mathrm{gefg}, \mathrm{ggef}, \mathrm{ggg}\}
\end{aligned}
$$

4.4 Closure

The (concatenative) *closure* of a language X is the infinite union of zero or more concatenations of X with itself. It is denoted by a superscripted asterisk; e.g., $X^* = \bigcup_{i \geqslant 0} X^i = \{\epsilon\} \cup X \cup XX \cup XXX \cup \ldots$. One variant of closure, indicated with a superscripted plus-sign, excludes the empty string; e.g., $X^+ = \bigcup_{i>0} X^i = X \cup XX \cup XXX \cup \ldots$, or equivalently, $X^+ = XX^*$. These two variants of closure are colloquially referred to as *Kleene star* and *Kleene plus*, respectively. Finally, a superscripted question mark indicates optionality; e.g., $X^? = \{\epsilon\} \cup X$.

5 Regular languages

We are now in a position to define a class of formal languages known as the *regular languages* first characterized by Kleene (1956). Since languages are sets (of strings) we will denote them using capital Italic letters. The regular languages are a set of languages such that:

- The empty language $\varnothing$ is a regular language.
- The empty string language $\{\epsilon\}$ is a regular language.
- For every symbol $s \in \Sigma$, the singleton language $\{s\}$ is a regular language.
- If X is a regular language then the closure X^* is a regular language.
- If X and Y are regular languages then:
  - their union $X \cup Y$ is a regular language and
  - their concatenation XY is a regular language.
- Languages not so derived are not regular languages.

6 Regular expressions

The *regular expressions* are a terse representation of regular languages which use closure, union, and concatenation.[2] As Jurafsky and Martin (2008:17f.) write, regular expressions are among

[2] While regular languages are closed with respect to intersection and difference as well, neither are supported by regular expressions.

the "unsung successes in standardization in computer science". Regular expression matching is supported by Python's `re` module, command-line tools like `grep` and `sed`, and nearly all of these use roughly the same terse algebraic notation. Following the convention introduced by the Perl programming language, we write regular expressions in typewriter text and surrounded by forward slashes.

6.1 Correspondences

- Concatenation is implicit in regular expressions.

$$\texttt{/ab/} = \{\texttt{ab}\}$$

- Kleene star corresponds to the quantifier `*`.

$$\texttt{/a*(bb)*/} = \{\texttt{a}\}^*\{\texttt{bb}\}^*$$

- Kleene plus corresponds to the quantifier `+`.

$$\begin{aligned}\texttt{/yes+/} &= \{\texttt{ye}\}\{\texttt{s}\}^+ \\ &= \{\texttt{yes}, \texttt{yess}, \texttt{yesss}, \ldots\}\end{aligned}$$

- The "Kleene question mark" corresponds to the quantifier `?`.

$$\begin{aligned}\texttt{/colou?r/} &= \{\texttt{colo}\}\{\texttt{u}\}^?\{\texttt{r}\} \\ &= \{\texttt{color}, \texttt{colour}\}\end{aligned}$$

- Union $\cup$ corresponds to several notations:
  - Square brackets indicate the union of single characters.

$$\begin{aligned}\texttt{/[Dd]addy/} &= (\{\texttt{D}\} \cup \{\texttt{d}\})\{\texttt{addy}\} \\ &= \{\texttt{Daddy}, \texttt{daddy}\}\end{aligned}$$

- Square brackets can also be used to indicate a union of a range of single characters.

$$\begin{aligned}\texttt{/Rocky\_[1-3]/} &= \{\texttt{Rocky\_}\}(\{\texttt{1}\} \cup \{\texttt{2}\} \cup \{\texttt{3}\}) \\ &= \{\texttt{Rocky\_1}, \texttt{Rocky\_2}, \texttt{Rocky\_3}\}\end{aligned}$$

- The | operator indicates unions of arbitrary-length character sequences.

$$\begin{aligned}\texttt{/gupp(y|ies)/} &= \{\texttt{gupp}\}(\{\texttt{y}\} \cup \{\texttt{ies}\}) \\ &= \{\texttt{guppy}, \texttt{guppies}\}\end{aligned}$$

Problem List all the strings the following regular expression matches:

$$\texttt{/(Hellrais|Highland|Loop|Sleep|Zooland)er/}$$

Solution

$$\{\texttt{Hellraiser}, \texttt{Highlander}, \texttt{Looper}, \texttt{Sleeper}, \texttt{Zoolander}\}$$

Problem List some strings matched by the following regular expressions:

$$\begin{aligned}\texttt{/[a-z]*burger/} \\ \texttt{/(in)?(de)?fatigable/}\end{aligned}$$

Solution

$$\begin{aligned}\texttt{/[a-z]*burger/} &= \{\texttt{burger}, \texttt{cheeseburger}, \texttt{veggieburger}, \ldots\} \\ \texttt{/(in)?(de)?fatigable/} &= \{\texttt{fatigable}, \texttt{infatigable}, \texttt{indefatigable}, \ldots\}\end{aligned}$$

6.2 Extensions

It should be noted that many regular expression engines (including Python's `re`) have additional "extended" features that allow them to exceed the capacity of the regular languages.

References

Gorman, Kyle, and Richard Sproat. 2021. *Finite-State Text Processing*. Morgan & Claypool.

Hopcroft, John E., Rajeev Motwani, and Jeffrey D. Ullman. 2008. *Introduction to Automata Theory, Languages, and Computation*. Pearson.

Jurafsky, Daniel, and James Martin. 2008. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* Pearson Prentice Hall, 2nd edition.

Kleene, Stephen C. 1956. Representation of events in nerve nets and finite automata. In *Automata Studies*, ed. Claude E. Shannon and J. McCarthy, 3–42. Princeton University Press.

Partee, Barbara H., Alice ter Meulen, and Robert E. Wall. 1993. *Mathematical Methods in Linguistics.* Kluwer Academic Publishers, 2nd edition.