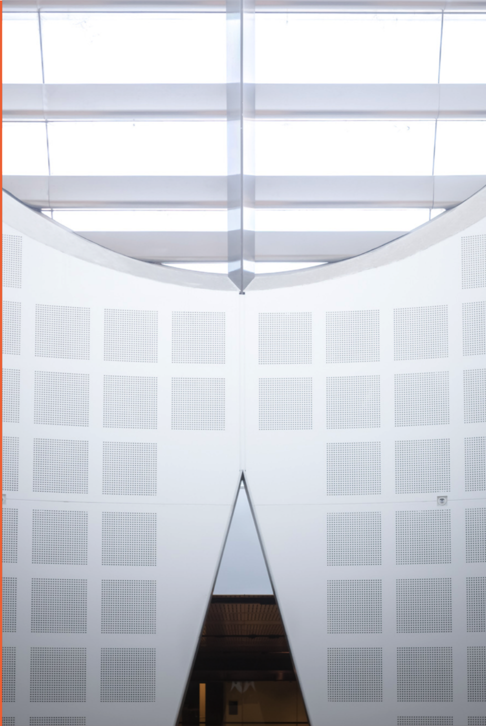
COMP2022|2922 Models of Computation

Formal Language Theory and Recursive Definitions

Sasha Rubin

August 4, 2022

THE UNIVERSITY OF SYDNEY



Agenda

1. Computational problems and formal languages
2. Regular Expressions

Computational problems

A computational problem specifies what the input can be and what the output should be.

Examples

- Given a list of integers, sort it.
- Given a list of integers, decide if it is sorted.
- Given a graph G and vertices s, t, find a shortest path from s to t (or say there is none).
- Given a graph G and vertices s, t and an integer k, decide if there is a path s to t of length at most k.

A decision problem only allows Yes (1) or No (0) outputs.

Decision problems on strings

We focus on decision problems where the input is a string.

Examples

- Given an ASCII string, is it a legal `Python` program?
- Given a decimal string, does it represent a prime number?

Tutorial question: is focusing on strings a serious restriction?

Sets of strings

- Every decision problem on strings can be viewed as a set of strings

  i.e., the set of strings for which the answer is "Yes".

Examples

- The set of ASCII strings that are legal `Python` programs.
- The set of decimal strings that represent prime numbers.

Terminology

- A set of strings is also called a **language**.
- Their study is called *Formal Language Theory*.
- They are inspired by but different from natural languages, like English.

Let's dive into some basic concepts about Formal Languages.

Formal Languages

Definition

An alphabet Σ (read 'Sigma') is a nonempty finite set, whose elements are called symbols.

Examples

- $\Sigma = \{0, 1\}$ is the classic binary alphabet.
- $\Sigma = \{a, b, c, \cdots, z\}$ is the lower-case English alphabet.
- $\Sigma = \{0, 1, 2, +, -, \div, \times, (,)\}$ is the arithmetic expression alphabet for base 3.
- $\Sigma =$ ASCII characters
- $\Sigma =$ Kanji characters
- We usually do not allow certain characters to be part of an alphabet, e.g., ϵ (read 'epsilon' or 'empty string') or $\emptyset$ (read 'empty set') or Σ (read 'Sigma')

Formal Languages

Definition

A string over Σ is a finite sequence of symbols from Σ. The number of symbols in a string is its length.

Examples

- Like `Python`'s `str` object
- 0110 is a string of length 4 over alphabet $\{0, 1\}$
- bob is a string of length 3 over alphabet $\{a, b, c, \cdots, z\}$.
- If w has length n we write $w = w_1 w_2 \cdots w_n$ where each $w_i \in \Sigma$.
- There is only one string of length 0. It is denoted ϵ (read 'empty string'). It is not a symbol from Σ! You can think of it like `''` in `Python`.
- The set of all strings over Σ is denoted Σ^* (read 'Sigma star') — this notation will be explained later.

Formal Languages

Definition

The concatenation of strings x, y is the string xy formed by appending y to the end of x.

Examples

- The concatenation of $x = 010$ and $y = 01$ is 01001.
- $w\epsilon = \epsilon w = w$ for all strings w.
  e.g., $01\epsilon = 01$.
- For $k \in \mathbb{N}$, we define w^k to be $\overbrace{ww\cdots w}^{k}$.
- And w^0 is defined to be ϵ.

Formal Languages

Definition

A set $L \subseteq \Sigma^*$ of strings is called a language over Σ.

Examples

- $L =$ the set of all strings representing legal `Python` programs
- $L =$ the set of all binary strings representing prime numbers
- $L =$ the set of all true statements about arithmetic.
- $L =$ the set of all English words (in a given dictionary)
- $L =$ the set of all English sentences (in a given book).
- Note. The empty set $\emptyset$, also written $\{\}$, is a language with no elements in it, but the set $\{\epsilon\}$ is a language with one element in it, the empty string.
- Note. The languages $\{0, 1\}$ and $\{1, 0\}$ are equal, but the strings 01 and 10 are not.

Important operations on languages

Definition

Let A, B be languages over Σ.

– The union of A and B, written $A \cup B$, is the language $\{x \in \Sigma^* : x \in A \text{ or } x \in B\}$.

– The concatenation of A and B, written AB (or $A \circ B$), is the language $\{xy \in \Sigma^* : x \in A, y \in B\}$.

– Write A^k for $\overbrace{AA \cdots A}^{k}$.

and define A^0 to mean $\{\epsilon\}$.

– The star (aka iteration) of A, written A^*, is the language

$$\bigcup_{n \in \mathbb{N}} A^n = A^0 \cup A^1 \cup A^2 \cup A^3 \cup \dots$$

i.e., $A^* = \{x_1 x_2 \cdots x_k \in \Sigma^* : k \geq 0, \text{each } x_i \in A\}$.

Important operations on languages

Examples

1. $\{a, ab\} \cup \{ab, aab\} =$
2. $\{a, ab\}\{ab, aab\} =$
3. $\{a, ab\}^* =$

To think about

1. Are the following formal languages? If so, what is the alphabet and what are the strings?
   - English text
   - HTML
   - Sheet music
2. What are the similarities and differences between formal languages and natural languages like English?

Terminology so far...

- Alphabet (= finite set of symbols)
- String (= finite sequence of symbols)
- Language (= set of strings)
- Operations on strings (concatenation)
- Operations on languages (union, concatenation, star)

Agenda

1. Computational problems and formal languages
2. Recursive definitions and recursive programs
3. Regular Expressions

Regular expressions in a nutshell

1. Expressions that describe "simple" languages
2. Extremely useful
   - Text processing, e.g., pattern matching
   - In Natural Language Processing as features in (machine learning) classifiers

     `COMP5046:Natural Language Processing`
   - Scanners (aka Lexical analysers, Tokenisers)

     `COMP3109:Programming Languages and Paradigms`
   - Specification of data formats
   - Foundations of query languages for graph databases
3. Based on three operations on languages: Union, Concatenation, Star

NB. Many implementations of Regular Expressions have extra features (which we will not study in this course).

WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

https://xkcd.com/208/

Regular expressions: Syntax

Definition

Let Σ be an alphabet. The regular expressions over Σ are strings defined by the following recursive process:

1. The symbols $\emptyset$ and ϵ are regular expressions
2. Each symbol a from Σ is a regular expression
3. If R_1, R_2 are regular expressions then so is $(R_1 \mid R_2)$
4. If R_1, R_2 are regular expressions then so is $(R_1 R_2)$
5. If R is a regular expressions then so is R^*

Regular expressions

Examples

Let $\Sigma = \{a, b\}$.

- $(a \mid \emptyset)$
- $(a\epsilon)$
- b^*
- $((a^* \mid b^*)(ac))^*$

Notation

We may drop the outermost parentheses to improve readability.

- we may write $a \mid \emptyset$ instead of $(a \mid \emptyset)$
- we may write ab^* instead of (ab^*), which is different from $(ab)^*$.

Regular expressions: Semantics (wordy)

We say that a string x matches a RE R according to the following rules:

1. No string matches the RE $\emptyset$, and

   only the empty string matches the RE ϵ.
2. For alphabet symbol $a \in \Sigma$, only the string a matches the RE a.
3. A string x matches $(R_1 \mid R_2)$ if x matches R_1 or x matches R_2.
4. A string x matches $(R_1 R_2)$ if x can be written as $x = uv$ where u matches R_1 and v matches R_2.
5. A string x matches R^* if x can be expressed as the concatenation of zero or more strings, all of which match R.

The set of all strings that match R is written $L(R)$, and is called

- the language specified by R,
- the language represented by R, or
- the language of R.

Regular expressions

Examples

Let $\Sigma = \{a, b\}$. Write in plain English the languages represented by the following regular expressions.

- $L(a \,|\, \emptyset) =$
- $L(a\epsilon) =$
- $L(b^*) =$
- $L(a^* \,|\, b^*) =$
- $L((a \,|\, b)^*) =$
- $L((ab)^*) =$

Regular expressions: Semantics (mathy)

Here is a recursive definition of $L(R)$.

Definition

Let Σ be an alphabet. The language $L(R)$ of a RE R is defined by the following recursive procedure:

1. $L(\emptyset) = \{\}$ and $L(\epsilon) = \{\epsilon\}$.
2. $L(a) = \{a\}$ for $a \in \Sigma$.
3. $L(R_1 \mid R_2) = L(R_1) \cup L(R_2)$.
4. $L(R_1 R_2) = L(R_1) L(R_2)$.
5. $L(R^*) = L(R)^* = \{\epsilon\} \cup L(R)^1 \cup L(R)^2 \cup L(R)^3 \cup \cdots$

For a string x and RE R, if $x \in L(R)$ we say that x matches R.

Regular expressions

Notation.

1. We may drop parentheses for associative operations.
   - we may write $(R_1 \mid R_2 \mid R_3)$ instead of $((R_1 \mid R_2) \mid R_3)$
   - we may write $(R_1 R_2 R_3)$ instead of $((R_1 R_2) R_3))$
2. If $A = \{a, b, c\}$ we may write A instead of $(a \mid b \mid c)$.
   - we may write A^* instead of $(a \mid b \mid c)^*$.
   - we may do this for any finite set A of strings.
3. Pedantic note. We are **overloading** notation for convenience.
   - $\emptyset$ is a **symbol** used in REs and the empty **set** of strings.
   - ϵ is a **symbol** used in REs and a **string**.
   - $a \in \Sigma$ is a **symbol** used in REs and an **alphabet symbol** and a **string** of length 1.
   - * is a **symbol** used in REs and an **operation** on sets of strings.
   - some texts use $+$ or $\cup$ in regular expressions instead of $|$.

Regular expressions

Do you think that regular expressions specify all languages?

Vote now! (on mentimeter)

Regular expressions

Which languages can be specified by regular expressions?

In the study of programming languages...

1. Regular expressions can be used to specify keywords and identifiers

   The set of identifiers in `C`.
   Signed and unsigned integers.
   Hexadecimal numbers prefixed with '0x'

2. However, for specifying expressions and statements in programming languages, we will need a more powerful model (later).

Coda: Algorithmic issues

1. Membership problem for REs: Given a regular expression R and string s, decide if $s \in L(R)$.
   - Is there an algorithm that solves this problem?
2. Equivalence problem for REs: Given regular expressions R_1, R_2, decide if $L(R_1) = L(R_2)$.
   - Is there an algorithmm that solves this problem?

p.s. Can you see why the membership problem is actually a *decision* problem? Same goes for the equivalence problem.

Recursion

We will provide recursive procedures for defining many objects in this course. Why?

1. Precise, unambiguous.
2. Can be implemented as is!

Tutorial question: Write a recursive algorithm for solving the membership problem for REs.

There is also an algorithm that solves the equivalence problem, although it is probably too hard right now... will see elegant solutions to both problems once we study an important model of computation called "finite automata".