Languages

COMPSCI 3331

Languages: Outline

- Languages: definitions and examples.
- Language operations.
- Proofs involving Languages.

What is a language?

Natural Languages

- Natural languages are governed by rules, exceptions ...
- Different alphabets to represent written words.
- Not very formal.
- Very complex.

Programming Languages: C++, Java, Basic, etc.

- Easier to “understand” (parsing).
- Well-defined: we can determine what is and is valid.

Formal Languages

We will deal with **formal languages**:

- A **symbolic** representation of a language.
- Does not necessarily have any communicative value.
- Some formal languages do represent something meaningful.
  - e.g., Language of Java identifiers.
- Allows **formal** reasoning (proofs).

Where do we use formal languages?

- Lexical analysis: convert a program from sequences of characters to units (variable names, keywords, numeric literals, etc.)
- Parsing: build an internal representation of a program (parse tree)
- Compiler Optimization: optimize code to speed execution
- Compiling and interpreting.

Formal Languages: Alphabets and Words

Let Σ be a **finite** set of symbols. $\Sigma = \{a, b, c, \dots\}$.

- The symbols $a, b, c, \dots$ are called **letters**.
- The set Σ of letters is called an **alphabet**.

A **word** over an alphabet Σ is finite sequence of letters from Σ:

- If $\Sigma = \{a, b, c\}$, then *babc* is a word over Σ.
- If Σ is Unicode, then αüb í is a word over Σ.
- So are `Western`, `computer` and
  `for (i=0;i<=10;i++) { n = n*i; }`
- We will usually denote letters by a, b, c, d, e and words by w, x, y, z or α, β, γ.

bbaabc

Formal Languages

- The **empty word** is the word with no letters.
- It is denoted by ε.
- (Other sources may use λ or Λ.)

Length: The length of a word is the number of letters in the word. We denote the length of a word w by $|w|$.

- e.g., $|abc| = 3$, $|aabaab| = 6$.
- The empty word has length zero: $|\varepsilon| = 0$.

Sometimes need to refer to the number of times a letter appears in a word.

- $|w|_c$ is the number of times c appears in w.
- e.g., $|cbaabcc|_b = 2$

Operations on words

Concatenation: given two words x, y, xy is the sequence of all letters in x followed by all the letters of y.

- $x = abba$, $y = caa$.
  $xy = abbacaa$
- Note that $w\varepsilon = w$ for all words w. $\varepsilon x = x$.
  $\varepsilon x = x$

Repetition: x^i is the concatenation of i copies of x:

- $w^0 = \varepsilon$
- $w^i = w^{i-1}w$ for all $i \geq 1$

$w^2 = ww$

Relations on words

Given words w, x, y, z:

- if $w = xyz$ then y is a **subword** of w.
- if $w = xy$ then x is a **prefix** of w.
- Also in this case, y is a **suffix** of w.

$w = abaabc$

$y = ba$

Reversal

If w is a word, then w^R is the reversal of the word w, where the letters appear in the reverse order.

- For all words x, y, $(xy)^R = y^R x^R$.

$w =$ abaacca

$w^R =$ accaaba

Formal Languages

Given an alphabet Σ, the **set of all words** over Σ is denoted Σ^*

1. If $\Sigma = \{a, b, c\}$, then
   $\Sigma^* = \{\varepsilon, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, aab, \ldots\}$.
2. If $\Sigma = \{a\}$, then $\Sigma^* = \{\varepsilon, a, aa, aaa, aaaa, \ldots, \}$.
3. If $\Sigma = \emptyset$, then $\Sigma^* = ?$.

$\Sigma^* = \{\varepsilon\}$

Formal Languages

Languages: A language (over an alphabet Σ) is any set of words over Σ, i.e., any subset $L \subseteq \Sigma^*$ is a language.

For $\Sigma = \{a, b\}$, we have
$\Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \dots\}$.
Here are some examples of languages over Σ:

- $L = \{a, ba\}$ is a language. It is finite.
- $L = \{x \in \{a, b\}^* \ : \ |x| \leq 100\}$.
- $L = \{\varepsilon, ab, aabb, aaabbb, aaaabbbb, \dots\}$ is an infinite language. It consists of all words of the form $a^n b^n$ for some $n \geq 0$.

$$L = \{a^n b^n \ : \ n \geq 0\}$$

More Formal Languages

Let $\Sigma = \{0, 1\}$.

- Let L be the set of all words which are binary encodings of the positive integers that do **not** go to ~~zero~~ under repeated application of the Collatz function. one

Let $\Sigma = \mathtt{UNICODE}$.

- Let $L \subseteq \Sigma^*$ be the set of all Java programs which compute π to 1,000,000 places.

Some descriptions of languages are more useful than others.

Some Special Languages

1
- $\emptyset$: the language containing no words at all;

2
- $\{\varepsilon\}$: the language consisting of one word ε (the word with no symbols).
- ALWAYS REMEMBER: the last two languages are different!
- Σ^+: all non-empty words (i.e., all of Σ^* *except* ε).
- Σ^* itself is a language.

What Can We Do with Languages?

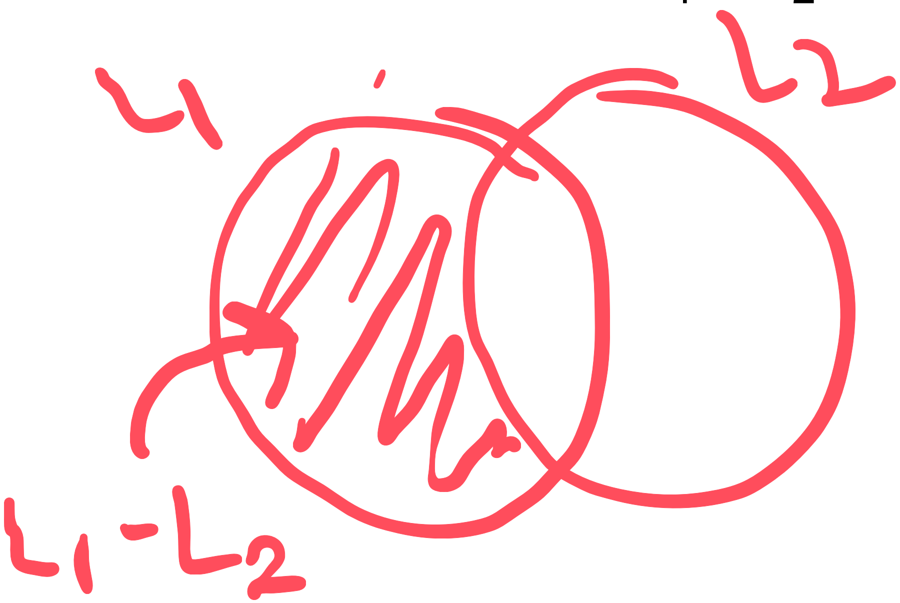
What can we do with languages?

- **classify them**: how difficult are they?
- Use to model things: e.g., $\Sigma = \{\mathtt{S}, \mathtt{R}, \mathtt{A}, \dots\}$. $L \subseteq \Sigma^*$: sequence of possible events under given communication protocols.
- **combine them**: language operations.

SRAS . . .

Language Operations

- What is an **operation**?
- Example: **arithmetic operations**: addition, multiplication, exponentiation.
- If $L_1, L_2 \subseteq \Sigma^*$ are languages, then we can combine them using the operations
  - union: $L_1 \cup L_2 = \{x \in \Sigma^* \ : \ x \in L_1 \text{ or } x \in L_2\}$.
  - intersection: $L_1 \cap L_2 = \{x \in \Sigma^* \ : \ x \in L_1 \text{ and } x \in L_2\}$.
  - difference: $L_1 - L_2 = \{x \in \Sigma^* \ : \ x \in L_1 \text{ and } x \notin L_2\}$.



Language Operations

Complement: If $L \subseteq \Sigma^*$ is a language, then $\overline{L} \subseteq \Sigma^*$ is the complement of L.

- $\overline{L} = \Sigma^* - L$.
- e.g., if $\Sigma = \{a\}$ and $L = \{a^i \ : \ i \text{ is even.}\}$ then $\overline{L} = \{a^i \ : \ i \text{ is odd.}\}$.

$L = \{\varepsilon, aa, aaaa, aaaaaa, \ldots\}$

$\overline{L} = \{x \in \{a, b\}^* : |x| > 100\}$

Language Operations: Concatenation

If $L_1, L_2 \subseteq \Sigma^*$ are languages, then

$$L_1 L_2 = \{xy \ : \ x \in L_1, y \in L_2\}.$$

$L_1 L_2$ is the **concatenation** of L_1 and L_2.

Example: $L_1 = \{ab, a, b\}$, $L_2 = \{\varepsilon, b\}$.

$L_1 L_2 = \{ab, a, b, abb, bb\}$

Special Concatenations

- concatenation with the empty language: $L\emptyset = ?$
- concatenation with the language consisting of empty word: $L\{\varepsilon\} = ?$

$$L\emptyset = \emptyset$$

$$L\{\varepsilon\} = L$$

Laws involving Concatenation

$$
\begin{aligned}
L_1(L_2L_3) &= (L_1L_2)L_3 \\
L_1(L_2 \cup L_3) &= L_1L_2 \cup L_1L_3 \\
(L_2 \cup L_3)L_1 &= L_2L_1 \cup L_3L_1
\end{aligned}
$$

$L_1L_2L_3$

Powers of Languages

Powers of Languages:

- $L^2 = LL$.
- e.g., $L = \{a, b, aa\}$.
- $L^2 = ?$.

$$L^2 = \{aa, ab, aaa, ba, bb, baa, aab, aaaa\}$$

Powers of Languages

- $L^n = L^{n-1}L$ for $n \geq 2$; ($L^1 = L$).
- We also define $L^0 = \{\varepsilon\}$.
- Definition of L^*, L^+:

$$\boxed{L^* = \bigcup_{i \geq 0} L^i}$$

$$L^+ = \bigcup_{i \geq 1} L^i$$

- We call the operation L^* **Kleene star** (or **Kleene closure**).

Powers and Kleene star

$$L^* = \bigcup_{i \geq 0} L^i$$

- If $x \in L^*$, then $x \in L^n$ for some $n \geq 0$.
- What does such an x look like?
- **x is the result of concatenating n strings from L together: $x = x_1 x_2 \cdots x_n$ where $x_i \in L$.**
- Each of these x_i can be the same, or different.

$\Sigma^* =$ set all words over Σ

Powers and Kleene star: Examples

- If $L = \{a, b\}$, then L^n is all words over $\{a, b\}$ of length n.
- If $L = \{a, ba\}$, then L^* contains the words:

$$\varepsilon, a, aa, ba, aaa, aba, baa, aaaa, aaba, abaa, baaa, baba, \ldots$$

L^1 L^1

L^2

a ba

a ba a

Proofs involving Languages

Languages are sets; certain proof techniques are typically used.

- **to show** $L_1 = L_2$, we need to show that (a) $L_1 \subseteq L_2$ and (b) $L_2 \subseteq L_1$.
- **to show** $L_1 \subseteq L_2$, a proof would follow the general pattern: "Let $x \in L_1$ be arbitrary. Then (use some property of words in L_1). Therefore, $x \in L_2$."

Other proofs on languages are proofs by induction, usually on the length of words in the language.

Problems vs Languages

- In this course, we will focus on languages (sets of words over an alphabet).
- We will consider a lot of decisions about languages
  - Is a word in a language?
  - How *difficult* is a language?
- But languages can represent complex problems through *encoding*.
- Provides a different, consistent way to think about problems: through the language they encode.

Encodings

$= \{10, 11, 101, \ldots\}$

- "Is a number prime?" vs.
  $\{x \in \{0,1\}^* \ : \ x \text{ is a prime number in binary. } \}$
- "Compute the intersection of two lists" vs
  $\{L1\#L2\#L3 \ : \ L1, L2, L3 \text{ are lists and } L1 = L2 \cap L3\}$
- "Does a C++ program compile without errors?" vs
  $\{x \ : \ x \text{ is a C++ program that compiles successfully.}\}$

Encodings

- Encoding a problem as a language means **membership** is important.
  - Membership: “is the word x in the language?”
- Encodings are up to us - anything can be encoded (data structures, programs, ...)
- Encodings don’t change how hard a problem is: e.g., if you can solve a problem, then you can determine membership in an encoded language.