Théorie des Langages Rationnels

A Framework for Languages

Adrien Pommellet, LRE

February 1, 2023

Deciding Languages

A premature goal

Before giving meaning to a word in a language, a computer should be able to tell **whether said word belongs to the language** in the first place.

Decidable language

A language L over an alphabet Σ is said to be decidable (or **recursive**) if there exists an algorithm `A` such that, on any input $w \in \Sigma^*$:

- If $w \in L$, then `A`(w) answers `true`.
- If $w \notin L$, then `A`(w) answers `false`.

We keep the **exact** definition of an algorithm vague on purpose: assume for now it is any method we can implement using a computer program, memory and speed issues notwithstanding.

Practical Application

Exercise 1. Are the following languages decidable?

1. The set of prime numbers.
2. The empty set $\emptyset$.
3. The set Σ^* of all words.
4. The set of C source codes of programs that halt on a given input x.
5. The set of C source codes of programs that halt in less than ten seconds on a given input x.

Answer

1. It is decidable: we can decide whether an integer is prime or not, as an example by using the sieve of Eratosthenes.
2. It is decidable: always answer `false`.
3. It is decidable: always answer `true`.
4. This language $\mathcal{H}$, **known as the halting problem**, is undecidable (regardless of the programming language): no code analysis can solve this problem. **You must admit and know this classical result**.
5. It is decidable: we merely have to compile the program, then run it with a ten second timeout. We then answer `true` if it has halted before the timeout, and `false` otherwise.

Some problems can't be solved by computers, regardless of their computing power.

Semi-decidable language

A language L over an alphabet Σ is said to be semi-decidable (or **recursively enumerable**) if there exists an algorithm $\mathtt{A}$ such that, on any input $w \in \Sigma^*$:

- If $w \in L$, then $\mathtt{A}(w)$ answers `true`.
- If $w \notin L$, then $\mathtt{A}(w)$ answers `false` or **does not end**.

Semi-decidable languages are said to be recursively enumerable because it is possible to design an algorithm that **enumerates** all the words in said language.

Exercise 2. Is the halting problem $\mathcal{H}$ recursively enumerable?

Answer

The halting problem $\mathcal{H}$ may not be recursive, but **it is still recursively enumerable**.

Consider indeed an algorithm `A` that merely runs the input program P on the input data x then returns `true` once it ends: it returns `true` if $P(x)$ ends, and does not halt otherwise.

Theorem

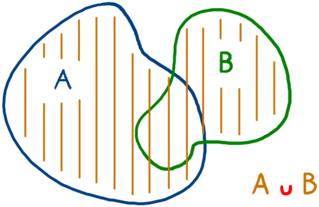
If a language L is recursive, then it is recursively enumerable as well.

The **converse is false**: consider indeed the halting problem $\mathcal{H}$.

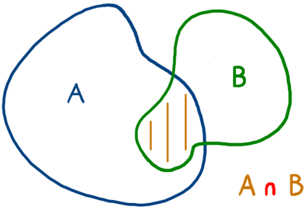
Set-theoretic Operations

Union

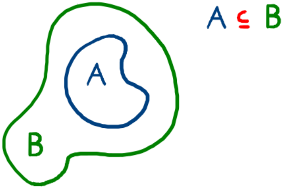
Given two languages A and B over an alphabet Σ, for all word $w \in \Sigma^*$, $w \in A \cup B$ if and only if $w \in A$ **or** $w \in B$.



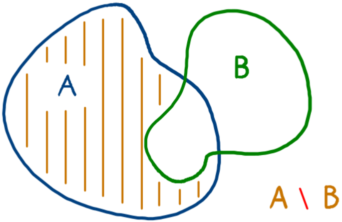
Given two languages A and B over an alphabet Σ, for all word $w \in \Sigma^*$, $w \in A \cap B$ if and only if $w \in A$ **and** $w \in B$.



Given two languages A and B over an alphabet Σ, $A \subseteq B$ if for all $w \in \Sigma^*$, $w \in A$ **implies that** $w \in B$.



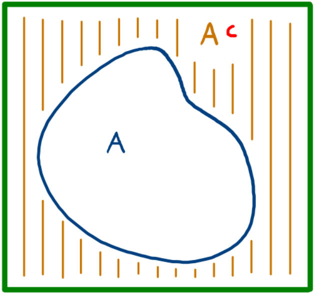
Given two languages A and B over an alphabet Σ, for all word $w \in \Sigma^*$, $w \in A \setminus B$ if and only if $w \in A$ **but** $w \notin B$.



Set-theoretic Operations

Complement

Given a language A over an alphabet Σ, for all word $w \in \Sigma^*$, $w \in A^\complement$ if and only if $w \notin A$. Note that $A^\complement = \Sigma^* \setminus A$. We also write $\bar{A} = A^\complement$.



Set-theoretic Operations

Common properties

Given three languages A, B and C over an alphabet Σ, these properties hold:

$$\begin{array}{rcl}
{A^{\complement}}^{\complement} & = & A \\
A \cup A^{\complement} & = & \Sigma^* \\
(A \cup B)^{\complement} & = & A^{\complement} \cap B^{\complement} \\
(A \cap B)^{\complement} & = & A^{\complement} \cup B^{\complement} \\
A \cap (B \cup C) & = & (A \cap B) \cup (A \cap C) \\
A \cup (B \cap C) & = & (A \cup B) \cap (A \cup C)
\end{array}$$

Some operations on words can be extended to languages.

Concatenation

Given two languages L_1 and L_2, we define their concatenation as the language $L_1 \cdot L_2 = \{w_1 \cdot w_2 \mid w_1 \in L_1, w_2 \in L_2\}$.

As an example, if $L_1 = \{ab, ac\}$ and $L_2 = \{cd, de\}$, then their concatenation is $L_1 \cdot L_2 = \{abcd, abde, accd, acde\}$.

Exponentiation of languages

Exponentiation

Given an integer $k \in \mathbb{N}$ and a language L, we define the languages:

- $L^0 = \{\varepsilon\}$.
- $L^k = \underbrace{L \cdot \ \cdots \ \cdot L}_{k \text{ times}}$.

L^k is the set of all words obtained by concatenating k words of L.

Note that $L^k \neq \{u^k \mid u \in L\}$. We can concatenate k **different** words together, we don't always have to repeat the **same** word k times.

As an example, if $L = \{ab, cd, e\}$, then $abcde \in L^3$.

Prefix of a language

Let L be a language on Σ. The language $\mathrm{Pref}(L)$ is the set of all words that are the prefix of **at least one** word of L.

Formally, $\mathrm{Pref}(L) = \{w \mid x \in L, w \in \mathrm{Pref}(x)\}$. We define $\mathrm{Suff}(L)$ and $\mathrm{Fact}(L)$ in a similar manner.

Note that $w \in \mathrm{Pref}(L)$ does not have to be a prefix of **every** word in L.

As an example, $\mathrm{Pref}(\{a, bc\}) = \{\varepsilon, a, b, bc\}$.

Exercise 3. Are the following properties true in the general case, assuming a generic language L? If they're not, use **counter-examples** to falsify them.

$$
\begin{aligned}
\mathrm{Fact}(L) &= \mathrm{Pref}(L) \cup \mathrm{Suff}(L) \\
\mathrm{Pref}(L) &= \mathrm{Pref}(\mathrm{Pref}(L)) \\
\mathrm{Pref}(\mathrm{Fact}(L)) &= \mathrm{Pref}(L) \\
\mathrm{Pref}(\mathrm{Fact}(L)) &= \mathrm{Fact}(L) \\
\mathrm{Fact}(L) &= \mathrm{Pref}(\mathrm{Suff}(L))
\end{aligned}
$$

Answer

$$
\begin{aligned}
\mathrm{Fact}(L) &\quad\neq\quad \mathrm{Pref}(L) \cup \mathrm{Suff}(L) \\
b \in \mathrm{Fact}(\{abc\}) &\quad\text{but}\quad b \notin \mathrm{Pref}(\{abc\}) \cup \mathrm{Suff}(\{abc\}) \\
\mathrm{Pref}(L) &\quad=\quad \mathrm{Pref}(\mathrm{Pref}(L)) \\
\mathrm{Pref}(\mathrm{Fact}(L)) &\quad\neq\quad \mathrm{Pref}(L) \\
b \in \mathrm{Pref}(\mathrm{Fact}(\{abc\})) &\quad\text{but}\quad b \notin \mathrm{Pref}(\{abc\}) \\
\mathrm{Pref}(\mathrm{Fact}(L)) &\quad=\quad \mathrm{Fact}(L) \\
\mathrm{Fact}(L) &\quad=\quad \mathrm{Pref}(\mathrm{Suff}(L))
\end{aligned}
$$

Kleene star

Given a language L, we introduce $L^* = L^0 \cup L_1 \cup L^2 \cup \cdots = \bigcup_{n \geq 0} L^n$, that is, the set of words than we can obtain by concatenating a **finite** number of words (possibly **none**) of L.

In a similar manner, we define $L^+ = \bigcup_{n \geq 1} L^n$ as the set of words than we can obtain by concatenating **at least one** word of L.

Other Operations

Properties of the Kleene star

For any language L, the following properties hold:

- $\varepsilon \in L^*$, even if $\varepsilon \notin L$. We can always pick 0 words of L to concatenate.
- However, $\varepsilon \in L^+$ if and only if $\varepsilon \in L$.
- $\emptyset^* = \{\varepsilon\}$. Again, we pick 0 words to concatenate.
- $\Sigma^* = \Sigma^*$. A word is indeed a finite concatenation of letters, and our initial notation makes sense.

Exercise 4. Are the following properties true in the general case, assuming three generic languages L_1, L_2, L_3?

$$
\begin{array}{rcl}
\{ab\} \cup \{ba\} & = & \{abba\} \\
\{a\}^n & = & \{a^n\} \\
\{a\}^* & = & \{a^n \mid n \geq 0\} \\
\{a, b\}^n & = & \{a^n, b^n\} \\
\{a, b\}^3 & = & \{aaa, aab, aba, abb, baa, bab, bba, bbb\} \\
(L_1 \cup L_2)^2 & = & L_1^2 \cup L_1 L_2 \cup L_2 L_1 \cup L_2^2 \\
L_1 \cdot (L_2 \cup L_3) & = & (L_1 \cdot L_2) \cup (L_1 \cdot L_3)
\end{array}
$$

Answer

$$
\begin{aligned}
\{ab\} \cup \{ba\} &\neq \{abba\} \\
\{a\}^n &= \{a^n\} \\
\{a\}^* &= \{a^n \mid n \geq 0\} \\
\{a, b\}^n &\neq \{a^n, b^n\} \\
\{a, b\}^3 &= \{aaa, aab, aba, abb, baa, bab, bba, bbb\} \\
(L_1 \cup L_2)^2 &= L_1^2 \cup L_1 L_2 \cup L_2 L_1 \cup L_2^2 \\
L_1 \cdot (L_2 \cup L_3) &= (L_1 \cdot L_2) \cup (L_1 \cdot L_3)
\end{aligned}
$$

See you next class!