∫(͡° ͜ʖ ͡°)dx

CS 719

Topics in Mathematical Foundations of Formal Verifications

Notes By: Aryaman Maithani

Spring 2020-21

Lecture 1 (11-01-2021)

11 January 2021 09:34

Recap of Regular Languages

Different formalisms surprisingly describe the same class of lang- regular languages. Regular expressions, DFA, NFA, MSO logic.

Notation and Setup (for the rest of course)

Fix a finite alphabet Σ.

A (finite) word over Σ is a finite sequence $a_0 a_1 \cdots a_n$ of elements of Σ. $u, v, w \ldots$ are used for words.

$$w = a_0 a_1 \ldots a_n \quad \text{where each } a_i \in \Sigma.$$

The empty sequence corresponds to the unique word of length 0 and is denoted by ϵ, the empty word.

$\Sigma^* =$ the set of all finite words over Σ. $(\epsilon \in \Sigma^*)$

$\Sigma^+ = \Sigma^* \setminus \{\epsilon\} =$ the set of all non-empty words over Σ.

Concatenation of words:

$$\cdot : \Sigma^* \times \Sigma^* \longrightarrow \Sigma^*$$
$$(u, v) \longmapsto u \cdot v$$

defined in the usual manner.

→ The $\cdot$ operation on Σ^* is associative.

That is, $\forall u, v, w \in \Sigma^* : (u \cdot v) \cdot w = u \cdot (v \cdot w)$.

→ ϵ acts as an (the) identity for $\cdot$

This is an example of a monoid $(X, *)$

$\forall w \in \Sigma^* : \quad \epsilon \cdot w = w \cdot \epsilon = w.$

(Another example of monoid: $(\mathbb{N}, +)$
$\mathbb{N} = \{0, 1, \ldots\}$ in this course)

(Later we'll look a finite monoids.)

$$\ell : \Sigma^* \longrightarrow \mathbb{N}$$
$$u \longmapsto \text{length of } u : \ell(u)$$

Note $\ell(u \cdot w) = \ell(u) + \ell(w)$
$\ell(\varepsilon) = \ell(0)$

Thus, ℓ is a monoid morphism.

Defn. A language L is simply a subset of Σ^*. (Language)

Given languages $L_1, L_2 \subset \Sigma^*$, we define

$$L_1 \cdot L_2 = \{ w_1 \cdot w_2 \mid w_1 \in L_1, w_2 \in L_2 \}.$$

REGULAR EXPRESSIONS (Regular expressions)

$$r \equiv \phi \mid \epsilon \mid \underset{\Sigma}{\underset{\cap}{a}} \mid r_1 + r_2 \mid r_1 \cdot r_2 \mid r^*$$

$r \rightsquigarrow L(r)$ language associated to r

$L(r)$ is defined by structural induction on r.

- $L(\phi) = \phi$
- $L(\epsilon) = \{\epsilon\}$
- $L(a) = \{a\} \quad (a \in \Sigma)$
- $L(r_1 + r_2) = L(r_1) \cup L(r_2)$
- $L(r_1 \cdot r_2) = L(r_1) \cdot L(r_2)$ (RHS defined earlier)

- $L(r^*) = \{\varepsilon\} \cup L(r) \cup L(r)\cdot L(r) \cup L(r)\cdot L(r)\cdot L(r) \cup \cdots$

$$= \bigcup_{i=0}^{\infty} L^i \qquad \left(L^0 = \{\varepsilon\},\ L^1 = L(r),\ L^{i+1} = L^i \cdot L \ \ {}_{i \geq 1} \right)$$

[Example. $(ab)^* = \{\varepsilon, ab, abab, \ldots\}$.]

Def^n. A language $L \subseteq \Sigma^*$ is said to be regular if there exists a regular expression r such that $L(r) = L$.

(Regular language)

Thm. Regular languages are closed under union, intersection, complementation, concatenation.

(As per our def^n using regular expressions, union & concatenation are obvious.)

Some of the above is easier to prove under diff. formalisms. One first shows that two diff. formalisms are actually same.

Def^n. (Extended reg. expressions)

(Extended regular expressions)

$$r \equiv \phi \mid \epsilon \mid \underset{\substack{\Uparrow \\ \Sigma}}{a} \mid r_1 + r_2 \mid r_1 \cap r_2 \mid \neg r \mid r_1 \cdot r_2 \mid r^*$$

these we can add, in view of th^m, w/o changing the class of languages

Q: What subclass of language will we get if we restrict ourselves to a subset of the operators?

Def^n. (Star-free extended reg. expressions) Exclude the $*$ operator.

(Star-free regular expressions)

Def^n. (Star-free reg. expressions) Exclude the $*$ operator.

(Star-free regular expressions)

Q. Which regular languages admit a star free representation?

(Non?) Example : $r = (ab)^*$

Can we rewrite this without $*$?

The "extended" is important. Else, we get trivial classes.

Lecture 2 (14-01-2021)

14 January 2021 11:35

Note that $\neg \phi = \Sigma^*$ ← can use this freely

Observe: $a^* = \neg(\underbrace{\Sigma^* \cdot b \cdot \Sigma^*}_{\text{words containing at least } b})$

Similarly $(ab)^* \longrightarrow$ words starting with a, ending with b, no consecutive a or b (or ϵ)

$$(ab)^* = \epsilon + [a \Sigma^* b \cap \neg(\Sigma^* aa \Sigma^* + \Sigma^* bb \Sigma^*)]$$

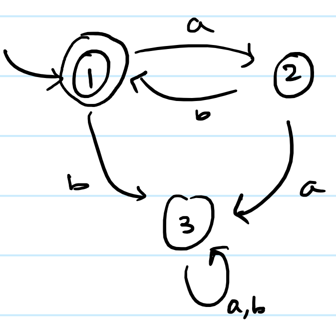
It is not even clear a priori whether the question "Which languages have $*$-free expression" is even decidable.

Finite state Automata (Finite state automata)

(NFA)

$$A = (\underbrace{Q}_{\text{finite set}}, \Sigma, \underbrace{Q_0 \subseteq Q}_{\text{initial states}}, \underbrace{\Delta \subseteq Q \times \Sigma \times Q}_{\substack{\text{transition} \\ (q, a, q') \in \Delta}}, \underbrace{F \subseteq Q}_{\text{final states}})$$

EXAMPLES ①



$Q = \{1, 2, 3\}$

$Q_0 = F = \{1\}$

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

Language accepted: $(ab)^*$

Def^n Suppose $w = a_0 \cdots a_n \in \Sigma^*$.

A **run** ρ of A on w is a sequence of states

$$\rho = q_0, \ldots, q_{n+1}$$

note $n+1$

such that
- $q_0 \in Q_0$
- $(q_i, a_i, q_{i+1}) \in \Delta \quad \forall i = 0, \ldots, n$

The run ρ is **accepting** if $q_{n+1} \in F$.

(Note that a word may have no run or even multiple runs.)

The **language** $L(A)$ of A is defined as

$$L(A) = \{ w \in \Sigma^* : A \text{ has at least one accepting run} \}.$$

A is **deterministic** if $|Q_0| = 1$ and

$$\forall q \in Q, \forall a \in \Sigma, \underbrace{\exists!}_{\text{there exists unique}} q' \in Q \text{ s.t. } (q, a, q') \in \Delta.$$

In other words, $\Delta \subseteq (Q \times \Sigma) \times Q$ is a function $Q \times \Sigma \to Q$.

The example above was actually deterministic. It is called a DFA.

Thm. [TOC] (Kleene's Theorem)

Regular expressions $\equiv$ NFA $\equiv$ DFA.

(That is, all three formalisms talk about the same class of language — regular languages.)

(Recap of Proof.) Reg. Exp $\subseteq$ NFA

$r \longmapsto A_r$

reg exp; NFA

$L(r) = L(A_r)$.

The way to do this is by induction.

- For ε and 'a', easy.
- $r = r_1 + r_2$. We have NFAs for r_1 and r_2.
  Then, the NFA $A_{r_1} \sqcup A_{r_2}$ works.

  Allowed since non-determinism →

- $r_1 \cdot r_2$. Use ε-transitions. Idea is to take union and put ε transitions from F of A_{r_1} to Q_0 of A_{r_2}. The final states are now F of A_{r_2} and initial is Q_0 of A_{r_1}.

- r^*. Same sort of idea as above but loop on self.

NFA $\subseteq$ Reg. Exp.

$Q = \{1, \ldots, n\}$

r_{ij} = a reg. exp. which captures the words which allow to go from i to j.

Then $r := \bigcup_{\substack{i \in Q_0 \\ j \in F}} r_{ij}$ works.

Thus, only need to figure out r_{ij}.

"Dynamic Programming"

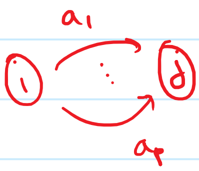
Introduce a third parameter k.

r_{ij}^{k} = reg. expression words w which have a

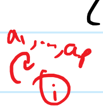
run ρ of A s.t.
(i) ρ starts at i
(ii) ρ end at j
(iii) all intermediates states of ρ are in $\{1, \ldots, k\}$.
(include $k = 0$)

Start building r_{ij}^k going from $k = 0$ to $k = n$.
Note that $r_{ij} = r_{ij}^n$. ✓

$r_{ij}^0 \equiv$ start at i, end at j, no intermediate state
$\equiv$ all those letters which allow transition from i to j.
(if any)



$$r_{ij} = a_1 + a_2 + \cdots + a_p \qquad (i \neq j)$$
$$= a_1 + \cdots + a_p + \varepsilon \qquad (i = j)$$



$$r_{ij}^k = r_{ij}^{k-1} + r_{ik}^{k-1} \cdot \left(r_{kk}^{k-1}\right)^* \cdot r_{kj}^{k-1}$$

(We build for lower k first for all (i, j))

Thus, Reg Exp $\equiv$ NFA.

NFA $\equiv$ DFA. DFA $\subseteq$ NFA is obvious.
Converse:

$$A = (Q, \Sigma, Q_0, \Delta, F).$$

The idea to get an equivalent DFA is the powerset construction.

$$B = (2^Q, \Sigma, \{Q_0\}, \delta : 2^Q \times \Sigma \to 2^Q, F')$$

Idea is to keep track of all the states that you can reach from given state.

$$\delta(X, a) = \{q \in Q : \exists q' \in X, \quad q' \xrightarrow{a} q\}.$$