∫(͡° ͜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.

$w = a_0 a_1 \ldots a_n$ 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.

Σ^* = 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 $(X, \star)$

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

(Another example of monoid: $(\mathbb{N}, +)$
$\mathbb{N} = \{0, 1, \dots\}$ 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.

Def^n. 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)

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

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

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

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

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

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

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

Def

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)

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

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 $\overset{\text{extended}}{\wedge}$ reg. expressions) Exclude the $*$ operator.

(Star-free regular expressions)

Defⁿ. (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)

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

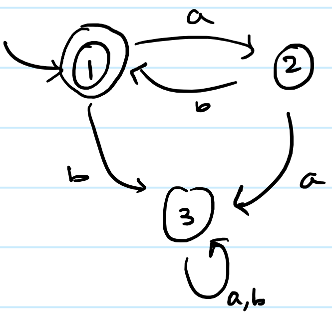
It is not even clear a priori whether the question "Which languages have $*$-free expression" is even decidable.

Finite state Automata (Finite state automata)

(NFA)

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

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

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

note n+1

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

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

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

there exists unique

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

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

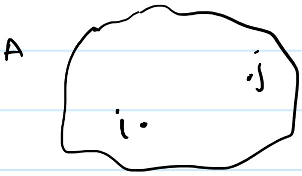
The way to do this is by induction.

- 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 ⊆ Reg. Exp.



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

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.

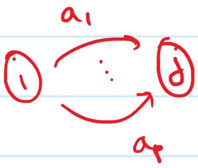
$r_{ij}^k \equiv$ reg. expression words w which have a

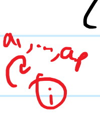
run ρ of A s.t.
(i) ρ starts at i
(ii) ρ ends at j
(iii) all intermediates states of ρ are in $\{1, \dots, k\}$.
(include $k = 0$)

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

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

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





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

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

Thus, Reg Exp ≡ NFA.

NFA ≡ DFA. DFA ⊆ 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.

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

Lecture 3 (18-01-2021)

18 January 2021 09:04

Today, we see another formalism to describe regular languages.
A natural way to describe a language is to give a "property" of words.

Examples:

1) Every (occurence of an) 'a' is eventually followed by a 'b'.
   aabaab ✓ bababac ×
2) There is exactly one 'a' in the word.
3) The first position is labelled 'a'.
4) There are even number of 'a's.

We need a formal language to do so.

Formal Language : Should allow us to do "Boolean" properties like "and", "or", et cetera.

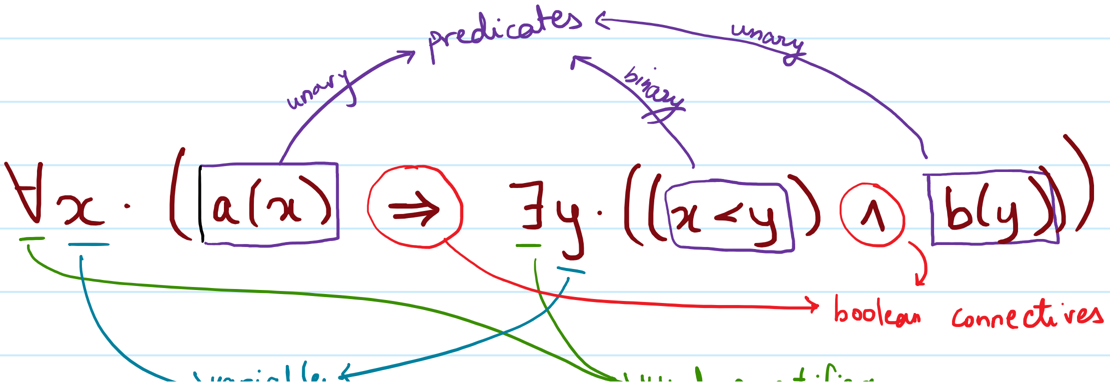
Going to use a Mathematical Logic for doing so.

First-Order Logic (over words)

(First Order Logic)

Before formal defn & syntax:

An example of a formula in this logic:



$$\forall x \cdot (\boxed{a(x)} \Rightarrow \exists y \cdot ((\boxed{x<y}) \wedge \boxed{b(y)}))$$

variables going to range over positions in the word

usual quantifiers

boolean connectives

FO $[\Sigma]$ - variables :- $x, y, z, \dots$ | range over
$x_0, x_1, x_2, \dots$ | positions

predicates :- • letter predicates
$a \in \Sigma,\ a(x)$ says the letter 'a' at position 'x' (true/false)
• binary predicates, $y < z$
• equality $x = y$

$$\varphi \equiv a(x) \mid x < y \mid x = y \mid \varphi \vee \varphi \mid \neg \varphi \mid \exists x \cdot \varphi$$

↳ can get $\varphi \wedge \varphi,\ \varphi \Rightarrow \varphi,\ \varphi \Leftrightarrow \varphi,\ \forall x \cdot \varphi$ using these

The above was a sentence, there was no free variable.

$$\text{first}(x) \equiv \forall y \cdot [(x = y) \vee (x < y)]$$ ← here x is free

Given this formula, if we wish to find truth of first(x) on some word w, we need to give x.

$w = \overset{1}{a}\overset{2}{b}\overset{3}{a}\overset{4}{a}\overset{5}{b}$

$w,\ x \leftarrow 2 \overset{?}{\models} \text{first}(x)$?

if true, we write: $w,\ x \leftarrow 2 \models \text{first}(x)$

else : $w,\ x \leftarrow 2 \not\models \text{first}(x)$

Easy to see $w,\ x \leftarrow 2 \not\models \text{first}(x)$. Why?

We need to check if for all positions 'p' in w:

$$w,\ x \leftarrow 2,\ y \leftarrow p \models (x = y) \vee (x < y)$$

If $P = 4$, then we check $$\underbrace{\underbrace{(2=4)}_{\text{false}} \vee \underbrace{(2<4)}_{\text{true}}}_{\text{true!}}$$

However, if $P = 1$, then $$\underbrace{\underbrace{(2=1)}_{\text{false}} \vee \underbrace{(2<1)}_{\text{false}}}_{\text{false}}$$

Thus, $w, x \leftarrow 2 \not\models \text{first}(x)$. (We had the universal quantifier.)

Easy to see $\text{first}(x)$ is true iff x is the first position.
Now, we can use $\text{first}(x)$.

Defn. A sentence is a formula without free variables.

(Sentence)

Example

- $\varphi \equiv \exists x \cdot [\text{first}(x) \wedge b(x)]$ is a sentence.
  Now makes sense to ask "$abaab \models \varphi$" without any assignment. $(abaab \not\models \varphi)$
  → the first position is labelled 'b'

- Exactly one 'a':
  $$\{\forall x\, \forall y\, [(a(x) \wedge a(y)) \Rightarrow x = y]\} \wedge \{\exists x \cdot a(x)\}$$

- $(ab)^*$ ← can you write a first order sentence which gives this regex?
  $$\{\forall x\, [\text{first}(x) \Rightarrow a(x)]\} \wedge \{\forall x \cdot [a(x) \Rightarrow \exists y \cdot (s(x,y) \wedge b(y))]\}$$
  $s(x,y) \equiv (x<y) \wedge \forall z\, (z<x \vee y<z)$.

- There are even numbers of 'a's → is regular, can come up with an automata

The other three examples were also regular. (Also expressible by FOL)

However, FOL cannot describe this logic!

But every language definable by FOL WILL be regular!

FO → FO- definable languages

REG → collection of reg. languages

$\boxed{FO \subsetneq REG}$.

We shall extend FO to MSO → Monadic Second Order (Logic).