Natural Language Processing - 2nd Semester (2024-2025)
1038141

1.4 - Finite State Automata and Regular Expressions

SAPIENZA
UNIVERSITÀ DI ROMA

Prof. Stefano Faralli
faralli@di.uniroma1.it

Prof. Iacopo Masi
masi@di.uniroma1.it

**credits are reported in the last slide

SAPIENZA
UNIVERSITÀ DI ROMA

4 - Finite State Automata and Regular Expressions

- finite state automata
- the sheep language
- state-transition table
- recognition
- acceptors vs. generators
- languages, regular expressions, finite state automata
- exercise
- Q&A

Milestones in NLP

rule-based systems

statistical and classical machine learning models

deep learning models

Finite State Automata

Definition

A finite state automata is a 5-tuple (**Q**, **Σ**, **δ**, $\mathbf{q_0}$, **A**) where:

- **Q** is a finite set of states.
- **Σ** is a finite set of symbols called the alphabet.
- **δ:QxΣ →Q**, is the transition function. The inputs to this function are the current state and the last input symbol. The function value **δ(q,x)** is the state the automaton goes to from state **q** after reading the symbol **x**.
- $\mathbf{q_0} \in \mathbf{Q}$, is the start state.
- **A** is the set of accepting states.

Running Example

«draw a sheep attending the natural language processing course at Sapienza University of Rome»

Created with DALL· E 3 technology.

Finite State Automata

Example

Let's look at the sheep language:

L_{Sheep}={
baa!,
baaa!,
baaaa!,
baaaaa!,
baaaaaa!,
…}

Finite State Automata

Example

Let's look at the sheep language:

L_{Sheep}={
baa!,
baaa!,
baaaa!,
baaaaa!,
baaaaaa!,
…}

Regular expression is:

/baa+!/

Finite State Automata

Example

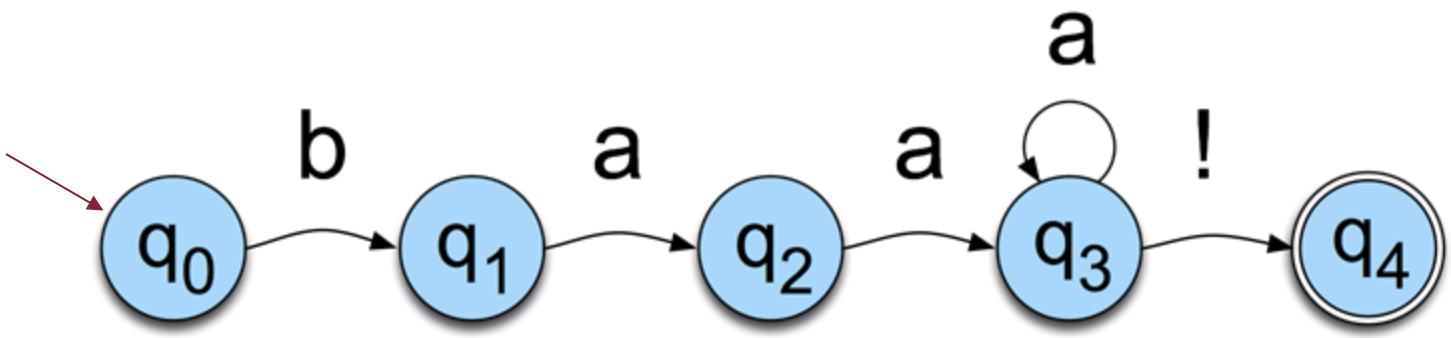
Let's look at the sheep language:

Regular expression is:

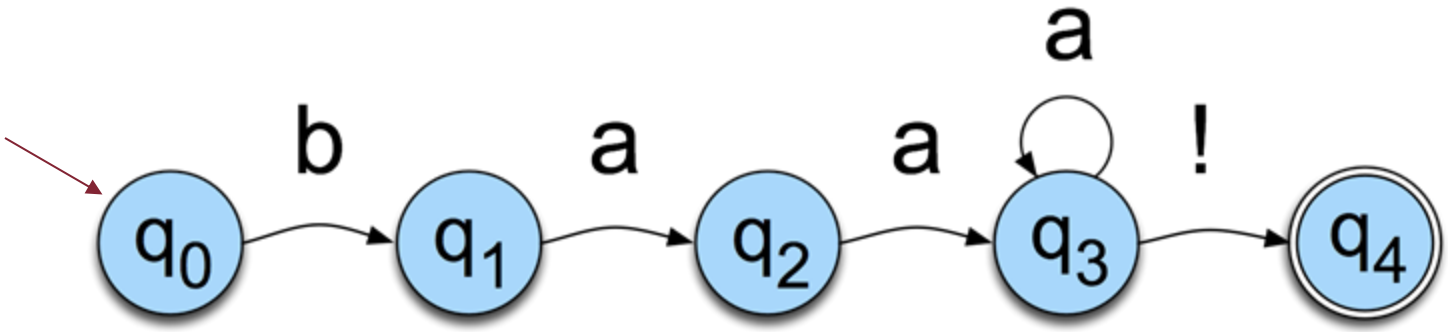
/baa+!/

L_{Sheep}={
baa!,
baaa!,
baaaa!,
baaaaa!,
baaaaaa!,
...}

The Finite State Automaton recognizing the sheep language :



Finite State Automata



RE

/baa+!/

$FA_{sheep}=(Q, \Sigma, \delta, q_0, A)$

- Q is a finite set of states. $Q=\{q_0, q_1, q_2, q_3, q_4\}$
- Σ is a finite set of symbols called the alphabet. $\Sigma=\{a, b, !\}$
- $\delta: Q \times \Sigma \rightarrow Q$, is the transition function.

  $\delta(q_0, b) = q_1$

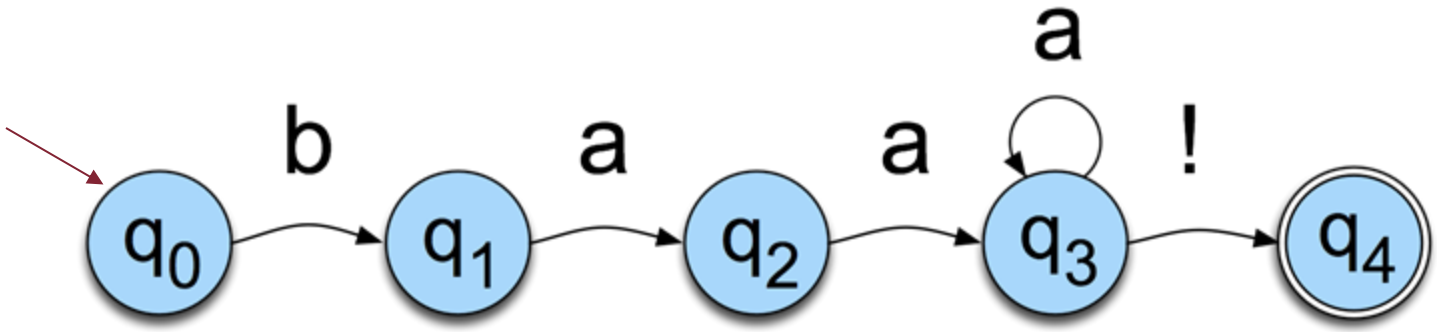
  $\delta(q_1, a) = q_2$

  $\delta(q_2, a) = q_3$

  $\delta(q_3, a) = q_3$

  $\delta(q_3, !) = q_4$
- $q_0 \in Q$, is the start state.
- A is the set of accepting states. $A=\{q_4\}$

Finite State Automata: state-transition table



RE

/baa+!/

$FA_{sheep}=(Q, \Sigma, \delta, q_0, A)$

- Q is a finite set of states. $Q=\{q_0, q_1, q_2, q_3, q_4\}$
- Σ is a finite set of symbols called the alphabet. $\Sigma=\{a, b, !\}$
- $\delta: Q \times \Sigma \rightarrow Q$, is the transition function.
  $\delta(q_0, b) = q_1$
  $\delta(q_1, a) = q_2$
  $\delta(q_2, a) = q_3$
  $\delta(q_3, a) = q_3$
  $\delta(q_3, !) = q_4$
- $q_0 \in Q$, is the start state.
- A is the set of accepting states. $A=\{q_4\}$

	Input		
State	b	a	!
0	1	Ø	Ø
1	Ø	2	Ø
2	Ø	3	Ø
3	Ø	3	4
4:	Ø	Ø	Ø

Finite State Automata: recognition

- Recognition is the process of determining if a string should be accepted by a machine
- Or… it's the process of determining if a string is in the language we're defining with the machine
- Or… it's the process of determining if a regular expression matches a string
- Those all amount the same thing in the end

Finite State Automata: recognition

The machine starts in the start state and iterates the following process:

- Check the next letter of the input
- Consult the state-transition table: if it matches the symbol on an arc leaving the current state, then cross that arc, move to the next state, and also advance one symbol in the tape pointer
- If we are in the accepting state when we run out of input, the machine has successfully **recognized** an instance of the input
- If the machine never gets to the final state, either because it runs out of input, or it gets some input that doesn't match an arc or if it just happens to get stuck in some non-final state, we say the machine **rejects** or **fails to accept** an input

Acceptors vs. Generators

- *Formal Languages* are sets of strings composed of symbols from a finite set (i.e, an alphabet). (e.g., the L_{sheep})
- Finite-state automata define formal languages (without having to enumerate all the strings in the language)
- The term *Generative* is based on the view that you can run the machine as a generator to get strings from the language.
- FSAs can be viewed from two perspectives:
  - **Acceptors** that can tell you if a string is in the language
  - **Generators** to produce *all and only* the strings in the language

Finite State Automata

- Any regular expression can be implemented as a finite-state automaton.

- Symmetrically, any finite-state automaton can be described with a regular expression.

- A regular expression is one way of characterizing a particular kind of formal language called a regular language. Both regular expressions and finite-state automata can be used to describe regular languages.

Languages, Regular Expressions and Finite State Automata

definition

let M be an FSA, the set L(M)= { x | when M is run on the string x starting from the start state, the final state is in A} is called the language decided by M.

definition

let L_1 and L_2 be languages. The union of L_1 and L_2 is the set:

$$L_1 \cup L_2 = \{x \mid x \in L_1 \text{ or } x \in L_2\}$$

definition

let L_1 and L_2 be languages. The concatenation of L_1 and L_2 is the set

$$L_1 L_2 = \{xy \mid x \in L_1 \text{ and } y \in L_2\}$$

definition

Let L be a language, L* is defined as the set:

$$L^* = \bigcup_{k=0}^{\infty} L^k$$

and is called the Kleene closure of L.

Languages, Regular Expressions and Finite State Automata

if we observe the left parts of the following equivalences:

$L_1 \cup L_2 = \{x \mid x \in L_1 \text{ or } x \in L_2\}$

$L_1 L_2 = \{xy \mid x \in L_1 \text{ and } y \in L_2\}$

$$L^* = \bigcup_{k=0}^{\infty} L^k$$

we can see a parallelism with the basic regular expression operators.

let RE_{L1} be the regular expression for L_1 and RE_{L2} be the regular expression for L_2:

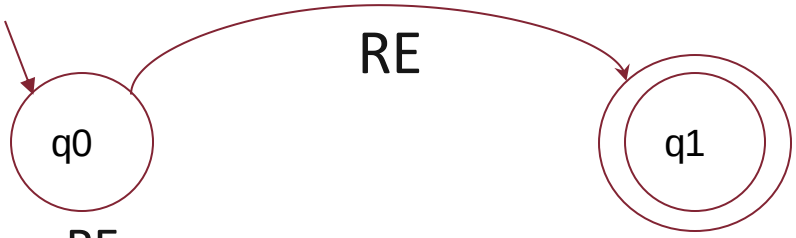
$RE_{L1 \cup L2} = (RE_{L1}) \mid (RE_{L2})$

$RE_{L1L2} = (RE_{L1})(RE_{L2})$

$RE_{L1^*} = (RE_{L1})^*$

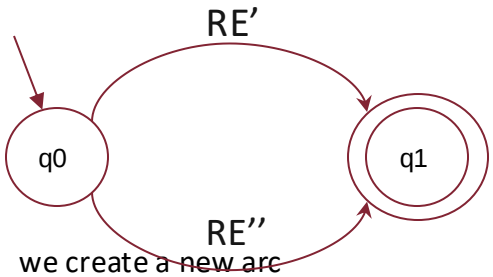
Languages, Regular Expressions and Finite State Automata

Let RE be a regular expression we start creating a finite state automata:



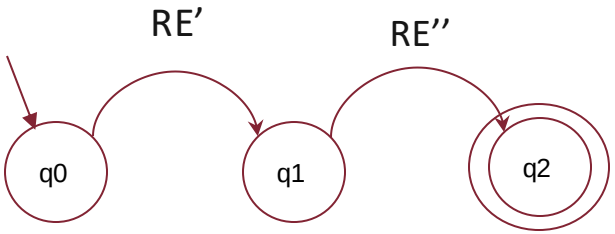
we iteratively decompose RE:

if RE = RE'|RE''



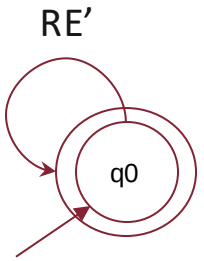
we create a new arc

if RE = RE'*



we create a new arc,
a new final state q2

if RE = RE'RE''



we remove q1 and
q0 is now final

Exercise:

given L_1 = {nlp, nat_lang_proc} and L_2 = { _is_cool} and $L_3 = L_1L_2^*$

- provide 4 examples of string x $\in L_3$
- write the regular expressions for L_1, L_2 and for L_3
- define the FSA for L_1, L_2 and for L_3

online tools:

- https://cyberzhg.github.io/toolbox/min_dfa?regex=YmxhYSsh

The notebook:

https://github.com/iacopomasi/NLP/blob/main/notebooks/Part_1_4_REs_and_finite_state_automata.ipynb

Q&A

**Credits

The slides of this part of the course are the result of a personal reworking of the slides and of the course material from different sources:

1. The NLP course of Prof. Roberto Navigli, Sapienza University of Rome
2. The NLP course of Prof. Simone Paolo Ponzetto, University of Mannheim, Germany
3. The NLP course of Prof. Chris Biemann, University of Hamburg, Germany
4. The NLP course of Prof. Dan Jurafsky, Stanford University, USA

Highly readable font Biancoenero© by biancoenero edizioni srl, designed by Umberto Mischi, with the consultancy of Alessandra Finzi, Daniele Zanoni and Luciano Perondi (Patent no. RM2011O000128).

Available free of charge for all institutions and individuals who use it for non-commercial purposes.