

FSA and regular languages

Data Structures and Algorithms for Computational Linguistics III
(ISCL-BA-07)

Çağrı Çöltekin
ccoltekin@ufa.uni-tuebingen.de

University of Tübingen
Seminar für Sprachwissenschaft

Winter Semester 2021/22

www.iscl.uni-tuebingen.de

Languages and automata

- Recognizing strings from a language defined by a grammar is a fundamental question in computer science
- The efficiency of computation, and required properties of computing device depends on the grammar (and the language)
- A well-known hierarchy of grammars both in computer science and linguistics is the *Chomsky hierarchy*
- Each grammar in the Chomsky hierarchy corresponds to an abstract computing device (an automaton)
- The class of *regular grammars* are the class that corresponds to *finite state automata*

C. Çöltekin, MB | University of Tübingen

Winter Semester 2021/22 1 / 22

Languages and automata Regular expressions Operations on FSA

How to describe a language?

Formal grammars

A formal grammar is a finite specification of a (formal) language.

- Since we consider languages as sets of strings, for a finite language, we can (conceivably) list all strings
- How to define an infinite language?
- Is the definition {ba, ba², ba³, ba⁴, ...} 'formal enough'?
- Using regular expressions, we can define it as *ba**
- But we will introduce a more general method for defining languages

C. Çöltekin, MB | University of Tübingen

Winter Semester 2021/22 2 / 22

Languages and automata Regular expressions Operations on FSA

Phrase structure grammars

- A phrase structure grammar is a generative device
- If a given string can be generated by the grammar, the string is in the language
- The grammar generates *all* and the *only* strings that are valid in the language
- A phrase structure grammar has the following components
 - Σ A set of terminal symbols
 - N A set of non-terminal symbols
 - $S \in N$ A special non-terminal, called the start symbol
 - R A set of rewrite rules or production rules of the form:

$$\alpha \rightarrow \beta$$

which means that the sequence α can be rewritten as β (both α and β are sequences of terminal and non-terminal symbols)

C. Çöltekin, MB | University of Tübingen

Winter Semester 2021/22 3 / 22

Languages and automata Regular expressions Operations on FSA

Chomsky hierarchy and automata

Grammar class	Rules	Automata
Unrestricted grammars	$\alpha \rightarrow \beta$	Turing machines
Context-sensitive grammars	$\alpha A \beta \rightarrow \alpha \gamma \beta$	Linear-bounded automata
Context-free grammars	$A \rightarrow \alpha$	Pushdown automata
Regular grammars	$A \rightarrow a$ $A \rightarrow aB$	Finite state automata

C. Çöltekin, MB | University of Tübingen

Winter Semester 2021/22 4 / 22

Languages and automata Regular expressions Operations on FSA

Regular grammars: definition

A regular grammar is a tuple $G = (\Sigma, N, S, R)$ where

Σ is an alphabet of terminal symbols

N is a set of non-terminal symbols

S is a special 'start' symbol $\in N$

R is a set of rewrite rules following one of the following patterns ($A, B \in N, a \in \Sigma, \epsilon$ is the empty string)

Left regular

- $A \rightarrow a$
- $A \rightarrow Ba$
- $A \rightarrow \epsilon$

Right regular

- $A \rightarrow a$
- $A \rightarrow aB$
- $A \rightarrow \epsilon$

C. Çöltekin, MB | University of Tübingen

Winter Semester 2021/22 5 / 22

Languages and automata Regular expressions Operations on FSA

Regular languages: some properties/operations

$L_1 L_2$ Concatenation of two languages L_1 and L_2 : any sentence of L_1 followed by any sentence of L_2

L^* Kleene star of L : L concatenated with itself 0 or more times

L^R Reverse of L : reverse of any string in L

\bar{L} Complement of L : all strings in Σ^* except the ones in L ($\Sigma^* - L$)

$L_1 \cup L_2$ Union of languages L_1 and L_2 : strings that are in any of the languages

$L_1 \cap L_2$ Intersection of languages L_1 and L_2 : strings that are in both languages

Regular languages are closed under all of these operations.

C. Çöltekin, MB | University of Tübingen

Winter Semester 2021/22 6 / 22

Languages and automata Regular expressions Operations on FSA

Three ways to define a regular language

- A language is regular if there is regular grammar that generates/recognizes it
- A language is regular if there is an FSA that generates/recognizes it
- A language is regular if we can define a regular expressions for the language

C. Çöltekin, MB | University of Tübingen

Winter Semester 2021/22 7 / 22

Languages and automata Regular expressions Operations on FSA

Regular expressions

- Every regular language (RL) can be expressed by a regular expression (RE), and every RE defines a RL
- A RE a defines a RL $L(a)$
- Relations between RE and RL

$$\begin{aligned} L(\emptyset) &= \emptyset, \\ L(a) &= a, \\ L(a^*) &= a^+, \\ L(a|b) &= L(a) \cup L(b), \\ L(a^*b) &= L(a)^+ L(b), \\ L(a^*b^*) &= L(a)^+ L(b)^+ \end{aligned}$$

$$\begin{aligned} L(a|b) &= L(a) \cup L(b) \\ &\text{(some author use the notation } a+b, \\ &\text{we will use } a|b \text{ as in many practical} \\ &\text{implementations)} \end{aligned}$$

where, $a, b \in \Sigma$, ϵ is empty string, \emptyset is the language that accepts nothing (e.g., $\Sigma^* - \Sigma^*$)

- Note: no standard complement and intersection in RE

C. Çöltekin, MB | University of Tübingen

Winter Semester 2021/22 8 / 22

Languages and automata Regular expressions Operations on FSA

Regular expressions and some extensions

- Kleene star (a^*), concatenation (ab) and union ($a|b$) are the basic operations
- Parentheses can be used to group the sub-expressions. Otherwise, the priority of the operators are as listed above: $a|bc^* = a|(b(c^*))$
- In practice some short-hand notations are common

$$\begin{aligned} \cdot &= (a_1 | \dots | a_n), & \neg &= [a^*c] = \cdot - (a|b|c) \\ \text{for } \Sigma &= \{a_1, \dots, a_n\}, & \forall d &= (0|1|\dots|8|9) \\ a^* &= a^{**}, & & \\ \neg[a^*c] &= (a|b|c) & - & \end{aligned}$$

- And some non-regular extensions, like $(a^*)^*b|1$ (sometimes the term *xyzzy* is used for expressions with non-regular extensions)

C. Çöltekin, MB | University of Tübingen

Winter Semester 2021/22 9 / 22

Languages and automata Regular expressions Operations on FSA

Some properties of regular expressions

Useful identities for simplifying regular expressions

- $u|(v|w) = (u|v)|w$
- $u|v = v|u$
- $u|(v|w) = uv|vw$
- $u|(\emptyset) = u$
- $u|v = cu = u$
- $\emptyset|u = \emptyset$
- $u|(v|w) = (uv)|w$
- $\emptyset^* = \epsilon$
- $\epsilon^* = \epsilon$
- $(u^*)^* = u^*$
- $u|u = u$
- $(u|v)^* = (u^*|v^*)^*$
- $u^*|c = u^*$

An exercise

$$\begin{aligned} \text{Simplify } a|ab^* \\ a|ab^* &= a|ab^* \\ &= a(c|b)^* \\ &= ab^* \end{aligned}$$

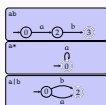
Note: some of these are direct statements of Kleene algebra, others can be derived from them.

C. Çöltekin, MB | University of Tübingen

Winter Semester 2021/22 10 / 22

Languages and automata Regular expressions Operations on FSA

Converting regular expressions to FSA



- For more complex expressions, one can replace the paths for individual symbols with corresponding automata
- Using ϵ transitions may ease the task
- The reverse conversion (from automata to regular expressions) is also easy:
 - identify the patterns on the left, collapse paths to single transitions with regular expressions

C. Çöltekin, MB | University of Tübingen

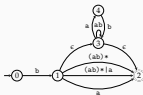
Winter Semester 2021/22 11 / 22

Exercise

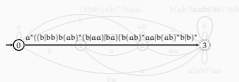
convert $b^*(ab)^*a$ to an NFA

Exercise

convert $b^*(ab)^*a$ to an NFA



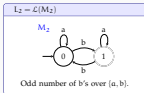
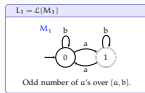
Converting FSA to regular expressions



- The general idea: remove (intermediate) states, replacing edge labels with regular expressions
- An exercise: simplify the resulting regular expressions

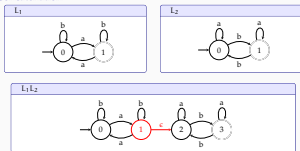
Two example FSA

what languages do they accept?

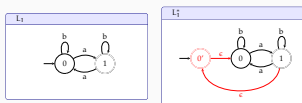


We will use these languages and automata for demonstration.

Concatenation

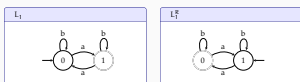


Kleene star



- What if there were more than one accepting states?

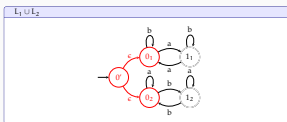
Reversal



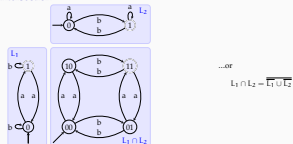
Complement



Union



Intersection



Closure properties of regular languages

- Since results of all the operations we studied are FSA: Regular languages are closed under
 - Concatenation
 - Kleene star
 - Reversal
 - Complement
 - Union
 - Intersection

Wrapping up

- FSA and regular expressions express regular languages
- Regular languages and FSA are closed under
 - Concatenation
 - Kleene star
 - Complement
 - Reversal
 - Union
 - Intersection
- To prove a language is regular, it is sufficient to find a regular expression or FSA for it
- To prove a language is not regular, we can use pumping lemma (see Appendix)

Next:

- Parsing

Acknowledgments, credits, references

- The classic reference for FSA, regular languages and regular grammars is Hopcroft and Ullman (1979) (there are recent editions).

1. Hopcroft, John E., Rajeev Motwani, and Jeffrey D. Ullman (2007). *Introduction to Automata Theory, Languages, and Computation*. 3rd. Pearson/Addison Wesley. isbn: 9780321462251.

2. Hopcroft, John E. and Jeffrey D. Ullman (1979). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Series in Computer Science and Information Processing. Addison-Wesley. isbn: 9780201029888.

Another exercise on intersection

Construct the intersection of the automata below (adapted from Hopcroft, Motwani, and Ullman (2007), Fig. 4.4)



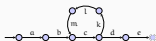
Is a language regular?

— or not

- To show that a language is regular, it is sufficient to find an FSA that recognizes it.
- Showing that a language is not regular is more involved
- We will study a method based on *pumping lemma*

Pumping lemma

intuition



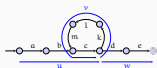
- What is the length of longest string generated by this FSA?
- Any FSA generating an infinite language has to have a loop (application of recursive rule(s) in the grammar)
- Part of every string longer than some number will include repetition of the same substring ('cklm' above)

Pumping lemma

definition

For every regular language L , there exist an integer p such that a string $x \in L$ can be factored as $x = uvw$,

- $uv^i w \in L, \forall i \geq 0$
- $v \neq \epsilon$
- $|uv| \leq p$



How to use pumping lemma

- We use pumping lemma to prove that a language is not regular
- Proof is by contradiction:
 - Assume the language is regular
 - Find a string x in the language, for all splits of $x = uvw$, at least one of the pumping lemma conditions does not hold
 - $uv^i w \in L (\forall i \geq 0)$
 - $v \neq \epsilon$
 - $|uv| \leq p$

Pumping lemma example

prove $L = a^n b^n$ is not regular

- Assume L is regular: there must be a p such that, if uvw is in the language
 - $uv^i w \in L (\forall i \geq 0)$
 - $v \neq \epsilon$
 - $|uv| \leq p$
- Pick the string $a^p b^p$
- For the sake of example, assume $p = 5$, $x = aaaaaabbbb$
- Three different ways to split

