

INF2080

2. Regular Expressions and Nonregular languages

Daniel Lupp

Universitetet i Oslo

26th January 2016



Department of
Informatics



University of
Oslo

- Deterministic finite automata (DFA)
- Regular languages are those languages accepted by DFA's
- Nondeterministic automata (NFA)
- DFA \leftrightarrow NFA

Fixing last weeks horrible yet useful analogy for NFA's:

- Think of an NFA as an unnecessarily complicated coffee machine (\rightarrow kaffeautomat!).
- Each state is a container that can hold water, the transitions are tubes connecting these containers.
- In order to get coffee you must type in a code (the input)
- You start by pouring water into the starting container. The tubes are generally closed by gates, yet can open given a specific input. Some tubes, however, are always open and water always flows through them (ϵ transitions).
- For each symbol you input, "gates" corresponding to that input open up in all containers containing water .
- If water reaches any of the coffee filters (accept states), the water can finally drip through!
- you get a cup of coffee \leftrightarrow your input has been accepted!.

Regular Expressions

Definition (Regular Expression)

Given an alphabet Σ , a *regular expression* is

- a for some $a \in \Sigma$,
- ε ,
- \emptyset ,
- $(R_1 \cup R_2)$ for regular expressions R_1, R_2 ,
- $(R_1 R_2)$ for regular expressions R_1, R_2 ,
- R_1^* for a regular expression R_1 .

→ Regular expressions represent languages!

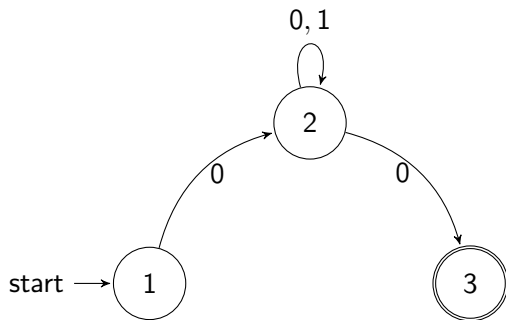
Regular Expressions - Examples

What languages do the following regular expressions (RE) represent?

- 0^*
- 10^*1
- $(1(0 \cup 1)^*1) \cup (0(0 \cup 1)^*0) \cup 0 \cup 1$

What is the connection between RE and DFA/NFA?

Language $0(0 \cup 1)^*0$:



What is the connection between RE and DFA/NFA?

- Can all RE be represented using DFA/NFA?
- Can all DFA/NFA be described by RE?

Yes!

Regular Expressions and Automata

Proposition

Every language described by an RE is regular.

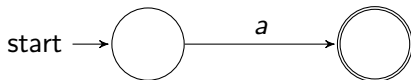
Proof based on inductive definition of RE!

Definition (Regular Expression)

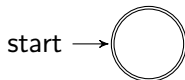
Given an alphabet Σ , a *regular expression* is

- a for some $a \in \Sigma$,
- ε ,
- \emptyset ,
- $(R_1 \cup R_2)$ for regular expressions R_1, R_2 ,
- $(R_1 R_2)$ for regular expressions R_1, R_2 ,
- R_1^* for a regular expression R_1 .

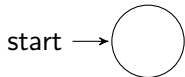
if $R = a$ for $a \in \Sigma$, then $L(R) = \{a\}$ is accepted by



If $R = \varepsilon$, then $L(R) = \{\varepsilon\}$ is accepted by



If $R = \emptyset$, then $L(R) = \emptyset$ is accepted by



Definition (Regular Expression)

Given an alphabet Σ , a *regular expression* is

- a for some $a \in \Sigma$,
- ε ,
- \emptyset ,
- $(R_1 \cup R_2)$ for regular expressions R_1, R_2 ,
- $(R_1 R_2)$ for regular expressions R_1, R_2 ,
- R_1^* for a regular expression R_1 .

The rest is union, concatenation and Kleene star of regular languages, as discussed last week!

Regular Expressions and Automata

So we've just proven

Proposition

Every language described by a RE is regular.

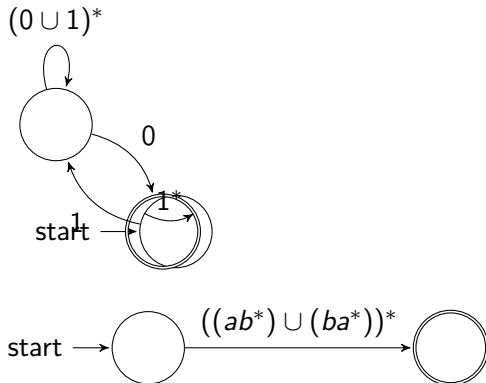
Next:

Proposition

Every regular language can be described using a RE.

Generalized Nondeterministic Finite Automaton (GNFA):

- NFA where the transitions are RE, not only symbols from Σ .
- some other assumptions for convenience:
- start state goes to every other state, but has no incoming states
- every state goes to the unique accepting state, which is different from the starting state. The accepting state does not have any outgoing arrows.
- all other states have one transition to all other states, including themselves.



Generalized Nondeterministic Finite Automata

Definition

A generalized nondeterministic finite automaton (GNFA) is a 5-tuple $(Q, \Sigma, \delta, q_{start}, q_{accept})$ where

- 1 Q is a finite set of states
- 2 Σ is the input alphabet
- 3 $\delta : (Q \setminus \{q_{accept}\}) \times (Q \setminus \{q_{start}\}) \rightarrow \mathcal{R}$ is the transition function, where \mathcal{R} is the set of all RE's over Σ ,
- 4 q_{start} is the start state, and
- 5 q_{accept} is the accept state.

Regular Expressions and Automata

Proposition

Every regular language can be described using a RE.

Proof idea: take DFA and transform into a GNFA that accepts the same language. Iteratively remove (non-starting and non-final) states so that the same language is accepted, until only the starting and accepting state remain. Then the RE along the transition between the two states describes the regular language.

Proposition

Every regular language can be described using a RE.

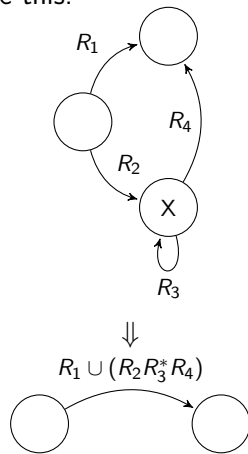
Proof:

- Given a DFA M , we construct an equivalent GNFA G by adding a new start state q_{start} with an ε transition to the old start state q_0 , as well as a new accepting state q_{accept} , with ε transitions from all old accept states.
- add \emptyset transitions for all state pairs that do not have a transition in M .

Regular Expressions and Automata

- Recall the “convenient” properties of GNFA:
 - start state goes to every other state, but has no incoming states
 - every state goes to the unique accepting state, which is different from the starting state. The accepting state does not have any outgoing arrows.
 - all other states have one transition to all other states, including themselves.

⇒ When removing X, we only need to consider situations like this:



Regular Expressions and Automata

Let's formalize this! Let us define a procedure $\text{CONVERT}(G)$:

- 1 If $k = 2$ then G only has one start and one accept state, so return the regular expression R of the transition connecting them.
- 2 if $k > 2$ select a state $q' \notin \{q_{\text{accept}}, q_{\text{start}}\}$. Define $G' = \{Q', \Sigma, \delta', q_{\text{start}}, q_{\text{accept}}\}$ with $Q' = Q \setminus \{q'\}$ and

$$\delta'(q_i, q_j) = R_1 \cup R_2 R_3^* R_4$$

where $R_1 = \delta(q_i, q_j)$, $R_2 = \delta(q_i, q')$, $R_3 = \delta(q', q')$, $R_4 = \delta(q', q_j)$, and .

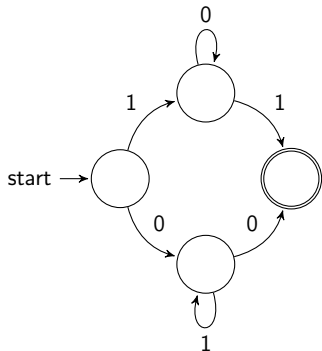
- 3 Return the result of $\text{CONVERT}(G')$.
- 4 correctness still remains to be shown! See book for details!

Regular Expressions and Automata

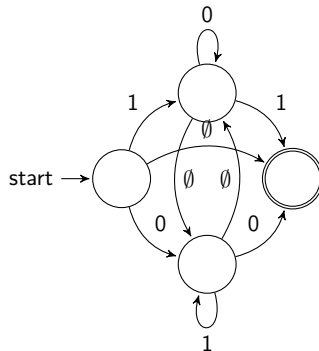
Proposition

Every regular language can be described using a RE.

Example:
DFA:



GNFA:



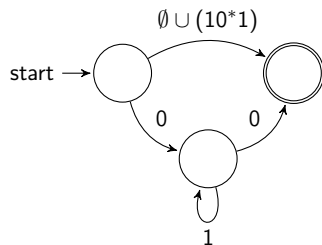
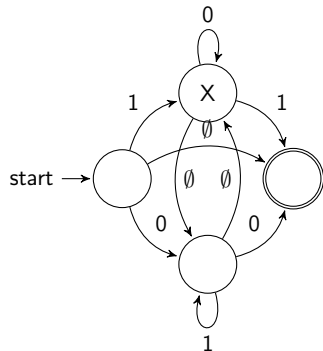
Regular Expressions and Automata

Proposition

Every regular language can be described using a RE.

Example:

Remove state X:

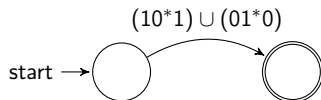
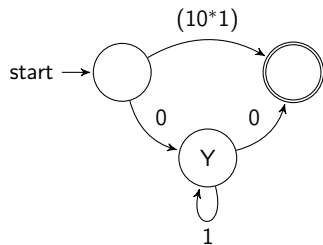


Regular Expressions and Automata

Proposition

Every regular language can be described using a RE.

Example:
Remove state Y:



Summary

So RE = GNFA = DFA = NFA = Regular languages...

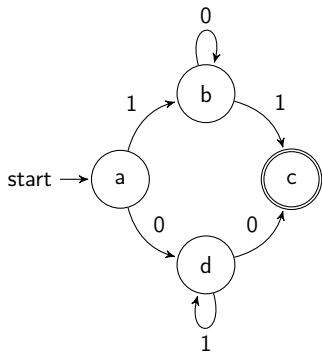
But when is a language *nonregular*? How can we check?

⇒ Pumping Lemma!

Pumping Lemma

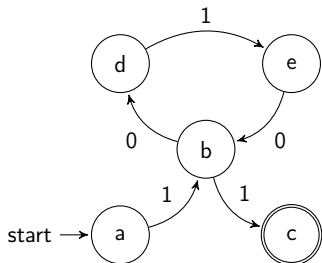
- DFAs only have *finite* memory, aka states.
- Pumping lemma gives a *pumping length*: if a string is longer than the pumping length, it can be *pumped*, i.e., there is a substring that can be repeated arbitrarily often such that the string remains in the language
- If a DFA has p states, and a string has length $\geq p$, then the accepting path in the DFA must visit at least $p + 1$ states. In other words, at least one state appears twice. \Rightarrow loop!
- This loop can be repeated while staying in the language.

Pumping Lemma - Example



- Language $(10^*1) \cup (01^*0)$
- DFA has 4 states
- consider string 10001, length 5
- \Rightarrow path must contain a loop (in this case, at node b)

Pumping Lemma - Example



- Language $1(010)^*1$
- DFA has 5 states
- consider string 10101, length 5
- \Rightarrow path must contain a loop (in this case, at nodes b,d,e)
- \Rightarrow 10100101 is also a word!

Pumping Lemma

Lemma (Pumping Lemma)

If A is a regular language, then there is a number p , called the pumping length, where if s is a word in A of length $\geq p$ then s can be divided into three parts, $s = xyz$, such that

- 1 $xy^i z \in A$ for every $i \geq 0$,
- 2 $|y| > 0$,
- 3 $|xy| \leq p$.

Pumping Lemma

- very useful for determining if a language is nonregular
- \rightarrow find a string with length $\geq p$ such that the pumping lemma does not hold
- *not* very useful for proving a language is regular
- \rightarrow not an if and only if statement!

Pumping Lemma - Applied

Lemma (Pumping Lemma)

If A is a regular language, then there is a number p , called the pumping length, where if s is a word in A of length $\geq p$ then s can be divided into three parts, $s = xyz$, such that

- 1 $xy^iz \in A$ for every $i \geq 0$,
- 2 $|y| > 0$,
- 3 $|xy| \leq p$.

- Let $A = \{0^n1^n \mid n \geq 0\}$.
- Is A regular?
- If it is, then the pumping lemma gives us a pumping length p .
- Let $s = 0^p1^p$.

Pumping Lemma - Applied

Lemma (Pumping Lemma)

If A is a regular language, then there is a number p , called the pumping length, where if s is a word in A of length $\geq p$ then s can be divided into three parts, $s = xyz$, such that

- 1 $xy^iz \in A$ for every $i \geq 0$,
- 2 $|y| > 0$,
- 3 $|xy| \leq p$.

- Let $A = \{0^n1^n \mid n \geq 0\}$.
- Let $s = 0^p1^p$.
- Condition 3 tells us that y consists of only 0s.
- \Rightarrow then xy^iz for $i \geq 2$ has more 0s than 1s. Contradiction! $\Rightarrow A$ is nonregular.

Pumping Lemma - Applied

- Even if a language is nonregular, it might contain strings for which the pumping lemma is true!
- We have to be careful!

Pumping Lemma - Applied

Lemma (Pumping Lemma)

$|s| \geq p, s = xyz, s.t.$

- 1 $xy^iz \in A$ for every $i \geq 0$,
- 2 $|y| > 0$,
- 3 $|xy| \leq p$.

- Let $B = \{\omega \mid \omega \text{ contains an equal number of 0s and 1s}\}$.
- Let $s = (01)^p$.
- $x = \varepsilon, y = 01, z = (01)^{p-1}$
- all conditions are met!

Pumping Lemma - Applied

Lemma (Pumping Lemma)

- 1 $xy^iz \in A$ for every $i \geq 0$,
- 2 $|y| > 0$,
- 3 $|xy| \leq p$.

- Let $B = \{\omega \mid \omega \text{ contains an equal number of 0s and 1s}\}$.
- Let $s = 0^p 1^p$.
- $x = \varepsilon, y = 0^p 1^p, z = \varepsilon$
- looks like it can be pumped, but are all conditions met?
- condition 3 $\Rightarrow y$ must contain only 0s, so it cannot be pumped $\Rightarrow B$ nonregular!

Pumping Lemma - Applied

- $A = \{0^n 1^n \mid n \geq 0\}$.
- $B = \{\omega \mid \omega \text{ contains an equal number of 0s and 1s}\}$
- Another way of showing B is nonregular is to reduce it to the nonregularity of A :
- regular languages are closed under intersection
- and $A = B \cap 0^*1^*$
- if B is regular and since 0^*1^* is regular, then A must be as well, contradiction!
- another way of saying this is: if a language contains a nonregular language, it must be nonregular as well!

Summary

- regular expressions are shorthand notations for languages
- $RE = GNFA = DFA = NFA$, i.e., regular expressions are shorthand for *regular* languages
- proof involved transforming a DFA to a GNFA then reducing the number of states to 2 while accepting the same language
- \rightarrow the regular expressions describe the paths in the DFA
- every regular language has a pumping length
- useful for determining if a language is *nonregular*