Theory of Computation

Paolo Bettelini

Contents

1 Fields of Study

1.1 Complexity Theory

Classify problems according to their degree of ”difficulty”.

1.2 Computability Theory

Classify problems as being solvable or unsolvable.

1.3 Automata Theory

Compare different computation models.

2 Alphabets and Languages

An *alphabet* is a finite set of *symbols*. For example: $\{a, b, c, \cdots, z\}$
The set $\{0, 1\}$ is the binary set. The empty string is denoted λ.
Note that $\lambda \neq \varnothing \neq \{\lambda\}$.
The length of a string w is denoted as $|w|$.

If Σ is an alphabet,

$$\Sigma_\lambda = \lambda \cup \Sigma$$

A set of strings is called a *language*.

3 Deterministic Finite Automaton

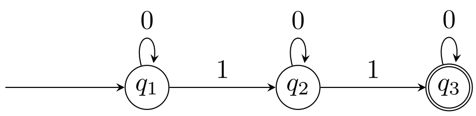
A deterministic finite automaton (DFA) is a state-machine which processes a string symbol by symbol from left to right. The automaton is in one of his *states* after processing a symbol. The machine might terminate in an *accept state* or not.

A DFA $M = (Q, \Sigma, \delta, q, F)$

- Q is a finite set of *states*
- Σ is an alphabet
- $\delta : Q \times \Sigma \to Q$ is the *transition function*
- q is an element of Q called the *start state*
- F is a subset of Q which contains the *accept states*

The transition function is the logical components, it determines in which state the machine will be after processing a symbol at any state.

The following automaton processes a binary string. The start state is q_1 and the only accept state is q_3. The program moves to the next state only if the symbol is 1, so it will reach q_3 only if the input string contains at least two 1s.



If a DFA is in a state r and it reads the symbol a, then it will uniquely switch to the state $\delta(r, a)$

The language of M, denoted $L(M)$ is the set of all accepted strings by M.

$$L(M) = \{w \in \Sigma^* \,|\, M \text{ accepts } w\}$$

4 Operations

4.1 Concatenation

If A and B are two languages over the same alphabet, the concatenation of A and B is defined as

$$AB = \{ab \,|\, a \in A \wedge b \in B\}$$

4.2 Kleene star operator

The kleene star operator can be applied to alphabets or languages. It represent the union of all n-permutations of the set.
The set $\{0,1\}^*$ is the set of all binary strings. If Σ is an alphabet, Σ^* is the set of all strings over Σ

$$\Sigma^* = \lambda \cup \bigcup_{n \in \mathbb{N}} \Sigma^n$$

5 Regular language

A language is regular if an automaton that accepts said language exists. A is regular iff

$$\exists M \,|\, L(M) = A$$

5.1 Closure under union (extra)

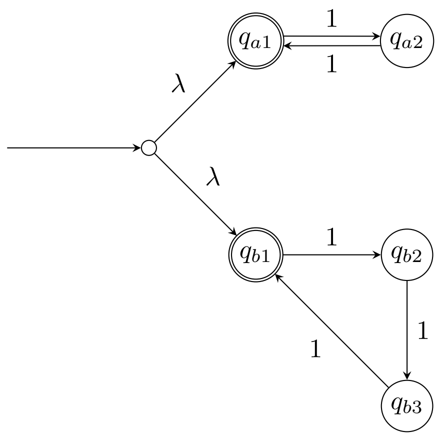
If A and B are two regular languages over the same alphabet Σ, then $A \cup B$ is also regular.

We can prove this by making a DFA that accepts both languages. Let's say that $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ accepts A and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ accepts B. The automaton $M = (Q, \Sigma, \delta, q, F)$ must run M_1 and M_2 *simultaneously*, so any state must represent the current states of M_1 and M_2. This means that the states of M must represent any combination of state between M_1 and M_2, meaning $Q = Q_1 \times Q_2$. The transition function is now in the form $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$ where $a \in \Sigma$. The initial state is the state in Q which contains the initial state of M_1 and M_2, namely (q_1, q_2). Finally, the set of accept states is every tuple in Q_1 containing a state in F_2 or in Q_2 containing a state in F_1, namely $Q_1 \times F_2 \cup Q_2 \times F_1$.
We can conclude that $M = (Q_1 \times Q_2, \Sigma, \delta((r_1, r_2), a), (q_1, q_2), Q_1 \times F_2 \cup Q_2 \times F_1)$ accepts $A \cup B$ so $A \cup B$ is regular.

6 Nondeterministic Finite Automaton

Nondeterministic finite automata (NFA) are state-machines like DFAs but can change multiple states at a time by processing empty strings λ and when processing a symbol a may have multiple possible states to switch to. The NFA will choose the "correct" switch in order to end in an accept state, if possible.

The following automaton where $\Sigma = \{1\}$ will end in an accept state if the input has length which is a multiple of 2 or 3.



The first switch is done by processing an empty string and the direction is chosen magically in order to end in an accept state.

A NFA is defined as $M = (Q, \Sigma, \delta, q, F)$ where

- Q is a finite set of *states*
- Σ is an alphabet
- $\delta : Q \times \Sigma_\lambda \to \mathcal{P}(Q)$ is the *transition function*
- q is an element of Q called the *start state*
- F is a subset of Q which contains the *accept states*

7 Equivalente of DFAs and NFAs

Anything that can be computed by a NFA can also be computed by a DFA and vice versa.

7.1 DFA to NFA conversion

Let $M = (Q, \Sigma, \delta, q, F)$ be a DFA. δ is not a transition function of a NFA, so we need to redefine it as δ'. Since δ cannot process λ, δ' it is defined as

$$\delta'(r, a) = \begin{cases} \delta(r, a) & x \neq \lambda \\ \emptyset & x = \lambda \end{cases}$$

where r is a state in Q and a is a symbol in Σ_λ.
We can conclude that $N = (Q, \Sigma, \delta', q, F)$.

7.2 NFA to DFA conversion

Let $N = (Q, \Sigma, \delta, q, F)$ be a NFA. The idea is to construct a DFA $M = (Q', \Sigma, \delta', q', F')$ that runs all the possible combinations that could be run by N at the same time. Any state of M is a set of states $R \in \mathcal{P}(Q)$, so we will say that $Q' = \mathcal{P}(Q)$. The set of accept states is any state R which contains an accept state of N

$$F' = \{R \in Q' \mid R \cap F \neq \emptyset\}$$

Let's assume that N does not execute any λ-transitions. q' would be $\{q\}$ and δ' would be

$$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$$

which is the union of all possible states N could switch to. Recall that for every $r \in R$, $\delta(r, a)$ is a set of all possible states to switch to.

Let's now remove the previous assumption. Now, M must also consider every state that could be reached by making zero or more λ-transitions. The λ-closure for a state r, $C_\lambda(r)$, is defined as the set of all possible states that can be reached from r by making zero or more λ-transitions. The λ-closure for a set of states R is defined as

$$C_\lambda(R) = \bigcup_{r \in R} C_\lambda(r)$$

The initial state q' is now given by $C_\lambda(q)$ and the transition function

$$\delta'(R, a) = \bigcup_{r \in R} C_\lambda(\delta(r, a))$$

8 Closure under regular operations

We proved using DFAs that if A and B are two regular languages over Σ, then $A \cup B$ is also regular.

$$\exists M \mid L(M) = A \cup B$$

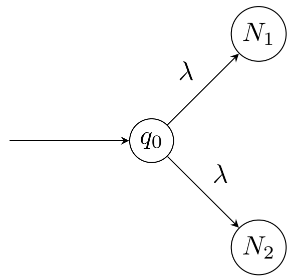
We can prove the closure under regular operations using NFAs.

8.1 Closure under union

If A and B are two regular languages over the same alphabet Σ, then $A \cup B$ is also regular.

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ be two NFAs such that $A_1 = L(N_1)$ and $A_2 = L(N_2)$. We can construct another NFA $N = (Q, \Sigma, \delta, q_0, F)$ such that $L(N) = A \cup B$. N will either go to N_1 or N_2 by making a λ-transition.

- $Q = \{q_0\} \cup Q_1 \cup Q_2$
- $F = F_1 \cup F_2$
- $$\delta(r, a) = \begin{cases} \delta_1(r, a) & r \in Q_1 \\ \delta_2(r, a) & r \in Q_2 \\ \{q_1, q_2\} & r = \lambda \\ \emptyset & r \neq \lambda \end{cases}$$

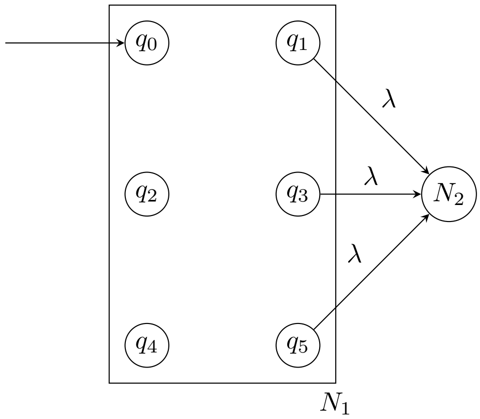


8.2 Closure under concatenation

If A and B are two regular languages over the same alphabet Σ, then AB is also regular.

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ be two NFAs such that $A_1 = L(N_1)$ and $A_2 = L(N_2)$. We can construct another NFA $N = (Q, \Sigma, \delta, q_0, F)$ such that $L(N) = AB$. N will start by executing N_1, meaning $q_0 = q_1$. If N switches to a state $r \in F_1$ it can move to executing N_2 with a λ-transition. The accepted states are only the ones of N_2 meaning $F = F_2$. $Q = Q_1 \cup Q_2$. The transition function is hence defined as

$$\delta(r,a) \begin{cases} \delta_1(r,a) & (r \in Q_1 \wedge r \notin F_1) \vee (r \in F_1 \wedge r \neq \lambda) \\ \delta_1(r,a) \cup \{q_2\} & r \in F_1 \wedge r = \lambda \\ \delta_2(r,a) & r \in Q_2 \end{cases}$$



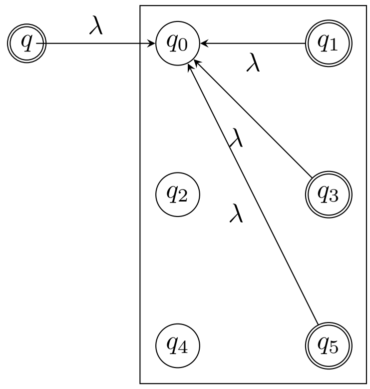
Here $F_1 = \{q_1, q_3, q_5\}$ but the actual accept states are the ones for N_2.

8.3 Closure under Kleene star

If A is a regular language, then A^* is also regular.

Let $N_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$ be a NFAs such that $A_1 = L(N_1)$. We can construct another NFA $N = (Q, \Sigma, \delta, q, F)$ such that $L(N) = A_1^*$. We want N_1 to be able to switch back to its initial point when it is in a state $r \in F_1$. This means that the concatenation of accepted strings can cycle one after the other. Since λ also needs to be accepted we need a new start state which is an accept state.

- $Q = \{q_a\} \cup Q_1$
- $q = q_a$
- $F = F_1 \cup \{q\}$
- $$\delta(r,a) = \begin{cases} \delta_1(r,a) & (r \in Q_1 \wedge r \notin F_1) \vee (r \in F_1 \wedge a \neq \lambda) \\ \delta_1(r,a) \cup \{q_0\} & r \in F_1 \wedge a = \lambda \\ \{q_0\} & r = q \wedge a = \lambda \\ \emptyset & r = q \wedge a \neq \lambda \end{cases}$$

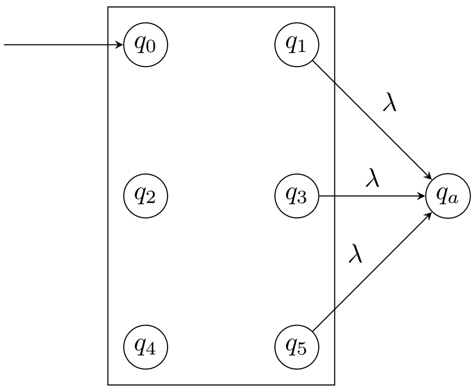


8.4 Closure under complement

If A is a regular language, then $\bar{A}$ is also regular.

Let $N_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$ be a NFAs such that $A_1 = L(N_1)$. We can construct another NFA $N = (Q, \Sigma, \delta, q, F)$ such that $L(N) = \bar{A}_1$. Any state $r \notin F_1$ will be able to switch to a new state q_a, which is the only accept state of N, with a λ-transition. This will negate any accepting state of N_1 and vice versa.

- $Q = \{q_a\} \cup Q_1$
- $q = q_0$
- $F = \{q_a\}$
- $$\delta(r, a) = \begin{cases} \delta_1(r, a) \cup \{q_a\} & r \in F_1 \\ \delta_1(r, a) & r \notin F_1 \end{cases}$$



Here $F_1 = \{q_0, q_2, q_4\}$, but the only actual accept state is q_a.

8.5 Closure under intersection

If A and B are two regular languages over the same alphabet Σ, then $A \cap B$ is also regular.

Since $A \cup B$ is regular and $A \cap B \subseteq A \cup B$, $A \cap B$ is also regular.

9 Regular Expressions

A regular expression is a mean to express a language. The class of languages that can be described by regular expressions coincides with the class of regular languages.

10 Properties

Let R_1 be a regular expression describing L_1 and R_2 a regular expression describing L_2.

- λ is a regular expression describing $\{\lambda\}$
- $\emptyset$ is a regular expression describing $\emptyset$
- $\emptyset^*$ is a regular expression describing $\{\lambda\}$
- Let Σ be a non-empty alphabet, $\forall a \in \Sigma, a$ is a regular expression describing $\{a\}$
- R_1R_2 is a regular expression describing L_1L_2
- $R_1 \cup R_2$ is a regular expression describing $L_1 \cup L_2$
- $R_1 \cap R_2$ is a regular expression describing $L_1 \cap L_2$
- R_1^* is a regular expression describing L_1^*
- $\bar{R_1}$ is a regular expression describing $\bar{L_1}$

If $L_1 = L_2$, then we say$R_1 = R_2$ (e.g. $\lambda = \emptyset^*$).

Let R_1, R_2 and R_3 be regular expressions

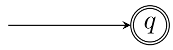
- $R_1\emptyset = \emptyset R_1 = \emptyset$
- $R_1\lambda = \lambda R_1 = R_1$
- $R_1 \cup R_2 = R_2 \cup R_1$
- $R_1 \cup \emptyset = R_1$
- $R_1 \cup R_1 = R_1$
- $R_1(R_2 \cup R_3) = R_1R_2 \cup R_1R_3$
- $(R_1 \cup R_2)R_3 = R_1R_3 \cup R_2R_3$
- $R_1(R_2R_3) = (R_1R_2)R_3$
- $\emptyset^* = \lambda$
- $\lambda^* = \lambda$
- $(\lambda \cup R_1)^* = R_1^*$
- $(\lambda \cup R_1)(\lambda \cup R_1)^* = R_1^*$
- $R_1^*(\lambda \cup R_1) = (\lambda \cup R_1)R_1^* = R_1^*$
- $R_1^*R_2 \cup R_2 = R_1^*R_2$
- $R_1(R_2R_1)^* = (R_1R_2)^*R_1$
- $(R_1 \cup R_2)^* = (R_1^*R_2)^*R_1^* = (R_2^*R_1)^*R_2^*$

11 Equivalence of regular expressions and regular languages

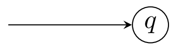
11.1 A regular expression describes a regular language

Let R be a regular expression over Σ.

Assume that $R = \lambda$. Then R describes $\{\lambda\}$. This language is regular and we can prove it by constructing an NFA $N = (Q, \Sigma, \delta, q, F)$ such that $L(N) = \{\lambda\}$. q is the start state, $Q = \{q\}$, $F = \{q\}$ and $\delta(r, a) = \emptyset$ where $a \in \Sigma_\lambda$.



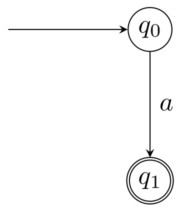
Assume that $R = \emptyset$. Then R describes $\emptyset$. This language is regular and we can prove it by constructing an NFA $N = (Q, \Sigma, \delta, q, F)$ such that $L(N) = \emptyset$. q is the start state, $Q = \{q\}$, $F = \emptyset$ and $\delta(r, a) = \emptyset$ where $a \in \Sigma_\lambda$.



Assume that $R = a$ where $a \in \Sigma$. Then R describes $\{a\}$. This language is regular and we can prove it by constructing an NFA $N = (Q, \Sigma, \delta, q, F)$ such that $L(N) = \{a\}$. q_0 is the start state, $Q = \{q_0, q_1\}$, $F = \{q_1\}$ and

$$\delta(r, b \in) = \begin{cases} \{q_1\} & b = a \land r = q_0 \\ \emptyset & \text{otherwise} \end{cases}$$

where $b \in \Sigma_\lambda$.



Assume that $R = R_1 \cup R_2$ where R_1 and R_2 are regular expressions. Let L_1 and L_2 be the languages described by R_1 and R_2 respectively. Assuming that L_1 and L_2 are regular, R then describes $L_1 \cup L_2$ which by theorem is regular.

Assume that $R = R_1 \cap R_2$ where R_1 and R_2 are regular expressions. Let L_1 and L_2 be the languages described by R_1 and R_2 respectively. Assuming that L_1 and L_2 are regular, R then describes $L_1 \cap L_2$ which by theorem is regular.

Assume that $R = R_1 R_2$ where R_1 and R_2 are regular expressions. Let L_1 and L_2 be the languages described by R_1 and R_2 respectively. Assuming that L_1 and L_2 are regular, R then describes $L_1 L_2$ which by theorem is regular.

Assume that $R = (R_1)^*$ where R_1 is a regular expression. Let L_1 be the language be described by R_1. Assuming that L_1 is regular, then R describes $(L_1)^*$ which by theorem is regular.

Assume that $R = \bar{R}_1$ where R_1 is a regular expression. Let L_1 be the language be described by R_1. Assuming that L_1 is regular, then R describes $\bar{L}_1$ which by theorem is regular.

This proves that every regular expression describes a regular language since every regular expression can be broken down to regular languages.

11.2 A DFA can be converted into a regular expression

11.2.1 Solving recurrence relations

Let Σ be an alphabet and let B, C and L be languages in Σ^* where $\lambda \notin B$

$$L = BL \cup C \implies L = B^* C$$

In order to prove this we first show that $B^*C \subseteq L$ by induction. Let $w \in B^*C$, then w is $k \geq 0$ strings in B followed a string in C. When $k = 0$, $w \in C$ which implies $w \in BL \cup C$ which implies $w \in L$. When $k > 0$ we can write w as xyz where x is a string in B, y is the concatenation of $k - 1$ strings in B and $c \in C$. Let $q = yz$. Since q is a concatenation of $k - 1$ strings of B followed by a string in C, $q \in L$. Hence, $w = xq$ where $x \in B$ and $q \in L$. This shows that $w \in BL$. Hence, $w \in BL \cup C$. Since $BL \cup C = L$, $w \in L$. This proves that $B^*C \subseteq L$.
Now we show that $L \subseteq B^*C$ by induction to conclude the proof. Let $w \in L$. When $|w| = 0$, $w = \lambda$. Since $\lambda \notin B$, $w \notin BL$. This means that $w \in C$. Since $C \subseteq B^*C$ the string w is also in B^*C. When $|w| > 0$, if $w \in C$ then $w \in B^*C$, If $w \notin C$, $w \in BL$ since $w \in L$ and $L = BL \cup C$. Hence, $w = bl$ where $b \in B$ and $l \in L$. Since $\lambda \notin B$, $|b| > 0$. This means that $|l| < |b|$ since $|l| + |b| = |w|$. By induction, l is a string in B^*C. Hence, $w = bl$ where $b \in B$ and $l \in B^*C$. This shows that $w \in B(B^*C)$. Since $B(B^*C) \subseteq B^*C$ it follows that $w \in B^*C$.

11.2.2 The conversion

We will now prove that every DFA can be converted into a regular expression. Let $M = (Q, \Sigma, \delta, q, F)$ be a DFA. We will prove that there exists a regular expression describing $L(M)$.

We define L_r where $r \in Q$ as the set of strings $w \in \Sigma_\lambda$ that would be accepted by M if r were its start state, meaning the language of M if r were its start state. Note that $L(M) = L_q$.

11.3 Conclusion

Since any DFA M can be converted into a regular expression that describes $L(M)$ and every regular expression describes a regular language, we can conclude that a language L is regular if there exists a regular expression that describes L.