Models of Computation

4: Regular expressions, finite automaton

Regular expressions

Applications

- search and replace dialogs of text editors
- search engines
- text processing utilities (e.g. sed and AWK)
- programming languages, lexical analysis
- genom analysis (genom as string)
- spam/malware filter
- ...

Regular expressions

Let *V* and *V'* = {ε, ·, +, * , (,)} be disjoint alphabets. A **regular expression** over V is defined recursively as follows:

1. ε is a regular expression over *V*,
2. all a ∈ *V* are regular expressions over *V*,
3. If *R* is a regular expression over *V*, then *R** is also a regular expression over *V*,
4. If *Q* and *R* are regular expressions over *V*, then (*Q* · *R*) and (*Q* + *R*) are also regular expressions over *V.*

* denotes the closure of iteration,
· the concatenation, and
+ union.

Regular expressions

Each regular expression **represents a regular language**, which is defined as:

1. ε represents the language {ε},
2. Letter *a* (∈ *V*) represents the language {*a*},
3. if *R* is a regular expression over *V*, which represents the language *L*, then *R** represents *L**,
4. if *Q* and *R* are regular expressions over *V*, that represent the languages *L* and *L'*, then
   ($Q \cdot R$) represents the language *LL'*,
   ($Q + R$) represents the language *L* U *L'*.

Regular expressions

Parentheses can be omitted when defining precedence on operations. The the usual sequence is: *, ·, +. The following regular expressions are equivalent:

- a^* is the same as $(a)^*$ and represent the language $\{a\}^*$.
- $(a + b)^*$ is the same as $((a) + (b))^*$ and represents the language $\{a, b\}^*$.
- $a^* \cdot$ b is the same as $((a)^*) \cdot (b)$ and represents the language $\{a\}^*b$.
- $b + ab^*$ is the same as $(b) + ((a) \cdot (b)^*)$ and represents the language $\{b\} \cup \{a\}\{b\}^*$.
- $(a + b) \cdot a^*$ is the same as $((a) + (b)) \cdot ((a)^*)$ and represents the language $\{a, b\}\{a\}^*$.

Regular expressions

Let P, Q, an R be regular expressions. Then following hold:

- $P + (Q + R) = (P + Q) + R$
- $P \cdot (Q \cdot R) = (P \cdot Q) \cdot R$
- $P + Q = Q + P$
- $P \cdot (Q + R) = P \cdot Q + P \cdot R$
- $(P + Q) \cdot R = P \cdot R + Q \cdot R$
- $P^* = \varepsilon + P \cdot P^*$
- $\varepsilon \cdot P = P \cdot \varepsilon = P$
- $P^* = (\varepsilon + P)^*$

Regular expressions

Example:

The language represented the regular expressions

$(a + b)a^*$ and $aa^* + ba^*$ is the same:

$\{ aa^n \mid n \in \mathrm{N}\} \cup \{ ba^n \mid n \in \mathrm{N} \}$.

The language represented by $a + ba^*$ is:

$\{ a, b, ba, ba^2, ba^3, \ldots\}$.

Expressive power of regular expressions

Theorem:

1) Every regular expression represents a regular (3-type) language.

2) For every regular (3-type) language, there is a regular expression representing the language.

Proof:

1) follows from the fact that the class of regular languages $\mathcal{L}_3$ is closed for the regular operations.

Expressive power of regular expressions

Proof:

For 2), we show that for every regular language L generate by a grammar $G = (N, T, P, S)$, a regular expression can be constructed, that represents L.

- Let $N = \{A_1, \ldots, A_n\}$, $n \geq 1$, $S = A_1$.
  - Each rule of G is of form $A_i \to aA_j$ or $A_i \to \varepsilon$, where $a \in T$, $1 \leq i, j \leq n$.
- We say that a non-terminal A_m is **affected** by the derivation $A_i \Rightarrow^* uA_j$ ($u \in T^*$), if A_m occurs in a intermediate string between A_i and uA_m in the derivation.

Expressive power of regular expressions

Proof (cont.):

- A derivation $A_i \Rightarrow^* uA_j$ is called **k-bounded** if $0 \leq m \leq k$ holds for all non-terminals A_m occurring in the derivation.
- Let $E^k_{i,j} = \{u \in T^* \mid \exists\ A_i \Rightarrow^* uA_j$ k-bounded derivation$\}$.
- We show by induction on k, that for language $E^k_{i,j}$, there is a regular expression representing $E^k_{i,j}$, where $0 \leq i,j,k \leq n$.

Expressive power of regular expressions

Proof (cont.):

- k=0 (induction start):
  - For $i \neq j$, $E^0_{i,j}$ is eighter empty, or it consists of symbols of T ($a \in E^0_{i,j}$ if and only if $A_i \rightarrow aA_j \in P$.)
  - For $i = j$, $E^0_{i,j}$ consists of ε and zero or more elements of T, so $E^0_{i,j}$ can be represented by a regular expression.

Expressive power of regular expressions

Proof (cont.):

- k-1 –> k (induction step):
  - Assume that for fixed k, $0 < k \leq n$, $E^{k\text{-}1}_{i,j}$ can be represented by a regular expression.
  - Then for all i, j, k it holds that
    - $E^{k}_{i,j} = E^{k\text{-}1}_{i,j} + E^{k\text{-}1}_{i,k} \cdot (E^{k\text{-}1}_{k,k})^* \cdot E^{k\text{-}1}_{k,j}$.
  - Therefore, $E^{k}_{i,j}$ can also be represented by a regular expression.
- Let I_ε be the set of indices i for which $A_i \rightarrow \varepsilon$.
  - Then $L(G) = \bigcup_{i \in I_\varepsilon} E^{n}_{1,i}$ can be representd by a regular expression. The claim of the theorem follows.

Finite Automata (FA)

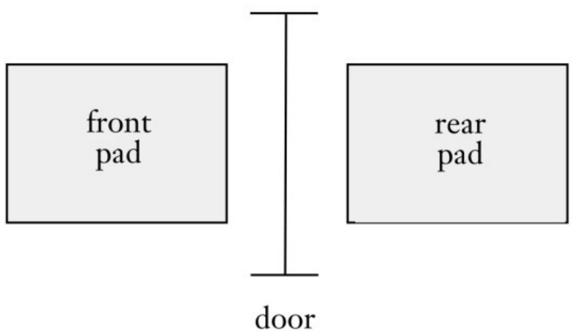
- Identifying formal languages is also possible with recognition devices, i.e. by automata.
- An automaton can process and identify words.
- Grammars use a synthesizing approach, while automata an analytic one.
- In response to a word, the automaton can either accept or reject.

Finite Automata (FA)

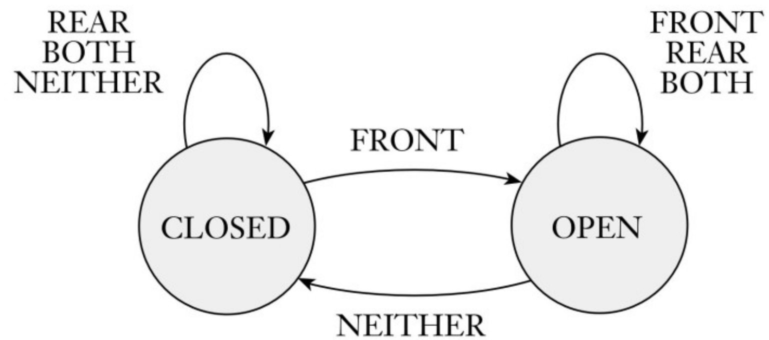
- A finite automaton performs a sequence of steps in discrete time intervals
- It starts in the initial state.
- The input word is located on the input tape and the reading head is on the leftmost symbol of an input word.
- After reading a symbol, the automaton moves the reading head to one position to the right, then the state changes, regarding the state transition function.
- If the automaton has read the input, it stops (accepts or rejects the input).

Finite Automata (FA)

- Example: automatic door control



State transition diagram:



State transition table:

state	input signal: NEITHER	FRONT	REAR	BOTH
CLOSED	CLOSED	OPEN	CLOSED	CLOSED
OPEN	CLOSED	OPEN	OPEN	OPEN

Finite Automata (FA)

- Application examples:
  - Automatic door control
  - Coffee machine
  - Pattern recognition
  - Markov chains – pattern recognition
  - Speech processing
  - Optical character recognition
  - Predictions of share prizes in the stock exchange
  - ...

Finite Automata (FA)

A finite automaton is a 5-tuple $A = (Q, T, \delta, q_0, F)$, where

- Q is a finite, nonempty set of **states**,
- T is the finite **alphabet of input symbols**,
- $\delta : Q \times T \rightarrow Q$ is the **state transition function**
- $q_0 \in Q$ is the **initial state** or **start state**,
- $F \subseteq Q$ is the set of **acceptance states** or **end states**.

Finite Automata (FA)

Remark:

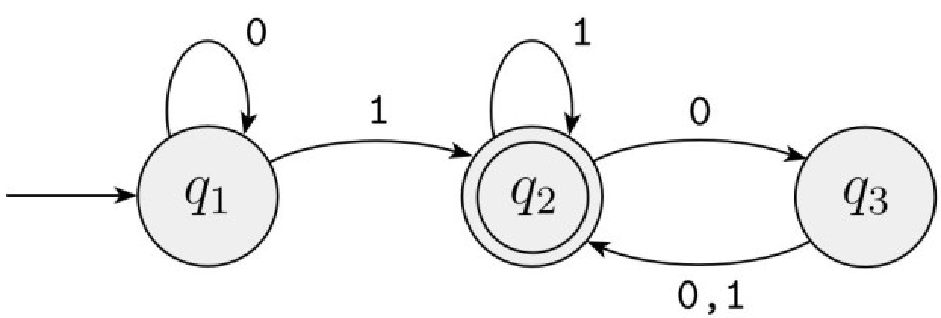
- The function δ can be extended to a function $\hat{\delta} : Q \times T^* \rightarrow Q$ as follows:
  - $\hat{\delta}(q, \varepsilon) = q$,
  - $\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$ for all $x \in T^*$ and $a \in T$.

Finite Automata (FA)

Example:

- Let $A = (Q, T, \delta, q_1, F)$ be a FA, where
  $Q = \{q_1, q_2, q_3\}$, $T = \{0, 1\}$, $F = \{q_2\}$, and
  $\delta(q_1, 0) = q_1$, $\delta(q_1, 1) = q_2$, $\delta(q_2, 0) = q_3$, $\delta(q_2, 1) = q_2$,
  $\delta(q_3, 0) = \delta(q_3, 1) = q_2$.

  State transition diagram:



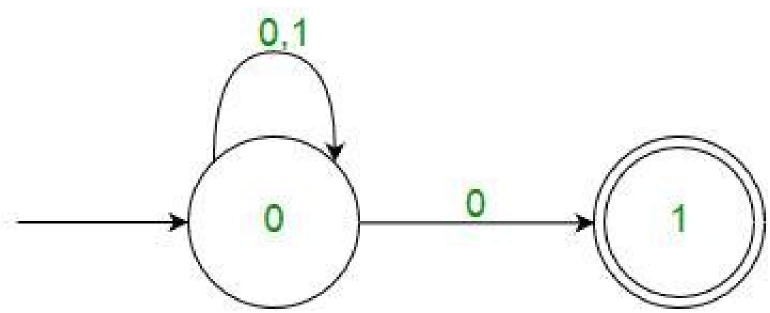
  State transition table:

δ	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

- The accepted language is $L(A)=\{w \mid w$ conains at least one 1 and the last 1 is not followed by an odd number of 0s$\}$

Deterministic and non-deterministic finite automata

- **Deterministic finite automaton (DFA)**: Function δ is single-valued, i.e. $\forall (q, a) \in Q \times T$ there is exactly one state s, s.t. $\delta(q, a) = s$.
- **Nondeterministic finite automaton (NFA)**: Function δ is multi-valued, i.e. $\delta : Q \times T \rightarrow 2^Q$.
  Multiple initial states are allowed (the set of initial states $Q_0 \subseteq Q$).
  It is allowed that $\delta(q, a) = \emptyset$ for some (q,a), i.e. the machine gets stuck.
  Null (or ε) move is allowed, i.e. it can move forward without reading symbols.



NFA example

Deterministic and non-deterministic finite automata

- New features of non-determinism
  - Multiple paths are possible (multiple choises at each step).
  - ε-transition is a “free” move without reading input.
  - Accept input if <u>some</u> path leads to an accepting state.

Deterministic and non-deterministic finite automata

- Alternative notation:
- State transitions can also be given in the form $qa \to p$, where $p \in \delta(q, a)$.
- Let M_δ be set of rules of the state transition of an NFA $A = (Q, T, \delta, Q_0, F)$.
- If M_δ contains exactly one rule $qa \to p$ for each pair (q,a), then the FA is deterministic, oherwise non-deterministic.

FA – reduction

- Let $A = (Q, T, \delta, Q_0, F)$ be a FA and $u,v \in QT^*$ words. The FA A **reduces** the u **in one step** (**directly**) to v (notation: $u \Rightarrow_A v$, or short: $u \Rightarrow v$), if there are a rule $qa \rightarrow p \in M_\delta$ (i.e. $\delta(q, a) = p$) and a word $w \in T^*$, s.t. $u = qaw$ and $v = pw$ hold.
- The FA $A = (Q, T, \delta, Q_0, F)$ **reduces** $u \in QT^*$ to $v \in QT^*$ (notation: $u \Rightarrow_A^* v$, or short: $u \Rightarrow^* v$, if
  - either $u = v$,
  - or $\exists$ a word $z \in QT^*$, s.t. $u \Rightarrow^* z$ and $z \Rightarrow$ v.
- Remark: $\Rightarrow^*$ is the reflexive, transitive closure of $\Rightarrow$.

FA – accepted language

- The **language accepted/recognized** by the FA $A = (Q, T, \delta, Q_0, F)$ is:
  $L(A) = \{u \in T^* \mid q_0 u \Rightarrow^* p$ for some $q_0 \in Q_0$ and $p \in F\}$
- For a deterministic FA A, there is one single start state $Q_0=\{q_0\}$. The language accepted by DFA A is:
  $L(A) = \{u \in T^* \mid q_0 u \Rightarrow^* p$ for some $p \in F\}$

Computing power of non-deterministic FA

- **Theorem**: For all non-deterministic FA $A = (Q, T, \delta, Q_0, F)$ a deterministic FA $A' = (Q', T, \delta', q'_0, F')$ can be constructed, s.t. $L(A) = L(A')$ holds.

- Idea: DFA keeps track of the subset of possible states in NFA
- Remark: In worst case $|Q'| = 2^{|Q|}$.