Contents

- Regular languages
- Automata
  - Deterministic finite automata DFA
  - Nondeterministic finite automata NFA
- Regular expressions
- Mapping to NFA
- NFA to DFA
- Matching
- ...

Regular expression

- **Definition:** A regular expression RE is a string on the set of symbols $\Sigma \cup \{ \epsilon, |, *, (, ) \}$, which is recursively defined as follows. RE is
  - an empty character $\epsilon$,
  - a character $\alpha \in \Sigma$,
  - $(RE_1)$,
  - $(RE_1 \cdot RE_2)$,
  - $(RE_1 | RE_2)$,
  - $(RE_1^*)$,
  - where $RE_1$ and $RE_2$ are regular expressions

Example

$((A \cdot T) | (G \cdot A)) \cdot ((A \cdot G) | ((A \cdot A) \cdot A))^*)$

- we can simplify

  $(AT|GA)(AG|AAA)^*)$

- Often also this is used:

  $RE^* = RE \cdot RE^*$

Why?

- Regular expression defines a language

  A set of words from $\Sigma^*$

  A convenient short-hand

  $(AT|GA)(AG|AAA)^*) => AT, ATAG, GAAAA, GAAGAAAAA, ...$

  Infinite set

Links

- Navarro and Raffinot Flexible Pattern Matching in Strings. (Cambridge University Press, 2002), ch. 5: Regular Expression Matching (pp. 99–143)
- Teoreetiline Informaatika (Jaan Penjam, TTÜ), Peatükk 5.
- Regular expression search using a DFA (relative difficulty: medium-hard) [ASU1996, pp. 92-105, 113-146], [NaRa2002, ch. 5], [Orponen1994, ch. 2]
- Google – Query
  - GNU grep manual (grep = Global Search for Regular Expression and Print)
  - https://www.gnu.org/software/grep/
  - FSA Utilities toolbox: a collection of utilities to manipulate regular expressions, finite-state automata and finite-state transducers. [Erik wik page]
  - Finnish-language course Model for Programming and Computing: essential regular expressions and automata theory...
  - http://www.regular-expressions.info/
**Definition:** A language represented by a regular expression RE is a set of strings over Σ, which is defined recursively on the structure of RE as follows:
- If RE is ε, then L(RE) = {ε}, the empty string.
- If RE is α ∈ Σ, then L(RE) = {α}, a single string of one character.
- If RE is of the form (RE_1)^*, then L(RE) = L(RE_1)^* = {α^n | n ≥ 0}, the set of all strings of α concatenated together.
- If RE is of the form (RE_1 | RE_2), then L(RE) = L(RE_1) ∪ L(RE_2), the union of two sets.
- If RE is of the form (RE_1)*, then L(RE) = (L(RE_1))^*, the set of all strings of RE_1 concatenated together.

**Closure:** If RE is of the form α^*, then L(RE) = L(α)^* = {α^n | n ≥ 0}, the set of all strings of α concatenated together.

**Operands in a regular expression can be:**
- Variables whose values are any pattern defined by a regular expression.
- Operations used in regular expressions include:
  - Union: If R1 and R2 are regular expressions, then R1 ∪ R2 (also written as R1 + R2) is also a regular expression.
  - Concatenation: If R1 and R2 are regular expressions, then R1·R2 is also a regular expression.
  - Star: If R1 is a regular expression, then R1* (also written as R1^*) is also a regular expression.
  - Parenthesis: Operations used in regular expressions include: (RE_1) is also a regular expression.

**A different example definition**

- Just as finite automata are used to recognize patterns of strings, regular expressions are used to generate patterns of strings. A regular expression is an algebraic formula whose value is a pattern consisting of a set of strings, called the language of the expression.
- Regular expressions are used to recognize patterns of strings.
- Operations used in regular expressions include:
  - Parenthesis: Operations used in regular expressions include: (RE_1) is also a regular expression.
  - Union: If R1 and R2 are regular expressions, then R1 ∪ R2 (also written as R1 + R2) is also a regular expression.
  - Concatenation: If R1 and R2 are regular expressions, then R1·R2 is also a regular expression.
  - Star: If R1 is a regular expression, then R1* (also written as R1^*) is also a regular expression.

**Languages:** A language represented by a regular expression RE is the number of characters of Σ in it.
- Many complexities are based on this measure.

**Regexp Matching**
- The problem of searching regular expression RE in a text T is to find all the factors of T that belong to the language L(RE).
  - Parsing
  - Thompsons NFA construction (1968)
  - Glushkov NFA construction (1961)
  - Search with the NFA
  - Determinization
  - Search with the DFA
  - Or, search by extracting set of strings, multi-pattern matching, and verification of real occurrences.
**q: what is the language?**

**Deterministic finite automaton DFA**

**Definition** DFA is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where
- $Q$ is the finite set of states of an automaton
- $\Sigma$ is the input alphabet
- $\delta : Q \times \Sigma \to Q$ is the transition function
- $q_0 \subseteq Q$ is the initial state
- $F \subseteq Q$ is a set of accepting final states

**Usage:**
- Transition step - $(q, aw) \to (q', w)$ if $\delta(q,a) = q'$, $w \in \Sigma^*$
- Accepted language: $L(M) = \{ w | (q_0, w) \to^* (q,\epsilon), q \in F \}$

**Non-deterministic finite automaton NFA**

**Definition** NFA is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where
- $Q$ is the finite set of states of an automaton
- $\Sigma$ is the input alphabet
- $\delta : Q \times \Sigma \to 2^Q$ is the transition function (a set)
- $q_0 \subseteq Q$ is the initial state
- $F \subseteq Q$ is a set of accepting final states

**Usage:**
- Transition step - $(q, aw) \to (q', w)$ if $q' \in \delta(q,a), a \in \Sigma \cup \{ \epsilon \}$, $w \in \Sigma^*$
- Accepted language: $L(M) = \{ w | (q_0, w) \to^* (q,\epsilon), q \in F \}$
\[ Q = \{ S_0, S_1, S_2 \} \]
\[ \Sigma = \{ a, b \} \]
\[ \delta : \]
\[ q_0 \in S_0 \]
\[ F = \{ S_2 \} \]

\[ S_0 \rightarrow a S_1 \]
\[ S_1 \rightarrow a S_1 \]
\[ S_1 \rightarrow b S_2 \]
\[ S_2 \rightarrow b S_2 \]

\[ S_0 \rightarrow a S_1 \rightarrow a S_1 \rightarrow a S_1 \rightarrow a S_1 \rightarrow b S_2 \rightarrow b S_2 \]

\[ a \ a \ a \ b \ b \]

**DFA**

\[ (AA)^*AT \]

\[ S_0 \rightarrow a S_1 \]
\[ S_1 \rightarrow a S_1 \]
\[ S_1 \rightarrow a S_2 \]
\[ S_2 \rightarrow t S_3 \]

\[ S_0 \rightarrow a S_1 \rightarrow a S_1 \rightarrow a S_1 \rightarrow a S_1 \rightarrow b S_2 \rightarrow b S_2 \]

\[ a \ a \ a \ b \ b \]
Regexp -> NFA / DFA

- **Construction of an automaton from the regular expression**
- Regular expressions are mathematical and human-readable descriptions of the language
- Automata represent computational mechanisms to evaluate the language
- One needs to be able to parse the regular expression and to construct an automaton for matching it.

Thompson construction

- Primitive automata
- Composition
- No optimality, no compression, etc.

Thompson construction: 2 primitive automata

- Symbol $\varepsilon$:

- Terminal symbol $a$:
Union and Concatenation

- $s|t$

- $st$

Closure

- $s^*$

Example

- $a^*(ba|c)$

• Produces up to 2m states, but it has interesting properties, such as ensuring a linear number of edges, constant indegree and outdegree, etc.

Simulation of an NFA

Input: NFA $M = (Q, \Sigma, \delta, q_0, F)$; Text $S = s[1..n]$

Output: States after each character read $Q_0, Q_1, ..., Q_n$

Note: $S \in L(M)$ only if $F \subseteq Q_n$.

1. Initially queue and sets $Q_i$ are empty
2. for $i = 0$ to $n$ do
3.     mark all $q \in Q$ unreached
4.     if ($i == 0$)
5.         $Q_0 = q_0$; queue $= q_0$; mark $q_0$ as reached
6.     else
7.         foreach $q \in Q_{i-1}$
8.             foreach $p \in \delta(q, s[i])$ // All transitions on $s[i]$
9.                 if $p$ not yet reached
10.                    $Q_i = Q_i \cup p$
11.                    push(queue, p)
12.                     mark $p$ as reached
13.     while queue $\neq \emptyset$
14.        $q = $ take(queue) // Follow up on all $\epsilon$-transitions
15.        foreach $p \in \delta(q, \epsilon)$ // All $\epsilon$-transitions
16.            if $p$ not yet reached
17.                $Q_i = Q_i \cup p$
18.                push(queue, p)
19.                mark $p$ as reached

Epsilons?
• **Theorem** Time complexity of the NFA simulation is \(O(||M_A|| \cdot n)\) where \(||M_A||\) is the total number of states and transitions of \(M_A\), \(||M_A|| \leq 6|A|\).

• **Proof** - During one step all states are manipulated only once, since all states are marked reached. There is at most \(n\) steps. The size of the automaton is at most \(6|A|\) where \(|A|\) is the length of the regular expression.

---

**Glushkov construction**

\[
(A_1 \; T_1 \; G_1 \; A_1) \; (A_2 \; G_2 \; A_2 \; A_3)^*
\]

---

**Matching of RE-s**

- No \(\epsilon\) links
- All incoming arcs have the same character label
- To reach a certain state always the same character from text had to be read.
- Construction: worst case is \(O(m^3)\) since poor performance for star closures...
- But this has been speeded up a bit
NFA -> DFA

• Why?

• More straightforward (i.e. faster) to match/simulate

Determinization of a NFA into a DFA

• Maintain at each stage a set of states reachable from previous set on the given character. (Remove ε transitions.)

• Represent every reachable combination of states of a NFA as a new state of DFA

• From each state there has to be only one transition on a given character.

• Automata for Matching Patterns Handbook of Formal Languages (kohalik)

• Navarro and Raffinot Flexible Pattern Matching in Strings. (Cambridge University Press, 2002). pp. 106 -> pp 115

Maintain at each stage a set of states reachable from previous set on the given character. (Remove ε transitions.)

Represent every reachable combination of states of a NFA as a new state of DFA

From each state there can be only one transition on a given character.

<table>
<thead>
<tr>
<th>States</th>
<th>A</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>E(0)</td>
<td>0,1,4</td>
<td>2</td>
</tr>
<tr>
<td>E(1)</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>E(2)</td>
<td>3,7,8,9,12,17</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 5.5. Thompson automaton construction for the regular expression (\(a|b|c\))\(^*\).

Fig. 5.11. Classical computation of the DFA from the NFA.
### DFA state 0

<table>
<thead>
<tr>
<th>States</th>
<th>A</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>E(0)</td>
<td>0,1,4</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>E(1)</td>
<td>2</td>
<td>-</td>
<td>3,7,8,9,12,17</td>
</tr>
<tr>
<td>E(2)</td>
<td>3,7,8,9,12,17</td>
<td>10,13</td>
<td>-</td>
</tr>
</tbody>
</table>

### NFA state 0

<table>
<thead>
<tr>
<th>DFA state</th>
<th>NFA States</th>
<th>A</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0,1,4</td>
<td>2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>-</td>
<td>3,7,8,9,12,17</td>
<td></td>
</tr>
<tr>
<td>2 F</td>
<td>3,7,8,9,12,17</td>
<td>10,13</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>10,13</td>
<td>14</td>
<td>-</td>
<td>8,9,11,12,16,17</td>
</tr>
<tr>
<td>4</td>
<td>14</td>
<td>15,16,17,8,9,12</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>8,9,12,15,16,17</td>
<td>10,13</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>8,9,11,12,16,17</td>
<td>10,13</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>10,13</td>
<td>14</td>
<td>-</td>
<td>11,16,17,8,9,12</td>
</tr>
</tbody>
</table>

### DFA state 1

<table>
<thead>
<tr>
<th>States</th>
<th>A</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>E(0)</td>
<td>0,1,4</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>E(1)</td>
<td>2</td>
<td>-</td>
<td>3,7,8,9,12,17</td>
</tr>
<tr>
<td>E(2)</td>
<td>3,7,8,9,12,17</td>
<td>10,13</td>
<td>-</td>
</tr>
<tr>
<td>E(3)</td>
<td>10,13</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

### NFA state 1

<table>
<thead>
<tr>
<th>DFA state</th>
<th>NFA States</th>
<th>A</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0,1,4</td>
<td>2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>-</td>
<td>3,7,8,9,12,17</td>
<td></td>
</tr>
<tr>
<td>2 F</td>
<td>3,7,8,9,12,17</td>
<td>10,13</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>10,13</td>
<td>14</td>
<td>-</td>
<td>8,9,11,12,16,17</td>
</tr>
<tr>
<td>4</td>
<td>14</td>
<td>15,16,17,8,9,12</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>8,9,12,15,16,17</td>
<td>10,13</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>8,9,11,12,16,17</td>
<td>10,13</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>10,13</td>
<td>14</td>
<td>-</td>
<td>11,16,17,8,9,12</td>
</tr>
</tbody>
</table>

### DFA state 2

<table>
<thead>
<tr>
<th>States</th>
<th>A</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>E(0)</td>
<td>0,1,4</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>E(1)</td>
<td>2</td>
<td>-</td>
<td>3,7,8,9,12,17</td>
</tr>
<tr>
<td>E(2)</td>
<td>3,7,8,9,12,17</td>
<td>10,13</td>
<td>-</td>
</tr>
<tr>
<td>E(3)</td>
<td>10,13</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

### NFA state 2

<table>
<thead>
<tr>
<th>DFA state</th>
<th>NFA States</th>
<th>A</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0,1,4</td>
<td>2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>-</td>
<td>3,7,8,9,12,17</td>
<td></td>
</tr>
<tr>
<td>2 F</td>
<td>3,7,8,9,12,17</td>
<td>10,13</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>10,13</td>
<td>14</td>
<td>-</td>
<td>8,9,11,12,16,17</td>
</tr>
<tr>
<td>4</td>
<td>14</td>
<td>15,16,17,8,9,12</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>8,9,12,15,16,17</td>
<td>10,13</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>8,9,11,12,16,17</td>
<td>10,13</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>10,13</td>
<td>14</td>
<td>-</td>
<td>11,16,17,8,9,12</td>
</tr>
</tbody>
</table>

### DFA state 3

<table>
<thead>
<tr>
<th>States</th>
<th>A</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>E(0)</td>
<td>0,1,4</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>E(1)</td>
<td>2</td>
<td>-</td>
<td>3,7,8,9,12,17</td>
</tr>
<tr>
<td>E(2)</td>
<td>3,7,8,9,12,17</td>
<td>10,13</td>
<td>-</td>
</tr>
<tr>
<td>E(3)</td>
<td>10,13</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

### NFA state 3

<table>
<thead>
<tr>
<th>DFA state</th>
<th>NFA States</th>
<th>A</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0,1,4</td>
<td>2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>-</td>
<td>3,7,8,9,12,17</td>
<td></td>
</tr>
<tr>
<td>2 F</td>
<td>3,7,8,9,12,17</td>
<td>10,13</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>10,13</td>
<td>14</td>
<td>-</td>
<td>8,9,11,12,16,17</td>
</tr>
<tr>
<td>4</td>
<td>14</td>
<td>15,16,17,8,9,12</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>8,9,12,15,16,17</td>
<td>10,13</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>8,9,11,12,16,17</td>
<td>10,13</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>10,13</td>
<td>14</td>
<td>-</td>
<td>11,16,17,8,9,12</td>
</tr>
</tbody>
</table>
Minimization of automata

• DFA construction does not always produce the minimal automaton

• Smaller -> better(?)

• Must still represent equivalent languages!

Minimization of automata

• Language is simply a subset of all possible strings of Σ* 
  • Regular language is the language that can be described using regular expressions and recognized by a finite automaton (no pushdown, multi-tape, or Turing machines)
  • Two automations that recognize exactly the same language are equivalent
  • The automaton that has fewest possible states from the same equivalence class, is the minimal
  • Automaton with more states is called redundant
  • The automation construction techniques do not create the minimal automaton
  • It would be easier to understand nonredundant automatons
  • Smaller automaton consumes less memory
  • The manipulation is faster

Minimization

• A compiler course subject
  • Minimization description ([L4_RegExp/min-fa.html])
  A: Merge all equivalent states until minimum achieved
  B: Start from minimal possible (2-state) and split states until no conflicts
• Fact. Equivalent states go to equivalent states under all inputs.
• Recognizer for \((aa \mid b)*ab(bb)^*\)

Step 1: Generate 2 equivalence classes: final and other states

Step 2: Create new class from 1 and 6 (conflict on b)

Step 3: Create new class from 3

Step 4: Create new class from 6

Minimal automaton

All states are consistent
Construct a DFA from the regular expression

- Usually NFA is constructed, and then determined
- McNaughton and Yamada proposed a method for direct construction of a DFA

- Example: Let’s analyze \( RE = (a \cup b)^*aba \)
- Add end symbol \# : \((a \cup b)^*aba#\)
- Make a parse tree
  - Leaves represent symbols of \( \Sigma \) from \( RE \)
  - Internal nodes - operators
- Give a unique numbering of leaves
- Position nr is active if this can represent the next symbol
- DFA states and transitions are made from the tree:
  - A state of DFA corresponds to a set of positions that are active after reading some prefix of the input
  - Initial state is \( \{1,2,3\} \) (when nothing has been read yet)
  - DFA contains transitions \( q \to q' \), where \( q' \) are position nrs that are activated when in positions of \( q \) the character a is read.
- Final states are those containing the position number of #
Construction of regular expressions from the automata

- It is possible to start from an automaton and then generate the regular expression that describes the language recognized by the automaton.


- The program JFLAP for transforming FSA to regular expressions can be downloaded from http://www.jflap.org/, or http://www.cs.duke.edu/~rodger/tools/jflap/indexold.html

- In the bottom of page there are links to "current version".
Filtering approaches for regular expression searches

- Identify a (sub)set of prefixes or factors that are necessarily present in the language represented by regexp.
- Use multi-pattern matching techniques for matching them all simultaneously
- In case of a match use the automaton to verify the occurrence

Prefixes

- \((GA|AAA)*(TA|AG)\) the set of 2-long prefixes is \{GA, AA, TA, AG\}
- \((AT|GA)(AG|AAA)((AG|AAA)+)\) lmin=6
- \{ATAGAG, ATAGAA, ATAAAA, GAAGAG, GAAGAA, GAAAAA\}

- \((AG|GA)\text{ATA}((TT)^*)\)
- The string ATA is a necessary factor.
- Gnu grep uses such heuristics
- Can be developed to utilise a lot of knowledge about possible frequencies of occurrences, speed of multi-pattern matchers etc.
### Summary

- Regular expression
- Parse
- NFA
- DFA
- Occurrences
- Minimize

### Learning languages

**Grammatical inference**

- AGAGGAT +
- ATGAGAA +
- ATGATTA –
- AA –
- AAATGA –
- AGATAG +

Q: What is the language represented by the positive examples?

A1: List of positive examples

A2: Minimal automation that recognises + examples, and none of the – examples?

Finding A2 in general a computationally hard problem

### Graph algorithms?

- Shortest path from start to end?

- Minimal cost path? (what would be the weights?)