Contents

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

Links

- Navarro and Raffinot: Flexible Pattern Matching in Strings. (Cambridge University Press, 2002), ch. 5. Regular Expression Matching (pp. 99–143)
- Pattern Matching (John Peltonen, TTI, Helsinki).
- Regular expression nondeterministic automata (Meeks Ross) tutorial
- Google - Query
- http://regex.info/f遣/g/Regular_expression
- dita-regex.html (game = Global search for Regular expression and Pratt)
- Regular expression tool for manipulating regular expressions: Find expressions, Perl-style backrefs, (Sipser’s book)
- Regular expression library for manipulating regular expressions: essential regular expressions and DFA (Kozen’s library)
- http://www.regexexpress.com/Resources/

Regular expression

- Definition: A regular expression (RE) is a string on the set of symbols $\Sigma \cup \{ \varepsilon, |, *, (, ) \}$, which is recursively defined as follows. RE is
  - an empty character $\varepsilon$,
  - a character $\alpha \in \Sigma$,
  - (RE$_1$),
  - (RE$_1$ · RE$_2$),
  - (RE$_1$ | RE$_2$), and
  - (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
- $$(AT|GA)((AG|AAA)^*) \Rightarrow AT, ATAG, GAAAA, GAAGAAAAAA, ...$$
- Infinite set
Language represented by RE

**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 \( L(RE) = \{ \varepsilon \} \), the empty string.
- If RE is a symbol in \( \Sigma \), then \( L(RE) = \{ \sigma \} \), a single string of one character.
- If RE is of the form \((RE_1)\), then \( L(RE) = L(RE_1) \), parentheses.
- If RE is of the form \((RE_1 \cdot RE_2)\), then \( L(RE) = L(RE_1) \cdot L(RE_2) \), where \( w_1 \cdot w_2 \) is in \( L(RE) \) if \( w_1 \in L(RE_1) \) and \( w_2 \in L(RE_2) \). (We call \( \cdot \) the concatenation operator)
- If RE is of the form \((RE_1 | RE_2)\), then \( L(RE) = L(RE_1) \cup L(RE_2) \), the union of two languages. (We call \( | \) the union operator)
- If RE is of the form \((RE_1)^*\), then \( L(RE) = L(RE)^* = \bigcup_{i \geq 0} L(RE_1)^i \), where \( L_0 = \{ \varepsilon \} \) and \( L_i = L \cdot L_{i-1} \). (We call \( ^* \) the star operator)

<table>
<thead>
<tr>
<th>Regular expression</th>
<th>Language ( L(RE) )</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε</td>
<td>{ε}</td>
<td>Empty string</td>
</tr>
<tr>
<td>( \sigma \in \Sigma )</td>
<td>{σ}</td>
<td>Single character</td>
</tr>
<tr>
<td>((RE_1))</td>
<td>(L(RE_1))</td>
<td>Parenthesis</td>
</tr>
<tr>
<td>((RE_1 \cdot RE_2))</td>
<td>(L(RE_1) \cdot L(RE_2))</td>
<td>Concatenation</td>
</tr>
<tr>
<td>((RE_1</td>
<td>RE_2))</td>
<td>(L(RE_1) \cup L(RE_2))</td>
</tr>
<tr>
<td>((RE_1)^*)</td>
<td>(L(RE)^* = \bigcup_{i \geq 0} L(RE_1)^i)</td>
<td>The star operator (Kleene star)</td>
</tr>
</tbody>
</table>

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.
- Operators in a regular expression can be:
  - characters from the alphabet over which the regular expression is defined.
  - variables whose values are any pattern defined by a regular expression.
  - epsilon which denotes the empty string containing no characters.
  - null which denotes the empty set of strings.
- Operators used in regular expressions include:
  - concatenation: \( R_1 \cdot R_2 \), if \( R_1 \) and \( R_2 \) are regular expressions, then \( R_1 \cdot R_2 \) is also a regular expression.
  - union: \( R_1 \cup R_2 \), if \( R_1 \) and \( R_2 \) are regular expressions, then \( R_1 \cup R_2 \) is also a regular expression.
  - Kleene closure: \( R_1^* \), if \( R_1 \) is a regular expression, then \( R_1^* \) (the Kleene closure of \( R_1 \)) is also a regular expression.
  - \( R_1 \cdot R_2 \cup R_2 \cdot R_1 \), if \( R_1 \) and \( R_2 \) are regular expressions, then \( R_1 \cdot R_2 \cup R_2 \cdot R_1 \) is also a regular expression.

- The size of a regular expression \( RE \) is the number of characters of \( \Sigma \) 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
  - Thompson's 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.

Matching of RE-s

<table>
<thead>
<tr>
<th>Regular expression</th>
<th>Parse</th>
<th>NFA</th>
<th>DFA</th>
<th>Occurrences</th>
</tr>
</thead>
</table>

03.12.18
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
- $\delta : Q \times \Sigma \rightarrow Q$ is the transition function
- $q_0 \in Q$ is the initial state
- $F \subseteq Q$ is a set of accepting final states

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

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
- $\delta : Q \times \Sigma \rightarrow \wp(Q)$ is the transition function (a set)
- $q_0 \in Q$ is the initial state
- $F \subseteq Q$ is a set of accepting final states

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

\[ Q = \{ S_0, S_1, S_2 \} \]
\[ \Sigma = \{ a, b \} \]
\[ \delta : q_0 \in S_0 \]
\[ F = \{ S_2 \} \]

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

\[ Q = \{ S_0, S_1, S_2 \} \]
\[ \Sigma = \{ a, b \} \]
\[ \delta : \]
\[ q_0 \in S_0 \]
\[ F = \{ S_2 \} \]

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

\[ (AA)^*AT \]

\[ Q = \{ S_0, S_1, S_2 \} \]
\[ \Sigma = \{ a, b \} \]
\[ \delta : \]
\[ q_0 \in S_0 \]
\[ F = \{ S_2 \} \]

\[ S_0 \rightarrow a S_1 \]
\[ S_1 \rightarrow a S_1 \]
\[ S_1 \rightarrow b S_2 \]
\[ S_2 \rightarrow b S_2 \]
• (AA)*AT

\[
\begin{align*}
S_0 & \rightarrow a S_1 \\
S_1 & \rightarrow a S_2 \\
S_2 & \rightarrow a S_1 \\
S_3 & \rightarrow t S_4
\end{align*}
\]

AAAAT

NFA – simultaneously in all reachable states

• (AA)*AT

\[
\begin{align*}
S_0 & \rightarrow a S_1 \\
S_1 & \rightarrow a S_0 \\
S_0 & \rightarrow a S_2 \\
S_2 & \rightarrow t S_3
\end{align*}
\]

AAAAT

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 \( \epsilon \):

\[
\begin{array}{c}
\text{i} \\
\text{c} \\
\text{f}
\end{array}
\]

• Terminal symbol \( a \):

\[
\begin{array}{c}
\text{i} \\
\text{a} \\
\text{f}
\end{array}
\]


Automaadi konstruktsioon: 

\[
\text{regexp to automaton: \text{Navarro and Raffinot} (flexible pattern matching in strings)}
\]

Tsiteat:

• Nii saadud lõplik automaat pole determineeritud, kuna me kasutame juba primitiivsete automaate või primitiivsete automaate kompositsiooni. Lõpliku automaadi võib hiljem muidugi eraldi determineerida.
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, Σ, q0, δ, F).
Output: States after each character read Q0, Q1, ..., Qn
NB: S ∈ L(M) only if F ⊆ Qn.

Initially queue and sets Qi are empty.

1. for i = 0 to m do
2. mark all q ∈ Qi unreached
3. if (i == 0) then
4. Q0 = q0, queue = q0, mark q0 as reached
5. else
6. foreach q ∈ Qi-1 // Main transitions on s[i]
7. foreach p ∈ δ(q, s[i]) // All transitions on s[i]
8. if p not yet reached
9. Q = Q + p
10. push(queue, p)
11. mark p as reached
12. while queue # 0
13. q = take(queue) // Follow up on all ε-transitions
14. foreach p ∈ δ(q, ε) // All ε-transitions
15. if p not yet reached
16. Q = Q + p
17. push(queue, p)
18. mark p as reached
19. 1 or more? Epsilon?
• **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**

$A_1 \rightarrow T_2 \rightarrow G_3 \rightarrow A_4 \rightarrow (A_2 \rightarrow G_4 \rightarrow A_5 \rightarrow A_4)^*$

---

**Matching of RE-s**

- No ε 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.
### DFA states | NFA states | A | T | G
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>E(0)</td>
<td>0,1,4</td>
<td>2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E(1)</td>
<td>2</td>
<td>-</td>
<td>3,7,8,9,12,17</td>
<td></td>
</tr>
<tr>
<td>E(2)</td>
<td>3,7,8,9,12,17</td>
<td>10,13</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E(3)</td>
<td>10,13</td>
<td>14</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### DFA states | NFA states | A | T | G
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>E(0)</td>
<td>0,1,4</td>
<td>2</td>
<td>-</td>
<td>- (57)</td>
</tr>
<tr>
<td>E(1)</td>
<td>2</td>
<td>-</td>
<td>3,7,8,9,12,17</td>
<td></td>
</tr>
<tr>
<td>E(2)</td>
<td>3,7,8,9,12,17</td>
<td>10,13</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E(3)</td>
<td>10,13</td>
<td>14</td>
<td>-</td>
<td>11,16,17,8,12</td>
</tr>
</tbody>
</table>

### DFA states | NFA states | A | T | G
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>E(0)</td>
<td>0,1,4</td>
<td>2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E(1)</td>
<td>2</td>
<td>-</td>
<td>3,7,8,9,12,17</td>
<td></td>
</tr>
<tr>
<td>E(2)</td>
<td>3,7,8,9,12,17</td>
<td>10,13</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E(3)</td>
<td>10,13</td>
<td>14</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### DFA states | NFA states | A | T | G
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>E(0)</td>
<td>0,1,4</td>
<td>2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E(1)</td>
<td>2</td>
<td>-</td>
<td>3,7,8,9,12,17</td>
<td></td>
</tr>
<tr>
<td>E(2)</td>
<td>3,7,8,9,12,17</td>
<td>10,13</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>E(3)</td>
<td>10,13</td>
<td>14</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### DFA states | NFA states | A | T | G
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0,1,4</td>
<td>2</td>
<td>-</td>
<td>5</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>15,16,17,8,12</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>11,16,17,8,12</td>
</tr>
</tbody>
</table>

### DFA states | NFA states | A | T | G
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0,1,4</td>
<td>2</td>
<td>-</td>
<td>5</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>15,16,17,8,12</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>11,16,17,8,12</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 $\Sigma^*$
- Regular language is the language that can be described using regular expressions and recognised by a finite automaton (no pushdown, multi-tape, or Turing machines)
- Two automatons 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 automaton construction techniques do not create the minimal automaton
- It would be easier to understand nonredundant automaton
- 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
Construct a DFA from the regular expression

- Usually NFA is constructed, and then determinized
- 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 - operations
- 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 \(q \rightarrow a \cdot q'\), where \(q'\) are position nums 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.
- In the bottom of page there are links to "current version".

---

[Diagram of automaton and regular expression conversion process]
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
  - Imín - the shortest occurrence length (to avoid missing short occurrences)
  - \((GA|AAA)^*(TA|AG)\) the set of 2-long prefixes is \{ GA, AA, TA, AG \}
  - \((AT)GA(AG|AAA)((AG|AAA)+)\) Imín=6
  - \{ ATAGAG, ATAGAA, ATAAAA, GAAGAG, GAAGAA, GAAAAA \}

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

regular expression → Parse → NFA → DFA → 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 automaton that recognizes + 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?)