Algorithmics (6EAP)

Regular Expressions and Automata

Jaak Vilo
2020 fall

Contents

• Regular expressions and regular languages
  • Automata
    — Deterministic finite automata DFA
    — Nondeterministic finite automata NFA
  • Regular expressions to NFA
  • NFA to DFA

Links

• Navarro and Raffinot, Flexible Pattern Matching in Strings. (Cambridge University Press, 2002).
  ch. 5: Regular Expression Matching (pp. 99–143)
• Regular expression search using a DFA (relative difficulty, handbook level) (IEEE TRANS. Comput., pp. 10–20, 113–134, [Nika2002], ch. 4)
• Regular expression search using a DFA (relative difficulty, handbook level) (IEEE TRANS. Comput., pp. 10–20, 113–134, [Nika2002], ch. 4)

Regular expression

• Definition: A regular expression RE is a string on the set of symbols Σ ∪ {ε, |, ·, *, (, )}, which is recursively defined as follows. RE is
  — an empty character ε,
  — a character α ∈ Σ,
  — ( RE1 ),
  — ( RE1 · RE2 ),
  — ( RE1 | RE2 ), and
  — ( RE1* ),
where RE1 and RE2 are regular expressions

Example

(( A - T ) | ( G - A )) · (( A - G ) | (( A - A ) - A )* )

• we can simplify
  (AT|GA)((AG)(AAA)*)

• Often also this is used:
  RE+ = RE · RE* 

Why?

• Regular expression defines a language
  • A set of words from Σ*
  • A convenient short-hand
    • (AT|GA)((AG)(AAA)*) ⇒ AT, ATAG, GAAAA, GAAGAAAAA, ...
  • 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) = {ε}, the empty string
- If RE is a ∈ Σ, then L(RE) = {a}, a single string of one character
- If RE is of the form (RE_1), then L(RE) = L(RE_1), the output of one language, (for call) (the concatenate operator)
- If RE is of the form (RE_1 · RE_2), then L(RE) = L(RE_1) · L(RE_2), the output of two languages, (for call) (the union operator)
- If RE is of the form (RE_1|RE_2), then L(RE) = L(RE_1) U 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 = {ε} and L_i = L · 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>a ∈ Σ</td>
<td>{a}</td>
<td>Single character</td>
</tr>
<tr>
<td>(RE_1)</td>
<td>L(RE_1)</td>
<td>Parenthesis</td>
</tr>
<tr>
<td>(RE_1 · RE_2)</td>
<td>L(RE_1) · L(RE_2)</td>
<td>Concatenation</td>
</tr>
<tr>
<td>(RE_1</td>
<td>RE_2)</td>
<td>L(RE_1) U L(RE_2)</td>
</tr>
<tr>
<td>(RE_1)*</td>
<td>L(RE_1)* = ( \bigcup_{i \geq 0} L(RE_1)^i )</td>
<td>The star operator (Kleene star)</td>
</tr>
</tbody>
</table>

A different example definition

- A different example definition
- Just as finite automata are used in a recognizer pattern of strings, regular expressions can create automatic patterns of regular expressions or an equivalent formula in regular expressions.
- A regular expression or regular expressions are particularly convenient in recognizing patterns of strings.
- A regular expression can describe a pattern consisting of a set of strings, called the language of the expression.
- Operands 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: If R1 and R2 are regular expressions, then R1R2 (also written as R1.R2) is also a regular expression.
    \( L(R1R2) = L(R1) \) concatenated with L(R2).
  - | Union: If R1 and R2 are regular expressions, then R1 | R2 (also written as R1 U R2 or R1 + R2) is also a regular expression.
    \( L(R1|R2) = L(R1) U L(R2) \).
  - * Kleene closure: If R1 is a regular expression, then R1* (the Kleene closure of R1) is also a regular expression
    \( L(R1*) = \epsilon \cup L(R1) \cup L(R1R1) \cup L(R1R1R1) \cup ... \).
- In order to simplify the expression, we define the following:
- A different example definition
- Just as finite automata are used in a recognizer pattern of strings, regular expressions can create automatic patterns of regular expressions.
- A regular expression or regular expressions are particularly convenient in recognizing patterns of strings.
- A regular expression can describe a pattern consisting of a set of strings, called the language of the expression.
- Operands 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: If R1 and R2 are regular expressions, then R1R2 (also written as R1.R2) is also a regular expression.
  - | Union: If R1 and R2 are regular expressions, then R1 | R2 (also written as R1 U R2 or R1 + R2) is also a regular expression.
  - * Kleene closure: If R1 is a regular expression, then R1* (the Kleene closure of R1) is also a regular expression.
  - Suffix of a regular expression is defined as a regular expression
  - Lexicon has the properties of a prefix, followed by a property.
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: $(q, a, w) \rightarrow (q', w)$ if $q' = \delta(q, a), w \in \Sigma^*$
- Accepted language: $L(M) = \{ w \mid (q_0, w) \rightarrow^* (q, \epsilon), q \in F \}$

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
- $\delta : Q \times (\Sigma \cup \{ \epsilon \}) \rightarrow P(Q)$ 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: $(q, a, w) \rightarrow (q', w)$ if $q' \in \delta(q, a), a \in \Sigma^*$
- Accepted language: $L(M) = \{ w \mid (q_0, w) \rightarrow^* (q, \epsilon), q \in F \}$

Lõplik automaat (näide)

13
14
15
16
17
18
DFA

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

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

\[ (AA)^*AT \]

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

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

\[ (AA)^*AT \]

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

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

\[ (AA)^*AT \]

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

\[ S_0 \rightarrow a S_1 \]
\[ S_0 \rightarrow a S_1 \]
\[ S_0 \rightarrow a S_1 \]
\[ S_1 \rightarrow a S_1 \]
\[ S_1 \rightarrow a S_1 \]
\[ S_1 \rightarrow a S_1 \]
\[ S_1 \rightarrow b S_2 \]
\[ S_2 \rightarrow b S_2 \]
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.
Union and Concatenation

- $s|t$
- $st$

### Example

- $a^* (ba|c)$

### Simulation of an NFA

**Input:** NFA $M=(Q, \Sigma, \delta, q_0, F)$

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

1. Initialize queue and sets $Q_i$ as empty
2. for $i = 0$ to $n$
   // for each symbol of text
   2. mark all $q \in Q$ unreached
   3. if ($i == 0$)
      // Initialise start state
      3. $Q_0 = q_0$;
         queue = $q_0$; mark $q_0$ as reached
   4. else
      // Main transitions on $s[i]
      4. foreach $q \in Q_{i-1}$
      5. foreach $p \in \delta(q, s[i])$
      // All transitions on $s[i]
      6. if $p$ not yet reached
          6. $Q_i = Q_i \cup p$
          7. push(queue, p)
          8. mark $p$ as reached
      9. while queue $\neq \emptyset$
      // Follow up on all $\varepsilon$-transitions
      10. $q = take(queue)$
      11. foreach $p \in \delta(q, \varepsilon)$
      // All $\varepsilon$-transitions
      12. if $p$ not yet reached
          12. $Q_i = Q_i \cup p$
          13. push(queue, p)
          14. mark $p$ as reached

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

\[
\]

---

**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

---

Determination 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 Adaptive Pattern Matching in Formal Languages* (Kohalik)

---

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>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>E(0)</td>
<td>0,1,4</td>
<td>2</td>
</tr>
<tr>
<td>E(1)</td>
<td>2</td>
<td>3,7,8,9,12,17</td>
</tr>
<tr>
<td>E(2)</td>
<td>3,7,8,9,12,17</td>
<td>F</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>States</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>E(0)</td>
<td>0,1,4</td>
<td>2</td>
</tr>
<tr>
<td>E(1)</td>
<td>2</td>
<td>3,7,8,9,12,17</td>
</tr>
<tr>
<td>E(2)</td>
<td>3,7,8,9,12,17</td>
<td>F</td>
</tr>
</tbody>
</table>
Minimization of automata

- DFA construction does not always produce the minimal automaton
- Smaller -> better (?)
- Must still represent equivalent languages!

Minimization

- A compiler course subject
  Addison Wesley 1986. (The Dragon Book)
- 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 \((aa \mid b)^*ab(bb)^*\)

![Diagram of states and transitions]

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 ∪ b)*aba
- Add end symbol # : (a ∪ b)*aba#
- Make a parse tree
- Leaves represent symbols of Σ 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
- Initial state is (1,2,3) (when nothing has been read yet)
- DFA contains transitions q → a q’ where q’ are positions 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.
- **Example**
  - 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
• $l_{\text{min}}$ - the shortest occurrence length (to avoid missing short occurrences)
• $[(GA|AAA)*(TA|AG)]$ the set of 2-long prefixes is $\{ GA, AA, TA, AG \}$
• $[(AT|GA)(AG|AAA)((AG|AAA)+)]$ $l_{\text{min}}=6$
• $\{ ATAGAG, ATAGAA, ATAAAA, GAAGAG, GAAGAA, GAAAAA \}$

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

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

Q: What is the language represented by the positive examples?
A1: List of positive examples
A2: Minimal recognizer that recognizes + examples, and none of the – examples?

Finding A2 in general is computationally hard problem.

Summary

Regular expression Parse NFA DFA Occurrences minimize

How to find primes

perl -e 'foreach $i (1000..1050) { print "$i
if ("a"*$i) !~ /^(aa+)(\1+)$/}'}

1009
1013
1019
1023
1031
1033
1039
1049