Algorithmics (6EAP)

Regular Expressions and Automata

Jaak Vilo
2019 fall

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. 95–143)
- Processing Informatics (Evan Perret; TTI, Helsinki)
- Regular expression nondeterministic automaton minimization (Melles Ross) Toolbox
- Google - Query
  - http://www.regular-expressions.info/
  - http://www.regular-expressions.info/re2
  - http://www.regular-expressions.info/re2java
  - http://www.regular-expressions.info/re2java2
  - http://www.regular-expressions.info/regex.html

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:
  \( \text{RE}^+ = \text{RE} \cdot \text{RE}^* \)

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

Example

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

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

• Often also this is used:
  \( \text{RE}^+ = \text{RE} \cdot \text{RE}^* \)
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 the form (RE₁), then L(RE)=L(RE₁)
- if RE is of the form (RE₁|RE₂), then L(RE)=L(RE₁) ∪ L(RE₂), where w=w₁w₂ is in L(RE₁) and w₁∉L(RE₂) and w₂∉L(RE₁)
- if RE is of the form (RE₁·RE₂), then L(RE)=L(RE₁)·L(RE₂), the union of two languages. (We call · the concatenation operator)
- if RE is of the form (RE₁*) then L(RE) = L(RE) ∗ = ∪_{i≥0} L(RE₁)^i, where L₀={ε} and Lᵢ=L·Lᵢ₋₁. (We call * the star operator)

### Regular expression Language (L(RE)) Comment

<table>
<thead>
<tr>
<th>RE</th>
<th>L(RE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε</td>
<td>{ε}</td>
</tr>
<tr>
<td>a ∈ Σ</td>
<td>{a}</td>
</tr>
<tr>
<td>(RE₁)</td>
<td>L(RE₁)</td>
</tr>
<tr>
<td>(RE₁</td>
<td>RE₂)</td>
</tr>
<tr>
<td>(RE₁·RE₂)</td>
<td>L(RE₁)·L(RE₂)</td>
</tr>
<tr>
<td>(RE₁*)</td>
<td>L(RE₁)^*</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.
- Regular expressions 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 R₁·R₂ (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* is also a regular expression.
  - Kleene plus: If R₁ is a regular expression, then R₁+ (the Kleene closure of R₁) is also a regular expression.
- The size of 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.

Matching of RE-s

- Regular expression Parse NFA DFA Occurrences
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 $\delta(q, a) = q'$, $w \in \Sigma^*$
- Accepted language: $L(M) = \{ w | (q_0, w) \rightarrow^* (q, \varepsilon), q \in F \}$

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

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

<table>
<thead>
<tr>
<th>State</th>
<th>Character</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>0, 1</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>1, 2</td>
</tr>
</tbody>
</table>

\[ q_0 \in S_0 \]
\[ F = \{ S_2 \} \]

\( a + b + \)

\( S_0 \rightarrow a S_0 \)
\( S_1 \rightarrow b S_1 \)
\( S_2 \rightarrow b S_2 \)

\( S_0 \rightarrow a S_0 \rightarrow a S_0 \rightarrow a S_0 \rightarrow a S_0 \rightarrow b S_2 \rightarrow b S_2 \)

\( a \ a \ a \ b \ b \)

\( (AA)^*AT \)

\( Q = \{ 0, 1, 2, 3 \} \)
\[ \Sigma = \{ A, T \} \]
\[ F = \{ 3 \} \]

\( (AA)^*AT \)
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

- $s | t$

- $st$

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 $Q_0, Q_1, ... Q_n$.

NB: $S \in L(M_A)$ only if $F \subseteq Q_n$.

Initially queue and sets $Q_i$ are empty.

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

$$\begin{array}{c}
\begin{array}{c}
A_1 \quad T_2 \quad G_1 \quad A_2 \\
\end{array}
\end{array}$$

---

**Matching of RE-s**

- No $\varepsilon$ 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>2</td>
<td>-</td>
</tr>
<tr>
<td>E(2)</td>
<td>3,7,8,12,17</td>
<td></td>
</tr>
</tbody>
</table>
### DFA states and NFA states

<table>
<thead>
<tr>
<th>DFA states</th>
<th>NFA states</th>
<th>A</th>
<th>B</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>15,16,17,8,12</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 and NFA states

<table>
<thead>
<tr>
<th>DFA states</th>
<th>NFA states</th>
<th>A</th>
<th>B</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>15,16,17,8,12</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 and NFA states

<table>
<thead>
<tr>
<th>DFA states</th>
<th>NFA states</th>
<th>A</th>
<th>B</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>15,16,17,8,12</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 and NFA states

<table>
<thead>
<tr>
<th>DFA states</th>
<th>NFA states</th>
<th>A</th>
<th>B</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>15,16,17,8,12</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 recognized by a finite automaton (no pushdown, multi-tape, or Turing machines)
- Two automata 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 automata
- It would be easier to understand nonredundant automata
- Smaller automaton consumes less memory
- The manipulation is faster

Minimization

- A compiler course subject
- Minimization description ([L4_RegExp/min-fa.html](https://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
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/](http://www.jflap.org/), or [http://www.cs.duke.edu/~rodger/tools/jflap/indexol.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
- Immin - the shortest occurrence length (to avoid missing short occurrences)
- \((GA|AAA)*(TA|AG)\) the set of 2-long prefixes is \(\{GA, AA, TA, AG\}\)
- \((AT|GA)(AG|AAA)((AG|AAA)+)\) Immin=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.
**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 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?)