Advanced Algorithmics & Text Algorithms
Regular expressions and automata
Jaak Vilo
2009 Spring

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)
• Regular expression search using a DFA (relative difficulty: medium-hard) (AVS1996, pp. 92–105, 113–146)
• Mapping to NFA
• NFA to DFA
• Matching
• ...

Regular expression
• Definition: A regular expression $RE$ is a string on the set of symbols $\Sigma \cup \{\epsilon, \cdot, *, (, )\}$, which is recursively defined as follows. $RE$ is
  – an empty character $\epsilon$,
  – a character $\alpha \in \Sigma$,
  – $(RE_1)$,
  – $(RE_1 \cdot RE_2)$,
  – $(RE_1 | RE_2)$, 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
  – $(AT | GA) ((AG | AAA)^*) \Rightarrow AT, ATAG, GAAAA, GAAGAAAAA, ...$
  – Infinite set
**Language represented by RE**

**Definition:** A language represented by a regular expression RE is a set of strings over $\Sigma$, which is defined recursively on the structure of RE as follows:
- If $RE$ is $\varepsilon$, then $L(RE) = \{\varepsilon\}$, the empty string.
- If $RE$ is $a \in \Sigma$, then $L(RE) = \{a\}$, a single string of one character.
- If $RE$ is of the form $(RE_1)_T$, then $L(RE) = L(RE_1)_T$, where $ww \in L(RE)$ if $w \in L(RE_1)$ and $w \notin L(RE_2)$. (We call $*$ the concatenation operator or $\cdot$)  
- If $RE$ is of the form $(RE_1) | (RE_2)$, then $L(RE) = L(RE_1) | L(RE_2)$, the union of two languages. (We call $|$ the union operator).
- If $RE$ is of the form $(RE_1^*)$, then $L(RE) = L(RE_1)^*$, where $L^* = \{\varepsilon\}$ and $L^+ = L - L^*$. (We call $^*$ the star operator)

<table>
<thead>
<tr>
<th>RE</th>
<th>Language $L(RE)$</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>${\varepsilon}$</td>
<td>Empty string</td>
</tr>
<tr>
<td>$a \in \Sigma$</td>
<td>${a}$</td>
<td>Single character</td>
</tr>
<tr>
<td>$(RE_1)_T$</td>
<td>$L(RE_1)_T$</td>
<td>Concatenation</td>
</tr>
<tr>
<td>$(RE_1)</td>
<td>(RE_2)$</td>
<td>$L(RE_1)</td>
</tr>
<tr>
<td>$(RE_1)^*$</td>
<td>$L(RE_1)^*$</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.
  - Variations whose values are any patterns defined by a regular expression.
  - An operator which denotes the empty string containing no characters.
  - A symbol which denotes the empty set of strings.
- Operators used in regular expressions include:
  - Concatenation: If $R_1$ and $R_2$ are regular expressions, then $R_1R_2$ (also written as $R_1 \cdot R_2$) is also a regular expression.
  - Union: $R_1 | R_2$ (also written as $R_1 \cup R_2$) is also a regular expression.
  - Star: $R^*$ (also written as $R^*$) is also a regular expression.
  - Kleene closure: $(R_1^*)$ is a regular expression, then $R_1^*$ is also a regular expression.
  - Union has the highest precedence, followed by concatenation, followed by union.

**Matching of RE-s**

- 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.
2.04.2009

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 \( - (q, aw) \rightarrow (q', w) \) if \( \delta(q, a) = q' \), \( w \in \Sigma^* \)
- 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
- \( \delta : Q \times \Sigma \rightarrow P(Q) \) 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 \( - (q, aw) \rightarrow (q', w) \) if \( \epsilon \in \delta(q, a) \), \( a \in \Sigma \cup \{ \epsilon \} \), \( w \in \Sigma^* \)
- Accepted language: \( L(M) = \{ w \mid (q_0, w) \rightarrow^* (q, \epsilon), q \in F \} \)
\[
\begin{align*}
Q &= \{ S_0, S_1, S_2 \} \\
\Sigma &= \{ a, b \} \\
\delta: \text{state} \times \text{char} &\rightarrow \text{state} \\
S_0 &\rightarrow a S_1 \\
S_1 &\rightarrow a S_1 \\
S_1 &\rightarrow b S_2 \\
S_1 &\rightarrow b S_2 \\
S_0 &\rightarrow a S_1 \\
S_1 &\rightarrow a S_1 \\
S_1 &\rightarrow b S_2 \\
S_2 &\rightarrow b S_2 \\
F &= \{ S_2 \}
\end{align*}
\]
Simulation of an NFA
Input: NFA \(M = (Q, \Sigma, \delta, q_0, F)\), Text \(S = s[1..n]\)
Output: States after each character read \(Q_0, Q_1, \ldots, Q_n\)
NB: \(S \in L(M)\) only if \(F \subseteq Q_n\).

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

Regexp \(\rightarrow\) 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

• Symbol \(\epsilon\):

\[
\begin{aligned}
\text{i} & \quad \epsilon \\
\text{j} & \quad \text{i} \\
\end{aligned}
\]

• Terminal symbol \(a\):

\[
\begin{aligned}
\text{i} & \quad a \\
\text{j} & \quad \text{i} \\
\end{aligned}
\]
Union and Concatenation

- $s \mid t$

- $st$

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.

- **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

( A₀, T₁ | G₁ A₁ ) ( ( A₂ G₃ | A₅ A₆ ) * )

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)
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 an NFA as a new state of DFA.

From each state there can be only one transition on a given character.
Fig. 5.6 Thompson automaton construction for the regular expression (A/A)(A/B/A/A/A/A/A).
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 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 automaton 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

Fact. Equivalent states go to equivalent states under all inputs.

Recognizer for (aa | b)*ab(bb)*

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
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

(\texttt{aa | b})^* \texttt{ab(bb)^*}
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 - 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 \(q \to a q'\), 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.
- **Example.**
- 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
- \( l_{\text{im}} \) - the shortest occurrence length (to avoid missing short occurrences)
- \((\text{GA}|\text{AAA})(\text{TA}|\text{AG})\) the set of 2-long prefixes is \{GA, AA, TA, AG\}
- \((\text{AT} | \text{GA})(\text{AG}|\text{AAA})((\text{AG}|\text{AAA})^+)\) \(l_{\text{im}}=6\)
- \{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

Learning languages

Grammaratical 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 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?)

NFA/DFA

- Create an automaton for matching a word approximately
- Allow 0, 1, \ldots, n errors

Approximate search: Problem statement

- Let $S=s_1s_2\ldots s_n \in \Sigma$ be a text and $P=p_1p_2\ldots p_m$ the pattern. Let $k$ be a preassigned constant.
- Main problems
  - $k$ mismatches
    - Find from $S$ all substrings $X$, $|X|=|P|$, that differ from $P$ at max $k$ positions (Hamming distance)
  - $k$ differences
    - Find from $S$ all substrings $X$, where $D(X,P) \leq k$ (Edit distance)
  - best match
    - Find from $S$ such substrings $X$, that $D(X,P)$ is minimal
- Distance $D$ can be defined using one of the ways from previous chapters

Algorithm for approximate search, $k$ edit operations

Input: $P$, $S$, $k$
Output: Approximate occurrences of $P$ in $S$ (with edit distance $\leq k$)

for $j=0$ to $m$ do $h_{j,0} = j$ // Initialize first column
for $i=1$ to $n$ do
  for $i1$ to $n$ do
    $h_{0,i} = 0$
    for $j=1$ to $m$ do
      $h_{j,i} = \min( h_{j-1,i-1} + (\text{if } p_j==s_i \text{ then } 0 \text{ else } 1), h_{j-1,i} + 1, h_{j,i-1} + 1 )$
      if $h_{m,i} \leq k$ Report match at $i$
      Trace back and report the minimizing path (from-to)
Fast matching

- Use bit-parallelism
- I.e. keep a bit vector expressing a set of reachable states
- Manipulate bit vectors by mask vectors based on the input character.
- agrep, shift-or ...

Automaton toolbox

- Roman Tehkov, Kristjan Vedel
- Features: NFA construction from regular expressions (Thompson and Glushkov)
- Determinisation of NFA
- Minimization of DFA
- Constructing automaton allowing errors from input automaton