Formal Languages Part 1
Including Regular Expressions

Basic Concepts for Symbols, Strings, and Languages

Alphabet
A finite set of symbols.

Example:
\( \Sigma_b = \{ 0, 1 \} \) binary alphabet
\( \Sigma_s = \{ A, B, C, ..., Z, Å, Ä, Ö \} \) Swedish characters
\( \Sigma_r = \{ \text{WHILE, IF, BEGIN}, ... \} \) reserved words

String
A finite sequence of symbols from an alphabet.

Example:
10011 from \( \Sigma_b \)
KALLE from \( \Sigma_s \)
WHILE DO BEGIN from \( \Sigma_r \)

Properties of Strings in Formal Languages
String Length, Empty String

Length of a string
Number of symbols in the string.

Example:
x arbitrary string, \(|x|\) length of the string x
\(|10011| = 5\) according to \( \Sigma_b \)
\(|\text{WHILE}| = 5\) according to \( \Sigma_s \)
\(|\text{BEGIN}| = 1\) according to \( \Sigma_r \)

Empty string
The empty string is denoted \( \epsilon \), \(|\epsilon| = 0\)

Properties of Strings in Formal Languages
Concatenation, Exponentiation

Concatenation
Two strings x and y are joined together \( xy = xy \)

Example:
x = AB, y = CDE produce \( xy = ABCDE \)
\(|xy| = |x| + |y|\)
\( xy = yx \) (not commutative)
\( \epsilon \cdot x = x = \epsilon \cdot x \)

String exponentiation
\( x^0 = \epsilon \)
\( x^1 = x \)
\( x^2 = xx \)
\( x^n = xx^{n-1}, n \geq 1 \)

Substrings: Prefix, Suffix

Example:
x = abc

Prefix: Substring at the beginning.
Prefix of x: abc (improper as the prefix equals x), ab, a, \( \epsilon \)

Suffix: Substring at the end.
Suffix of x: abc (improper as the suffix equals x), bc, c, \( \epsilon \)

Languages
A Language = A finite or infinite set of strings which can be constructed from a special alphabet.
Alternatively: a subset of all the strings which can be constructed from an alphabet.
\( \emptyset \) = the empty language. NB! \( \{ \epsilon \} \neq \emptyset \)

Example: \( S = \{ 0, 1 \} \)
\( L_1 = \{ 00, 01, 10, 11 \} \) all strings of length 2
\( L_2 = \{ 1, 01, 11, 001, ..., 111, ... \} \) all strings which finish on 1
\( L_3 = \emptyset \) all strings of length 1 which finish on 01
Closure

- $\Sigma^*$ denotes the set of all strings which can be constructed from the alphabet.

- Closure types:
  - $\star$ = closure, Kleene closure
  - $+$ = positive closure

- Example: $S = \{0,1\}$
  - $\Sigma^* = \{\varepsilon, 0, 00, 01, 10, 11, 101, \ldots\}$
  - $\Sigma^+ = \Sigma^* - \{\varepsilon\} = \{0, 00, 01, 10, 11, 101, \ldots\}$

Operations on Languages

Concatenation

- $L$, $M$ are languages.

- Concatenation operation $\cdot$ (or nothing) between languages
  - $L \cdot M = LM = \{xy | x \in L$ and $y \in M\}$
  - $L(\varepsilon) = (\varepsilon)L = L$
  - $L\emptyset = \emptyset L = \emptyset$

- Example:
  - $L = \{ab, cd\}$, $M = \{uv, yz\}$
  - gives us: $LM = \{abuv, abyz, cduv, cdyz\}$

Exponents and Union of Languages

- Exponents of languages
  - $L^0 = \{\varepsilon\}$
  - $L^1 = L$
  - $L^2 = L \cdot L$
  - $L^n = L \cdot L^{n-1}$, $n \geq 1$

- Union of languages
  - $L$, $M$ are languages.
  - $L \cup M = \{x | x \in L$ or $x \in M\}$
  - Example: $L = \{ab, cd\}$, $M = \{uv, yz\}$
  - gives us: $L \cup M = \{ab, cd, uv, yz\}$

- Closure
  - $L^* = L_0 \cup L_1 \cup \ldots \cup L^\infty$
  - Positive closure
    - $L^+ = L_1 \cup L_2 \cup \ldots \cup L^\infty$
  - $LL^* = L^* - \{\varepsilon\}$, if $\varepsilon$ not in $L$

- Example:
  - $A = \{a, b\}$
  - $A^* = \{a, b, aa, ab, ba, bb, \ldots\}$
  - All possible sequences of $a$ and $b$.

  - A language over $A$ is always a subset of $A^*$.

Closure of Languages

- Regular expressions
  - Regular expressions are used to describe simple languages, e.g. basic symbols, tokens.
    - Example: identifier = letter $\cdot$ (letter $|$ digit)$^*$

  - Regular expressions over an alphabet $S$ denote a language (regular set).

Small Language Exercise
Rules for constructing regular expressions

- S is an alphabet,
- the regular expression r describes the language \( L_r \),
- the regular expression s corresponds to the language \( L_s \), etc.

<table>
<thead>
<tr>
<th>Regular expression r</th>
<th>Language ( L_r )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \epsilon )</td>
<td>{ \epsilon }</td>
</tr>
<tr>
<td>a, a \in S</td>
<td>{ a }</td>
</tr>
<tr>
<td>union: ( (s) \cup (t) )</td>
<td>( L_s \cup L_t )</td>
</tr>
<tr>
<td>concatenation: ( s.t )</td>
<td>( L_sL_t )</td>
</tr>
<tr>
<td>repetition: ( s^* )</td>
<td>( L_s^* )</td>
</tr>
</tbody>
</table>

2.13 TDDD55/B44, P Fritzson, IDA, LIU, 2011.

- Each symbol in the alphabet S is a regular expression which denotes \{a\}.
- * = repetition, zero or more times.
- + = repetition, one or more times.
- concatenation can be left out

<table>
<thead>
<tr>
<th>Priorities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highest</td>
</tr>
<tr>
<td>*</td>
</tr>
<tr>
<td>+</td>
</tr>
<tr>
<td>Lowest</td>
</tr>
</tbody>
</table>

2.14 TDDD55/B44, P Fritzson, IDA, LIU, 2011.

Regular Expression Language Examples

- Examples: \( S = \{a, b\} \)
  1. \( r=a \) \( L_r=\{a\} \)
  2. \( r=a^* \) \( L_r=\{\epsilon, a, aa, aaa, ...\} = \{a\}^* \)
  3. \( r=a+b \) \( L_r=\{a,b\} = \{a\} \cup \{b\} \)
  4. \( r=(a|b)^* \) \( L_r=\{a,b\}^* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, ...\} \)
  5. \( r=(a^*b)^* \) \( L_r=\{a,b\}^* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, ...\} \)
  6. \( r=(a|b)^* \) \( L_r=\{a,b\}^* = \{a, b, ba, baa, baaa, ...\} \)

NB! \( \{a^n b^n \mid n \geq 0\} \) cannot be described with regular expressions.
- \( r=a^*b^* \) gives us \( L_r=\{\epsilon, a^i b^j \mid i,j \geq 0\} \) does not work.
- \( r=(ab)^* \) gives us \( L_r=\{\epsilon, a, ab, aab, ...\} \) does not work.
- Regular expressions cannot "count" (have no memory).

Finite state Automata and Diagrams

- (Finite automaton)
- Assume:
  - regular expression RU = \( ba^*b^* = baa \ldots abb \ldots b \)
  - \( L(RU) = \{ ba^*b^m \mid n, m \geq 1 \} \)

Recognizer

A program which takes a string x and answers yes/no depending on whether x is included in the language.
- The first step in constructing a recognizer for the language \( L(RU) \) is to draw a state diagram (transition diagram).

State Transition Diagram

- state diagram (DFA) for \( ba^*b^n \)

Interpret a State Transition Diagram

- Start in the starting node 0.
- Repeat until there is no more input:
  - Read input.
  - Follow a suitable edge.
- When there is no more input:
  - Check whether we are in a final state. In this case accept the string.
- There is an error in the input if there is no suitable edge to follow.
  - Add one or several error nodes.

Input and State Transitions

- Example of input: \( baab \)
- Then accept when there is no more input and state 3 is an accepting state.
Representation of State Diagrams by Transition Tables

- The previous graph is a DFA (Deterministic Finite Automaton).
- It is deterministic because at each step there is exactly one state to go to and there is no transition marked "ε".
- A regular expression denotes a regular set and corresponds to an NFA (Nondeterministic Finite Automaton).

<table>
<thead>
<tr>
<th>State</th>
<th>Accept</th>
<th>Found</th>
<th>Next state a</th>
<th>Next state b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>no</td>
<td>ε</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>no</td>
<td>b</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>no</td>
<td>ba*</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>yes</td>
<td>ba&quot;b&quot;</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Transition Table
(Suitable for computer representation).

NFA and Transition Tables

Example: NFA for (b|a)* ab

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
<th>Accept</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(0,1)</td>
<td>(0)</td>
<td>no</td>
</tr>
<tr>
<td>1</td>
<td>(2)</td>
<td>(2)</td>
<td>no</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>yes</td>
</tr>
</tbody>
</table>

It requires more calculations to simulate an NFA with a computer program, e.g. for input ab, compared to a DFA.

Transforming NFA to DFA

- Theorem
  - Any NFA can be transformed to a corresponding DFA.
- When generating a recognizer automatically, the following is done:
  - regular expression → NFA.
  - NFA → DFA.
  - DFA → minimal DFA.
  - DFA → corresponding program code or table.

DFA for (b|a)*ab

Small Regular Expression and Transition Diagram/Table Exercise