P. Thiemann, M. Neubauer Wintersemester 2004/2005

Compilerbau

http://proglang.informatik.uni-freiburg.de/teaching/compilerbau/2004/

Übungsblatt 2

Abgabe: 10.11.2004

Aufgabe 1 (Linksquotient von M nach L):

Für zwei beliebige Sprachen L, M definiert $L\backslash M = \{w \mid \exists v \in L \text{ so dass } vw \in M\}$ den *Linksquotienten von M nach L.*

Zeige, dass für zwei reguläre Sprachen R_1 und R_2 die Sprache $R_1\backslash R_2$ ebenso regulär ist.

Aufgabe 2 (Erweiterte reguläre Ausdrücke):

Scannergeneratoren, wie etwa Lex, benutzen als Beschreibungssprache für Tokens in der Regel nicht nur einfache reguläre Ausdrücke, sondern erweitern diese um zusätzliche Operatoren. Die folgende Tabelle zeigt die erweiterte Beschreibungssprache von Lex (wobei c für ein beliebiges Zeichen, r für einen regulären Ausdruck und s für eine Zeichenkette steht):

<table>
<tr><th>Ausdruck</th><th>Bedeutung</th><th>Beispiel</th></tr>
<tr><td>c</td><td>bel. (Nichtsonder-)Zeichen c</td><td>a</td></tr>
<tr><td>\c</td><td>Zeichen c</td><td>*</td></tr>
<tr><td>"s"</td><td>Zeichenkette s</td><td>"**"</td></tr>
<tr><td>.</td><td>bel. Zeichen außer newline</td><td>a.*b</td></tr>
<tr><td>^</td><td>Zeilenanfang</td><td>^abc</td></tr>
<tr><td>$</td><td>Zeilenende</td><td>abc$</td></tr>
<tr><td>[s]</td><td>bel. Zeichen aus der Klasse s</td><td>[abc]</td></tr>
<tr><td>[^s]</td><td>bel. Zeichen nicht aus der Klasse s</td><td>[^abc]</td></tr>
<tr><td>r*</td><td>null oder mehr rs</td><td>a*</td></tr>
<tr><td>r+</td><td>ein oder mehr rs</td><td>a+</td></tr>
<tr><td>r?</td><td>null oder ein r</td><td>a?</td></tr>
<tr><td>r{m,n}</td><td>m bis n Wiederholungen von r</td><td>a{1,5}</td></tr>
<tr><td>r1r2</td><td>erst r1 dann r2</td><td>ab</td></tr>
<tr><td>r1|r2</td><td>r1 oder r2</td><td>a|b</td></tr>
<tr><td>(r)</td><td>r</td><td>(a|b)</td></tr>
<tr><td>r1/r2</td><td>r1 falls gefolgt von r2</td><td>abc/123</td></tr>
</table>

(i) Die Spezialbedeutung der Operatorsymbole (`\ " . ^ [ ] * + ? { } | /`) muss ausgeschaltet werden, falls diese selber übereinstimmen sollen. Es gibt zwei Möglichkeiten, dies zu bewerkstelligen. Falls kein `"`-Zeichen in einer Zeichenkette s vorkommt, kann diese mit `"`s`"` exakt auf Übereinstimmung überprüft werden (etwa in `"**"`). Alternativ können die Zeichen separat mit Hilfe von `\` angegeben werden (etwa in `\*\*`).

Gib einen regulären Ausdruck für Lex an, der die Zeichenkette `"\` beschreibt.

(ii) Eine Zeichenklasse, die mit dem Symbol `^` beginnt, beschreibt eine *Komplementär-Zeichenklasse*. Mit einer Komplementär-Zeichenklasse erkennt man alle Zeichen, die *nicht* in einer bestimmten Zeichenklasse enthalten sind (`[^a]` etwa beschreibt alle Zeichen außer `a`).

Zeige, dass zu jedem regulären Ausdruck mit Komplementär-Zeichenklassen ein äquivalenter regulärer Ausdruck ohne Komplement-Zeichenklassen existiert.

(iii) Der reguläre Ausdruck r`{`m`,`n`}` erkennt m bis n Wiederholungen des Musters r. Beispielsweise erkennt `a{1,5}` Zeichenketten mit ein bis fünf `a`s.

Zeige, dass zu jedem regulären Ausdruck mit Wiederholungsoperator ein äquivalenter regulärer Ausdruck ohne Wiederholungsoperator existiert.

Aufgabe 3 (Ableitungen regulärer Ausdrücke):

In der Vorlesung wurde die Ableitung eines regulären Ausdrucks r bezüglich eines Symbols a, $D(r, a)$, zusammen mit einer Hilfsfunktion E definiert.

Beweise die folgenden Hilfslemmas über Ableitungen regulärer Ausdrücke:

(i) $L(E(r)) = L(r) \cap \{\epsilon\}$

(ii) Für alle $a \in \Sigma$ gilt: $w \in L(D(r, a)) \quad \text{iff} \quad aw \in L(r)$.

Beweise mit Hilfe der beiden Hilfslemmas das Repräsentationslemma für reguläre Sprachen:

(iii) $L(r) = L(E(r)) \cup \bigcup_{a \in \Sigma} aL(D(r, a))$

Aufgabe 4 (SAX-Parser):

SAX[1] ist eine einheitliche Schnittstelle für XML-Parser, die über eine Event-Schnittstelle mit der Außenwelt kommunizieren.

Implementiere in Java[2] einen (vereinfachten) SAX-konformen Parser für XML-Dokumenten mit Hilfe des Scannergenerators `JLex`[3] oder `JFlex`[4].

Gehe etwa nach folgendem "Kochrezept" vor:

(i) Untersuche die SAX-Parser-API-Klasse `DefaultHandler` [5] und finde heraus, welche Art von Tokens Dein Scanner/Parser erkennen muss. (Dein Parser sollte später zumindest sinnvolle Events durch Methodenaufrufe von `characters`, `endDocument`, `endElement`, `fatalError`, `ignorableWhitespace`, `startDocument` und `startElement` generieren können, wobei Attribute eines Elements ignoriert oder als gemeinsamer String behandelt werden können.)

(ii) Untersuche die Spezifikation von XML [6] und definiere für jede Klasse von Tokens einen passenden regulären Ausdruck.

(iii) Implementiere Deinen SAX-Parser, indem Du eine passende JLex-Spezifikation schreibst, wobei die zu den Regeln assoziierten *Actions* passende SAX-Events erzeugen.

---

[1] `http://sax.sourceforge.net/`

[2] Die Aufgabe kann auch in jeder anderen Programmiersprache bearbeitet werden, für die es einen Scannergenerator gibt.

[3] `http://www.cs.princeton.edu/~appel/modern/java/JLex/`

[4] `http://jflex.de/`

[5] `http://sax.sourceforge.net/apidoc/org/xml/sax/helpers/DefaultHandler.html`

[6] `http://www.w3.org/TR/REC-xml`