Theoretische Informatik 2

Tutorium #1 18.4.2002

(Fabian Wleklinski)

Reguläre Ausdrücke

- Unterschied zwischen leerer Sprache und Sprache aus leerem Wort!
  - $R_1 := \emptyset$ $L(R_1) = \{\}$
  - $R_2 := \varepsilon$ $L(R_2) = \{\varepsilon\}$

- In vielen Programmiersprachen:
  - PERL, PHP, Java, C++, VisualBasic, JavaScript, ...
- In vielen Werkzeugen:
  - grep, find, vi, Emacs, WinWord,...
- In SQL-Datenbanken:
  - MSSQL, Oracle, DB/2, MySQL, PostgreSQL, Ingres, ...
- Als „Subtechnologie“:
  - z.B. XML Standard: XPath

Reguläre Ausdrücke

- Implementierung basiert auf einer von drei Varianten:
  - DFA,
  - NFA (z.B. PERL) oder
  - POSIX-NFA.
- Einsatzgebiete:
  - Im Web:z.B. Formularvalidierung, ...!
  - Clientseitig: z.B. Suchfunktionen, Filterregeln, ...

Reguläre Ausdrücke

- Beispiel aus PHP:

```
$datum = "2001-05-12";
if (ereg("([0-9]{4})-([0-9]{1,2})-([0-9]{1,2})",
    $datum, $regs)) {

    echo "$regs[3].$regs[2].$regs[1]";
} else {
    echo "Ungültiges Datumsformat: $datum";
}
```

Reguläre Ausdrücke

- Beispiel aus JavaScript:

```
<html><head><title>Test</title>

<!-- vertauscht Vor- und Nachname eines Namens -->

<script type="text/javascript"><!--
function Ausgabe(Wert) {
    var Ausdruck = /(\w.+)\s(\w.+)/;
    Ausdruck.exec(Wert);
    alert(RegExp.$2 + ", " + RegExp.$1);
}
//-->
</script>
...
```

Reguläre Ausdrücke

- >= JDK 1.4:
  - `Package java.util.regex`
- < JDK 1.4:
  - z.B: http://jakarta.apache.org/regexp/index.html
- Beispiel:

```
public static void main(String[] args) {
    String input = "Test für Regex 1xxx2 n444n.";
    Pattern p = Pattern.compile("\\d\\D+\\d");
    Matcher m = p.matcher(input);
    if (m.find()) {
          System.out.println("Pos. " + m.start());
          System.out.println("Text: "+m.group());
    }
}
```

Reguläre Ausdrücke

- Ein Mechanismus/Formalismus, um reguläre Sprachen (d.h. Mengen) zu beschreiben!
- Es gibt weitere Mechanismen!
  - natürlichsprachlich
  - Grammatiken (siehe Backus-Naur-Form)
  - aussagenlogische Prädikate
  - ...

Reguläre Ausdrücke

- Beschränkt!
- Z.B. gegebener XHTML-Code
-
```
<body>
    <p>
          <b>Franz</b> fühlt sich unheimlich
          <b>fettgedruckt</b>!
    </p>
</body>
```
- Wie nach beliebigen Elementen suchen?
- => kontextfreie Sprachen!

Reguläre Ausdrücke

- Aber warum doch REGEXP, wenn beschränkt?
  - relativ leicht implementierbar
  - relativ intuitive Syntax
    - verglichen mit anderen Notationen... J
  - Endliche Automaten ohne Speicher und ohne Stack können Wortproblem für alle (!) reguläre Sprachen in Linearzeit lösen!

Aufgabe 1.1 – Divison by Zero Bugs

- „Gibt es eine Eingabe, für die durch Null dividiert wird?“
- Nur 1 Division im Code: „q=m/zaehler“
  - „n % i“dividiert auch, aber i kann nicht 0 werden!
- „Gibt es eine Eingabe, für die „zaehler“ Null wird?“

  ⇒ was ist der „zaehler“?
- „zaehler“ ist Anzahl gefundener Primzahlpaare
- „Gibt es eine Eingabe n, für die sich 2n nicht als Summe zweier Primzahlen darstellen lässt?
  - => Goldbach‘sche Vermutung: Solche Eingaben gibt es nicht!
- Compiler müsste die Goldbach‘sche Vermutung widerlegen oder bestätigen!

Aufgabe 1.1 – Divison by Zero Bugs

- Goldbachsche Vermutung:
  - „Jede gerade Zahl > 2 lässt sich als Summe zweier Primzahlen ausdrücken.“
  - Christian Goldbach: preußischer Mathemat.
  - 1742 in pers. Brief an Euler formuliert
  - brit. Verlag „Faber and Faber“ hat 1 Mio. $ für Beweis der Goldbachschen Vermutung ausgelobt
  - für alle Zahlen unter 400 Billionen bewiesen!
  - unbezweifelt, aber seit 250 Jahren unbewiesen!

Aufgabe 1.1 – Divison by Zero Bugs

- Deprimierend:
  - Nach Kurt Gödel: nicht jeder wahre Satz der Mathematik ist beweisbar! J
- Konsequenzen für Entwicklungsabteilung
  - Ziel mit an Sicherheit grenzender Wahrscheinlichkeit nicht erreichbar!
- Quintessenz:
  - Komplexität vererbt sich von Problem zu Problem!
  - Zurückführung andere Probleme, an denen sich Andere bereits die Zähne ausgebissen haben, spart viel sinnlose Arbeit!!!

Aufgabe 1.2 – Überflüssige Programmteile

- „Gibt es eine Anweisung im Programmcode, die nicht erreicht wird?“
  - Nehme Code aus 1.1
  - ersetze „q=m/zaehler“:

```
if (zaehler==0){
 printf(„Ich werde
 berühmt!“);
}
```

- Problem lässt sich auch auf andere Probleme zurückführen
- z.B. auf den großen Fermat:
  - $x^n+y^n=z^n$ für n>2 ist nicht für alle Zahlen lösbar
  - Beweis erst Jahrhunderte später (1995) vollbracht

Aufgabe 1.3 – Arbeiten mit formalen Sprachen

- $L_1=(\{a\}^*\bullet\{b\}^*\bullet\{c\}^+)^*$
  z.B. abc,ac,bc,c
- $L_2=(\{a,b\}^*\bullet\{c\}^+)^*$
  z.B. bac,abababcc
- $L_3=((\{a\}^*\bullet\{c\}^+) \cup (\{b\}^*\bullet\{c\}^+))^*$
  z.B. a..c..,b..c..
- $\Rightarrow$ Vermutung:
  $L_3 \subset L_1 \subset L_2$

- Beweis von $L_1 \subseteq L_2$:
  - für jedes w aus L_1 gilt:
    - $w=w_1\bullet w_2\bullet w_3\bullet\ldots\bullet w_n$
  - mit w_i‘s der Form:
    - $\{a\}^*\bullet\{b\}^*\bullet\{c\}^+$
  - Für jedes w_i gilt:
    - $w_i= w_{i,1}\bullet w_{i,2}\bullet w_{i,3}$
  - so dass:
    - $w_{i,1}$ aus $\{a\}^*$
    - $w_{i,2}$ aus $\{b\}^*$
    - $w_{i,3}$ aus $\{c\}^+$

Aufgabe 1.3 – Arbeiten mit formalen Sprachen

- $\{a\} \subseteq \{a,b\}$, $\{b\} \subseteq \{a,b\}$
  $\Downarrow$
  $\{a\}^* \subseteq \{a,b\}^*$, $\{b\}^* \subseteq \{a,b\}^*$
- $w_{i,1}$ und $w_{i,2} \in \{a,b\}^*$
- $w_{i,1} \bullet w_{i,2} \in \{a,b\}^*$
- Daher gilt für alle w_i:
  - $w_i = w_{i,1} \bullet w_{i,2} \bullet w_{i,3}$
  - $w_i \in \{a,b\}^* \bullet \{c\}^+$
- $w = w_1 \bullet w_2 \bullet w_3 \bullet \ldots \bullet w_n$
  $\Downarrow$
  $w \in (\{a,b\}^* \bullet \{c\}^+)^* = L_2$
- $L_1 \subseteq L_2$ gezeigt!
- Auch $L_1 \subset L_2$??? Ja!!!
  - $bac \in L_2$
  - $bac \notin L_1$
- ◊

Aufgabe 1.3 – Arbeiten mit formalen Sprachen

- Beweis von $L_3 \subseteq L_1$:
  - wieder für w aus L_3:
    - $w = w_1 \bullet w_2 \bullet w_3 \bullet \ldots \bullet w_n$
  - mit w_i's der Form:
    - $(\{a\}^* \bullet \{c\}^+) \cup (\{b\}^* \bullet \{c\}^+)$
  - also auch aus
    - $\{a\}^* \bullet \{b\}^* \bullet \{c\}^+$
  - also gilt für w:
    - $w \in (\{a\}^* \bullet \{b\}^* \bullet \{c\}^+)^*$
  - $L_3 \subseteq L_1$ gezeigt!
  - Auch $L_3 \subset L_1$??? Ja!!!
    - $abc \in L_1$
    - $abc \notin L_3$
- ◊
- Gesamtbeweis:
  - $L_1 \subset L_2$ √
    $L_3 \subset L_1$ √
    $\Downarrow$
    $L_3 \subset L_1 \subset L_2$
- ◊

Aufgabe 1.4 – Finden regulärer Ausdrücke

- $\Sigma=\{a,b,c\}$
- Ausdruck für L_1
  - Wörter über Σ, welche „abc“ enthalten!
    - z.B. cbac**abc**ccab
  - „abc enthalten“ = „irgendwas - abc - irgendwas“
  - $L_1=L(R)$ mit
  - $R = (a+b+c)^* \bullet a\bullet b\bullet c \bullet (a+b+c)^*$
- Ausdruck für L_2
  - Wörter über Σ, welche „abc“ **nicht** enthalten!
  - REGEXP bieten keinen Komplement-Operator
    - Aber: für jedes Komplement eines REGEXP gibt es einen REGEXP!
  - Idee: Zerlege alle Wörter aus L in Teilwörter!

Aufgabe 1.4 – Finden regulärer Ausdrücke

- L‘ sei die Menge aller Wörter über {a,b,c}, die
  1. auf „c“ enden, und
  2. davor nur a‘s und b‘s haben
  3. „abc“ nicht enthalten!
     $\Rightarrow$ kein „ab“ vorm „c“!
- Behauptung:
  - $L=(L‘)^* \bullet \{a,b\}^*$
- Schritt 1: Beweis von $L \supseteq (L‘)^* \bullet \{a,b\}^*$
  - kein Wort aus L‘ enthält „abc“ (per Def.)
  - weil jedes Wort aus L‘ auf „c“ endet $\Rightarrow$ kein Wort aus $(L‘)^*$ enthält „abc“ – auch nicht über Wortgrenzen hinaus!
  - also ist jedes Wort aus $(L‘)^* \bullet \{a,b\}^*$ auch in L

Aufgabe 1.4 – Finden regulärer Ausdrücke

- Schritt 1: Beweis von $L \subseteq (L')^* \bullet \{a,b\}^*$
  - Jedes Wort w aus L ist zerlegbar in
    - einige auf „c“ endende Teilwörter, sowie
    - einen „c“ nicht enthaltenden Rest
    - $w=w_1 \bullet w_2 \bullet w_3 \bullet \ldots \bullet w_n$
  - Teilwörter w_i aus L‘, letztes Teilwort aus $\{a,b\}^*$ $\Rightarrow$ w liegt in $(L')^* \bullet \{a,b\}^*$
- Behauptung bewiesen: $L=(L')^* \bullet \{a,b\}$!!!
- ◊
- Regulärer Ausdruck:
  - Wörter aus L‘ sind
    - „c“ oder
    - „bc“
  - bzw. enden auf
    - „ac“ oder
    - „bbc“

Aufgabe 1.4 – Finden regulärer Ausdrücke

- REGEXP für L‘
  - L‘=L(R) mit
  - $R=(c+bc)+((a+b)^* \bullet ((ac)+(bbc)))$
- REGEXP für L insgesamt:
  - L=L(R) mit
  - $R=[(c+bc)+((a+b)^* \bullet ((ac)+(bbc)))]^* \bullet (a+b)^*$
- „ausklammern“ des „c“ ergibt:
  - $R=[(\varepsilon+b)+((a+b)^* \bullet ((a)+(bb))) \bullet c]^* \bullet (a+b)^*$
- ◊

Aufgabe 1.4 – Finden regulärer Ausdrücke

- Leichte Sprachen erfordern manchmal komplizierte REGEXP!
  - oder Beschreibungen allgemein!
- Jeder Formalismus hat Stärken u. Schwächen
- DFA ist für beide Sprachen banal!
  - DFAs machen Sinn! J
- Wie vollführt man das Komplement algorithmisch?
  - REGEXP => DFA
  - Komplement bilden
  - DFA => REGEXP
- Konvertierung von DFAs/NFAs/REGEXP macht Sinn!