Technische Universität München
Fakultät für Informatik
Dr. H. Ehler, Dr. M. Pizka,
W. Prenninger, Dr. M. Schneider

WS 2001
Lösungsvorschläge zu Blatt 7
29. November 2001

Übungen zu Einführung in die Informatik I

Aufgabe 27 Termersetzung: Terminierung (Lösungsvorschlag)

a) Der Einfachheit halber werden wieder nur Grundterme betrachtet, die aus `empty`, `make` und `conc` aufgebaut sind. Die ersten drei Regeln aus Aufgabe 20 zur Überführung von Konstruktortermen in Normalform sind:

```
1. conc(empty, y) → y
2. conc(x, empty) → x
3. conc(conc(x, y), z) → conc(x, conc(y, z))
```

Wir definieren $g : \{t \in W_{\text{SEQ}} : t^{\mathbf{SEQ}} \neq \perp\} \to \mathbb{N}$ wie folgt:

```
g("empty") = 0,
g("make(a)") = 1,
g("conc(t1,t2)") = 1+2*g("t1")+g("t2"),
```

Sei $l \to r$ eine der obigen drei Regeln, so gilt $g(l) > g(r)$:

```
1. g("conc(empty, y)") = 1+2*0+g("y") > g("y")
2. g("conc(x, empty)") = 1+2*g("x")+0 > g("x")
3. g("conc(conc(x,y), z)")
   = 1+2*(1+2*g("x")+g("y"))+g("z")
   = 3+4*g("x")+2*g("y")+g("z")
   > 2+2*g("x")+2*g("y")+g("z")
   = 1+2*g("x")+1+2*g("y")+g("z")
   = g("conc(x, conc(y,z))")
```

b) Aus der Existenz dieser Bewertungsfunktion mit der Eigenschaft $g(l) > g(r)$ für alle Regeln $l \to r$ in R folgt die Terminierung des Termersetzungssystems für beliebige Terme $t \in W_{\text{SEQ}}$ mit $t^{\mathbf{SEQ}} \neq \perp$, da höchstens $g(t)$ Mal eine Regel anwendbar ist.

Anmerkung: Durch folgende Erweiterung der Definition von g läßt sich sogar zeigen, daß das gesamte Termersetzungssystem von Aufgabe 20 für beliebige Terme $t \in W_{\text{SEQ}}$ mit $t^{\mathbf{SEQ}} \neq \perp$ immer terminiert:

```
g("true") = 1,
g("false") = 1,
g("t1 ∧ t2") = g("t1") + g("t2"),
g("t1 =? t2") = 1,
g("kompl(t1,t2)") = 2+g("t1")+g("t2")
```

Für die totale Korrektheit von R wäre überdies noch zu zeigen, daß für beliebige verschiedene terminale Grundterme t_1, t_2 mit $t_1^{\mathbf{SEQ}} \neq \perp$, $t_2^{\mathbf{SEQ}} \neq \perp$ stets gilt $t_1^{\mathbf{SEQ}} \neq t_2^{\mathbf{SEQ}}$.

Aufgabe 28 BNF (Lösungsvorschlag)

a) <ab_sprache> ::= <A> | <B>
<A> ::= a <B> | a
<B> ::= b <A> | b

b) In der angegebenen BNF-Regel existiert lediglich genau ein Nonterminal: <K>, d.h. es ist lediglich X_j für $j = 1$ von Interesse. Wir verzichten daher bei X_j^i und R_j^i auf den unteren Index.

$$\begin{array}{lll lll l}
R^0 & ::= & \{\} & X^0 & = & \emptyset & \\
R^1 & ::= & \{(<R^0>)\}^* & & & & \\
 & ::= & \{(\{\})\}^* & & & & \\
 & ::= & \varepsilon & X^1 & = & X^0 \cup \{\varepsilon\} & \\
R^2 & ::= & \{(<R^1>)\}^* & & & & \\
 & ::= & \{(\varepsilon)\}^* & X^2 & = & X^1 \cup \{(), ()(), \ldots\} & \text{Klammergebirge bis Tiefe 1} \\
R^3 & ::= & \{(<R^2>)\}^* & X^3 & = & X^2 \cup \{(()), (()()), \ldots\} & \text{Klammergebirge bis Tiefe 2}
\end{array}$$

D. h. in der Iteration $i = n$ erhalten wir den regulären Ausdruck R^n mit der zugehörigen Sprache X^n: Klammergebirge bis Schachtelungstiefe $n-1$.
Mit $X = \cup_{i \in \mathbb{N}} X^i$ ergibt sich daraus allgemein: Die angegebene Regel <*K*> beschreibt die formale Sprache der Klammergebirge beliebiger Schachtelungstiefe.

c) Als “korrekt geklammerte” boolesche Terme akzeptieren wir i.d.R. sowohl Zeichensequenzen, deren Aufbau entweder der induktiven Definition boolescher Terme (s. Vorlesung) entspricht als auch die vereinfachende klammerfreie Schreibweise (wobei wir dann bei der Interpretation die Bindungsstärken berücksichtigen müssen) und zusätzlich auch boolesche Terme mit zusätzlichen (unnötigen) Klammerpaaren.

d) <BT> ::= true | false | <ID> | <BT> <binop> <BT> | ¬ <BT> | (<BT>)
<ID> ::= x | y | z
<binop> ::= ∨ | ∧ | ⇒ | ⇔

Aufgabe 29 Beschreibung von HTML-Dokumenten mit BNF (Lösungsvorschlag)

a) Es folgt der Anfang einer vollständigen Ableitung:

<HTML-Dokument> = `<html>` <Dokumentinhalt> `</html>`
= `<html><head>` <Kopf> `</head>`
`<body>` <Rumpf> `</body></html>`
= `<html><head><title>` <Text> `</title></head>`
`<body>` <Rumpf> `</body></html>`
= `<html><head><title> Informatik I </title></head>`
`<body>` <Rumpf> `</body></html>`
⋮ ⋮ ⋮
= `<html><head><title> Informatik I </title></head>`
`<body>` <Überschrift><Überschrift>
<Absatz><Absatz><Absatz><Absatz>`</body></html>`
⋮ ⋮ ⋮

Als weiteres Beispiel greifen wir die Ableitung des dritten Absatzes heraus:

<Absatz> = <p> <HTMLtext><HTMLtext><HTMLtext> </p>
<p> <HTMLtext>
<HTMLtext><HTMLtext> </p>
<p> <Text>
<Hyperlink> <Text> </p>
<p> <Text>
<a href="<URL>"> <Text> <Text> </p>
= <p>Übungen:

<a href="http://www4.in.tum.de/lehre/vorlesungen
/info11ws01/zentraluebung.html">Zentralübung:
Freitag 12:15-13:00, Hörsaal 1200 </p>

Der vollständige HTML-Quelltext lautet:

```
<html>
<head>
<title>Informatik I</title>
</head>
<body>
<h1>Einführung in die Informatik I</h1>
<h2>Wintersemester 2001/2002</h2>
<p>Bereich:
<br>Vorlesung im Grundstudium für Diplom- und Bachelor-Studiengang
</p>
<p>...</p>
<p>Übungen:
<br><a href="http://www4.in.tum.de/lehre/vorlesungen/info11ws01/
zentraluebung.html">Zentralübung:</a>
Freitag 12:15-13:00, Hörsaal 1200
</p>
<p>...</p>
</body>
</html>
```

b) Wir modifizieren die Menge der BNF-Regeln wie folgt:

<Rumpf> ::= { <Überschrift> | <Absatz> | <Aufzählung> }*
<Aufzählung> ::= { <HTMLtext> }*

Aufgabe 30 Pattern matching (Lösungsvorschlag)

a)
```
isEmpty :: [m] -> Bool
isEmpty [] = True
isEmpty _  = False

rest :: [m] -> [m]
rest [] = undefined
rest (x:xs) = xs
```

b)
```
data List a = Empty | Append (a, List a)

istleer :: List a -> Bool
istleer Empty = True
istleer _ = False
```

```
erstesElement :: List a -> a
erstesElement Empty = undefined
erstesElement (Append (x, l)) = x

verbinde :: (List a, List a) -> List a
verbinde (Empty, l) = l
verbinde (Append (a, s), l) = Append(a, verbinde(s, l))
```

c)
```
data Datum = Date (Int, Int, Int)

isdef :: Datum -> Bool
isdef (Date (d, m, y)) = d > 0 && m > 0 && m < 13 &&
                (((elem m [1, 3, 5, 7, 8, 10, 12]) && (d< 32)) ||
                 ((elem m [4, 6, 9, 11]) && (d < 31)) ||
                 ((m==2 && (mod y 4)==0) && (d < 30)) ||
                 ((m==2 && not((mod y 4)==0)) && (d < 29))
                )

nextDay :: Datum -> Datum
nextDay t@(Date (d, m, y))
       | (isdef t) && (isdef (Date (d+1, m, y))) = Date (d+1, m, y)
       | (isdef t) && (isdef (Date (1, m+1, y))) = Date (1, m+1, y)
       | (isdef t)  = Date (1, 1, y+1)

{- Testtage fuer die Funktion nextDay -}

normalTag = Date (15, 4, 2001)
monatEnde = Date (31, 5, 2001)
silvester = Date (31, 12, 2000)
nichtSchalt = Date (28, 2, 1901)
schalt = Date (28, 2, 2000)
```