SE1 – Freiwillige Zusatzaufgaben

Christian Rehn

5. Dezember 2010

Hinweise

- Zu diesen Aufgaben wird es keine Musterlösung und keine Korrektur geben. Ihr könnt mich aber jederzeit fragen.
- Ich hab die Aufgaben weder selbst gelöst, noch größer durchdacht. Die meisten Aufgaben sollten nicht besonders schwer sein, jedoch kann ich nicht ausschließen, dass ich mich in der Schwierigkeit verschätzt habe oder eine Formulierung missverständlich ist. Fragen und Hinweise zu den Aufgaben sind jederzeit gerne willkommen.
- Die Aufgaben hier bilden nicht alle Aufgabentypen der Klausur ab. Ich habe mich hier auf Grammatiken und Rekursion konzentriert ohne die Aufgabenstellung jeweils zu variieren. Um einen besseren Eindruck von den Klausuraufgaben zu bekommen, solltet ihr euch Altklausuren ansehen und natürlich die Probeklausur mitschreiben.

Teil 1 – Grammatiken

Aufgabe 1: Mailadresse, URL und IP

Insbesondere bei Webanwendungen ist es wichtig, Eingaben zu validieren, d. h. zu überprüfen, ob der vom User angegebene String wirklich eine Mailadresse, eine URL, ein Name oder was auch immer ist. Ansonsten baut man sich dabei wunderschöne Sicherheitslücken[1] mit denen man Passwörter stehlen, die Datenbank verändern und ähnliche Späße treiben kann.

Häufig verwendet man hierzu reguläre Ausdrücke[2]. Hier tun wir das aber mal mit Hilfe von kontextfreien Grammatiken.

a) Definiere eine kontextfreie Grammatik, die die Sprache der Mailadressen beschreibt. Gehe davon aus, dass die Top-Level-Domain (de, com, net) immer zwei oder drei Stellen (Buchstaben, keine Zahlen) hat[3]. In Benutzername und Domain dürfen Buchstaben,

[1] SQL Injection

[2] http://de.wikipedia.org/wiki/Regular_Expression; Diese sind generell sehr hilfreich; guckt euch das bei Gelegenheit mal an.

[3] Obwohl es auch .info, .name und .museum gibt

Zahlen, der Unterstrich und der Bindestrich vorkommen. Sonderregeln wie die Kodierung von Umlauten können ignoriert werden.

b) Definiere eine Kontextfreie Grammatik, die die Sprache der URLs beschreibt. Zum generellen Aufbau von URLs findest du Informationen in der Wikipedia. Auch hier können Sonderregeln ignoriert werden.

c) Eine IPv4-Adresse ist eine 32 bit-Zahl, die meist in vier Blöcken zu je einem Byte geschrieben werden (z. B. `127.0.0.1`). Schreibe eine Grammatik, die die Sprache der IP-Adressen definiert.

Aufgabe 2: Nummernschilder

a) Definiere eine Kontextfreie Grammatik, die die Sprache der deutschen KFZ-Kennzeichen[4] beschreibt. Gehe dabei davon aus, dass das Unterscheidungszeichen (KL für Kaiserslautern, MZ für Mainz, etc.) aus zwei oder drei Buchstaben besteht und ignoriere Sonderregeln.

b) Die KFZ-Kennzeichen bieten diverse Möglichkeiten um weitere Grammatiken zu definieren. Saisonkennzeichen, Kennzeichen der Bundes- und Landesorgane, Rote Kennzeichen, Oldtimer-Kennzeichen, Kennzeichen in anderen Ländern u. v. m.

Teil 2 – Rekursion mit Haskell

Aufgabe 3: Kontrabass

Im Kinderlied „Drei Chinesen mit dem Kontrabass“[5] werden alle Vokale (und Umlaute) nacheinander durch a, e, i, o und u ersetzt. Schreibe eine Funktion, die das erledigt:

```
kontrabass 'a' "Drei Chinesen mit dem Kontrabass"
    == "Draa Chanasan mat dam Kantrabass"
kontrabass 'u' "saßen auf der Straße und erzählten sich was"
    == "sußun uuf dur Strußu und urzuhltun such wus"
```

Aufgabe 4: Geheimschrift

Eine Möglichkeit der Unkenntlichmachung von Texten ist das Entfernen und neu Setzen der Leerzeichen:

```
"Das kann jeder problemlos lesen."
   => "Da ska nnje derpr obl eml osle sen."
```

[4] http://de.wikipedia.org/wiki/Kfz-Kennzeichen_(Deutschland)
[5] http://de.wikipedia.org/wiki/Drei_Chinesen_mit_dem_Kontrabass

a) Schreibe eine Funktion `trim :: String -> String`, die aus einem gegebenen String alle Leerzeichen entfernt.

```
trim "Das kann jeder problemlos lesen." == "Daskannjederproblemloslesen."
```

b) Schreibe eine Funktion `insertSpaces :: Int -> String -> String`, die ein Wort ohne Leerzeichen nimmt und Leerzeichen in regelnmäßigen Abständen einfügt:

```
insertSpaces 4 "Daskannjederproblemloslesen." == "Dask annj eder prob
    leml osle sen."
```

c) Schreibe eine Funktion `countCharsBetweenSpaces :: String -> [Int]`, die ein Wort nimmt und jeweils die Zeichen zwischen den Leerzeichen zählt:

```
countCharsBetweenSpaces "Das kann jeder problemlos lesen." = [3, 4, 5,
    10, 6]
```

d) Schreibe eine Funktion `insertSpacesByList :: [Int] -> String -> String`, die eine Liste mit Abständen und ein Wort ohne Leerzeichen nimmt und Leerzeichen in den gegebenen Abständen einfügt. Eventuell überzählige Abstände in der Liste sollen ignoriert werden.

```
insertSpacesByList [2, 4, 2, 7, 5, 3] "Daskannjederproblemloslesen." == "
    Da skan nj ederpro bleml osl esen."
```

e) Schreibe eine Funktion `rotateSpaces :: Int ->String -> String`, die die Leerzeichen in einem String um eine gewisse Anzahl Zeichen rotiert.

```
rotateSpaces 7 "Das kann jeder problemlos lesen." == "D askannjed erpr
    oblem loslesen."
```

Aufgabe 5: Hauptbahnhof

a) Schreibe eine Funktion `split :: String -> Char -> [String]`, die einen String an dem gegebenen Zeichen trennt und die einzelnen Wörter in einer Liste zurück gibt.

```
split "Mit dem Transrapid braucht man vom Hauptbahnhof zum Flughafen nur
    zehn Minuten." ' ' == [["Mit"], ["dem"], ["Transrapid"], ["braucht"],
    ["man"], ["vom"], ["Hauptbahnhof"], ["zum"], ["Flughafen"], ["nur"], [
    "zehn"], ["Minuten."]]
```

b) Schreibe eine Funktion `glue :: [String] -> Char -> String` die aus einer Liste vom Strings und einem Verbindungszeichen wieder einen zusammenhängenden String macht.

```
glue [["Mit"], ["dem"], ["Transrapid"], ["braucht"], ["man"], ["vom"], ["
    Hauptbahnhof"], ["zum"], ["Flughafen"], ["nur"], ["zehn"], ["Minuten."
    ]] ' ' == "Mit dem Transrapid braucht man vom Hauptbahnhof zum
    Flughafen nur zehn Minuten."
```

c) Schreibe eine Funktion `zerstoib :: String -> String`, die einen String folgendermaßen verändert:

- Nach jedem dritten Wort wird ein „ähh...“ eingefügt.
- Das Wort „Transrapid“ wird durch „Hauptbahnhof“ ersetzt.
- Das Wort „Hauptbahnhof“ wird ersetzt durch „Flughafen ... am am Hauptbahnhof in München“.
- Jedes zweite „in“ wird durch „mit“ ersetzt.

```
zerstoib "Sie steigen in den Trasrapid ein und fahren in zehn Minuten zum
    Flughafen. Dann starten Sie im Grunde genommen am Hauptbahnhof Ihren
   Flug."
    == "Sie steigen in ähh... den Hauptbahnhof ein ähh... und fahren mit
       ähh... zehn Minuten zum ähh... Flughafen. Dann starten ähh... Sie
       im Grunde ähh... genommen am Flughafen ... am am Hauptbahnhof in M
       ünchen ähh... Ihren Flug."
```

Aufgabe 6: Rekursion auf Binärbäumen

a) Definiere eine Datenstruktur `BinTree a`, die Binärbäume darstellt, die an Blättern und inneren Knoten markiert sind.

b) Schreibe eine Funkion `sumTree :: BinTree Int -> Integer`, die alle Markierungen eines Baumes addiert.

c) Wie muss die Funktion abgewandelt werden, wenn nur die Blätter (oder nur die inneren Knoten) aufsummiert werden sollen?

d) Schreibe eine Funktion `countEdges :: BinTree a -> Integer`, die die Anzahl der Kanten in einem Baum berechnet.

e) Schreibe eine Funktion `treeToList :: BinTree Int -> [Int]`, die einen Baum „flachklopft“, d. h. eine Liste zurück liefert, die alle Markierungen des Baumes von links nach rechts[6] enthält.

f) Schreibe eine Funktion `isSearchTree :: BinTree -> Bool`, die einen Binärbaum auf die Suchbaumeigenschaft überprüft.

g) Schreibe eine Funktion `mapTree :: (a -> b)-> BinTree a -> BinTree b`, die `map` auf Binärbäumen definiert.

[6] in-order; siehe http://de.wikipedia.org/wiki/In-order#Traversierung

Aufgabe 7: Quicksort mit Filter

In der Vorlesung würde der QuickSort-Algorithmus behandelt. Mit Hilfe der Funktion `filter` lässt sich dieser Algorithmus noch kompakter und übersichtlicher definieren. Tu das.

Teil 3 – Kleinkram

Aufgabe 8: Pattern Matching

a) Schreibe deine Lösungen für die Aufgaben von Blatt 4 mit Hilfe von Pattern-Matching.

b) Schreibe deine Lösungen für die Aufgaben von Blatt 5 mit Hilfe von Pattern-Matching (solltest du das nicht schon getan haben).

Aufgabe 9: Vereinfachung von Bool-Ausdrücken

Das war nicht explizit Teil der Vorlesung und ist demnach auch nur bedingt klausurrelevant. Jedoch könnte das euch helfen übersichtlichere Programme zu schreiben und so weniger Fehler zu machen.

Vereinfache die folgenden Ausdrücke:

```
a = if x > y then True else False
b = if x > y
       then if x > z
           then 42
           else if x /= z
               then 17
               else 42
       else 17
c = (x > y) == True
d = (x > y) == False
e = (x > y) || (even y) || False
f = (x /= y) && (x /= z) && True == False
g = (x > y) && (y > z) && (z >= w) && (y /= w) && (w >= x)
h = if (prime x) == True then False else True
```

Aufgabe 10: Einrücken

Das war nicht explizit Teil der Vorlesung und ist demnach auch nur bedingt klausurrelevant. Jedoch könnte das euch helfen übersichtlichere Programme zu schreiben und so weniger Fehler zu machen.

Rücke den folgenden Code übersichtlich ein:

```
somefunc x y z = if x > y then if y > z then
   let a = 2 * pi * z in 6 * a + otherfunc a else 42
   else if isThursday then negate 13 else 12
```