Prof. Dr. Ralf Hinze
Cass Alexandru, M.Sc.
Alexander Dinges, M.Sc.
Sebastian Schloßer, M.Sc.

RPTU
Fachbereich Informatik
AG Programmiersprachen

Übungsblatt 8: Grundlagen der Programmierung (WS 2024/25)

Ausgabe: 17. Dezember 2024
Abgabe: 06./07./08. Januar 2025, siehe Homepage

Sprechstunden zu den Übungen Sie haben Schwierigkeiten mit den Übungsaufgaben und machen sich Sorgen, dass es Ihnen nicht gelingen wird die zur Klausurzulassung nötigen 60% der erreichbaren Punkte zu erlangen?

Dann besuchen Sie unsere Sprechstunden zu den Übungen! Dort erhalten Sie Tipps und Lösungshinweise, wenn Sie mit einer Aufgabe nicht weiterkommen. Sie können dort auch zu früheren Aufgaben Fragen stellen. Alle Informationen zu den Übungssprechstunden finden Sie auf unserer Homepage.

Aufgabe 1 Reguläre Ausdrücke (Präsenzaufgabe)

Motivation: In dieser Aufgabe sollen Sie sich mit regulären Ausdrücken beschäftigen. Die Aufgabe soll Ihnen dabei helfen den Weg von einem regulären Ausdruck schrittweise bis zu einer Akzeptorfunktion nachzuvollziehen. Sie können sich an den Vorlesungsfolien 553 bis 623 sowie am Skript Kapitel 6.1 und 6.2 orientieren.

Wir betrachten den regulären Ausdruck `b*a` über dem Alphabet Alphabet $A = \{\mathsf{a}, \mathsf{b}\}$.

a) Bestimmen Sie **alle** Rechtsfaktoren (inkl. Rechtsfaktoren der Ergebnisse). Geben Sie dabei in der Rechnung jeweils den ersten Schritt explizit an, nachfolgende Zwischenschritte dürfen Sie zusammenfassen.

b) Zeichnen Sie den Aufrufgraphen für den Akzeptor (wie auf Vorlesungsfolie 599).

   Umranden Sie Ausdrücke, die nullable sind, doppelt. Wenn wir beim Einlesen eines Wortes an einem solchen nullable Ausdruck landen und keine weitere Eingabe mehr folgt, gehört das eingelesene Wort zur durch den regulären Ausdruck beschriebenen Sprache. Durch die doppelte Umrandung können wir einfacher ablesen, dass wir an einem möglichen Ende angekommen sind (daher werden solche Knoten auch „Endzustände“ genannt).

c) Implementieren Sie die Akzeptorfunktionen. Gehen Sie dabei **streng nach dem Verfahren aus der Vorlesung** vor (Folie 600). Nutzen Sie für das Alphabet den Typ `type Alphabet = | A | B`.

   Hinweis: Wir empfehlen die einzelnen Akzeptorfunktionen als verschränkt rekursive Hilfsfunktionen innerhlab von `accept` *zu definieren und am Ende die Start-Akzeptorfunktion mit der Eingabe aufzurufen.*

Aufgabe 2 Weihnachtsbaum schmücken (Einreichaufgabe, 11 Punkte)

Motivation: In dieser Aufgabe arbeiten Sie in weihnachtlicher Stimmung mit Bäumen und Listen.

Schreiben Sie Ihre Lösungen in die Datei `Weihnachtsbaum.fs` *aus der Vorlage* `Aufgabe-8-2.zip`.

Wir betrachten folgenden Typ für Bäume:

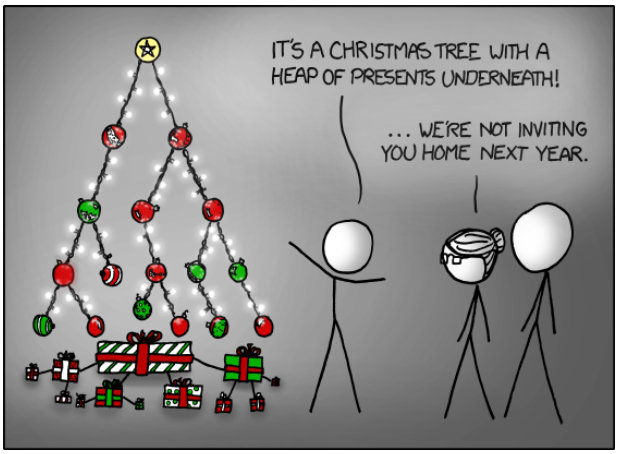
```
type Tree<'a> =
    | Leaf
    | ENode of Tree<'a> *       Tree<'a>
    | Node  of Tree<'a> * 'a * Tree<'a>
```

`ENode` Knoten (empty node) können, genauso wie Blätter, keine Elemente aufnehmen. `Node` Knoten möchten wir mit folgendem Schmuck versehen:

```
type Schmuck =
    | Kugel
    | Lametta
```

Ein Weihnachtsbaum ist ein geschmückter Baum:

```
type Weihnachtsbaum = Tree<Schmuck>
```



Tree, xkcd.com/835, CC BY-NC 2.5
Der dargestellte Baum entspricht *nicht* unserem Datentyp!

Ziel ist es, einen Baum so zu schmücken, dass er balanciert ist, sodass er nicht umfällt. Dazu muss das Gewicht des Schmucks beachtet werden: Eine `Kugel` hat ein Gewicht von `2` und `Lametta` das Gewicht `1`.

a) Schreiben Sie eine Funktion `schmuckGewicht: Schmuck -> Nat`, die das Gewicht des Schmucks berechnet.

b) Schreiben Sie eine Funktion `baumGewicht: Weihnachtsbaum -> Nat`, die das Gesamtgewicht des Schmucks am Baum berechnet.

c) Schreiben Sie eine Funktion `istBalanciert: Weihnachtsbaum -> Bool`, die überprüft, ob der Baum balanciert ist. Ein Baum ist balanciert, wenn das Gewicht des linken Teilbaums gleich dem Gewicht des rechten Teilbaums ist und beide Teilbäume ebenfalls balanciert sind.

d) Schreiben Sie eine Funktion `moeglicheGewichte<'a>: Tree<'a> -> List<Nat>`, die eine Liste aller möglichen Gewichte des Baums zurückgibt, wenn man ihn so schmückt, dass er balanciert ist.

Hinweis: Überlegen Sie, welche Gewichte möglich sind, wenn Sie die möglichen Gewichte des rechten und linken Teilbaums kennen.

e) Schreiben Sie eine Funktion `schmuecken: Tree<Unit> -> Nat -> Option<Weihnachtsbaum>`, die einen Baum und ein Zielgewicht nimmt und den Baum geschmückt zurückgibt, sodass das Gesamtgewicht des Baums gleich dem Zielgewicht ist und der geschmückte Baum balanciert ist. Falls dies nicht möglich ist, soll `None` zurückgegeben werden.

f) Schreiben Sie eine Funktion `schmueckungen: Tree<Unit> -> List<Weihnachtsbaum>`, die alle möglichen balancierten Schmückungen des Baums zurückgibt.

Aufgabe 3 Reguläre Ausdrücke (Einreichaufgabe, 10 Punkte)

Motivation: In dieser Aufgabe sollen Sie sich mit regulären Ausdrücken beschäftigen. Die Aufgabe soll Ihnen dabei helfen den Weg von einem regulären Ausdruck schrittweise bis zu einer Akzeptorfunktion nachzuvollziehen. Sie können sich an den Vorlesungsfolien 553 bis 623 sowie am Skript Kapitel 6.1 und 6.2 orientieren.

Praxistipp: Das UNIX- bzw. Linuxprogramm `grep`[1] erlaubt die Suche in Dateien und Datenströmen anhand von regulären Ausdrücken. Unter Windows stellt das PowerShell Kommando `Select-String -Pattern` eine ähnliche Funktionalität zur Verfügung.

Hinweis: Die Präzedenz der Operatoren ist wie folgt definiert: Die Alternative (`|`) bindet am schwächsten, dann folgt die Konkatenation (·) und zuletzt der Stern (`*`).

Schreiben Sie Ihre Lösungen in die Datei `RegExp.fs` aus der Vorlage `Aufgabe-8-3.zip`.

Wir betrachten den regulären Ausdruck $((\mathtt{a}|\mathtt{aa})|\varepsilon)((\mathtt{b}|\mathtt{bb})(\mathtt{a}|\mathtt{aa}))*$ über dem Alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.

a) Bestimmen Sie **alle** Rechtsfaktoren (inkl. Rechtsfaktoren der Ergebnisse). Geben Sie dabei in der Rechnung jeweils den ersten Schritt explizit an, nachfolgende Zwischenschritte dürfen Sie zusammenfassen.

   Hinweis: Verwenden Sie als Abkürzung `A = (a|aa)` und `B = (b|bb)`, sonst wird die Berechnung unübersichtlich. Wir merken ausserdem an, dass:

   ```
   a \ A = (ε|a)
   b \ A = ∅
   b \ B = (ε|b)
   a \ B = ∅
   ```

   Wir erinnern zusätzlich an die *Rechenregeln* für reguläre Ausdrücke, die es erlauben Ausdrücke zu vereinfachen.

b) Zeichnen Sie den Aufrufgraphen für den Akzeptor (wie auf Vorlesungsfolie 599).

   Umranden Sie Ausdrücke, die nullable sind, doppelt. Wenn wir beim Einlesen eines Wortes an einem solchen nullable Ausdruck landen und keine weitere Eingabe mehr folgt, gehört das eingelesene Wort zur durch den regulären Ausdruck beschriebenen Sprache. Durch die doppelte Umrandung können wir einfacher ablesen, dass wir an einem möglichen Ende angekommen sind (daher werden solche Knoten auch „Endzustände“ genannt).

c) Implementieren Sie die Akzeptorfunktionen. Gehen Sie dabei **streng nach dem Verfahren aus der Vorlesung** vor (Folie 601). Nutzen Sie für das Alphabet den Typ `type Alphabet = | A | B`.

   Hinweis: Wir empfehlen die einzelnen Akzeptorfunktionen als verschränkt rekursive Hilfsfunktionen innerhlab von `accept` zu definieren und am Ende die Start-Akzeptorfunktion mit der Eingabe aufzurufen.

[1] https://de.wikipedia.org/wiki/Grep