Musteraufgaben zur Klausur

A107 – Programmierparadigmen

2021 Prof. Dr. Baltasar Trancón Widemann

1. (5 Punkte)

Beurteilen Sie die folgenden Aussagen:	richtig	falsch
Funktionales und prädikatives Programmieren fasst man unter dem Oberbegriff deklaratives Programmieren zusammen.		
Relationen (im Sinne der relationalen Programmierung) sind vielseitiger verwendbar als Funktionen (im Sinne der funktionalen Programmierung).		
Rennen in nebenläufigen Programmen können durch den Verzicht auf Zuweisungen an Variablen vermieden werden.		
Logische Programme bestehen aus Fakten, Regeln und Variablenzuweisungen.		
Programmiersprachen mit einem dynamischen Typsystem verknüpfen Datentypen mit Variablen, solche mit einem statischen Typsystem mit Konstanten.		

2. (10 Punkte)
   Beantworten Sie kurz die folgenden Fragen:

   (a) (3 Punkte) Wodurch ist ein strenges, statisches und implizites Typsystem gekennzeichnet?

   (b) (1 Punkt) Nennen Sie ein Beispiel für eine Programmiersprache mit einem strengen, statischen und implizitem Typsystem.

   (c) (6 Punkte) Nennen Sie die wichtigsten Probleme, die bei der Programmierung nebenläufiger Programme gelöst werden müssen.

3. (14 Punkte)
   Machen Sie beim Aufschreiben der folgenden Funktionen von der in SML gegebenen Möglichkeit des *pattern matching* Gebrauch.

   Hinweis: Zur Erinnerung hier ein Beispiel für die Anwendung für in SML eingebaute Listen:

```
fun sum_list xs = case xs of
    []     => 0
  | x::xs' => x + sum_list xs'
```

   Alternative Schreibweise:

```
fun sum_list [] = 0
  | sum_list (x::xs') = x + sum_list xs'
```

   (a) (2 Punkte) Schreiben Sie die folgende Funktion äquivalent um:

```
fun fac n = if (n=0) then 1 else n*(fac (n-1));
```

   (b) (4 Punkte) Gegeben sei die folgende Datentypdefinition für Binärbäume:

```
datatype 'a bintr = LEAF of 'a
                  | NODE of 'a bintr * 'a bintr
```

Schreiben Sie eine Funktion, die für einen Baum, dessen Blätter ganze Zahlen sind, diese aufsummiert.

(c) (8 Punkte) Gegeben sei die folgende Datentypdefinition für arithmetische Aus- drücke, bestehend aus Konstanten, Negationen, Additionen und Multiplikationen:

```
datatype exp = Constant of int
             | Negate of exp
             | Add of exp * exp
             | Multiply of exp * exp
```

Schreiben Sie eine Funktion `eval`, die einen arithmetischen Ausdruck auswertet; z. B. sollte der Aufruf

```
eval (Add (Constant 19, Negate (Constant 4)))
```

das Resultat `15` liefern.

4. (9 Punkte) Implementierung von Listen durch Funktionen

Gegeben seien folgende Definitionen für die Clojure-Funktionen **`cons`** und **`first`** (Wir setzen dabei voraus, dass die Definition in einem eigenen Namensbereich stattfindet, so dass es zu keiner Kollision mit den gleichnamigen Standardfunktionen kommt):

```
(def cons
    (fn [x y]
        (fn [m] (m x y))))
(def first
    (fn [z] (z (fn [p q] p))))
```

(a) (5 Punkte) Verifizieren Sie, dass der Ausdruck `(first (cons x y))` als Resultat `x` liefert.

(b) (4 Punkte) Fügen Sie die passende Definition von **`rest`** hinzu, so dass der Ausdruck `(rest (cons x y))` als Resultat `y` liefert.

5. (15 Punkte) Gegeben sei eine Menge von Prolog-Fakten von der allgemeinen Form `father(name1,name2)`. Dabei soll gelten: `name1` ist Vater von `name2`; alle Personen seien männlichen Geschlechts.

(a) (1 Punkt) Definieren Sie ein Prädikat `brother(X,Y)`, das genau dann zutrifft, wenn `X` und `Y` Brüder sind.

(b) (2 Punkte) Definieren Sie ein Prädikat `cousin(X,Y)`, das genau dann zutrifft, wenn `X` und `Y` Cousins sind.

(c) (2 Punkte) Definieren Sie ein Prädikat `grandson(X,Y)`, das genau dann zutrifft, wenn `X` Enkel von `Y` ist.

(d) (2 Punkte) Definieren Sie ein Prädikat `descendant(X,Y)`, das genau dann zutrifft, wenn `X` von `Y` abstammt.

(e) (2 Punkte) Betrachten Sie die folgenden Fakten:

```
father(a,b).
father(a,c).
father(b,d).
father(b,e).
father(c,f).
```

Zeichnen Sie den dazu gehörenden Stammbaum auf.

(f) (6 Punkte) Geben Sie die Antworten, die von Ihren Prädikaten erzeugt werden, in der von Prolog ermittelten Reihenfolge für die folgenden Fragen an:

```
?- brother(X,Y).
?- cousin(X,Y).
?- grandson(X,Y).
?- descendant(X,Y).
```

6. (15 Punkte) Die Software-Gleichung von Larry Putnam beschreibt den phänomenologischen Zusammenhang zwischen der Produktgröße G (gemessen in lines of code), dem Aufwand A (gemessen in Personenstunden), einem Technologiefaktor P und der Projektdauer t:

$$G = P \cdot A^{\frac{1}{3}} \cdot t^{\frac{4}{3}}$$

Entwickeln Sie eine Lösung dieser Gleichung mithilfe des aus Vorlesung bekannten Constraint-Propagation-Systems. Die Existenz der Basis-Bausteine für die Addition, Multiplikation und Potenzbildung darf vorausgesetzt werden.

(a) (6 Punkte) Zeichnen Sie die Gleichung als Datenflussnetz aus Rechenelementen und Konnektoren.

(b) (4 Punkte) Schreiben Sie eine entsprechende Prolog-Regel, welche auf der Löser-Bibliothek `clpr` aufbaut.

(c) (5 Punkte) Geben Sie möglichst aussagekräftige Templates für die Schnittstellendokumentation des von Ihnen definierten Prädikates an.