Musteraufgaben zur Klausur (mit Lösungen)

A107 – Programmierparadigmen

2022 Prof. Dr. Baltasar Trancón Widemann

1. (6 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 wiederholte 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.		×
Pattern Matching ist ein Sprachmittel zur Reproduktion von Programmiermustern.		×

2. (13 Punkte)
   Beantworten Sie kurz die folgenden Fragen:

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

   Antwort: *Typen sind nicht deklaratiospflichtig, sondern werden inferiert (implizit); die Prüfung erfolgt vor der Ausführung des Programms für alle denkbaren Variablenbelegungen (statisch); Inkompatibilitäten sind (Compiler-)Fehler und das Programm wird zurückgewiesen (streng).*

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

   Antwort: *ML (oder z. B. Haskell, ...); Java in speziellen Kontexten (Lambdas, neues Schlüsselwort `var`, ...)*

   (c) (3 Punkte) Welche Rolle spielt das Substitutionsprinzip im Typsystem der Sprache Java?

   Antwort: *Das Typsystem von Java ist nominell. Das Programm enthält Subtypdeklarationen (`extends`, `implements`); die Vererbungsregeln stellen dann das Substitutionsprinzip sicher. Wo dies zum Widerspruch führt (z. B. Einschränkung einer Methode beim Überschreiben von `public` auf `private`), wird ein Fehler gemeldet.*

   (d) (6 Punkte) Nennen Sie die wichtigsten Probleme, die bei der Programmierung nebenläufiger Programme gelöst werden müssen. Geben Sie jeweils eine stichpunktartige Erläuterung.

Antwort: *Mindestens drei aus ...*

- *Rennen – Konflikt um schreibbare Variablen*
- *Ressourcenkonflikte allgemein (Dateien, Peripheriegeräte, ...)*
- *Deadlock/Livelock – Situation ohne möglichen Fortschritt wegen unauflösbarer Konkurrenz um Ressourcen*
- *Fairness – Unmöglichkeit des Verhungerns*
- *Synchronisation – Warten auf nebenläufig produzierte Daten/Ereignisse*

3. (17 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))
```

Antwort:

```
fun fac 0 = 1
  | fac n = n*(fac (n-1))
```

(b) (3+5 Punkte) Gegeben sei die folgende Datentypdefinition für Binärbäume:

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

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

Antwort:

```
fun sum_tree t = case t of
    LEAF n      => n
  | NODE (l, r) => sum_tree l + sum_tree r
```

Schreiben Sie eine Funktion `count_sl`, die für einen Baum, dessen Blätter beliebige Werte sind, die Anzahl von Einzelblättern ermittelt. Ein Einzelblatt ist eines, dessen Geschwisterknoten *kein* Blatt ist.

Antwort:

```
fun count_sl t = case t of
    LEAF n                => 0
  | NODE (LEAF m, LEAF n) => 0
  | NODE (LEAF m, r)      => 1 + count_sl r
  | NODE (l, LEAF n)      => 1 + count_sl l
  | NODE (l, r)           => count_sl l + count_sl r
```

(c) (7 Punkte) Gegeben sei die folgende Datentypdefinition für arithmetische Ausdrücke, 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.

Antwort:

```
fun eval (Constant n)      = n
  | eval (Negate a)        = ~(eval a)
  | eval (Add (a, b))      = eval a + eval b
  | eval (Multiply (a, b)) = eval a * eval b
```

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 durch schrittweise Auswertung, dass der Ausdruck `(first (cons x y))` als Resultat `x` liefert.

Antwort: *(Kommentare optional)*

```
;; first -> (fn ...) laut def
((fn [z] (z (fn [p q] p))) (cons x y))
;; Argument zuerst auswerten:
;; cons -> (fn ...) laut def
((fn [z] (z (fn [p q] p)))
  ((fn [x y] (fn [m] (m x y))) x y))
;; Jetzt bekannte Funktion aufrufen:  Muster "((fn" finden!
;; fn faellt weg
;; Parameterliste mit Argumenten paaren: z -> ((fn ...))
;; Im Rumpf ersetzen
(((fn [x y] (fn [m] (m x y))) x y) (fn [p q] p))
;; dito, hier x -> x und y -> y
((fn [m] (m x y)) (fn [p q] p))
;; nochmal, hier m -> (fn [p q] p)
((fn [p q] p) x y)
;; und... hier p -> x, q -> y (kommt nicht vor)
x
```

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

Antwort:

```
(def rest
    (fn [z] (z (fn [p q] q))))
```

5. (15 Punkte) Gegeben sei eine Menge von Prolog-Fakten von der allgemeinen Form `father(Name1, Name2)`. Dabei soll gelten: `Name2` ist Vater von `Name1`; 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.

   Antwort:

   ```
   brother(X, Y) :- father(X, F), father(Y, F), X \= Y.
   ```

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

   Antwort:

   ```
   cousin(X, Y) :- father(X, F), father(Y, G), brother(F, G).
   %% Annahme: verschiedene Vaeter haben verschiedene Kinder!
   ```

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

   Antwort:

   ```
   grandson(X, Y) :- father(Y, F), father(F, X).
   ```

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

   Antwort:

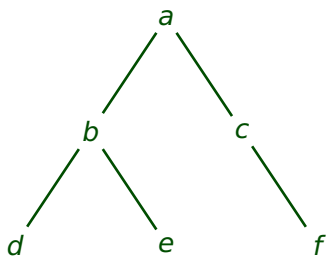
   ```
   descendant(X, Y) :- father(X, Y).
   descendant(X, Y) :- father(X, F), descendant(F, Y).
   ```

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

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

   Zeichnen Sie den dazu gehörenden Stammbaum auf.

   Antwort:



   (f) (8 Punkte) Geben Sie die Antworten, die auf dieser Datenbasis von Ihren Prädikaten erzeugt werden, in der von Prolog ermittelten Reihenfolge für die folgenden Fragen an:

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

Antwort: *(Kann bei anderen Definitionen abweichen)*

```
?- brother(X, Y).
X = b, Y = c ;
X = c, Y = b ;
X = d, Y = e ;
X = e, Y = d

?- grandson(X, Y).
X = a, Y = d ;
X = a, Y = e ;
X = a, Y = f

?- cousin(d, e).
false

?- descendant(X, f).
X = c ;
X = a
```

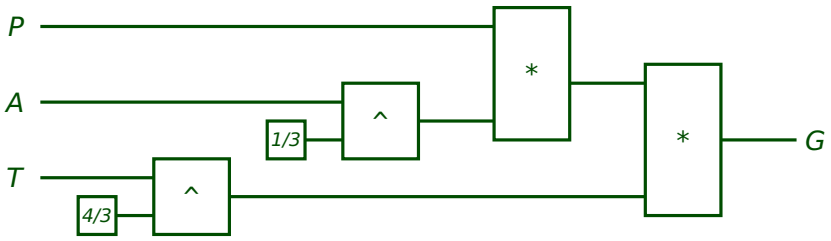
6. (15 Punkte) Die Software-Gleichung von Larry Putnam beschreibt den phänomenologischen Zusammenhang zwischen der Produktgröße G (gemessen in lines of code), dem Aufwand A (gemessen in Personenstunden), einem Technologiefaktor P und der Projektdauer t:

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

Entwickeln Sie eine relationale Lösung dieser Gleichung mithilfe des aus Vorlesung bekannten Constraint-Propagation-Systems. Die Existenz von Elementen für die Addition, Multiplikation und Potenzierung darf vorausgesetzt werden.

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

   Antwort:



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

   Antwort:

```
software(G, A, P, T) :- { G = P * A ^ (1/3) * T ^ (4/3) }.
```

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

   Antwort:

```
%! software(-Groesse:float, +Aufwand:float,
%!          +Techno:float, +Dauer:float) is det
% ... Rest vereinfacht ...
%! software(+, -, +, +) is det
%! software(+, +, -, +) is det
%! software(+, +, +, -) is det
```

7. (9 Punkte) Das Prolog-Prädikat `between(+X, +Y, ?Z)` hat als Lösung alle ganzen Zahlen Z mit $X \leq Z \leq Y$. Gegeben ist folgendes Programm nach dem Generieren-und-Testen-Muster:

```
find(A, B, C) :-
    between(1, 20, A),
    between(1, 20, B),
    between(1, 25, C),
    A =< B,
    C^2 =:= A^2 + B^2,
    C mod 4 =:= 1.
```

   (a) (3 Punkte) Beschreiben Sie die Menge der Lösungen exakt. (Die Reihenfolge ist irrelevant.)

   Antwort: *Es handelt sich um alle ganzzahligen Tripel der Form* (a, b, c)*, mit* $1 \leq a, b \leq 20$ *und* $1 \leq c \leq 25$*, wobei zusätzlich* $a \leq b$ *und* $c^2 = a^2 + b^2$ *(pythagoräisches Tripel) und* $c \equiv 1 \pmod 4$ *gelten muss.*

   (b) (2 Punkte) Dieses Programm wurde auf ineffiziente Weise implementiert. Erläutern Sie kurz, durch welche Art von Refaktorisierung die Effizienz (ohne detaillierte Mathematikkentnisse oder die Verwendung eines Constraint Solvers) verbessert werden könnte.

   Antwort: *Test-Klauseln, die einzelne Constraints überprüfen, sollten möglichst weit vorne im Suchverfahren stehen, damit möglichst große „hoffnungslose“ Teilbereiche des Lösungsraums vermieden werden können; idealerweise direkt hinter den Generatoren der vorkommenden Variablen.*

   (c) (4 Punkte) Geben Sie eine äquivalente, möglichst effiziente Implementierung an. (Die Reihenfolge der Lösungen darf verändert werden.)

   Antwort:

```
find(A, B, C) :-
    between(1, 25, C),
    C mod 4 =:= 1,
    between(1, 20, A),
    between(A, 20, B),  %% Bonuspunkt!
    C^2 =:= A^2 + B^2.
```