# Musteraufgaben zur Klausur (mit Lösungen)

## *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?

   ***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 implizitem Typsystem.

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

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

   ***Antwort:***
   - *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*
   - *(ggf. Prioritätsinversion)*

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));
```

***Antwort:***

```
fun fac 0 = 1
  | fac n = 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.

***Antwort:***

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

(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.

***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, dass der Ausdruck `(first (cons x y))` als Resultat `x` liefert.

   ***Antwort:***

```
;; 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 die passende Definition von **rest** hinzu, so dass der Ausdruck `(rest (cons x y))` als Resultat `y` liefert.

   ***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: `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.

   ***Antwort:***

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

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

***Antwort:***

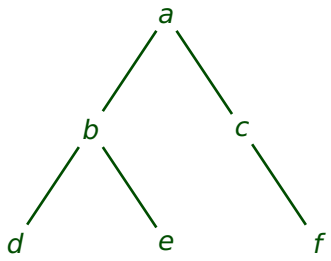
```
descendant(X, Y) :- father(Y, X).
descendant(X, Y) :- father(F, X), descendant(F, Y).
```

(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.

***Antwort:***




(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).
```

***Antwort:***

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

?- cousin(X, Y).
X = d, Y = f ;
X = e, Y = f ;
X = f, Y = d ;
X = f, Y = e ;
false.
```

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

?- descendant(X, Y).
X = b, Y = a ;
X = c, Y = a ;
X = d, Y = b ;
X = e, Y = b ;
X = f, Y = c ;
X = d, Y = a ;
X = e, Y = a ;
X = f, Y = a ;
false.
```

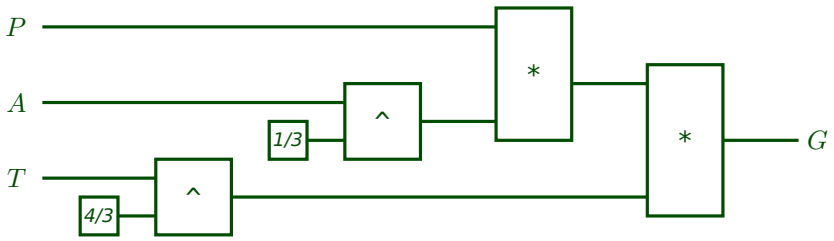
6. (15 Punkte) Die Software-Gleichung von Larry Putnam beschreibt den phänomenologischen Zusammenhang zwischen der Produktgröße $G$ (gemessen in lines of code), dem Aufwand $A$ (gemessen in Personenstunden), einem Technologiefaktor $P$ und der Projektdauer $t$:

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

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.

***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:***

```
% descendant(-Groesse, +Aufwand, +Techno, +Dauer) is det
% descendant(+Groesse, -Aufwand, +Techno, +Dauer) is det
% descendant(+Groesse, +Aufwand, -Techno, +Dauer) is det
% descendant(+Groesse, +Aufwand, +Techno, -Dauer) is det
```