Philipps-Universität Marburg
Fachbereich Mathematik und Informatik
AG Programmiersprachen und -werkzeuge

Sommersemester 2024
17.07.2023

Prof. Dr. Christoph Bockisch
MSc. Steffen Dick

Klausur zur Vorlesung

Deklarative Programmierung

Wichtige Hinweise:

- Schalten Sie, falls noch nicht geschehen, umgehend ihr Mobiltelefon aus!
- Schalten Sie außerdem alle nicht medizinisch notwendigen Lärmquellen aus.
- Entfernen Sie jetzt alle unerlaubten Gegenstände vom Tisch. Erlaubt sind nur ein Stift (kein Rot-, Grün- oder Bleistift) und Getränke. Halten Sie außerdem Ihren Studienausweis, so wie Ihren Personalausweis oder Reisepass bereit.
- Die Bearbeitungszeit beträgt **2** Zeitstunden.
- Verwenden Sie kein eigenes Papier für Notizen. Am Ende der Klausur befindet sich 1 Extrablatt. Sie können auf Anfrage weitere Blätter erhalten. Machen Sie gut kenntlich, wenn Sie Zusatzblätter für Lösungen verwenden und tragen Sie dort ebenfalls Namen und Matrikelnummer ein.
- Es sind keine weiteren Hilfsmittel erlaubt. Zuwiderhandlung zieht einen Ausschluss von der Klausur nach sich.
- Mehrere, widersprüchliche Lösungen zu einer Aufgabe werden mit 0 Punkten bewertet.
- Sollten Sie eine Frage haben, wenden Sie sich bitte leise an die Tutor:innen.
- Schreiben Sie auf jedes Blatt Ihren Namen und Matrikelnummer. Blätter ohne Namen werden nicht korrigiert und ergeben 0 Punkte! Füllen Sie insbesondere die folgende Tabelle in Druckbuchstaben aus:

Vorname	
Nachname	
Matrikelnummer	
Studienfach	
Angestrebter Abschluss	

Möchten Sie, dass Ihr Ergebnis mit Ihrer Matrikelnummer im Ilias veröffentlicht wird?

☐ Ja ☐ Nein

Bei fehlendem Kreuz oder "Nein" erfahren Sie Ihr Ergebnis erst in der Einsicht.

Übersicht der erreichbaren Punkte:

Aufgabe	1	2	3	4	5	6	7	Gesamt
Punkte	14	17	13	11	10	15	20	100

Vorname:	Nachname:	Matrikelnummer:

Aufgabe 1: Knowledge questions

14 Punkte

Answer the following questions in 1-2 short sentences!

a) What does the term shadowing mean? **2**

b) What must be taken into account when evaluating the order of cond expressions? **2**

c) What does structural recursion mean? **2**

d) What does the term atom mean in Prolog? Give an example! **2**

Vorname:	Nachname:	Matrikelnummer:

e) What is a scope in programming? **2**

f) Briefly explain the term "accumulator invariant". **2**

g) What is syntactic sugar? **2**

Vorname:	Nachname:	Matrikelnummer:

Aufgabe 2: expressions

17 Punkte

For subtasks a) to d): What is the result of the following programs? It is sufficient to write down the result. The rules applied do not have to be written down. If an error occurs during evaluation, describe the cause of the error. The language level **"Intermediate Student Language with Lambdas" (ISL+)** applies to the first four subtasks.

a) **2**

```
1       (define (tick x) (* (trick x) x))
2       (define (trick x) (+ x track))
3       (define track 23)
4
5       (tick 3)
```

b) **2**

```
1       (define gauckeley 666)
2       (if (positive? gauckeley)
3       (+ gauckeley glueckstaler)
4       (- gauckeley glueckstaler))
```

c) **2**

```
1       (define-struct ente (name networth))
2       (define bertel (make-ente "Scrooge McDuck" 2147483647))
3       (define klaas (make-ente "Klaas Klever" 21474836))
4       (if (> (ente-networth bertel) (ente-networth klaas))
5       "Scrooge McDuck is richer!"
6       "Klaas Klever is richer!")
```

d) Write lists in the result in the representation (list ...). **2**

note: The character before the first opening bracket is a slanted apostrophe.

```
1       ‘(,(+ 1 2) (rest 3 4))
```

Vorname:	Nachname:	Matrikelnummer:

The language level **"Beginning Student Language" (BSL)** applies to the following subtask.

e) Consider the following code in DrRacket: **3**

```
1       (define taler 1.05)
2       (define (euro-to-taler e)
3       (* e taler))
```

Now reduce the expression `(euro-to-taler 10)` step by step using the given environment. **Indicate the rule used for each transformation step**. Only use the transformation rules of the BSL.

Note: You can use the distributed sheet of paper with the transformation rules in this task.

Vorname:	Nachname:	Matrikelnummer:

f) The following function should actually calculate the wealth of a certain richest duck in the world in thalers. Here, 100 kreutzers correspond to one thaler and 5 thalers to one doubloon. However, **2 errors** have crept into the code. You can assume that `count-net-worth` always contains correct entries. **6**

Enter the following for each error:

1. An example call of `count-net-worth` where the error occurs.
2. A description of the error, for example in the form of an error message that Racket would output when called, or an explanation of why the result is incorrect.
3. A short explanation of how to fix the error.

```
1       (define-struct money store (thaler kreutzer doubloons))
2       (define (count-net-worth assetList)
3       (+ (cond
4       [(money store? (first assetList))
5       (+ (money-store thaler (first assetList))
6       (/ (geldspeicher-kreutzer (first assetList)) 100)
7       (* (geldspeicher-dublonen (first assetList)) 1))]
8       [(number? (first assetList)) (first assetList)])
9       (if (empty? assetList) 0
10      (count-net-worth (rest assetList)))))
```

Vorname:	Nachname:	Matrikelnummer:

Aufgabe 3: Algebraic data types

13 Punkte

In this task, you are to implement an auxiliary function for searching within a music database. This checks for an artist (band or solo artist) and a name whether it is the name of the band, the solo artist or a band member.

A band (`Band`) consists of the following information:

- The name of the band
- The list of members (personal names)

A solo artist (`Soloist`) consists of the following information:

- The artist name
- The civil name

An artist (`Artist`) is either a band or a solo artist.

a) Define the algebraic data type *Artist* and all associated `struct`s. Also specify example values. **6**

Vorname:	Nachname:	Matrikelnummer:

b) Implement a function `(matches artist name)` that returns `true` if the name corresponds to the band name or the name of a band member or, in the case of a soloist, the artist name or the civil name. **Apply the design recipe for functions over algebraic data types**. **7**

Note: Outsource any required functions.

Vorname:	Nachname:	Matrikelnummer:

Aufgabe 4: Higher-order lists and functions — 11 Punkte

***hint*: You may use the following higher-order functions known from the lecture in task part a):**

```
1     ; [X] (X -> Boolean) (listof X) -> (listof X)
2     ; Returns a list containing all elements from l
3     ; that fulfill the predicate p
4     (filter p l)
5
6     X Y] (X -> Y) (listof X) -> (listof Y)
7     ; Maps all elements from l with f and returns
8     ; the list of results.
9     (map f l)
10
11    ; [X Y] (X Y -> Y) Y (listof X) -> Y
12    ; Combines all elements of the list l by f. The
13    ; empty list is mapped to base, the elements
14    ; are run through from right to left.
15    (foldr f base l)
```

a) Recreate the functionality of `(myFoldl f base l)`. Apart from `append`, only use the **above-mentioned higher-order functions and lambda expressions** in your implementation. **5**

```
1       ; [X Y] (X Y -> Y) Y (listof X) -> Y
2       ; Combines all elements of the list l by f. The
3       ; empty list is mapped to base, the elements
4       ; are run through from left to right.
5       (check-expect (myFoldl cons empty (list 1 2 3)) (list 3 2 1))
6       (define (myFoldl f base l)
```

Vorname:	Nachname:	Matrikelnummer:

b) Implement the function `(myOrmap proc lst)` **without using higher order functions**. **6**

```
1       (check-expect (myOrmap positive? '(1 2 a)) true)
2       ; [X] (X -> boolean) List-of-X -> boolean
3       ; Applies proc to all elements of the list lst and
4       ; combines the partial results with a logical or.
5       (define (myOrmap proc lst)
```

Vorname:	Nachname:	Matrikelnummer:

Aufgabe 5: recursion & termination

10 Punkte

a) Implement the function `(myFilter proc lst)`, which filters a list `lst` by the predicate `proc`, using an accumulator. The original order of the list should be retained. Define all auxiliary functions only locally within the function. Also specify the **accumulator invariant**. **7**

Vorname:	Nachname:	Matrikelnummer:

b) Consider the following function, which adds up the elements of a list, which can also contain lists again. **3**

```
1       (define (mySumList lst)
2       (cond
3       [(emtpy? lst) 0]
4       [(number? (first lst)) (+ (first lst) (mySumList (rest lst)))]
5       [else (+ (mySumList (first lst)) (mySumList (rest lst)))]))
```

Explain why the function `mySumList` terminates and also specify which type of recursion (*generative*, *accumulative* or *structural*) is involved. Explain your answer in 1-2 short sentences!

Vorname:	Nachname:	Matrikelnummer:

Aufgabe 6: Prolog

15 Punkte

a) Define the following procedure in Prolog. The use of library procedures from Prolog is not permitted. However, arithmetic operations such as + or - may be used. You may also use `append/3`. The relation `append(L1, L2, E)` links two lists (L1 and L2) to a third (E). Define the required auxiliary procedures yourself. **5**

`is_palindrome(L)`, which is fulfilled if L is a list whose elements result in a palindrome. A palindrome is a word that can be read the same forwards and backwards (e.g. Girafarig or Farigiraf).

Vorname:	Nachname:	Matrikelnummer:

Given the following definitions of procedures b - d and the queries in subtasks b) - d) that use them. For each query, specify what the result is. If a query is satisfied, specify a valid substitution of all variables. In case of an error or an unfulfilled query, give a short justification.

```
1     b([], r()).
2     b([R|[T|L]],r(Two)) :- R < T, b([T|L], two).
3     b([S|[A|T]],One) :- S > A, b([A|T], One).
4
5     c([Marjory, kasmeer], braham, [rox|[Frostbite]], Taimi).
6
7     d(0,1).
8     d(M,I) :- I is M + 2, B is M -1, d(B, _).
```

b) `b([1,2,3], E).` **3**

c) `c([canach, kasmeer], braham, [rox, marjory], taimi).` **4**

d) `d(6, E).` **3**

Vorname:	Nachname:	Matrikelnummer:

Aufgabe 7: reduction and equivalence — 20 Punkte

For the following subtasks a)-c), you can assume that the following definitions are in the environment:

```
1     ; Number -> Number
2     ; Calculates the sum of all squares of the numbers from 0-N with recursion
3     (define (sum-pow n)
4     (cond
5     [(= n 0) 0]
6     [else (+ (* n n) (sum-pow (- n 1)))]))
7
8     ; Number -> Number
9     ; Calculates the sum of all squares of the numbers from 0-N with a formula
10    (/ (* n (+ n 1) (+ (* n 2) 1)) 6)
```

The equivalence of `(sum-pow n)` ≡ `(/ (* n (+ n 1)(+ (* n 2)1))6)` is to be shown by structural induction via `n`. The following equivalence rules may be used without proof:

EPRIM-mult-0 `(* 0 X1 ... XN)` ≡ `0`

default `(/ (* (+ n 1)(+ (+ n 1)1)(+ (* (+ n 1)2)1))6)` ≡
`(+ (* (+ n 1)(+ n 1))(/ (* n (+ n 1)(+ (* n 2)1))6))`

Furthermore, if two identical steps directly follow each other, you may omit the first step. Parts in which no change takes place may be abbreviated with `....`

a) Set up the equivalence to be proven in **start of induction** and perform **start of induction**. **5**

Vorname:	Nachname:	Matrikelnummer:

b) Set the **induction assumption**. **1**

c) Now establish the equivalence to be proven in the **induction step** and then carry out the **induction step**. **8**

Vorname:	Nachname:	Matrikelnummer:

The following subtask is no longer part of the proof of equivalence. This means that `sum-pow` and `(/ (* n (+ n 1)(+ (* n 2)1))6)` are no longer required.

d) Implement the function (flatten lst) in Racket using **Pattern-Matching**. **You may not use selector functions in this subtask! This includes, for example, the list functions `rest` and `first`.** **6**

The `flatten` function receives a list as an argument, which can contain simple elements or additional lists. The result of `flatten` should be a list that contains all elements but no further lists.

```
1       (check-expect (flatten '(1 2 ((3 4) 5 (6 7)))) (list 1 2 3 4 5 6 7))
2       (check-expect (flatten empty) empty)
```

Vorname:	Nachname:	Matrikelnummer:

Extra Seite