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

Sommersemester 2024
18.09.2023

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	18	12	11	10	15	20	100

Vorname:	Nachname:	Matrikelnummer:

Aufgabe 1: Knowledge questions 14 Punkte

Answer the following questions in 1-2 short sentences!

a) In which order are expressions evaluated in DrRacket? **2**

b) What does the acronym *DRY* stand for? Also explain the meaning. **2**

c) Explain the term accumulator. **2**

d) What is a *syntax error* and at what point is it recognized? **2**

Vorname:	Nachname:	Matrikelnummer:

e) What does the term *Magic Numbers* mean and how can they be avoided? **2**

f) What is meant in Prolog when talking about a *common instance*? **2**

g) What does the term *unifier* mean in Prolog? **2**

Vorname:	Nachname:	Matrikelnummer:

Aufgabe 2: expressions

18 Punkte

For subtasks a) to d): What is the result of the following programs? It is sufficient to note 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

```
(define (huey x) (+ (dewey x) x))
(define (dewey x) (- x louie))
(define louie 5)
(define phooey 4)

(+ (huey 3) phooey)
```

b) 2

```
(define-struct duck (name luck))
(define gander (make-duck "Gladstone Gander" 2147483647))
(define duck (make-duck "Donald Duck" -2147483647))
(if (> (duck-luck gander) (duck-luck duck))
(string-append (duck-name gander) " has more luck")
(string-append (duck-name duck) " has more luck"))
```

c) 2

```
(define magica 5)
(cond
[(deSpell? magica) true]
[else -1])
```

d) In the result, write lists in the display (list ...).

2

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

```
‘((string-append "Drake" Mallard) ,(rest (list "is" "Darkwing")))
```

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

e) Consider the following code in DrRacket:

4

```
(define blathering "blatherskite")
(define (secret x)
(cond
[(< 0 x) blathering]
[else "Let’s get dangerous!"]))
```

Now reduce the expression `(secret 5)` 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.

f) The following function should actually convert the treasures that a certain richest duck in the world has brought back from one of his travels into coins. A ruby is worth its weight in talers, an emerald is worth twice its weight in talers and a sapphire is worth three times its weight in talers. However, **2 errors** have crept into the code. You can assume that `treasure->taler` always receives a list as input, and that all elements are correct instances of the structures `ruby`, `emerald` or `saphire`. **6**

Specify the following for each error:

1. An example call to `treasure->taler` 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 brief explanation of how to fix the error.

```
(define-struct ruby (mass))
(define-struct emerald (mass))
(define-struct sapphire (mass))
(define (treasure->taler treasure)
(if (empty? treasure) 0
(*
(cond
[(ruby? (first treasure)) (* (ruby-mass (first treasure)) 1)]
[(emerald? (first treasure)) (* (emerald-mass (first treasure)) 2)]
[else (* (ruby-mass (first treasure)) 3)]])
(treasure->taler (rest treasure)))))
```

Vorname:	Nachname:	Matrikelnummer:

Aufgabe 3: Algebraic data types 12 Punkte

In this task, you are to implement a helper function for calculating the performance of a computer player character over several recordings.

A player character is either

- a damage dealer (`DD`)
- a supporter (`Support`)

A `DD` has

- a name
- a damage-per-second value (DPS)

A `Support` has

- a name
- a healing-per-second value (HPS)
- has a damage-per-second value (DPS)

a) Define the algebraic data type *player character* and all associated `structs`. Also specify example values. **6**

b) Implement a function `(performanceAnalysis performances)` that adds up the values (dps and hps) of all player characters in `performances`. The argument `performances` is simply a list of player characters. Apply the design recipe for functions over algebraic data types. 6

hint: Outsource any required functions.

Aufgabe 4: Lists and higher-order functions

11 Punkte

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

```
; [X] (X -> Boolean) (listof X) -> (listof X)
; Returns a list containing all elements from l
; that fulfill the predicate p
(filter p l)
X Y] (X -> Y) (listof X) -> (listof Y)
; Maps all elements from l with f and returns
; the list of results.
(map f l)
; [X Y] (X Y -> Y) Y (listof X) -> Y
; Combines all elements of the list l by f. The
; empty list is mapped to base, the elements
; are run through from right to left.
(foldr f base l)
```

a) Recreate the functionality of `(andMap proc lst)`. In your implementation, only use the **above-mentioned higher-order functions and lambda expressions**. You may also use the and function. 4

```
; [X] (X -> boolean) List-Of-X -> boolean
; Applies the function proc to all elements of the list
; and combines the results with and.
(check-expect (andMap positive? (list 1 2 3 4)) true)
(check-expect (andMap positive? (list -1 2 3 4)) false)

(define (andMap proc lst)
```

Vorname:	Nachname:	Matrikelnummer:

b) Implement the following function in DrRacket, **without** using a higher order function or lambda expressions. **7**

Note: You may define your own auxiliary functions and work with accumulators.

```
(define-struct doubleList (one two))

(check-expect
(partition positive? (list -4 -2 1 2 3))
(make-doubleList (list 1 2 3) (list -4 -2)))
X] List-Of-X (X -> boolean) -> doubleList
; Splits a list lst into two lists based on a given predicate function per
; given predicate function proc.
; The order of the individual elements is retained.
(define (partition proc lst)
```

Aufgabe 5: recursion & termination 10 Punkte

a) Implement the function `(sumQuad n)`, which sums the squares of all even natural numbers up to and including n, using a **accumulator**. Define all auxiliary functions only locally within the function. Also specify the **accumulator invariant**. 7

b) Consider the following function `conc`, which converts a list of lists into a list of all elements. 3

```
(define (conc lsts)
(local
[(define (conc-helper lsts acc)
(cond
[(empty? lsts) acc]
[(cons? (first lsts))
(conc-helper (rest lsts) (append acc (first lsts)))]
[else
(conc-helper (rest lsts) (append acc (list (first lsts))))]))]
(conc-helper lsts ’())))
```

Explain why the function `conc` terminates. Also state what type of recursion (*generative*, *accumulative* or *structural*) the function is. Explain your choice in 1-2 short sentences.

Aufgabe 6: Prolog

15 Punkte

Define the following procedure in Prolog. The use of other procedures from Prolog is not permitted. Define any additional procedures you require yourself.

a) `reverse(L1, L2)`: `L1` is a list, `L2` is the reverse list `L1`. Take the following queries as an example: **6**

- `?— reverse([1,2,3,4], E).` is fulfilled with `E = [4,3,2,1]`.
- `?— reverse([1,2],[1,2]).` is not fulfilled.

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 (**true**, `false` or error). If a query is fulfilled, specify a valid substitution of all variables. In the event of an error or an unfulfilled query, provide a brief explanation.

```
b([], r()).
b([Azzarath, Metrion, Zinthos], r(Raven)) :- b([Azzarath, Metrion], Raven).
b([Richard|Kori],r(Gar)) :- b(Kori, Gar).

c([Logan | [Rytlock, [caithe | Zojja], eir]]).

d(0,1).
d(Grayson, Todd) :- Grayson is Drake + 1, Wayne is Todd - 1, d(Brown, _).
```

b) `?- b([teen,titans,go], Vik).` **3**

c) `?- c([logan, rytlock, caithe, zojja, eir]).` **3**

d) `?- d(6, Robin).` **3**

Aufgabe 7: reduction and equivalence

20 Punkte

For the following subtasks a)-c), you can assume that the following function definition is in the environment, which recursively calculates the sum of the first $n + 1$ positive odd numbers:

```
; Number -> Number
(check-expect (formula 1) (+ 1 3))
(define (formula n)
(cond
[(= 0 n) 1]
[else (+ (* 2 n) 1 (formula (- n 1)))]))
```

Prove by structural induction over `n` that the equivalence `(formula n)` ≡ `(expt (+ n 1)2)` holds. Where `(expt a b)` is the primitive function that calculates a^b. Specify the rules used for each transformation step. Steps where no rule is specified are not evaluated. Parts in which no change takes place may be abbreviated with `...`.

The following equivalence rules may be used without proof:

EHTRIVIAL: `(expt 1 2)` ≡ 1

EHKLAR: `(expt (+ (+ n 1)1)2)` ≡ `(+ (* 2 (+ n 1)1 (expt (+ n 1)2))`

a) Set up the equivalence to be proven in **start of induction** and perform **start of induction**. **6**

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**. 7

The following subtask is no longer part of the proof of equivalence. Therefore, `(formula n)` is no longer required.

d) Implement the function `(count lst e)` in Racket using **Pattern-Matching**. **You may not use selector functions in this subtask. This includes, among other things `rest` and `first` and all functions with similar functionality**. 6

The function `count` receives a list and an element as an argument and returns the number of all occurrences of the element in the transferred list. The transferred list can also contain other lists that are also to be searched for the element.

```
(check-expect (count (list (list 1 2 2 3) 1 2 3) 2) 3)
(check-expect (count (list 1 2 2 3) 2) 2)
```

Vorname:	Nachname:	Matrikelnummer:

Extra Seite