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

Sommersemester 2021
26.07.2021

Prof. Dr. Christoph Bockisch

Klausur zur Vorlesung

Deklarative Programmierung

Wichtige Hinweise:

- Schalten Sie jetzt Ihr Mobiltelefon und alle medizinisch nicht notwendigen potentiellen Lärmquellen aus.
- Entfernen Sie jetzt alle unerlaubten Gegenstände vom Tisch. Erlaubt sind nur ein Stift (kein Rot-, Grün-, oder Bleistift) und Getränke. Halten Sie Ihren Studentenausweis und Personalausweis/Reisepass bereit.
- Es sind keine eigenen Hilfsmittel erlaubt. Zuwiderhandlungen führen zum Ausschluss.
- Schreiben Sie Ihren Namen und Ihre Matrikelnummer auf jedes Blatt in Druckbuchstaben. Blätter ohne Namen ergeben 0 Punkte! Füllen Sie folgende Tabelle in Druckbuchstaben aus:

Vorname	
Nachname	
Matrikelnummer	
Fachbereich	
Studienfach	
Angestrebter Abschluss	

- Die Bearbeitungszeit beträgt **2** Stunden.
- Verwenden Sie kein eigenes Papier. Sie können die Extrablätter am Ende des Klausurbogens nutzen oder nach Extrablättern fragen. Machen Sie ggf. im vorgesehenen Lösungsfeld kenntlich, wo die Lösung zu finden ist und tragen Sie auch auf Extrablättern Ihren Namen und Ihre Matrikelnummer ein.
- Mehrere widersprüchliche Lösungen zu einer Aufgabe werden mit 0 Punkten bewertet.
- Falls Sie eine Frage haben, so wenden Sie sich bitte leise an einen der Tutoren.
- Die Klausur ist ab 45 Punkten definitiv bestanden. Mit mindestens 95 Punkten erreichen Sie definitiv die Note 15.

Wird bei der Korrektur ausgefüllt:

Aufgabe	1	2	3	4	5	6	7	8	Total
Punkte	8	11	18	9	20	11	10	13	100
Punkte erreicht									

Vorname:	Nachname:	Matrikelnummer:

Aufgabe 1: Wissensfragen 8 Punkte

Beantworten Sie die folgenden Fragen in 1–2 kurzen Sätzen!

(a) Beschreiben Sie, was ein **Literal** ist. Geben Sie ein **Beispiel**. **2**

(b) Beschreiben Sie, was **Konfluenz** bei der Auswertung von Ausdrücken bedeutet ist. Geben Sie ein **Beispiel**. **2**

(c) Was ist syntaktischer Zucker? **2**

(d) Beschreiben Sie, wie man beim Top-Down Entwurf von Funktionen vorgeht. **2**

Aufgabe 2: Ausdrücke 11 Punkte

Für die Teilaufgaben a) bis c): Welches Ergebnis liefern die folgenden Ausdrücke? Es genügt, wenn Sie das Ergebnis notieren. Die angewendeten Regeln müssen nicht aufgeschrieben werden. Tritt beim Auswerten ein Fehler auf, geben Sie die Art und Ursache des Fehlers an.

(a)
```
(cond [(and (> -1 1) (< -1 (/ 10 0))) 1]
      [(< 1 -1) 2])
```
2

(b) Schreiben Sie in dem Ergebnis Listen in der Darstellung `(list ... )`. **2**

```
(cons (+ 4 5) '(make-posn 1 2))
```

(c)
```
((lambda (x) (+ x x)) (* 2 2))
```
2

Vorname:	Nachname:	Matrikelnummer:

(d) Reduzieren Sie den Ausdruck (Zeile 5) in dem folgenden Programm unter der gegebenen Umgebung zu einem Wert. **Geben Sie alle nötigen Reduktionsschritte unter Angabe der Regelnamen an.** Benutzen Sie die Reduktionsregeln der BSL-Sprache. **5**

```
1 ; In der Umgebung befindet sich diese Definition
2 (define (f x y) (cond [x (posn-x y)] [else (posn-y y)]))
3
4 ; reduzieren Sie den folgenden Aufruf
5 (f (empty? '()) (make-posn 3 4))
```

Aufgabe 3: Funktionsdefinitionen und Signaturen 18 Punkte

(a) Geben Sie die Signatur der folgenden Funktion an. Verwenden Sie soweit nötig Typparameter, welche die Beziehungen zwischen den Parametern in der Signatur korrekt repräsentieren. Gehen Sie davon aus, dass **alle verwendeten Listen homogen** sind, also nur Werte von einem Typ enthalten. Dabei kommen als Element-Typ einer List **keine** Summentypen in Frage. Begründen Sie alle Teile der Signatur, und nehmen Sie dabei auf die Funktionsdefinition Bezug! **7**

```
(define (f x y z)
  (cond [(empty? x) empty]
        [(first y) (cons (z (first x)) (f (rest x) (rest y) z))]
        [else (f (rest x) (rest y) z)]))
```

Vorname:	Nachname:	Matrikelnummer:

Gegeben sind folgende Struktur- und Datendefinitionen:

```
(define-struct cell (value priority rest))
; Eine Cell ist eine Struktur: (make-cell String Priority PrioList)
; interp. eine Zelle einer Prioritätsliste

; Eine Priority kann einen der folgenden Werte annehmen:
; - "high"
; - "low"
; interp. jedes Element von Priority wird durch eine String-Beschreibung
; der Dringlichkeit dargestellt.


; Ein PrioList ist eins von:
; - Cell
; - empty
; interp. eine Liste von String Einträgen, die eine Priorität haben.

; Ein StringOrFalse ist eins von:
; - String
; - false
; interp. ein String-Wert oder false, wenn kein String-Wert ermittelt werden konnte
```

(b) Um was für eine Art Datentyp handelt es sich jeweils bei den Typen? **3**

- Cell ist ein _____.
- Priority ist ein _____ und hat die Kardinalität _____.

(c) Die Funktion `poll` soll einen PrioList-Wert als Argument entgegen nehmen und den Eintrag mit der höchsten Priorität zurückliefern, bzw. `false`, wenn die Prioritätsliste leer ist. Geben Sie für diese Funktion folgendes an: Signatur, Testfälle in ausreichender Anzahl (gemäß des zutreffenden Entwurfsrezepts), die Schablone der Implementierung. **8**

Vorname:	Nachname:	Matrikelnummer:

Aufgabe 4: Algebraische Datentypen 9 Punkte

Es soll ein Programm zur Verwaltung von Immobilien geschrieben werden. Im ersten Schritt soll das Programm mit Einfamilienhäusern und Mehrfamilienhäusern arbeiten. Ein Einfamilienhaus wird durch die Adresse und die Anzahl der Zimmer beschrieben. Mehrfamilienhäuser werden beschrieben durch die Adresse, sowie eine Liste, die für jede Wohnung die Anzahl an Zimmern enthält.

(a) Definieren Sie geeignete Strukturen zur Repräsentation von *Einfamilienhäusern* (EFH) und *Mehrfamilienhäusern* (MFH) und geben Sie eine Datendefinition für den **Algebraischen Typen** *Immobilie* an. **3**

(b) Implementieren Sie eine Funktion `(miete immo)` zur Berechnung der Mieteinnahmen einer Immobilie, basierend auf der Definition aus Aufgabenteil a). Gehen Sie dabei davon aus, dass pro Raum eine Miete von 100 Euro fällig ist. **Wenden Sie die Schablone für algebraische Datentypen an.** **6**

```
(define (miete immo)
```

Vorname:	Nachname:	Matrikelnummer:

Aufgabe 5: Reduktion und Äquivalenz

20 Punkte

Für die folgenden Teilaufgaben a) und b) gehen Sie davon aus, dass sich die folgende Definition in der Umgebung befindet. Die Funktion berechnet die geometrische Reihe für die Zahl 2 bis zum Index n, also $\sum_0^n 2^n$. Das Argument der Funktion ist eine natürliche Zahl, also eine ganze Zahl ≥ 0. Die Exponentialfunktion a^n in Racket ist `(expt a n)`.

```
; Nat -> Nat
; Berechnet die geometrische Reihe für die 2 bis zum Index n.
(define (geosum n)
  (cond
        [(zero? n) 1]
        [true (+ (expt 2 n) (geosum (sub1 n)))])
)
```

Es soll die Äquivalenz `(geosum n)` $\equiv$ `(- (expt 2 (+ n 1)) 1)` durch strukturelle Induktion über `n` bewiesen werden. Dabei dürfen Sie mehrere algebraische Umformungen zu einem Schritt zusammenfassen. Insbesondere kann diese Äquivalenz hilfreich sein: $2^{n+1} + (2^{n+1} - 1) \equiv 2^{n+1+1} - 1$. Außerdem dürfen Sie in einem Schritt mittels der Regel *PRIM* die folgende Äquivalenzumformung durchführen: `(add1 n)` $\equiv$ `(+ n 1)`.

(a) Stellen Sie die im **Induktionsanfang zu beweisende Äquivalenz** auf und führen Sie den **Induktionsanfang** durch. **7**

(b) Stellen Sie die im **Induktionsschritt zu beweisende Äquivalenz** und die **Induktionsannahme** auf. Führen Sie anschließend den **Induktionsschritt** durch. **9**

Vorname:	Nachname:	Matrikelnummer:

Die folgende Teilaufgabe ist nicht mehr Teil des Äquivalenzbeweises und die Funktion `geosum` wird nicht mehr benötigt.

(c) Implementieren Sie die Funktion `(flat l)` in Racket **mithilfe von Pattern-Matching**. **Sie dürfen in dieser Teilaufgabe keine Selektorfunktionen verwenden!** Diese Funktion bekommt als Argument eine Liste, die sowohl Zahlen als auch Instanzen der Struktur `pair` (Definition, siehe unten) enthält. Das Ergebnis soll eine Liste sein, in der jede Strukturinstanz durch die Elemente des Paars ersetzt wurde. **4**

```
(define-struct pair (left right))

(check-expect (flat '() ) '())
(check-expect (flat (list 1 (make-pair 2 3))) (list 1 2 3))

(define (flat l)
```

Aufgabe 6: Listen und Funktionen höherer Ordnung 11 Punkte

Hinweis: Die folgenden aus der Vorlesung bekannten Funktionen höherer Ordnung dürfen Sie in Aufgabenteil a) und b) verwenden:

```
; [X] (X -> Boolean) (listof X) -> (listof X)
; Gibt eine Liste zurück mit allen Elementen aus l, die das Prädikat p erfüllen
(filter p l)

; [X Y] (X -> Y) (listof X) -> (listof Y)
; Bildet alle Elemente aus l mit f ab und liefert die Liste der Ergebnisse.
(map f l)

; [X Y] (X Y -> Y) Y (listof X) -> Y
; Kombiniert alle Elemente der Liste l durch f, wobei das letzte Element mit
; base kombinitert wird. Die leere Liste wird auf base abgebildet, die Elemente
; werden von rechts nach links durchlaufen.
(foldr f base l)
```

Definieren Sie die nachfolgenden Racket-Funktionen in Teilaufgabe a) und b) **nur durch Aufrufe der oben genannten Funktionen höherer Ordnung**. Das heißt, der Body-Ausdruck Ihrer Funktion **darf keine *Aufrufe* an andere primitive oder selbst-definierte Funktionen enthalten**. Primitive oder selbst-definierte Funktionen dürfen aber als Argument an die Funktionen höherer Ordnung übergeben werden.

(a) **2**

```
; [X] (list-of (list-of X)) -> (list-of Number)
; Für eine Liste l, deren Elemente beliebige Listen sind, gibt die
; Funktion eine Liste der Längen der Elemente zurück
(check-expect (lengths '((1 2 3 4) (5 7 8))) '(4 3))
(define (lengths l)
```

Vorname:	Nachname:	Matrikelnummer:

(b) **3**

```
; (list-of String) Number -> (list-of String)
; Gibt für eine Liste l von Strings eine neue Liste
; zurück, die nur die Strings mit einer Länge > n enthält.
(check-expect (f '("" "a" "ab") 1) '("ab"))
(check-expect (f '("" "a" "ab") 2) '())
(check-expect (f '() 3) '())
(define (f l n)
```

(c) Implementieren Sie die untenstehende Funktion, **ohne** die Funktionen `foldr`, `map` und `filter` zu benutzen. **6**

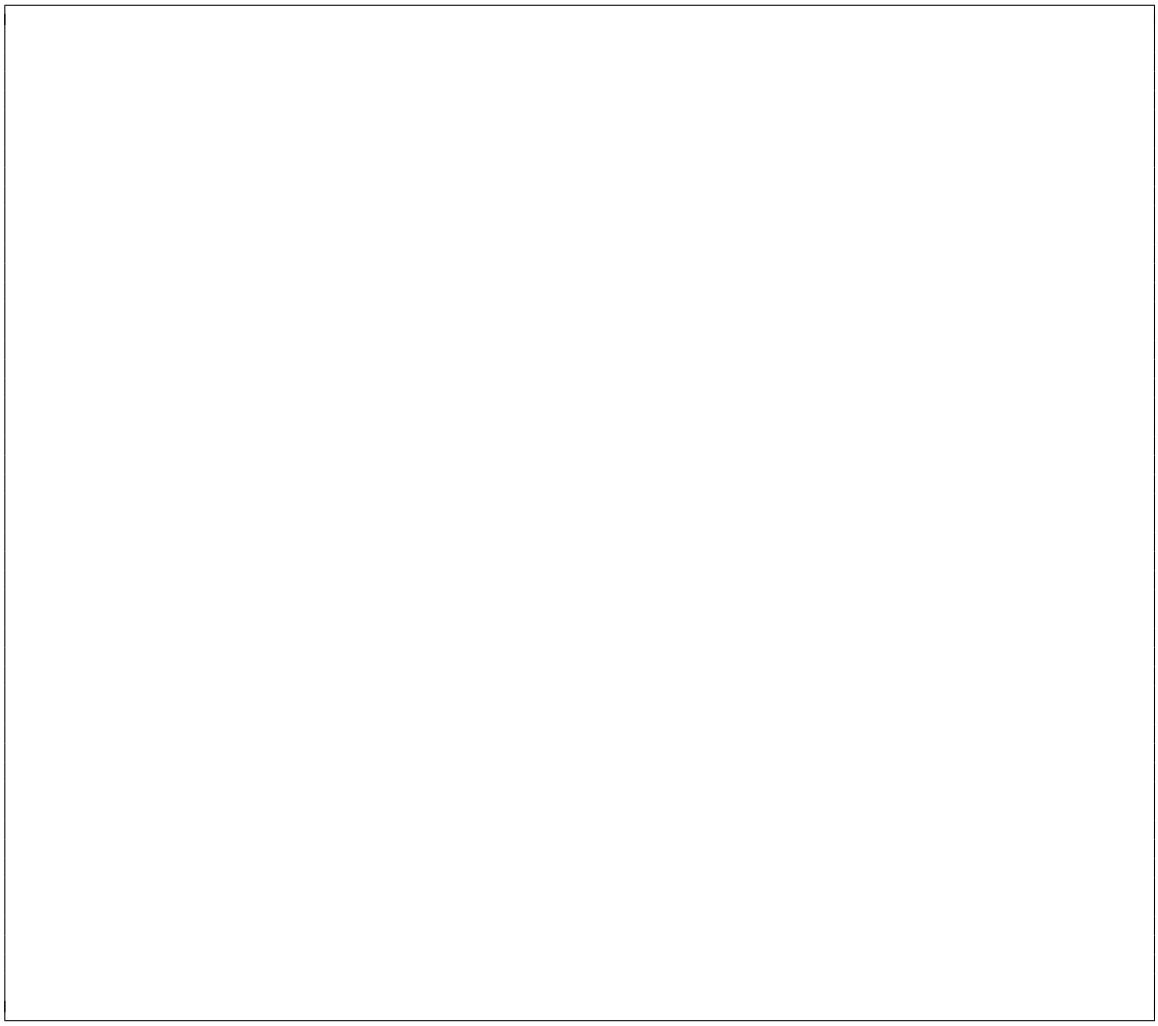
```
; Ein List-or-Number ist eins von
; - (list-of List-or-Number)
; - Number

; (Number -> Number) List-or-Number -> List-or-Number
; Die Funktion deep-map wendet die Funktion f auf jede Zahl an in val
; an und gibt eine verschachtelte Liste mit derselben Struktur wie val zurück.
(check-expect (deep-map add1 1) 2)
(check-expect (deep-map add1 '()) '())
(check-expect (deep-map add1 '(1)) '(2))
(check-expect (deep-map add1 '(1 (2 (3)))) '(2 (3 (4))))
(define (deep-map f val)
```

Aufgabe 7: Rekursion & Terminierung 10 Punkte

(a) Implementieren Sie mittels eines Akkumulators die Funktion `(count-until alon max)`. Die Funktion bestimmt, wie viele Elemente vom Anfang der List `alon` mindestens aufsummiert werden müssen, um eine Zahl $\geq$ `max` zu erhalten. Wenn `max` nicht erreicht werden kann, soll die Funktion die Anzahl der Elemente in der Liste zurückgeben. **Geben Sie die Akkumulator-Invariante mit an. Befolgen Sie das Entwurfsrezept für Funktionen mit Akkumulator.** **7**

```
; (list-of Number) Number -> Number
(check-expect (count-until '(1 3 5 7) 1) 1)
(check-expect (count-until '(1 3 5 7) 5) 3)
(check-expect (count-until '(1 3 5 7) 0) 0)
(check-expect (count-until '(1 3 5 7) 100) 4)
(check-expect (count-until '() 0) 0)
(define (count-until alon max)
```



Vorname:	Nachname:	Matrikelnummer:

(b) Betrachten Sie die folgende Funktion, die eine binäre Suche nach einem Element in einer aufsteigend sortierten Liste durchführt. Dabei gibt die Funktion `get` das Element einer List an der angegebenen Position zurück. Die Funktion `head` gibt die Teilliste vor und `tail` die Teilliste nach der angegebenen Position zurück. **3**

```
(define (binary-search l e)
  (if (empty? l) false
      (local [(define middle (quotient (length l) 2))
              (define middle-element (get l middle))]
        (cond [(= e middle-element) true]
              [(> e middle-element) (binary-search (tail l middle) e)]
              [(< e middle-element) (binary-search (head l middle) e)]))))
```

Begründen Sie, warum die Funktion `binary-search` stets terminiert.

Aufgabe 8: Prolog 13 Punkte

(a) Definieren Sie die folgende Prozedur in Prolog. Die Verwendung von Bibliotheks-Prozeduren aus Prolog ist dabei **nicht** gestattet. Definieren Sie benötigte Hilfsprozeduren selbst. **4**

Hinweis: Die arithmetischen Vergleichsprädikate <, >, =<, >=, == (Gleichheit), \= (Ungleichheit) dürfen verwendet werden und werden in Prolog infix geschrieben, zum Beispiel: `?- 2 =< 3.`

`count(L, V, C)`: `L` ist eine Liste, `V` ist ein beliebiger Wert und `C` ist eine Zahl. Dabei ist `C` die Anzahl, wie oft das Element `V` in der Liste `L` vorkommt. Als Beispiel nehmen Sie die Abfragen:

- `?- count( [1, 2, 1], 1, Cnt).`
- `?- count( [1, 2, 1], 2, Cnt).`
- `?- count( [1, 2, 1], 3, Cnt).`

Alle drei abfragen sind erfüllt jeweils mit den Substitutionen `Cnt = 2`, `Cnt = 1` beziehungsweise `Cnt = 0`.

Vorname:	Nachname:	Matrikelnummer:

Gegeben seien die folgenden Definitionen der Prozeduren b – d und die Abfragen in Teilaufgaben b) – d), die diese verwenden. Geben Sie für jede Abfrage an, was das **Resultat** ist. Wenn eine Abfrage erfüllt ist, geben Sie eine gültige **Substitution** aller Variablen an. Im Fall eines **Fehlers** geben Sie eine kurze **Begründung** an.

```
b(l( X ), [ X ]).
b(n(L, R),[ X, Y ]) :- b(L, X), b(R, Y).

c(X, Y, Z) :- Z is X + Y.

d([ insekt(biene), fisch(guppy) ], X).
```

(b) `?- b( T, [ [ 1 ] , [ 2 ] ] ).` **3**

(c) `?- c(1, U, 3).` **3**

(d) `?- d([ insekt(Guppy), fisch(Biene) ], [ Guppy, biene ]).` **3**