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

Sommersemester 2020
01.10.2020

Prof. Dr. Christoph Bockisch

Klausur zur Vorlesung

Deklarative Programmierung

Wichtige Hinweise:

- Schalten Sie, soweit noch nicht geschehen, sofort Ihr Mobiltelefon aus!
- Schalten Sie auch alle anderen 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 außerdem Ihren Studentenausweis und Ihren Personalausweis/Reisepass bereit.
- Schreiben Sie Ihren Namen und Ihre Matrikelnummer auf jedes Blatt in Druckbuchstaben. Blätter ohne Namen ergeben 0 Punkte und werden nicht korrigiert! Füllen Sie insbesondere auch folgende Tabelle in Druckbuchstaben aus:

Vorname	
Nachname	
Matrikelnummer	
Fachbereich	
Studienfach	
Angestrebter Abschluss	

- Die Bearbeitungszeit beträgt **2** Stunden.
- Verwenden Sie kein eigenes Papier für Notizen. Am Ende des Klausurbogens befindet sich eine Extraseite. In dringenden Fällen erhalten Sie auf Anfrage Zusatzblätter. Machen Sie gut kenntlich, wenn Sie Zusatzblätter für Lösungen verwenden und tragen Sie dort Name und Matrikelnummer ein.
- Es sind keine eigenen Hilfsmittel erlaubt. Zuwiderhandlungen führen zum Ausschluss.
- 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	9	18	9	20	11	12	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 **gebundenes** Vorkommen einer Variablen ist. Geben Sie ein Beispiel. **2**

(b) Was bedeutet Typisomorpie? Verwenden Sie in Ihrer Erklärung **nicht die Begriffe** "Bijektion" oder "bijektiv". **2**

(c) Was ist in Prolog ein **universeller Fakt**? Beschreiben Sie, was für eine Form und was für eine Bedeutung so ein Fakt hat. **2**

(d) Beschreiben Sie die Aufgabe des "World State" in einem interaktiven Racket Programm, das mittels der Funktion `big-bang` implementiert ist. **2**

Aufgabe 2: Ausdrücke 9 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 (Syntax-, Typ- oder Laufzeitfehler) und Ursache des Fehlers an.

(a) `(and (< 0 -1) (< 0 (/ 1 0)))` **2**

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

```
`( (+ a b) (3 4))
```

(c) **2**

```
(define x "(/ 2 4)")
(cond [(equal? x "2") 1]
      [(equal? x "0.5") 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. **3**

```
1 ; In der Umgebung befindet sich diese Definition
2 (define (f a) (make-posn a a))
3
4 ; reduzieren Sie den folgenden Aufruf
5 (posn? (f 1))
```

Aufgabe 3: Typ- und Funktionsdefinitionen 18 Punkte

Sie entwickeln eine Software zur Planung der Bewässerung in einer Gärtnerei. Hier geht es darum, Pflanzkübel zu verwalten, und zu berechnen, wie viel Wasser zum Gießen benötigt wird. Sie müssen verschiedene Informationen verarbeiten:

- Es gibt verschiedene Arten von Erde, die verwendet wird, nämlich: Sandboden, Lehmboden, Tonboden.
- Die verschiedenen Erde-Arten können unterschiedlich viel Wasser aufnehmen. Pro Liter Erde nehmen sie so viel Wasser auf: Sandboden 0,3 Liter, Lehmboden 0,2 Liter, Tonboden 0,1 Liter.
- Ein Pflanzkübel hat zwei Eigenschaften: Das Volumen an enthaltener Erde in Liter und die Art der Erde.
- Eine Gärtnerei hat eine Liste von Pflanzkübeln.

(a) Definieren Sie Datentypen für `Erde` und `Kuebel` und geben Sie an, um welche Art von Datentyp es sich jeweils handelt. **5**

Vorname:	Nachname:	Matrikelnummer:

(b) Schreiben Sie eine Funktion (`wasser-kuebel k`) (Signatur `Kuebel -> Number`), die für einen Pflanzkübel (`k`) , die benötigte Menge Wasser zum Gießen berechnet. **Denken Sie daran, das Entwurfsrezept für Funktionen zu beachten.** **6**

(c) Schreiben Sie nun eine Funktion (`wasser-gesamt lok`), die für eine Liste von Kübeln die insgesamt benötigte Wassermenge berechnet. Sie müssen in dieser Teilaufgabe keine Tests definieren. **4**

```
(define (wasser-gesamt lok)
```

Für die folgenden Teilaufgaben benötigen Sie **nicht mehr die obigen Angaben**.

(d) Gegeben ist die untenstehende Funktion mit einer unvollständigen Signatur. Geben Sie den fehlenden Rückgabetyp der Funktion an und begründen Sie diesen indem Sie **auf die Funktionsdefinition Bezug nehmen**. **3**

```
1 ; [U] (Number -> U) U -> ________
2 (define (f x y)
3   (lambda (z)
4     (cond [(> z 0) (x z)]
5           [else y])))
```

Vorname:	Nachname:	Matrikelnummer:

Aufgabe 4: Algebraische Datentypen 9 Punkte

Es soll ein Programm zur Darstellung zweidimensionaler geometrischer Formen geschrieben werden. Im ersten Schritt soll das Programm mit Kreisen und rechtwinkligen Dreiecken arbeiten. Ein Kreis wird beschrieben durch die Position seines Mittelpunkts (x- und y-Koordinate) und einen Radius (r). Dreiecke werden durch die Position Ihres Eckpunkts mit dem rechten Winkel (x- und y-Koordinate) und die Längen (k1 und k2) der beiden von dort ausgehenden Seiten (Katheten) beschrieben.

(a) Definieren Sie geeignete Strukturen zur Repräsentation von *Kreis*- und *Dreieck*-Werten und geben Sie eine Datendefinition für den **Algebraischen Typen** *Geometrie* an. **Dokumentieren Sie die Datentypen so, wie Sie es in der Vorlesung gelernt haben**, die "Interpretation" dürfen Sie dabei weglassen. **5**

(b) Implementieren Sie eine Funktion `area` zur Berechnung der Fläche eines `Geometrie`-Wertes basierend auf der Definition aus Aufgabenteil a). Die Kreisfläche berechnet sich durch $3.14 \cdot radius^2$, die Dreieckfläche durch $\frac{katehete_1 \cdot katehete_2}{2}$. Wenden Sie die **Schablone für algebraische Datentypen** an, Tests dürfen Sie dabei auslassen. **4**

Aufgabe 5: Äquivalenz und Pattern Matching 20 Punkte

Betrachten Sie die folgende Funktionsdefinition in Racket, welche die Summe aller Quadratzahlen (daher der Name `sq`) bis n berechnet (z.B. $sq(1) = 1$, $sq(2) = 1 + 4 = 5$, etc.). Die Funktion `sq` berechnet das Ergebnis rekursiv. Es lässt sich auch ein Ausdruck aufstellen, der das Ergebnis direkt über eine Formel berechnet: `(/ (* n (+ n 1) (+ (* 2 n) 1)) 6)`.

Es soll die Äquivalenz `(sq n)` $\equiv$ `(/ (* n (+ n 1) (+ (* 2 n) 1)) 6)` durch strukturelle Induktion über `n` bewiesen werden. Gehen Sie daher davon aus, dass `n` eine natürliche Zahl (inklusive der Null) ist. Geben Sie jeweils die **Namen aller Auswertungs- oder Äquivalenzregeln** an, die sie benutzen.

Verwenden Sie in Ihrem Beweis die folgende **zusätzliche Äquivalenz-Regel**:
(**EVORGABE**) $(n+1)\cdot(n+1) + \frac{n\cdot(n+1)\cdot(2n+1)}{6} \equiv \frac{(n+1)\cdot((n+1)+1)\cdot(2(n+1)+1)}{6}$

Hinweis: Die Multiplikation in Racket kann beliebig viele Argumente miteinander multiplizieren. Zum Beispiel entspricht `(* a b c)` der Formel $a \cdot b \cdot c$.

Hinweis: Notieren Sie in jedem Schritt den relevanten Teil des Programms. Unveränderte Teile können durch „... “ abgekürzt werden.

Hinweis: Mehrere arithmetische Umformungen, die direkt hintereinander durchgeführt werden, können Sie zu einem Schritt in der Äquivalenzumformung zusammenfassen.

Hinweis: Sie können für diese Aufgabe den ausgeteilten Zettel mit Auswertungs- und Äquivalenzregeln verwenden.

```
; Number -> Number
(define (sq n)
  (cond [(zero? n) 0]
        [true (+ (* n n)
                 (sq (sub1 n)))]))
```

Vorname:	Nachname:	Matrikelnummer:

(a) Stellen Sie die im **Induktionsanfang** zu beweisende Äquivalenz auf und führen Sie den Induktionsanfang durch. **6**

(b) Stellen Sie die im **Induktionsschritt** zu beweisende Äquivalenz und die Induktionsannahme auf. Führen Sie anschließend den Induktionsschritt durch. **10**

Vorname:	Nachname:	Matrikelnummer:

Die folgende Teilaufgabe ist **nicht mehr Teil des Äquivalenzbeweises**.

(c) Implementieren Sie die Funktion `(to-point v)` in Racket **mithilfe von Pattern-Matching**. **4**
Sie dürfen in dieser Teilaufgabe **keine Selektorfunktionen verwenden**! Die Funktion `to-point` bekommt ein Argument, das entweder eine Instanz der Struktur `pair` oder eine Liste mit zwei Elementen ist. Das heißt, in beiden Fällen sind in dem Argument zwei Werte verpackt. Das Ergebnis der Funktion soll eine Instanz der Struktur `point` sein, die als Felder diese beiden Werte hat.

```
(define-struct pair (p1 p2))
(define-struct point (x y))

(check-expect (to-point (make-pair 1 2)) (make-point 1 2))
(check-expect (to-point '(1 2)) (make-point 1 2))
(define (to-point v)
```

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**

```
; (listof Number) -> (listof Boolean)
; Gibt für eine Liste von Zahlen l eine neue Liste mit derselben
; Anzahl an Boolean-Werten zurück, die angeben,
; ob die entsprechende Zahl (d.h., die Zahl an derselben Position in l)
; nicht kleiner als 1 (also 1 oder höher) ist.

(check-expect (f '(-1 0 1 2)) (list false false true true))
(check-expect (f '()) (list ))

(define (f l)
```

Vorname:	Nachname:	Matrikelnummer:

(b) **3**

```
; [X] (listof X) -> Boolean
; Gibt true zurück, wenn die Liste l eine gerade Anzahl von
; Elementen enthält und false sonst. Für die leere Liste
; soll true zurückgegeben werden.

(check-expect (g '()) true)
(check-expect (g '(1 2 3)) false)
(check-expect (g '(1 2 3 4)) true)

(define (g l)
```

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

```
; [X] (X -> Boolean) (listof X) -> (listof X)
; Gibt eine Liste zurück, die die Elemente der übergebenen Liste l
; enthält, wobei jedes Element, das das Prädikat p erfüllt
; dupliziert ist.

(check-expect (duplicateif even? '()) (list ))
(check-expect (duplicateif even? '(0 1 2 3)) (list 0 0 1 2 2 3))
(define (duplicateif p l)
```

Aufgabe 7: Rekursion & Terminierung

12 Punkte

(a) Implementieren Sie die folgende Funktion mittels eines **Akkumulators**. Die Funktion bekommt eine Liste von Zahlen (`l`) übergeben und gibt auch eine Liste von Zahlen zurück. Die Elemente der Ergebnisliste berechnen sich durch Addition des entsprechenden Elements in `l` mit der Position des Elements. Die Position des ersten Elements ist `1`. Verwenden Sie die **Schablone für Funktionen mit Akkumulator** und **geben Sie die Akkumulator-Invariante mit an**.

7

```
; (listof Number) -> (listof Number)
(check-expect (add-pos '()) '())
(check-expect (add-pos '(0 1 2 3)) '(1 3 5 7))
(define (add-pos l)
```

Vorname:	Nachname:	Matrikelnummer:

(b) Die folgende Funktion `loc-los` wandelt eine Liste von Zeichen (`loc`) in eine Liste von Sätzen um. Dabei wird ein Satz wiederum durch eine Liste von Zeichen dargestellt, die durch das Zeichen `"."` (Punkt) beendet wird. Alle Zeichen aus `loc`, die nach dem letzten Punkt folgen werden verworfen. In der Implementierung werden die folgenden primitiven Funktionen verwendet: **5**

```
; [X] (listof X) X -> NumberOrFalse
; Wenn die Liste l das Element e enthält, wird der Index des ersten
; Vorkommens zurückgegeben. Sonst wird false zurückgegeben.
; Der Index des ersten Elements ist 0.
(index-of l e)

; [X] (listof X) Number -> (listof X)
; Gibt die Teilliste aller Elemente aus l mit einem Index < n zurück.
(take l n)

; [X] (listof X) Number -> (listof X)
; Gibt die Teilliste aller Elemente aus l mit einem Index >= n zurück.
(drop l n)
```

Um welche Form der Rekursion handelt es sich bei der folgenden Funktion? Terminiert diese Funktion? Begründen Sie Ihre Antwort.

```
(check-expect (loc->los '()) '())
(check-expect (loc->los '("a" "." "b" ".")) '(("a" ".") ("b" ".")))
(check-expect (loc->los '("a" "." "b")) '(("a" ".")))

(define (loc->los loc)
  (local [(define pos (index-of loc "."))]
    (cond [(equal? pos false) '()]
          [else (cons (take loc (add1 pos))
                      (loc->los (drop loc (add1 pos))))])))
```

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. **6**

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

`prefix(L1, L2, P)`: Die Argumente `L1`, `L2` und `P` sind Listen. Dabei ist das Prädikat `prefix` erfüllt, wenn `P` das längste gemeinsame Präfix von `L1` und `L2` ist. Zum Beispiel ist die folgende Abfrage erfüllt:

```
?- prefix([a, b, c],[a, b, d], [a, b]).
```

Nicht erfüllt sind hingegen die Abfragen:

```
?- prefix([a, b, c],[a, b, d], [a]).
?- prefix([a, b],[c, d], [e]).
```

Vorname:	Nachname:	Matrikelnummer:

Gegeben seien die folgenden Definitionen der Prozeduren b – c und die Abfragen in Teilaufgaben b) – c), 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(t,Xs,Xs).
b(p(T), [X|Xs], Ys) :- b(T, Xs, Ys).

c([hund,katze,maus],Vogel).
```

(b) `?- b(Trace, [1, 2, 3], [X]).` **4**

(c) `?- c([Hund|Katze],maus).` **3**

Extraseite