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

Sommersemester 2021
24.09.2021

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 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 | 16 | 10 | 20 | 12 | 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 **universeller Fakt** in Prolog ist. Geben Sie ein **Beispiel**. **2**

(b) Beschreiben Sie, die **Kongruenz-Regel** bei der Auswertung von Ausdrücken. Geben Sie ein **Beispiel** an. **2**

(c) Was muss bei der Reihenfolge der Bedingungen in einem `cond` Ausdruck berücksichtigt werden? **2**

(d) Beschreiben Sie das Geheimnisprinzip (Information Hiding). **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) `((if (< 1 -1) 21 (lambda (x) 42)) 7)` **2**

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

```
(rest '(lambda (x) (sub1 x)))
```

(c) `(+ (string->number "(+ 1 2)") 1)` **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) (cond [(posn? x) (posn-y x)] [else false]))
3
4  ; reduzieren Sie den folgenden Aufruf
5  (f (make-posn 3 4))
```

**Aufgabe 3: Funktionsdefinitionen und Signaturen** 16 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! **8**

```
1  (define (f x y z)
2    (if (> (first y) 5)
3           ((first x) (first y))
4           z))
```

| Vorname: | Nachname: | Matrikelnummer: |
|---|---|---|
| | | |

Gegeben sind folgende Struktur- und Datendefinitionen:

```
(define-struct node (op left right))
; Ein Node ist eine Struktur: (make-branch Operator Tree Tree)
; interp. ein innerer Knoten in einem Ausdruck-Baum.
; Das heißt, jeder Knoten steht für einen binären arithmetischen
; Ausdruck mit dem Operator op, dem linken Operanden left
; und dem rechten Operanden right.

; Ein Operator ist eins von:
; - '+
; - '-
; interp. jedes Element steht für eine Grundrechenart.

; Ein Tree ist eins von:
; - Node
; - Number
; interp. ein Ausdruck-Baum, der für einen arithmetischen
; Ausdruck steht, der entweder zusammengesetzt ist oder
; eine Zahl.
```

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

- Tree ist ein _____.
- Operator ist ein _____ und hat

  die Kardinalität _____.

(c) Die Funktion `compute` soll einen Tree-Wert als Argument entgegen nehmen und den berechneten Wert des entsprechenden Ausdrucks zurückliefern. Geben Sie für diese Funktion folgendes an: Signatur, Testfälle in ausreichender Anzahl (gemäß des zutreffenden Entwurfsrezepts), die **Schablone** der Implementierung. Die Implementierung selbst ist nicht gefordert. **5**

| Vorname: | Nachname: | Matrikelnummer: |
|---|---|---|
| | | |

**Aufgabe 4: Algebraische Datentypen** 10 Punkte

Es soll ein Programm zur Verwaltung einer Bibliothek von Medien geschrieben werden. Als Medien sollen Bücher und Hörbücher unterstützt werden, wobei jedes Medium einen Titel und einen Autor hat, Hörbücher haben außerdem noch einen Sprecher. Ein Buch hat zusätzlich eine Liste, die für jedes Kapitel die Anzahl der Wörter speichert, während ein Hörbuch zusätzlich eine Gesamt-Spieldauer in Minuten hat.

(a) Definieren Sie geeignete Strukturen zur Repräsentation der Datentypen *Buch* und *Hörbuch* und geben Sie eine Datendefinition für den **Algebraischen Datentypen** *Medium* an. **4**

(b) Implementieren Sie eine Funktion `(dauer medium)` zur Berechnung der Zeit in Minuten, die man braucht, um das Buch ganz zu lesen beziehungsweise das Hörbuch ganz zu hören. Gehen Sie dabei von einer durchschnittlichen Lesegeschwindigkeit von 200 Worten pro Minute aus. **Wenden Sie die Schablone für algebraische Datentypen an.** Funktionssignaturen und Tests müssen **nicht** angegeben werden. **6**

```
(define (dauer medium)
```

| Vorname: | Nachname: | Matrikelnummer: |
|---|---|---|
| | | |

**Aufgabe 5: Reduktion und Äquivalenz** 20 Punkte

Für die Teilaufgaben a) und b) gehen Sie davon aus, dass sich die folgende Definition in der Umgebung befindet. Das Argument der Funktion ist eine natürliche Zahl, also eine ganze Zahl $\geq 0$. Die Funktion berechnet den Wert an Index $n$ für die Folge:

$$a_n = \begin{cases} 3 & \text{für } n = 0 \\ 2 + a_{n-1} & \text{sonst} \end{cases}$$

```
; Nat -> Nat
; Berechnet das Folgenelement an Index n.
(define (folge n)
  (cond [(zero? n) 3]
        [true (+ 2 (folge (sub1 n)))]))
```

Es soll die Äquivalenz `(folge n)` $\equiv$ `(+ 3 (* 2 n))` durch strukturelle Induktion über `n` bewiesen werden. Dabei dürfen Sie mehrere algebraische Umformungen zu einem Schritt zusammenfassen. Außerdem dürfen Sie in einem Schritt mittels der Regel *PRIM* die folgende Äquivalenzumformung durchführen: `(add1 n)` $\equiv$ `(+ n 1)`.

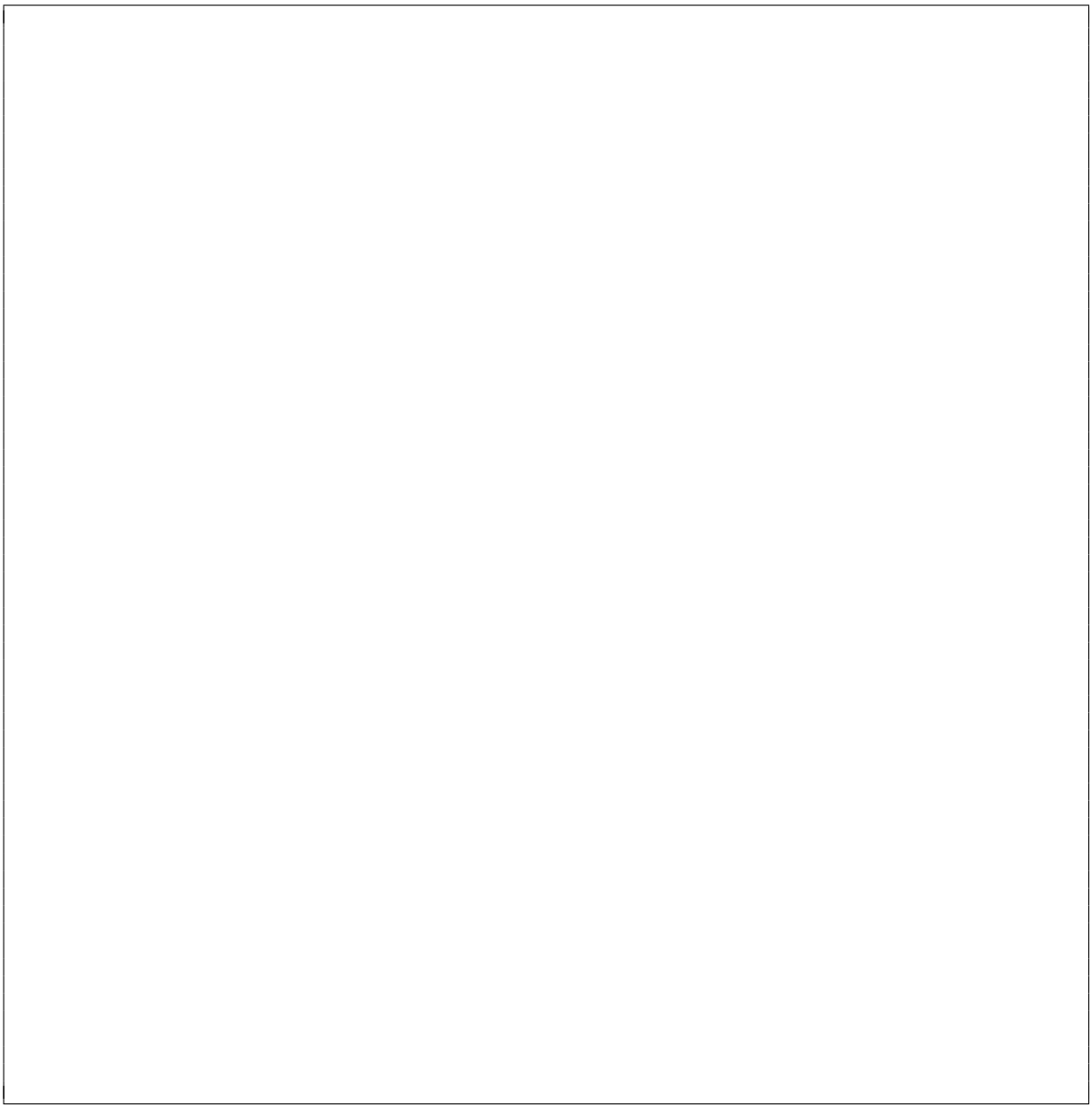
(a) Stellen Sie die im **Induktionsanfang zu beweisende Äquivalenz** auf und führen Sie den **Induktionsanfang** durch. **5**

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

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

(c) Implementieren Sie die Funktion `(eval l)` in Racket **mithilfe von Pattern-Matching**. **Sie dürfen in dieser Teilaufgabe keine Selektorfunktionen verwenden! Dazu gehören zum Beispiel auch die Listen-Funktionen `rest` und `first`.** Die Funktion `eval` bekommt als Argument eine S-Expression übergeben, die erzeugt wird, indem der Quote-Operator auf einen Ausdruck in Racket-Schreibweise angewendet wird. Unterstützt werden dabei nur Zahlen-Literale, der binäre Operator `+` und der unäre Operator `sqr` für die Quadrat-Funktion. Das Ergebnis von `eval` soll der Wert des Ausdrucks sein. **5**

```
(check-expect (eval 5) 5)
(check-expect (eval '(+ (sqr 2) 3)) 7)
(define (eval e)
```



| Vorname: | Nachname: | Matrikelnummer: |
|---|---|---|
| | | |

**Aufgabe 6: Listen und Funktionen höherer Ordnung** 12 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**

```
; (list-of Any) -> (list-of Number)
; Für eine nicht-homogene Liste l, in der jedes Element
; einen beliebigen Typ haben kann, gibt f eine Liste
; zurück, die nur die Zahlen-Werte aus der Liste enthält.
(check-expect (f '("a" 1 false 2)) '(1 2))
(check-expect (f '("a")) '())
(check-expect (f '()) '())
(define (f l)
```

(b) **3**

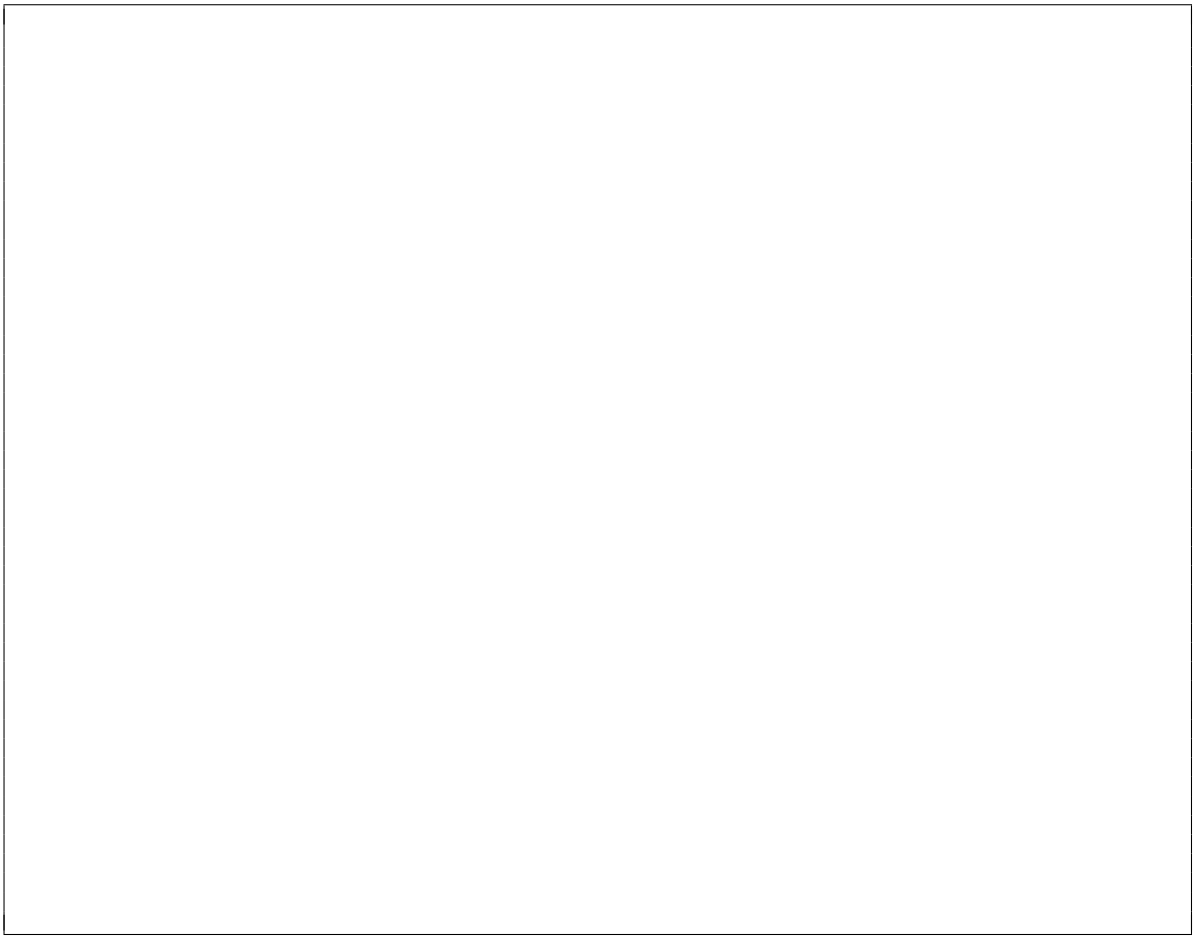
```
; [U] (list-of Posn) -> (list-of U)
; Für eine Liste von Instanzen der posn Struktur gibt f
; eine Liste der x-Komponenten dieser Instanzen zurück.
; Dabei steht U für den Typ der x-Komponente.
(check-expect (f (list (make-posn 1 2) (make-posn 3 4)))
              '(1 3))
(check-expect (f (list (make-posn "a" "b"))) '("a"))
(check-expect (f '()) '())
(define (f l)
```

| Vorname: | Nachname: | Matrikelnummer: |
|---|---|---|
| | | |

(c) Implementieren Sie die untenstehende Funktion, **ohne** die Funktionen `foldr`, `map` und `filter` zu benutzen. **7**

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

; (Number Number -> Number) Number List-or-Number -> Number
; Die Funktion deep-foldr Kombiniert alle Zahlen-Elemente
; in val durch f, wobei das letzte Element mit n
; kombinitert wird. Die leere Liste wird auf n abgebildet,
; und die Elemente werden von rechts nach links durchlaufen,
; wobei im Fall von verschachtelten Listen auch in die
; Listen abgestiegen wird.
(check-expect (deep-foldr + 0 1) 1)
(check-expect (deep-foldr + 0 '()) 0)
(check-expect (deep-foldr + 0 '(1)) 1)
(check-expect (deep-foldr + 0 '(1 2 (3))) 6)
(define (deep-foldr f n val)
```



**Aufgabe 7: Rekursion & Terminierung** 10 Punkte

(a) Implementieren Sie mithilfe von Akkumulatoren die Funktion `(average alon)`. Die Funktion bekommt eine Liste von Zahlen übergeben, die **nicht leer** ist. Das Ergebnis der Funktion soll der Durchschnitt aller Werte in der Liste sein. Verwenden Sie hierzu **zwei Akkumulatoren**. **Geben Sie für jeden Akkumulator die Akkumulator-Invariante mit an. Befolgen Sie das Entwurfsrezept für Funktionen mit Akkumulator.** **7**

```
; (list-of Number) -> Number
; (average '()) ist nicht definiert, daher gibt es
; hierfür keinen Test.
(check-expect (average '(1)) 1)
(check-expect (average '(1 2 3)) 2)
(define (average lon)
```

| Vorname: | Nachname: | Matrikelnummer: |
|---|---|---|
| | | |

(b) Betrachten Sie die folgende Funktion, die mithilfe des Euklidischen Algorithmus den größten gemeinsamen Teiler von zwei Zahlen ermittelt. Gehen Sie davon aus, dass die Funktion nur natürliche Zahlen ohne die Null, also **ganze Zahlen** $\geq$ **1**, übergeben bekommt (dieser Typ heißt `Nat*`). **3**

```
; Nat* Nat* -> Nat*
(define (ggt x y)
    (cond
      [(= x y) x]
      [(> x y) (ggt (- x y) y)]
      [(< x y) (ggt x (- y x))]))
```

Begründen Sie, warum die Funktion `ggt` stets terminiert.

**Aufgabe 8: Prolog** 13 Punkte

(a) Definieren Sie die folgende Prozedur in Prolog. Die Verwendung von Bibliotheksprozeduren 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.`

`max(L, M)`: `L` ist eine Liste von Zahlen und `M` ist eine Zahl. Dabei ist `M` die größte Zahl, die in der Liste `L` vorkommt. Für die leere Liste soll die Relation `max` für alle `M` **nicht erfüllt** sein. Als Beispiel nehmen Sie die Abfragen:

- `?- max([1, 4, 9, 5], M).` - Diese Abfrage ist erfüllt mit der Substitution `M = 9`.
- `?- max([], 0).` - Diese Abfrage ist nicht erfüllt.

| 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(e(),[]).
b(s(X),[X]).
b(p(X,Y),[X,Y]).
b(l(p(X,Y),U), [X,Y,Z|R]) :- b(U,[Z|R]).

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

d(X, [sauer(zitrone,essig), suess(zucker,apfel)], Z).
```

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

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

(d) `?- d(Essig,[sauer(Essig,essig), suess(zucker,Apfel)],Apfel).` **3**

Extraseite 1