Prof. Dr. Ralf Hinze
Cass Alexandru, M.Sc.
Alexander Dinges, M.Sc.
Sebastian Schloßer, M.Sc.

RPTU
Fachbereich Informatik
AG Programmiersprachen

Lösungshinweise/-vorschläge zum Übungsblatt 3: Grundlagen der Programmierung (WS 2024/25)

Aufgabe 1 Peano und Leibniz Entwurfsmuster (Präsenzaufgabe)

Motivation: Diese Aufgabe soll Sie auf die beiden folgenden Einreichaufgaben vorbereiten. Dazu schauen wir uns einige Beispielanwendungen des Peano und Leibniz Entwurfsmusters an und beleuchten die Unterschiede der beiden Ansätze. Sie können sich an den Vorlesungsfolien 199 bis 233 sowie an den Kapiteln 3.7.1 und 3.7.2 im Skript orientieren.

Schreiben Sie Ihre Lösungen in die Datei `Entwurfsmuster.fs` *aus der Vorlage* `Aufgabe-3-1.zip`.

a) Wiederholen Sie Peano und Leibniz Entwurfsmuster. Wo liegen die Unterschiede der beiden Entwurfsmuster? Warum verwenden wir diese?

Peano, linear viele Schritte

```
let rec f (n: Nat): t =
    if n = 0N then ...
    else ... f (n - 1N) ...
```

Leibniz, logarithmisch viele Schritte

```
let rec f (n: Nat): t =
    if n = 0N then ...
    else ... n % 2N ... f (n / 2N) ...
```

Skript: „Rekursion birgt die Gefahr der Nichtterminierung. Entwurfsmuster bannen diese Gefahr und unterstützen Programmierende bei der systematischen Lösung von Problemen.“

b) Schreiben Sie eine Funktion `add: Nat -> Nat`, die alle natürlichen Zahlen bis `n` aufaddiert. Gehen Sie nach Peano Entwurfsmuster vor.

Beispiele:

```
add 0N = 0N        add 2N = 3N        add 3N = 6N        add 10N = 55N
```

```
let rec add (n: Nat): Nat =
    if n = 0N then 0N
    else n + add (n - 1N)
```

c) Schreiben Sie eine rekursive Funktion `mod5: Nat -> Nat`, die für eine gegebene Zahl `n` den Rest der Ganzzahldivision durch `5` berechnet. Es soll also `mod5 n = n % 5` gelten. Gehen Sie nach Peano Entwurfsmuster vor. Verwenden Sie *nicht* die Funktionen `/` oder `%`.

Beispiele:

```
mod5 0N = 0N        mod5 2N = 2N        mod5 4N = 4N        mod5 19N = 4N       mod5 21N = 1N
mod5 1N = 1N        mod5 3N = 3N        mod5 5N = 0N        mod5 20N = 0N       mod5 22N = 2N
```

```
let rec mod5 (n: Nat): Nat =
    if n = 0N then 0N
    else
        let m = mod5 (n - 1N)
        if m = 4N then 0N
        else m + 1N
```

d) Schreiben Sie eine rekursive Funktion `mult42: Nat -> Nat`, die die übergebene Zahl `n` mit `42` multipliziert. Sie dürfen in Ihrer Lösung jedoch **nicht die Multiplikationsfunktion `*` verwenden**. Gehen Sie stattdessen **nach Leibniz Entwurfsmuster** vor, sodass die **Berechnung in logarithmisch vielen Schritten** erfolgt.

Beispiele:

```
mult42 0N = 0N          mult42 1N = 42N         mult42 2N = 84N         mult42 5N = 210N
```

```
let rec mult42 (n: Nat): Nat =
    if n = 0N then 0N
    else
        let r = mult42 (n / 2N)
        r + r + if n % 2N = 0N then 0N else 42N
```

e) Schreiben Sie eine Funktion `count5: Nat -> Nat`, die zählt, wie oft die Ziffer `5` in einer Zahl vorkommt. Gehen Sie nach Leibniz Entwurfsmuster vor.

Hinweis: Auch wenn das Leibniz Entwurfsmuster in der Vorlesung mit Divisor 2 eingeführt wurde, dürfen Sie wenn nötig einen anderen Divisor wählen. Wichtig ist lediglich, dass sowohl bei der Division als auch beim Modulo-Operator der selbe Divisor verwendet wird.

Beispiele:

```
count5 3N = 0N          count5 76567N = 1N      count5 1234N = 0N       count5 445566N = 2N
```

```
let rec count5 (n: Nat): Nat =
    if n = 0N then 0N
    else
        if n % 10N = 5N then 1N else 0N
        +
        count5(n / 10N)
```

Aufgabe 2 Peano Entwurfsmuster (Einreichaufgabe, 6 Punkte)

Motivation: In dieser Aufgabe sollen Sie das Peano Entwurfsmuster einüben. Sie können sich an den Vorlesungsfolien 199 bis 216 sowie am Skript Kapitel 3.7.1 orientieren.

Schreiben Sie Ihre Lösungen in die Datei `Peano.fs` *aus der Vorlage* `Aufgabe-3-2.zip`.

Anmerkung: Ihre Implementierung soll dem Peano Entwurfsmuster folgen, Sie sollen nicht die `peano-pattern` Funktion von Folie 214 verwenden.

a) Schreiben Sie eine rekursive Funktion `iterate`, die eine Funktion `f` vom Typ `Nat -> Nat` sowie eine natürliche Zahl `n` übergeben bekommt. Die Funktion `iterate f n` soll eine Funktion zurückgeben, die der `n`-fachen Anwendung von `f` entspricht. Für `n = 3` z.B. gilt: `(iterate f 3N) m = f (f (f m))`. Wir vereinbaren, dass `(iterate f 0) m = m`.

Beispiele:

```
(iterate (fun x -> 1N + x * x) 0N) 0N = 0N
(iterate (fun x -> 1N + x * x) 1N) 0N = 1N
(iterate (fun x -> 1N + x * x) 2N) 0N = 2N
(iterate (fun x -> 1N + x * x) 3N) 0N = 5N
(iterate (fun x -> 1N + x * x) 4N) 0N = 26N
(iterate (fun x -> 1N + x * x) 5N) 0N = 677N
```

```
let rec iterate (f : Nat -> Nat) (n : Nat) : Nat -> Nat =
    if n = 0N then (fun n -> n)
    else (fun m -> f ((iterate f (n - 1N)) m))
```

b) Schreiben Sie eine rekursive Funktion `lt: Nat -> Nat -> Bool`, die für zwei Natürliche Zahlen `n,m` den Wert $n < m$ berechnet, ohne dabei die Vergleischsoperationen `<`, `<=`, `>`, `>=` zu verwenden. Die Verwendung von `=` ist hingegen gestattet.

Beispiele:

```
lt 0N 1N = true
lt 0N 0N = false
lt 5N 3N = false
lt 1N 6N = true
```

Tipp: Denken Sie darüber nach, auf welches der beiden Argumente Sie das Peano Entwurfsmuster anwenden sollten!

```
let rec lt (n : Nat) (m : Nat) : Bool =
    if m = 0N then false
    else
        let pre = (m - 1N)
        let nminlt = lt n pre
        (n = pre) || nminlt
```

Aufgabe 3 Leibniz Entwurfsmuster (Einreichaufgabe, 6 Punkte)

Motivation: In dieser Aufgabe sollen Sie das Leibniz Entwurfsmuster einüben. Sie können sich an den Vorlesungsfolien 219 bis 234 sowie am Skript Kapitel 3.7.2 orientieren.

Schreiben Sie Ihre Lösungen in die Datei `Leibniz.fs` *aus der Vorlage* `Aufgabe-3-3.zip`.

Hinweis: Wie schon in der Präsenzaufgabe erwähnt muss der Divisor im Leibniz Entwurfsmuster nicht unbedingt 2 sein.

a) In der Vorlesung wurde die Funktion `binarySearch` (Folie 248) definiert, und verwendet um eine Inverse zur Quadratfunktion x^2, also $\lfloor \sqrt{x} \rfloor$ zu definieren. Verwenden Sie sie, um eine Inverse zur Funktion 2^x, also $\lfloor \log_2 x \rfloor$ zu definieren.

Beispiele:

```
log2 0N = 0N                    log2 2N = 1N
log2 1N = 0N                    log2 4711N = 12N
```

Tipp: Der Potenzoperator wird in F# „`**`“ geschrieben, also 2^x entspricht `2N ** x`.

```
let log2 (n : Nat) : Nat =
    binarySearch ((fun k -> n < 2N ** (k + 1N)) , 0N ,n)
```

b) Schreiben Sie eine rekursive Funktion `sortedDigits: Nat -> Nat`, die für eine gegebene Zahl `n` prüft, ob die Ziffern von `n` (in Zehnerdarstellung) aufsteigend sortiert sind.

Beispiele:

```
sortedDigits 0N = true      sortedDigits 159N = true      sortedDigits 1101N = false
sortedDigits 5N = true      sortedDigits 1111N = true     sortedDigits 543N = false
```

```
let rec sortedDigits (n: Nat) : Bool =
    if n = 0N then true
    else
        let firstDigitsSorted = sortedDigits (n / 10N)
        let lastDigit = n % 10N
        let secondLastDigit = (n / 10N) % 10N
        lastDigit >= secondLastDigit && firstDigitsSorted
```

Aufgabe 4 Rekursion mit natürlichen Zahlen (Einreichaufgabe, 6 Punkte)

Motivation: Als Teil einer vorlesungsbegleitenden Studie wollen wir untersuchen, welches mentale Modell von Rekursion die Studierenden haben. Bitte bearbeiten Sie diese Aufgabe daher gewissenhaft und ohne fremde Hilfe.

Betrachten Sie die nachfolgenden rekursiven Funktionen. Was ist das Ergebnis des jeweiligen Funktionsaufrufs? Zeigen Sie Ihre Vorgehensweise, indem Sie alle gemachten Schritte aufschreiben und gegebenenfalls erklären. **Das Endergebnis allein gibt keine Punkte! Erstellen Sie *keinen Beweisbaum*!**

Bearbeiten Sie diese Aufgabe auf Papier oder schreiben Sie Ihre Schritte digital in eine pdf-Datei.

a) Funktionsaufruf: `f 5`

```
let rec f (n: Nat): Nat =
  if n = 0 then 1
  else 2 * (f (n - 1)) + 1
```

```
f 5
= 2 * (f 4) + 1
= 2 * (2 * (f 3) + 1) + 1
= 2 * (2 * (2 * (f 2) + 1) + 1) + 1
= 2 * (2 * (2 * (2 * (f 1) + 1) + 1) + 1) + 1
= 2 * (2 * (2 * (2 * (2 * (f 0) + 1) + 1) + 1) + 1) + 1
= 2 * (2 * (2 * (2 * (2 * 1 + 1) + 1) + 1) + 1) + 1
= 2 * (2 * (2 * (2 * 3 + 1) + 1) + 1) + 1
= 2 * (2 * (2 * 7 + 1) + 1) + 1
= 2 * (2 * 15 + 1) + 1
= 2 * 31 + 1
= 63
```

Eine anderer sinnvoller Lösungsweg könnte der folgende tabellarische Ansatz sein:

Schritt	n	Ausdruck	Erklärung
1	5	2 * f(4) + 1	f(5) wird zu 2 * f(4) + 1 aufgelöst
2	4	2 * (2 * f(3) + 1) + 1	f(4) wird zu 2 * f(3) + 1 aufgelöst
3	3	2 * (2 * (2 * f(2) + 1) + 1) + 1	f(3) wird zu 2 * f(2) + 1 aufgelöst
4	2	2 * (2 * (2 * (2 * f(1) + 1) + 1) + 1) + 1	f(2) wird zu 2 * f(1) + 1 aufgelöst
5	1	2 * (2 * (2 * (2 * (2 * f(0) + 1) + 1) + 1) + 1) + 1	f(1) wird zu 2 * f(0) + 1 aufgelöst
6	0	2 * (2 * (2 * (2 * (2 * 1 + 1) + 1) + 1) + 1) + 1	f(0) wird zu 1 aufgelöst (Basisfall)
7		2 * (2 * (2 * (2 * (2 + 1) + 1) + 1) + 1) + 1	Vereinfachung
8		2 * (2 * (2 * (2 * 3 + 1) + 1) + 1) + 1	Vereinfachung
9		2 * (2 * (2 * (6 + 1) + 1) + 1) + 1	Vereinfachung
10		2 * (2 * (2 * 7 + 1) + 1) + 1	Vereinfachung
11		2 * (2 * (14 + 1) + 1) + 1	Vereinfachung
12		2 * (2 * 15 + 1) + 1	Vereinfachung
13		2 * (30 + 1) + 1	Vereinfachung
14		2 * 31 + 1	Vereinfachung
15		62 + 1	Vereinfachung
16		63	Endergebnis

b) Funktionsaufruf: `g 8`

```
let rec g (n: Nat): Nat =
  if n <= 1 then 1
  else 4 * (g (n / 2)) + (n % 2)
```

```
g 8
= 4 * (g 4) + 0
= 4 * (4 * (g 2) + 0) + 0
= 4 * (4 * (4 * (g 1) + 0) + 0) + 0
= 4 * (4 * (4 * 1 + 0) + 0) + 0
= 4 * (4 * 4 + 0) + 0
= 4 * 16 + 0
= 64
```

Eine anderer sinnvoller Lösungsweg könnte der folgende tabellarische Ansatz sein:

Schritt	**n**	**Ausdruck**	**Erklärung**
1	8	4 * g(4) + (8 % 2)	g(8) wird zu 4 * g(4) + 0 aufgelöst
2	4	4 * (4 * g(2) + (4 % 2)) + 0	g(4) wird zu 4 * g(2) + 0 aufgelöst
3	2	4 * (4 * (4 * g(1) + (2 % 2)) + 0) + 0	g(2) wird zu 4 * g(1) + 0 aufgelöst
4	1	4 * (4 * (4 * 1 + 0) + 0) + 0	g(1) wird zu 1 aufgelöst (Basisfall)
5		4 * (4 * (4 + 0) + 0) + 0	Vereinfachung
6		4 * (4 * 4 + 0) + 0	Vereinfachung
7		4 * (16 + 0) + 0	Vereinfachung
8		4 * 16 + 0	Vereinfachung
9		64 + 0	Vereinfachung
10		64	Endergebnis

Aufgabe 5 Dynamische Semantik (Trainingsaufgabe)

Motivation: Sie sollen anhand einiger Beispiele die Regeln der dynamischen Semantik für Funktionen üben. Sie können sich für diese Aufgabe an den Vorlesungsfolien 187 bis 197 sowie am Skript Kapitel 3.6 orientieren.

Werten Sie die folgenden Mini-F#-Ausdrücke mit den Regeln der **dynamischen Semantik** aus und geben Sie einen entsprechenden Beweisbaum an.

a) Werten Sie den Ausdruck `(let x = 2 in (let x = 3 in x + x) + x)` bezüglich der folgenden Umgebung aus:

$\delta := \{x \mapsto 1\}$

$$\dfrac{\dfrac{\dfrac{}{\delta \vdash 2 \Downarrow 2}}{\delta \vdash \texttt{let x = 2} \Downarrow \{x \mapsto 2\}} \quad \dfrac{\dfrac{\dfrac{}{\{x \mapsto 2\} \vdash 3 \Downarrow 3}}{\{x \mapsto 2\} \vdash \texttt{let x = 3} \Downarrow \{x \mapsto 3\}} \quad \dfrac{\dfrac{}{\{x \mapsto 3\} \vdash x \Downarrow 3} \quad \dfrac{}{\{x \mapsto 3\} \vdash x \Downarrow 3}}{\{x \mapsto 3\} \vdash x + x \Downarrow 6}}{\{x \mapsto 2\} \vdash \texttt{let x = 3 in x + x} \Downarrow 6} \quad \dfrac{}{\{x \mapsto 2\} \vdash x \Downarrow 2}}{\{x \mapsto 2\} \vdash \texttt{(let x = 3 in x + x) + x} \Downarrow 8} \quad \dfrac{}{\delta \vdash x \Downarrow 1}}{\delta \vdash \texttt{let x = 2 in ((let x = 3 in x + x) + x)} \Downarrow 8}$$

$$\delta \vdash \texttt{(let x = 2 in (let x = 3 in x + x) + x)} \Downarrow 9$$

b) Werten Sie den Ausdruck `(fun y -> y * x) (let x = 2 in x)` bezüglich der folgenden Umgebung aus:

$\delta := \{x \mapsto 3\}$

$$\dfrac{\dfrac{}{\delta \vdash \texttt{fun y -> y * x} \Downarrow \langle \delta, y, \texttt{y * x} \rangle} \quad \dfrac{\dfrac{\dfrac{}{\delta \vdash 2 \Downarrow 2}}{\delta \vdash \texttt{let x = 2} \Downarrow \{x \mapsto 2\}} \quad \dfrac{}{\{x \mapsto 2\} \vdash x \Downarrow 2}}{\delta \vdash \texttt{let x = 2 in x} \Downarrow 2} \quad \dfrac{\dfrac{}{\{x \mapsto 3, y \mapsto 2\} \vdash y \Downarrow 2} \quad \dfrac{}{\{x \mapsto 3, y \mapsto 2\} \vdash x \Downarrow 3}}{\{x \mapsto 3, y \mapsto 2\} \vdash \texttt{y * x} \Downarrow 6}}{\delta \vdash \texttt{(fun y -> y * x) (let x = 2 in x)} \Downarrow 6}$$

c) Werten Sie den Ausdruck `g (fun (n: Nat) -> n*7)` bezüglich der folgenden Umgebung aus:

$\delta := \{\mathtt{g} \mapsto \langle \emptyset, \mathtt{f}, \mathtt{(f\ 2N)} \rangle\}$

Definiere aus Platzgründen $\delta_2 := \{\mathtt{f} \mapsto \langle \delta, \mathtt{n}, \mathtt{n\ *\ 7} \rangle\}$

$$\dfrac{\dfrac{}{\delta \vdash \mathtt{g} \Downarrow \langle \emptyset, \mathtt{f}, \mathtt{f\ 2} \rangle} \quad \dfrac{}{\delta \vdash \mathtt{fun\ n\ ->\ n*7} \Downarrow \langle \delta, \mathtt{n}, \mathtt{n*7} \rangle} \quad \dfrac{\dfrac{}{\delta_2 \vdash \mathtt{f} \Downarrow \langle \delta, \mathtt{n}, \mathtt{n*7} \rangle} \quad \dfrac{}{\delta_2 \vdash \mathtt{2} \Downarrow 2} \quad \dfrac{\dfrac{}{\{\mathtt{g} \mapsto \langle \emptyset, \mathtt{f}, \mathtt{f\ 2} \rangle, \mathtt{n} \mapsto 2\} \vdash \mathtt{n} \Downarrow 2} \quad \dfrac{}{\{\mathtt{g} \mapsto \langle \emptyset, \mathtt{f}, \mathtt{f\ 2} \rangle, \mathtt{n} \mapsto 2\} \vdash 7 \Downarrow 7}}{\{\mathtt{g} \mapsto \langle \emptyset, \mathtt{f}, \mathtt{f\ 2} \rangle, \mathtt{n} \mapsto 2\} \vdash \mathtt{n*7} \Downarrow 14}}{\delta_2 \vdash \mathtt{f\ 2} \Downarrow 14}}{\delta \vdash \mathtt{g\ (fun\ n\ ->\ n*7)} \Downarrow 14}$$

Aufgabe 6 Regelsammlung Statische und Dynamische Semantik

In der Vorlesung erweitern wir unsere Programmiersprache Mini-F# Schritt für Schritt um neue Konzepte. Dabei werden immer neue Regeln zur statischen und dynamischen Semantik eingeführt. Um Aufgaben wie "Geben Sie einen Beweisbaum an" lösen zu können, müssen Sie die Regeln zur Hand haben. Derartige Aufgaben waren bislang in jeder GdP-Klausur enthalten. Es ist also wahrscheinlich, dass auch Ihre Klausur mindestens eine solche Aufgabe enthalten wird. Wir erlauben Ihnen in der Klausur einen Spickzettel mitzunehmen (mehr dazu zu einem späteren Zeitpunkt), dieser sollte auf jeden Fall die Regeln enthalten.

Legen Sie sich daher schon jetzt eine Zusammenstellung aller bislang eingeführten Regeln an und erweitern Sie diese im Laufe des Semesters kontinuierlich um neu hinzugekommene Regeln. So haben Sie es leichter, Beweisbaum-Aufgaben für die Hausaufgaben zu lösen und haben zudem schon einen Teil der Klausurvorbereitung erledigt.