Prof. Dr. Ralf Hinze
Cass Alexandru, M.Sc.
Alexander Dinges, M.Sc.
Felix Winkler, M.Sc.

RPTU
Fachbereich Informatik
AG Programmiersprachen

Lösungshinweise/-vorschläge zum Übungsblatt 3: Konzepte der Programmierung (WS 2025/26)

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, 10 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 `exp7: Nat -> Nat`, die für eine gegebene natürliche Zahl `x` den Wert 7^x berechnet, jedoch ohne dabei die F# eigene Potenzfunktion zu verwenden. Die Verwendung der Multiplikationsfunktion ist hingegen gestattet.

Beispiele:

```
exp7 0N = 1N                    exp7 1N = 7N                    exp7 2N = 49N
```

```
let rec exp7 (x: Nat): Nat =
    if x = 0N then 1N
    else exp7 (x - 1N) * 7N
```

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
```

c) Implementieren Sie die Funktionen `even: Nat -> Bool` und `odd: Nat -> Bool`, die berechnen, ob eine Zahl gerade bzw. ungerade ist (rufen Sie die Funktionen nicht gegenseitig auf). Wie würde eine alternative Implementierung mit Hilfe des Operators `%` aussehen?

```
let rec even (n: Nat): Bool =
    if n = 0N then true
    else not (even (n - 1N))

let rec odd (n: Nat): Bool =
    if n = 0N then false
    else not (odd (n - 1N))

// mit %
let even2 (n: Nat): Bool = (n % 2N) = 0N
let odd2 (n: Nat): Bool = (n % 2N) = 1N
```

Aufgabe 3 Leibniz Entwurfsmuster (Einreichaufgabe, 10 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) Schreiben Sie eine Funktion `iterate: (Nat -> Nat) -> Nat -> (Nat -> Nat)`, die eine Funktion `f` und eine natürliche Zahl `n` als Parameter entgegennimmt und eine neue Funktion zurückgibt, die `f` `n`-mal auf ihren Parameter anwendet.

Beispiele:

```
iterate (fun x -> x + 1N) 3N 0N = 3N          iterate (fun x -> x * 2N) 4N 1N = 16N
```

```
let rec iterate (f: Nat -> Nat) (n: Nat): Nat -> Nat =
    if n = 0N then fun x -> x
    else
        let g = iterate f (n / 2N)
        if n % 2N = 0N
            then fun x ->    g (g x)
            else fun x -> f (g (g x))
```

b) Schreiben Sie eine Funktion `isExp2: Nat -> Bool`, die für eine gegebene natürliche Zahl n überprüft, ob es eine natürliche Zahl m gibt, sodass $2^m = n$ ist. Es soll also überprüft werden, ob n eine Zweierpotenz ist.

Hinweis: Möglicherweise brauchen Sie zwei Basisfälle.

Beispiele:

```
isExp2 0N = false          isExp2 2N = true          isExp2 4N = true
isExp2 1N = true           isExp2 3N = false         isExp2 5N = false
```

```
let rec isExp2 (n: Nat): Bool =
    if n = 0N then false
    else if n = 1N then true
    else n % 2N = 0N && isExp2 (n / 2N)
```

c) Schreiben Sie eine Funktion `mirror: Nat -> Nat`, die eine Zahl spiegelt.

Orientieren Sie sich am Leibniz Entwurfsmuster.

Hinweis: Definieren Sie eine rekursive Hilfsfunktion mit zwei Parametern. Einer der Parameter kann als Akkumulator für das Ergebnis verwendet werden.

Beispiele:

```
mirror 5N = 5N      mirror 123N = 321N      mirror 4711N = 1174N      mirror 13531N = 13531N
```

```
let mirror (n: Nat): Nat =
    let rec help (n: Nat) (acc: Nat): Nat =
        if n = 0N then acc
        else help (n / 10N) (n % 10N + acc * 10N)
    help n 0N
```

Aufgabe 4 Statische und dynamische Semantik (Einreichaufgabe, 8 Punkte)

Motivation: Sie sollen anhand eines Beispiels die Regeln der statischen und dynamischen Semantik für Funktionen üben. Sie können sich für diese Aufgabe an den Vorlesungsfolien 165 bis 186 sowie am Skript Kapitel 3.4 und 3.5 orientieren.

Geben Sie für die folgenden Mini-F#-Ausdrücke den Typ bzw. das Auswertungsergebnis an, sowie den entsprechenden Beweisbaum dazu.

a) Bestimmen Sie mithilfe der Regeln der **statischen Semantik** den Typ für den Ausdruck `(f 1) 2` bezüglich der folgenden Signatur:

$\Sigma := \{\texttt{f} \mapsto \text{Nat} \to \text{Nat} \to \text{Nat}\}$

$$\dfrac{\dfrac{\dfrac{}{\Sigma \vdash \texttt{f} : \text{Nat} \to \text{Nat} \to \text{Nat}} \quad \dfrac{}{\Sigma \vdash 1 : \text{Nat}}}{\Sigma \vdash \texttt{f 1} : \text{Nat} \to \text{Nat}} \quad \dfrac{}{\Sigma \vdash 2 : \text{Nat}}}{\Sigma \vdash \texttt{(f 1) 2} : \text{Nat}}$$

b) Werten Sie mithilfe der Regeln der **dynamischen Semantik** den Ausdruck `(f 1) 2` bezüglich der folgenden Umgebung aus:

$\delta := \{\texttt{f} \mapsto \langle \emptyset, \texttt{x}, \texttt{fun y -> x + y} \rangle\}$

Definiere aus Platzgründen $\delta' := \{\texttt{x} \mapsto 1, \texttt{y} \mapsto 2\}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\delta \vdash \texttt{f} \Downarrow \langle \emptyset, \texttt{x}, \texttt{fun y -> x + y} \rangle} \quad \dfrac{}{\delta \vdash 1 \Downarrow 1} \quad \dfrac{}{\{\texttt{x} \mapsto 1\} \vdash \texttt{fun y -> x + y} \Downarrow \langle \{\texttt{x} \mapsto 1\}, \texttt{y}, \texttt{x + y} \rangle}}{\delta \vdash \texttt{f 1} \Downarrow \langle \{\texttt{x} \mapsto 1\}, \texttt{y}, \texttt{x + y} \rangle} \quad \dfrac{}{\delta \vdash 2 \Downarrow 2} \quad \dfrac{\dfrac{}{\delta' \vdash \texttt{x} \Downarrow 1} \quad \dfrac{}{\delta' \vdash \texttt{y} \Downarrow 2}}{\delta' \vdash \texttt{x + y} \Downarrow 3}}{\delta \vdash \texttt{(f 1) 2} \Downarrow 3}$$

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:

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

$$\dfrac{\dfrac{\dfrac{}{\delta \vdash 2 \Downarrow 2}}{\delta \vdash \texttt{let x = 2} \Downarrow \{x \mapsto 2\}} \quad \dfrac{\dfrac{\dfrac{}{\{x \mapsto 2\} \vdash 3 \Downarrow 3}}{\{x \mapsto 2\} \vdash \texttt{let x = 3} \Downarrow \{x \mapsto 3\}} \quad \dfrac{\dfrac{}{\{x \mapsto 3\} \vdash x \Downarrow 3} \quad \dfrac{}{\{x \mapsto 3\} \vdash x \Downarrow 3}}{\{x \mapsto 3\} \vdash x + x \Downarrow 6}}{\{x \mapsto 2\} \vdash \texttt{let x = 3 in x + x} \Downarrow 6} \quad \dfrac{}{\{x \mapsto 2\} \vdash x \Downarrow 2}}{\{x \mapsto 2\} \vdash \texttt{(let x = 3 in x + x) + x} \Downarrow 8}}{\delta \vdash \texttt{let x = 2 in ((let x = 3 in x + x) + x)} \Downarrow 8} \quad \dfrac{}{\delta \vdash x \Downarrow 1}}{\delta \vdash \texttt{(let x = 2 in (let x = 3 in x + x) + x) + x} \Downarrow 9}$$

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

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

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

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.