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 2: Konzepte der Programmierung (WS 2026/27)

Hinweis zu den Aufgabentypen:

- **Präsenzaufgaben** werden während der Übungsstunden gemeinsam in den Übungsgruppen zur Vorbereitung der Einreichaufgaben bearbeitet und besprochen. Es ist keine Abgabe erforderlich.
- **Einreichaufgaben** mit Punkten sind Pflichtaufgaben, die abgegeben werden müssen und für die Klausurzulassung relevant sind.
- **Trainingsaufgaben** werden nicht abgegeben, jedoch ist der dort behandelte Stoff durchaus klausurrelevant. Nutzen Sie diese Aufgaben zur Vertiefung des gelernten Stoffes und zur Klausurvorbereitung.

Sie können in den Sprech- und Übungsstunden zu allen Aufgabentypen Fragen stellen. Sie erhalten zu allen Aufgaben Lösungshinweise, jeweils nach der Abgabefrist für die letzte Übungsgruppe (Mittwoch Mittag).

Installationsanleitung Eine Installationsanleitung und Anweisungen zur Einrichtung der Entwicklungsumgebung finden Sie auf unserer Homepage.

Aufgabe 1 Statische und Dynamische Semantik (Präsenzaufgabe)

Motivation: Sie sollen anhand einiger Beispiele die bisher bekannten Regeln der statischen und dynamischen Semantik üben. Ein gutes Verständnis dieser Regeln und deren Anwendung in Beweisbäumen wird Ihnen im Laufe der Vorlesung an vielen Stellen helfen, sich Mini-F# systematisch zu erschließen. Sie können sich für diese Aufgabe an den Vorlesungsfolien 100 bis 125 sowie am Skript Kapitel 3.1 und 3.2 orientieren.

Prüfen Sie, ob die folgenden Mini-F#-Ausdrücke gültige Ausdrücke bezüglich der statischen Semantik sind. Geben Sie für die gültigen Ausdrücke deren Typ und einen entsprechenden Beweisbaum mit den Regeln der statischen Semantik an. Werten Sie dann gültige Ausdrücke mit den Regeln der dynamischen Semantik aus und geben Sie einen entsprechenden Beweisbaum an. Erklären Sie für die ungültigen Ausdrücke, wo der Fehler liegt.

Hier noch mal eine Übersicht der benötigten Regeln:

Syntax	**Statische Semantik**	**Dynamische Semantik**
n	$\dfrac{}{n : Nat}$	$\dfrac{}{n \Downarrow n}$
*	$\dfrac{e_1 : Nat \qquad e_2 : Nat}{e_1 * e_2 : Nat}$	$\dfrac{e_1 \Downarrow n_1 \qquad e_2 \Downarrow n_2}{e_1 * e_2 \Downarrow n_1 \cdot n_2}$
+	$\dfrac{e_1 : Nat \qquad e_2 : Nat}{e_1 + e_2 : Nat}$	$\dfrac{e_1 \Downarrow n_1 \qquad e_2 \Downarrow n_2}{e_1 + e_2 \Downarrow n_1 + n_2}$
false	$\dfrac{}{false : Bool}$	$\dfrac{}{false \Downarrow false}$
true	$\dfrac{}{true : Bool}$	$\dfrac{}{true \Downarrow true}$
if then else	$\dfrac{e_1 : Bool \qquad e_2 : t \qquad e_3 : t}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 : t}$	$\dfrac{e_1 \Downarrow true \qquad e_2 \Downarrow v}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \Downarrow v}$ $\dfrac{e_1 \Downarrow false \qquad e_3 \Downarrow v}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \Downarrow v}$
<	$\dfrac{e_1 : Nat \qquad e_2 : Nat}{e_1 < e_2 : Bool}$ Für andere Operationen genauso (z.B. <>)	$\dfrac{e_1 \Downarrow n + k \qquad e_2 \Downarrow n}{e_1 < e_2 \Downarrow false}$ $\dfrac{e_1 \Downarrow n \qquad e_2 \Downarrow n + 1 + k}{e_1 < e_2 \Downarrow true}$

a) `(4 * 5) > (17 + 3)`

Statische Semantik

$$\dfrac{\dfrac{\dfrac{}{\texttt{4} : \mathit{Nat}} \quad \dfrac{}{\texttt{5} : \mathit{Nat}}}{\texttt{4 * 5} : \mathit{Nat}} \quad \dfrac{\dfrac{}{\texttt{17} : \mathit{Nat}} \quad \dfrac{}{\texttt{3} : \mathit{Nat}}}{\texttt{17 + 3} : \mathit{Nat}}}{\texttt{(4 * 5) > (17 + 3)} : \mathit{Bool}}$$

Dynamische Semantik

$$\dfrac{\dfrac{\dfrac{}{\texttt{4} \Downarrow 4} \quad \dfrac{}{\texttt{5} \Downarrow 5}}{\texttt{4 * 5} \Downarrow 20} \quad \dfrac{\dfrac{}{\texttt{17} \Downarrow 17} \quad \dfrac{}{\texttt{3} \Downarrow 3}}{\texttt{17 + 3} \Downarrow 20}}{\texttt{(4 * 5) > (17 + 3)} \Downarrow \mathit{false}}$$

b) `(5 > 4) > 3`

Ungültiger Ausdruck, da `5 > 4 : Bool` und `3 : Nat`.

c) **`if`** `(2 + 6) < 10` **`then`** `42` **`else`** `9`

Statische Semantik

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\texttt{2} : \mathit{Nat}} \quad \dfrac{}{\texttt{6} : \mathit{Nat}}}{\texttt{2 + 6} : \mathit{Nat}} \quad \dfrac{}{\texttt{10} : \mathit{Nat}}}{\texttt{(2 + 6) < 10} : \mathit{Bool}} \quad \dfrac{}{\texttt{42} : \mathit{Nat}} \quad \dfrac{}{\texttt{9} : \mathit{Nat}}}{\texttt{if (2 + 6) < 10 then 42 else 9} : \mathit{Nat}}$$

Dynamische Semantik

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\texttt{2} \Downarrow 2} \quad \dfrac{}{\texttt{6} \Downarrow 6}}{\texttt{2 + 6} \Downarrow 8} \quad \dfrac{}{\texttt{10} \Downarrow 10}}{\texttt{(2 + 6) < 10} \Downarrow \mathit{true}} \quad \dfrac{}{\texttt{42} \Downarrow 42}}{\texttt{if (2 + 6) < 10 then 42 else 9} \Downarrow 42}$$

d) **`if`** `42` **`then false else true`**

Ungültiger Ausdruck, da Bedingung nicht vom Typ *Bool*.

Aufgabe 2 Typquiz (Einreichaufgabe, 6 Punkte)

Motivation: In dieser Aufgabe sollen Sie sich den Zusammenhang zwischen der Funktionsdefinition und der daraus resultierenden Signatur klarmachen. Es handelt sich um eine Transferaufgabe, Sie finden in den Vorlesungsfolien und im Skript keine direkten Beispiele. Schauen Sie sich ggf. noch einmal die Vorlesungsfolien 165 bis 186 oder im Skript die Kapitel 3.4 und 3.5 an.

In Aufgabe 1 haben wir für gegebene Ausdrücke deren Typ ermittelt und dafür die Regeln der statischen Semantik benutzt. Das war eine rein maschinelle Aufgabe: Die linke Seite (der Ausdruck) war jeweils gegeben, die rechte Seite (der Typ) errechnet sich ohne jegliche Kreativität aus den Regeln.

Nun wollen wir die Regeln in umgekehrter Richtung benutzen. Gegeben ist also ein bestimmter Typ und Sie sollen einen Ausdruck finden, der gemäß der Regeln der statischen Semantik den vorgegebenen Typ hat. Hierbei ist die Lösung keineswegs eindeutig: Meist gibt es verschiedene Ausdrücke, die den selben vorgegebenen Typ haben. Sie müssen also selbst kreativ werden und nicht nur stur die Regelschemata befolgen.

Geben Sie für die folgenden Typen jeweils einen gültigen Mini-F# Ausdruck an, der diesen Typ hat. Bei Funktionstypen müssen die Parameter im Rumpf der Funktion verwendet werden. Das heißt, für den Typ `Nat -> Bool` darf die konstante Funktion **`fun`** `(x : Nat) ->` **`true`** *nicht* als Lösung angegeben werden, da der Parameter `x` nicht verwendet wird.

a) `Bool`

> **`true`** oder **`false`**

b) `Bool -> Nat`

> **`fun`** `(x : Bool) ->` **`if`** `x` **`then`** `42N` **`else`** `7N`

c) `Nat -> (Bool -> Nat)`

> **`fun`** `(x : Nat) ->` **`fun`** `(a : Bool) ->` **`if`** `a` **`then`** `x` **`else`** `0N`
>
> F# erlaubt es uns, diesen Funktionsausdruck abzukürzen zu:
> **`fun`** `(x: Nat) (a: Bool) ->` **`if`** `a` **`then`** `x` **`else`** `0N`
> Um diesen Funktionsausdruck an den Bezeichner `f` zu binden:
> **`let`** `f (x: Nat) (a: Bool) =` **`if`** `a` **`then`** `x` **`else`** `0N`
>
> Der erste Funktionsausdruck macht deutlich, dass es sich um eine Funktion handelt, die eine Zahl `x` vom Typ `Nat` erwartet und dann eine Funktion vom Typ `Bool -> Nat` zurückgibt. Diese zurückgegebene Funktion können wir dann wiederum auf einen Booleschen Wert anwenden und erhalten ein Ergebnis vom Typ `Nat`. Beispiel: Der gesamte Funktionsausdruck wird auf das Argument `1N` angewendet und das Ergebnis davon auf das Argument **`true`**. Dies lässt sich durch folgenden Ausdruck beschreiben:
>
> `((`**`fun`** `(x: Nat) ->` **`fun`** `(a: Bool) ->` **`if`** `a` **`then`** `x` **`else`** `0N) 1N)` **`true`**
>
> Durch die Verwendung des Bezeichners `f` für den Funktionsausdruck ergibt sich: `(f 1N)` **`true`**.
> Funktionsapplikation ist linksassoziativ, das heißt die Klammern können weggelassen werden und der Ausdruck ist identisch zu `f 1N` **`true`**. Wir können das also einerseits so lesen, als würden wir die Funktion `f` mit zwei Argumenten `1N` und **`true`** aufrufen, oder andererseits als würden wir sie mit nur einem Argument (`1N`) aufrufen und die zurückgegebene Funktion wiederum mit einem Argument (**`true`**). Das Fachwort hierfür ist *Currying*, siehe auch https://de.wikipedia.org/wiki/Currying oder den Abschnitt über *Staffelung* im Skript auf Seite 59.

d) `(Nat -> Bool) -> Nat`

> **`fun`** `(f : Nat -> Bool) ->` **`if`** `f 4711N` **`then`** `3N` **`else`** `7N`

Aufgabe 3 Dynamische Semantik (Einreichaufgabe, 6 Punkte)

Motivation: Dies ist die Fortsetzung von Aufgabe 1. Hier betrachten wir ausschließlich die Dynamische Semantik, dieses Mal allerdings mit Wertedefinitionen. Sie können sich für diese Aufgabe an den Vorlesungsfolien 141 bis 146 sowie am Skript Kapitel 3.3 orientieren.

Werten Sie die Ausdrücke mit den Regeln der dynamischen Semantik aus und geben Sie einen entsprechenden Beweisbaum an.

Hinweis: Wir verwenden hier Wertedefinitionen. Sie müssen also, wie ab Vorlesungsfolie 142 gezeigt, entsprechend Umgebungen angeben.

a) `let x = false in (if x then 0 else 1)`

$$\dfrac{\dfrac{\dfrac{}{\emptyset \vdash \texttt{false} \Downarrow \mathit{false}}}{\emptyset \vdash \texttt{let x = false} \Downarrow \{\texttt{x} \mapsto \mathit{false}\}} \quad \dfrac{\dfrac{}{\{\texttt{x} \mapsto \mathit{false}\} \vdash \texttt{x} \Downarrow \mathit{false}} \quad \dfrac{}{\{\texttt{x} \mapsto \mathit{false}\} \vdash 1 \Downarrow 1}}{\{\texttt{x} \mapsto \mathit{false}\} \vdash \texttt{if x then 0 else 1} \Downarrow 1}}{\emptyset \vdash \texttt{let x = false in (if x then 0 else 1)} \Downarrow 1}$$

b) `let x = 1 in let x = 2 in x`

$$\dfrac{\dfrac{\dfrac{}{\emptyset \vdash 1 \Downarrow 1}}{\emptyset \vdash \texttt{let x = 1} \Downarrow \{\texttt{x} \mapsto 1\}} \quad \dfrac{\dfrac{\dfrac{}{\{\texttt{x} \mapsto 1\} \vdash 2 \Downarrow 2}}{\{\texttt{x} \mapsto 1\} \vdash \texttt{let x = 2} \Downarrow \{\texttt{x} \mapsto 2\}} \quad \dfrac{}{\{\texttt{x} \mapsto 2\} \vdash \texttt{x} \Downarrow 2}}{\{\texttt{x} \mapsto 1\} \vdash \texttt{let x = 2 in x} \Downarrow 2}}{\emptyset \vdash \texttt{let x = 1 in let x = 2 in x} \Downarrow 2}$$

Aufgabe 4 Programmieren mit Zahlen (Einreichaufgabe, 6 Punkte)

Motivation: Dies ist unsere erste Programmieraufgabe. Sie sollen sich mit dem Setup vertraut machen und erste kleine Funktionen implementieren.

Hinweis: Arbeiten Sie die auf der Homepage verlinkte Installationsanleitung ab. Sie enthält wertvolle Hinweise zum Bearbeiten von Programmieraufgaben. Laden Sie sich die Vorlage von der Website herunter, öffnen Sie den Ordner wie beschrieben in VS Code und schreiben Sie Ihre Lösungen in die Datei `Zahlen.fs`. **Laden Sie abschließend ausschließlich diese Datei im ExClaim-System hoch.**

a) Definieren Sie eine Funktion `dist`, welche den Abstand zweier natürlicher Zahlen berechnet.

Beispiele:

```
dist 2N 2N = 0N          dist 5N 3N = 2N          dist 3N 5N = 2N          dist 11N 6N = 5N
```

```
let dist (a: Nat) (b: Nat): Nat =
    max a b - min a b
```

b) Definieren Sie eine Funktion `avg3`, welche den ganzzahligen Mittelwert dreier natürlicher Zahlen berechnet.

Beispiele:

```
avg3 1N 2N 3N = 2N       avg3 4N 3N 7N = 4N       avg3 6N 3N 4N = 4N       avg3 9N 5N 3N = 5N
```

```
let avg3 (a: Nat) (b: Nat) (c: Nat): Nat =
    (a + b + c) / 3N
```

c) Definieren Sie eine Funktion `applyToSucc : (Nat -> Nat) -> (Nat -> Nat)`, die eine Funktion `f` als Argument nimmt und eine Funktion zurückgibt, die das Argument um eins erhöht, bevor sie `f` darauf anwendet.

Beispiele:

```
(applyToSucc (fun x -> x)) 5N  = 6N          (applyToSucc (fun x -> x + 2N)) 7N = 10N
(applyToSucc (fun x -> x * x)) 3N = 16N      (applyToSucc (fun x -> x - 1N)) 0N = 0N
```

```
let applyToSucc (f: Nat -> Nat): (Nat -> Nat) =
    fun n -> f (n + 1N)
```