

16. Knobelaufgabe #6

Harry Hacker hat mit dem folgenden Ausdruck den „International Obfuscated F# Code Contest“ gewonnen.

```
let a = fun b c → b (b c) in a a a a (fun d → d * d * d) 9
```

Zu welcher natürlichen Zahl wertet der Ausdruck aus?

III Werte

Ralf Hinze

Boolesche Werte

Natürliche Zahlen

Werte/-definitionen

Funktions/-definitionen

Funktionsausdrücke

Rekursion

Motivation

Abstrakte Syntax

Statische Semantik

Dynamische Semantik

Entwurfsmuster

186

16. Motivation

- ▶ Wieviele Möglichkeiten gibt es, n verschiedene Objekte in einer Reihe zu arrangieren?
 - ▶ Für die erste Position gibt es n Möglichkeiten,
 - ▶ für die zweite Position gibt es $n - 1$ Möglichkeiten,
 - ▶ ...
 - ▶ für die vorletzte Position gibt es 2 Möglichkeiten,
 - ▶ für die letzte Position gibt es 1 Möglichkeit.
- ▶ Insgesamt gibt es $1 * 2 * \dots * (n - 1) * n$ Möglichkeiten.
- ▶ Diese Zahl heißt auch n Fakultät, notiert $n!$.

III Werte

Ralf Hinze

Boolesche Werte

Natürliche Zahlen

Werte/-definitionen

Funktions/-definitionen

Funktionsausdrücke

Rekursion

Motivation

Abstrakte Syntax

Statische Semantik

Dynamische Semantik

Entwurfsmuster

187

16. Motivation

Wie können wir die Fakultätsfunktion in Mini-F# programmieren?

Ein erster Versuch:

```
let factorial (n : Nat) : Nat =  
  if n = 0 then 1  
  else if n = 1 then 1  
  else if n = 2 then 1 * 2  
  else if n = 3 then 1 * 2 * 3  
  else ...
```

☞ Der Wert von 0 Fakultät ist 1, da das leere Produkt von Zahlen vereinbarungsgemäß 1 ist.

III Werte

Ralf Hinze

Boolesche Werte

Natürliche Zahlen

Werte/-definitionen

Funktions/-definitionen

Funktionsausdrücke

Rekursion

Motivation

Abstrakte Syntax

Statische Semantik

Dynamische Semantik

Entwurfsmuster

188

16. Motivation

Beobachtung: in jedem der Fälle $n > 0$ ist der letzte Faktor die Zahl n selbst.

```
let factorial (n : Nat) : Nat =  
  if n = 0 then 1  
  else if n = 1 then n  
  else if n = 2 then 1 * n  
  else if n = 3 then 1 * 2 * n  
  else ...
```

III Werte

Ralf Hinze

Boolesche Werte

Natürliche Zahlen

Werte/-definitionen

Funktions/-definitionen

Funktionsausdrücke

Rekursion

Motivation

Abstrakte Syntax

Statische Semantik

Dynamische Semantik

Entwurfsmuster

189

16. Motivation

☞ Der Faktor n kann aus den Zweigen der Alternativen „herausgezogen“ werden.

```
let factorial (n : Nat) : Nat =  
  if n = 0 then 1  
  else ( if n = 1 then 1  
        else if n = 2 then 1  
        else if n = 3 then 1 * 2  
        else ...) * n
```

III Werte

Ralf Hinze

Boolesche Werte

Natürliche
Zahlen

Werte/-
definitionen

Funktions/-
definitionen

Funktions-
ausdrücke

Rekursion

Motivation

Abstrakte Syntax

Statische Semantik

Dynamische
Semantik

Entwurfsmuster

190

16. Motivation

Der Ausdruck in Klammern sieht dem ursprünglichen Funktionsrumpf sehr ähnlich. Die Konstanten können in Übereinstimmung gebracht bringen, indem wir links und rechts jeweils 1 subtrahieren.

```
let factorial (n : Nat) : Nat =  
  if n = 0 then 1  
  else ( if n - 1 = 0 then 1  
        else if n - 1 = 1 then 1  
        else if n - 1 = 2 then 1 * 2  
        else ...) * n
```

☞ Der Ausdruck in Klammern ist gleich dem Aufruf $factorial (n - 1)$.

III Werte

Ralf Hinze

Boolesche Werte

Natürliche
Zahlen

Werte/-
definitionen

Funktions/-
definitionen

Funktions-
ausdrücke

Rekursion

Motivation

Abstrakte Syntax

Statische Semantik

Dynamische
Semantik

Entwurfsmuster

191

16. Motivation

Erlauben wir bei der Definition einer Funktion den Rückgriff auf die definierte Funktion selbst (!), so erhalten wir:

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

☞ Greift man bei der Definition auf das definierte Objekt selbst zurück, spricht man von einer *rekursiven* Definition.

☞ Rekursive Definitionen werden mit dem Schlüsselwort **rec** gekennzeichnet. (Warum?)

III Werte

Ralf Hinze

Boolesche Werte

Natürliche
Zahlen

Werte/-
definitionen

Funktions/-
definitionen

Funktions-
ausdrücke

Rekursion

Motivation

Abstrakte Syntax

Statische Semantik

Dynamische
Semantik

Entwurfsmuster

192

16. Abstrakte Syntax

```
d ::= ...  
  | let rec f (x1 : t1) : t2 = e
```

Deklarationen:
rekursive Funktionsdefinition

☞ **let rec** $f(x) = e$ definiert genau wie **let** $f(x) = e$ eine Funktion, nur dass in e zusätzlich der Bezeichner f sichtbar ist.

III Werte

Ralf Hinze

Boolesche Werte

Natürliche
Zahlen

Werte/-
definitionen

Funktions/-
definitionen

Funktions-
ausdrücke

Rekursion

Motivation

Abstrakte Syntax

Statische Semantik

Dynamische
Semantik

Entwurfsmuster

193

16. Statische Semantik

Typregeln:

$$\frac{\Sigma, \{f \mapsto t_1 \rightarrow t_2, x_1 \mapsto t_1\} \vdash e_2 : t_2}{\Sigma \vdash (\mathbf{let\ rec\ } f (x_1 : t_1) : t_2 = e_2) : \{f \mapsto t_1 \rightarrow t_2\}}$$

Zum Vergleich die Regel für nicht-rekursive Definitionen:

$$\frac{\Sigma, \{x_1 \mapsto t_1\} \vdash e_2 : t_2}{\Sigma \vdash (\mathbf{let\ } f (x_1 : t_1) : t_2 = e_2) : \{f \mapsto t_1 \rightarrow t_2\}}$$

III Werte

Ralf Hinze

Boolesche Werte

Natürliche Zahlen

Werte/-definitionen

Funktions/-definitionen

Funktionsausdrücke

Rekursion

Motivation

Abstrakte Syntax

Statische Semantik

Dynamische Semantik

Entwurfsmuster

194

16. Dynamische Semantik

Der Bereich der Werte wird um rekursive Funktionsabschlüsse erweitert.

$$\nu ::= \dots \quad \text{Werte:} \\ | \langle \delta, f, x, e \rangle \quad \text{rekursiver Funktionsabschluss}$$

Auswertungsregeln:

$$\frac{\delta \vdash (\mathbf{let\ rec\ } f\ x = e) \Downarrow \{f \mapsto \langle \delta, f, x, e \rangle\}}{\delta \vdash e \Downarrow \nu \quad \delta \vdash e_1 \Downarrow \nu_1 \quad \delta', \{f \mapsto \nu, x_1 \mapsto \nu_1\} \vdash e' \Downarrow \nu'} \\ \delta \vdash e\ e_1 \Downarrow \nu'$$

mit $\nu = \langle \delta', f, x_1, e' \rangle$

☞ f wird an den rekursiven Funktionsabschluss gebunden, in dem f selbst aufgeführt wird. (Aus der Rekursion wird ein zyklisches Geflecht.)

III Werte

Ralf Hinze

Boolesche Werte

Natürliche Zahlen

Werte/-definitionen

Funktions/-definitionen

Funktionsausdrücke

Rekursion

Motivation

Abstrakte Syntax

Statische Semantik

Dynamische Semantik

Entwurfsmuster

195

16. Demo

Die Fakultät wächst sehr schnell, wie die folgenden Aufrufe zeigen.

Mini) *factorial* 10

3.628.800

Mini) *factorial* 100

93.326.215.443.944.152.681.699.238.856.266.700.490.715.968.264.

381.621.468.592.963.895.217.599.993.229.915.608.941.463.976.156.

518.286.253.697.920.827.223.758.251.185.210.916.864.000.000.000.

000.000.000.000.000

☞ Die Zahl der Atome im sichtbaren Weltall wird auf ungefähr 10^{79} geschätzt; *factorial* 100 mit seinen 158 Stellen übersteigt diese Zahl um ein Vielfaches.

III Werte

Ralf Hinze

Boolesche Werte

Natürliche Zahlen

Werte/-definitionen

Funktions/-definitionen

Funktionsausdrücke

Rekursion

Motivation

Abstrakte Syntax

Statische Semantik

Dynamische Semantik

Entwurfsmuster

196

17. Vertiefung

- Der Schritt von den nicht-rekursiven zu den rekursiven Funktionen ist ein gewaltiger.

Mini-F# ist berechnungsuniversell.

- Mit Mini-F# können wir die prinzipiellen Möglichkeiten eines Rechners ausnutzen.
- Mehr dazu später in der Theoretischen Abteilung der Informatik.
- Wir wenden uns an dieser Stelle den vergnüglichen Dingen zu, der Programmierung. Im folgenden:
 - Potenzfunktion,
 - Quadratwurzel,
 - ein Ratespiel,
 - und mehr.

III Werte

Ralf Hinze

Boolesche Werte

Natürliche Zahlen

Werte/-definitionen

Funktions/-definitionen

Funktionsausdrücke

Rekursion

Entwurfsmuster

Peano

Entwurfsmuster

Leibniz

Entwurfsmuster

Ratespiel

197

17. Vertiefung

```
let rec factorial (n : Nat) : Nat =  
  if n = 0 then 1  
  else factorial (n ÷ 1) * n
```

Die Definition basiert auf der Tatsache, dass eine natürliche Zahl entweder 0 oder größer als 0 ist.

Problemlösungsbrille:

- ▶ **Rekursionsbasis:** die Lösung muss unmittelbar angegeben werden.
- ▶ **Rekursionsschritt:** rekursiv wird eine Lösung für $n \div 1$ bestimmt, dann wird die Teillösung zu einer Gesamtlösung für n erweitert.

III Werte

Ralf Hinze

Boolesche Werte

Natürliche Zahlen

Werte/-definitionen

Funktions/-definitionen

Funktionsausdrücke

Rekursion

Entwurfsmuster

Peano Entwurfsmuster

Leibniz Entwurfsmuster

Ratespiel

198

17. Potenzfunktion

Wenden wir das Problemlösungsschema auf die Potenzfunktion x^n an.

```
let rec power (x : Nat, n : Nat) : Nat =  
  if n = 0 then ...  
  else ... power (x, n ÷ 1) ...
```

☞ Wir rekurren über n , nicht über x . (Warum?)

III Werte

Ralf Hinze

Boolesche Werte

Natürliche Zahlen

Werte/-definitionen

Funktions/-definitionen

Funktionsausdrücke

Rekursion

Entwurfsmuster

Peano Entwurfsmuster

Leibniz Entwurfsmuster

Ratespiel

199

17. Potenzfunktion

- ▶ **Rekursionsbasis:** $x^0 = 1$.

```
let rec power (x : Nat, n : Nat) : Nat =  
  if n = 0 then 1  
  else ... power (x, n ÷ 1) ...
```

- ▶ **Rekursionsschritt:** $x^n = x^{n-1} * x$.

```
let rec power (x : Nat, n : Nat) : Nat =  
  if n = 0 then 1  
  else power (x, n ÷ 1) * x
```

III Werte

Ralf Hinze

Boolesche Werte

Natürliche Zahlen

Werte/-definitionen

Funktions/-definitionen

Funktionsausdrücke

Rekursion

Entwurfsmuster

Peano Entwurfsmuster

Leibniz Entwurfsmuster

Ratespiel

200

17. Multiplikation

Die Potenzfunktion wird auf wiederholte Multiplikation zurückgeführt. Ebenso lässt sich die Multiplikation auf wiederholte Addition zurückführen.

```
let rec mul (m : Nat, n : Nat) : Nat =  
  if m = 0 then ...  
  else ... mul (m ÷ 1, n) ...
```

III Werte

Ralf Hinze

Boolesche Werte

Natürliche Zahlen

Werte/-definitionen

Funktions/-definitionen

Funktionsausdrücke

Rekursion

Entwurfsmuster

Peano Entwurfsmuster

Leibniz Entwurfsmuster

Ratespiel

201

17. Multiplikation

- ▶ **Rekursionsbasis:** $0 * n = 0$.

```
let rec mul (m : Nat, n : Nat) : Nat =  
  if m = 0 then 0  
  else ... mul (m - 1, n) ...
```

- ▶ **Rekursionsschritt:** $m * n = (m - 1) * n + n$.

```
let rec mul (m : Nat, n : Nat) : Nat =  
  if m = 0 then 0  
  else mul (m - 1, n) + n
```

III Werte

Ralf Hinze

Boolesche Werte

Natürliche
Zahlen

Werte/-
definitionen

Funktions/-
definitionen

Funktions-
ausdrücke

Rekursion

Entwurfsmuster

Peano

Entwurfsmuster

Leibniz

Entwurfsmuster

Ratespiel

202

17. Addition

Und die Addition lässt sich auf die Nachfolgerfunktion zurückführen.

```
let rec add (m : Nat, n : Nat) : Nat =  
  if m = 0 then ...  
  else ... add (m - 1, n) ...
```

III Werte

Ralf Hinze

Boolesche Werte

Natürliche
Zahlen

Werte/-
definitionen

Funktions/-
definitionen

Funktions-
ausdrücke

Rekursion

Entwurfsmuster

Peano

Entwurfsmuster

Leibniz

Entwurfsmuster

Ratespiel

203

17. Addition

- ▶ **Rekursionsbasis:** $0 + n = n$.

```
let rec add (m : Nat, n : Nat) : Nat =  
  if m = 0 then n  
  else ... add (m - 1, n) ...
```

- ▶ **Rekursionsschritt:** $m + n = (m - 1) + n + 1$.

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

III Werte

Ralf Hinze

Boolesche Werte

Natürliche
Zahlen

Werte/-
definitionen

Funktions/-
definitionen

Funktions-
ausdrücke

Rekursion

Entwurfsmuster

Peano

Entwurfsmuster

Leibniz

Entwurfsmuster

Ratespiel

204

17. Zwischenfazit

Die Beispielprogramme zeigen, dass wir theoretisch mit einigen wenigen vordefinierten Funktionen auskommen:

- ▶ der Zahl 0,
- ▶ dem Test „gleich 0“,
- ▶ der Nachfolgerfunktion und
- ▶ der Vorgängerfunktion.

☞ Minimalistische Sprachen sind von Vorteil, wenn man Aussagen über *alle* Programme einer Sprache machen möchte. Mehr zu diesem Thema später aus der Abteilung der Theoretischen Informatik.

III Werte

Ralf Hinze

Boolesche Werte

Natürliche
Zahlen

Werte/-
definitionen

Funktions/-
definitionen

Funktions-
ausdrücke

Rekursion

Entwurfsmuster

Peano

Entwurfsmuster

Leibniz

Entwurfsmuster

Ratespiel

205

17. Peano Entwurfsmuster

Haben wir die Aufgabe eine Funktion $f : \text{Nat} \rightarrow t$ zu erstellen, dann sieht ein erster Entwurf folgendermaßen aus.

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

Peano Entwurfsmuster:
Rekursionsbasis
Rekursionsschritt

Die Ellipsen müssen mit Leben gefüllt werden:

- ▶ *Rekursionsbasis*: ein Ausdruck des Typs t .
- ▶ *Rekursionsschritt*: ein Ausdruck, der die Teillösung $f(n \div 1)$ vom Typ t zu einer Gesamtlösung vom Typ t erweitert.

III Werte

Ralf Hinze

Boolesche Werte

Natürliche Zahlen

Werte/-definitionen

Funktions/-definitionen

Funktionsausdrücke

Rekursion

Entwurfsmuster

Peano Entwurfsmuster

Leibniz Entwurfsmuster

206

17. Giuseppe Peano (1858–1932)



Der italienische Mathematiker und Logiker Giuseppe Peano entwickelte, an die Algebra der Logik von Boole, Jevons, Schröder, Porezki anknüpfend, die mathematische Logik weiter.

Von Peano stammt ein bekanntes und noch heute verwendetes *Axiomensystem* für die natürlichen Zahlen.

- ▶ 0 ist eine natürliche Zahl.
- ▶ Für alle n gilt, dass, wenn n eine natürliche Zahl ist, auch die auf n folgende Zahl eine natürliche Zahl ist.
- ▶ Wenn auf zwei Zahlen dieselbe Zahl folgt, sind sie identisch.
- ▶ 0 kann nicht auf eine natürliche Zahl folgen.
- ▶ Das Induktionsaxiom: Wenn 0 eine Eigenschaft hat, und wenn jede auf eine natürliche Zahl folgende Zahl die Eigenschaft besitzt, sofern die Zahl selbst die Eigenschaft hat, dann haben alle natürlichen Zahlen die betreffende Eigenschaft.

III Werte

Ralf Hinze

Boolesche Werte

Natürliche Zahlen

Werte/-definitionen

Funktions/-definitionen

Funktionsausdrücke

Rekursion

Entwurfsmuster

Peano Entwurfsmuster

Leibniz Entwurfsmuster

207

17. Induktion: das Domino Prinzip



III Werte

Ralf Hinze

Boolesche Werte

Natürliche Zahlen

Werte/-definitionen

Funktions/-definitionen

Funktionsausdrücke

Rekursion

Entwurfsmuster

Peano Entwurfsmuster

Leibniz Entwurfsmuster

208

17. Quadratwurzel

Aufgabe: *square-root* n soll die Quadratwurzel der Zahl n bestimmen. Genauer: gesucht wird die *größte* Zahl r , so dass $r * r \leq n$.

```
Mini) square-root 4711  
68  
Mini) square 68  
4624  
Mini) square 69  
4761
```

In eine Formel gegossen suchen wir $\lfloor \sqrt{n} \rfloor$.

III Werte

Ralf Hinze

Boolesche Werte

Natürliche Zahlen

Werte/-definitionen

Funktions/-definitionen

Funktionsausdrücke

Rekursion

Entwurfsmuster

Peano Entwurfsmuster

Leibniz Entwurfsmuster

209

17. Quadratwurzel

Mit dem Peano Entwurfsmuster erhalten wir:

```
let rec square-root (n : Nat) : Nat =  
  if n = 0 then ...  
  else ... square-root (n ÷ 1) ...
```

III Werte

Ralf Hinze

Boolesche Werte

Natürliche
Zahlen

Werte/-
definitionen

Funktions/-
definitionen

Funktions-
ausdrücke

Rekursion

Entwurfsmuster

Peano
Entwurfsmuster

Leibniz
Entwurfsmuster

Ratespiel

210

17. Quadratwurzel

► *Rekursionsbasis*: $\lfloor \sqrt{0} \rfloor = 0$.

```
let rec square-root (n : Nat) : Nat =  
  if n = 0 then 0  
  else ... square-root (n ÷ 1) ...
```

► *Rekursionsschritt*: Wie lässt sich aus $\lfloor \sqrt{n-1} \rfloor$ eine Lösung für $\lfloor \sqrt{n} \rfloor$ herleiten? Es gilt:
 $0 \leq \lfloor \sqrt{n} \rfloor - \lfloor \sqrt{n-1} \rfloor \leq 1$ wenn $n > 0$. Also,

- $\lfloor \sqrt{n} \rfloor$ ist entweder identisch zu $\lfloor \sqrt{n-1} \rfloor$ oder
- $\lfloor \sqrt{n} \rfloor$ ist um eins größer.

Wir testen einfach, ob wir mit der Erhöhung über das Ziel hinausschießen.

```
let rec square-root (n : Nat) : Nat =  
  if n = 0 then 0  
  else let r = square-root (n ÷ 1)  
        in if n < square (r + 1) then r else r + 1
```

III Werte

Ralf Hinze

Boolesche Werte

Natürliche
Zahlen

Werte/-
definitionen

Funktions/-
definitionen

Funktions-
ausdrücke

Rekursion

Entwurfsmuster

Peano
Entwurfsmuster

Leibniz
Entwurfsmuster

Ratespiel

211

17. Peano Entwurfsmuster — programmiert

Was ist im Rekursionsschritt zu tun?

- *Rekursionsschritt*: ein Ausdruck, der die Teillösung $f (n \div 1)$ vom Typ t zu einer Gesamtlösung vom Typ t erweitert.

Einsicht: die Erweiterung der Teillösung zu einer Gesamtlösung ist der Natur nach eine Funktion.

Geben wir also den fehlenden Programmteilen im Schema einen Namen.

```
let rec f (n : Nat) : Nat =  
  if n = 0 then zero  
  else succ (f (n ÷ 1))
```

III Werte

Ralf Hinze

Boolesche Werte

Natürliche
Zahlen

Werte/-
definitionen

Funktions/-
definitionen

Funktions-
ausdrücke

Rekursion

Entwurfsmuster

Peano
Entwurfsmuster

Leibniz
Entwurfsmuster

Ratespiel

212

17. Peano Entwurfsmuster — programmiert

Abstrahieren wir von *zero* und *succ*, erhalten wir eine *Implementierung* des Peano Entwurfsmusters.

```
let peano-pattern (zero : Nat, succ : Nat → Nat) : Nat → Nat =  
  let rec f (n : Nat) : Nat =  
    if n = 0 then zero  
    else succ (f (n ÷ 1))  
  in f
```

III Werte

Ralf Hinze

Boolesche Werte

Natürliche
Zahlen

Werte/-
definitionen

Funktions/-
definitionen

Funktions-
ausdrücke

Rekursion

Entwurfsmuster

Peano
Entwurfsmuster

Leibniz
Entwurfsmuster

Ratespiel

213

17. Peano Entwurfsmuster — programmiert

Mit Hilfe von *peano-pattern* können wir *power* usw. kürzer aufschreiben.

```
let power (x, n) = (peano-pattern (1, fun s → s * x)) n
let mul (m, n) = (peano-pattern (0, fun s → s + n)) m
let add (m, n) = (peano-pattern (n, fun s → s + 1)) m
```

☞ Man sieht sehr schön, wie jeweils die Teillösung *s* zu einer Gesamtlösung erweitert wird.

III Werte

Ralf Hinze

Boolesche Werte

Natürliche
Zahlen

Werte/-
definitionen

Funktions/-
definitionen

Funktions-
ausdrücke

Rekursion

Entwurfsmuster

Peano
Entwurfsmuster

Leibniz
Entwurfsmuster

Ratespiel

214

17. Peano Entwurfsmuster — programmiert

Ganz perfekt ist die Umsetzung des Entwurfsmusters noch nicht.

- ▶ Der Typ ist zu speziell:

```
let peano-pattern (zero : Nat, succ : Nat → Nat) : Nat → Nat =
```

- ▶ Die Fakultätsfunktion und die Quadratwurzel lassen sich damit nicht definieren.

III Werte

Ralf Hinze

Boolesche Werte

Natürliche
Zahlen

Werte/-
definitionen

Funktions/-
definitionen

Funktions-
ausdrücke

Rekursion

Entwurfsmuster

Peano
Entwurfsmuster

Leibniz
Entwurfsmuster

Ratespiel

215