

35. Knobelaufgabe #19

Die *unendliche* Folge

```
0 1 0 2 0 1 0 3 0 1 0 2 0 1 0 4 0 1 0 2 0 1 0 3 0 1 0 2 0 1 0 5 ...
```

ist invariant unter der folgenden Transformation.

- ▶ Jedes Element wird um 1 erhöht.
- ▶ An den Anfang und zwischen je zwei Elemente wird eine 0 gesetzt.

Programmieren Sie eine Funktion, die alle Elemente der Folge nacheinander ausgibt.

VII Effekte
Ralf Hinze

Ein- und Ausgabe
Zustand
Motivation
Abstrakte Syntax
Statische Semantik
Dynamische Semantik
Vertiefung
Blick über den Tellerrand
Listenbeschreibungen
Kontrollstrukturen
Ausnahmen

750

35. Motivation

Assignment is undoubtedly the most characteristic feature of programming a digital computer, and one that most clearly distinguishes it from other branches of mathematics.

— C.A.R. Hoare (1934–)

VII Effekte
Ralf Hinze

Ein- und Ausgabe
Zustand
Motivation
Abstrakte Syntax
Statische Semantik
Dynamische Semantik
Vertiefung
Blick über den Tellerrand
Listenbeschreibungen
Kontrollstrukturen
Ausnahmen

751

35. Motivation

Wunsch: Die Protokollierung der Auswertung via *Return* und *Call* soll sich an- und ausschalten lassen.

☞ Ein *globaler Schalter* lässt sich mit den bisherigen Mitteln nicht realisieren. Funktionen lassen sich bis dato nur über ihre Funktionsargumente steuern — oder über Eingaben der Benutzer*in.

☞ Erweitert man *Return* und *Call* um ein Boolesches Argument, dann hat man viele *lokale Schalter*, aber keinen globalen.

VII Effekte
Ralf Hinze

Ein- und Ausgabe
Zustand
Motivation
Abstrakte Syntax
Statische Semantik
Dynamische Semantik
Vertiefung
Blick über den Tellerrand
Listenbeschreibungen
Kontrollstrukturen
Ausnahmen

752

35. Motivation

Idee: ein Gedächtnis in Form einer *Speicherzelle* einführen, die abgefragt und manipuliert werden kann.

☞ Ein Schalter kann durch eine Speicherzelle realisiert werden, die einen Booleschen Wert enthält.

```
let trace = ref false
```

☞ Der Ausdruck *ref false* legt eine neue Speicherzelle an, einen Behälter für einen Wert. Der initiale *Inhalt* der allokierten Speicherzelle ist *false*. Als Ergebnis des Aufrufs gibt *ref* die *Adresse* der allokierten Speicherzelle zurück.

VII Effekte
Ralf Hinze

Ein- und Ausgabe
Zustand
Motivation
Abstrakte Syntax
Statische Semantik
Dynamische Semantik
Vertiefung
Blick über den Tellerrand
Listenbeschreibungen
Kontrollstrukturen
Ausnahmen

753

35. Motivation

☞ Mit Hilfe der Adresse können wir den Inhalt der Speicherzelle abfragen.

```
!trace
```

☞ Wertet e zu einer Adresse aus, so ermittelt $!e$ den Inhalt der adressierten Speicherzelle (lies: „bang e “).

Damit können wir eine Version von *putline* definieren, die sich von außen über den Schalter steuern lässt.

```
let traceline (s : String) =  
  if !trace then putline s
```

☞ Nur wenn der Inhalt der Speicherzelle *true* ist, erfolgt die Ausgabe. (Wo ist das *else*?)

VII Effekte
Ralf Hinze

Ein- und
Ausgabe
Zustand
Motivation
Abstrakte Syntax
Statische Semantik
Dynamische
Semantik
Vertiefung
Blick über den
Tellerand
Listenbeschrei-
bungen
Kontrollstruktu-
ren
Ausnahmen

754

35. Motivation

☞ Mit Hilfe der *Zuweisung* $e_1 := e_2$ können wir den Inhalt der von e_1 adressierten Speicherzelle auf den Wert von e_2 setzen.

```
trace := true
```

Der Ausdruck schaltet die Protokollierung an.

```
trace := false
```

Der Ausdruck schaltet die Protokollierung aus.

☞ Der Wert der Zuweisung ist jeweils '()', unser liebgewonnener Dummywert.

In der Definition von *Return* und *Call* ersetzen wir *putline* durch *traceline*.

VII Effekte
Ralf Hinze

Ein- und
Ausgabe
Zustand
Motivation
Abstrakte Syntax
Statische Semantik
Dynamische
Semantik
Vertiefung
Blick über den
Tellerand
Listenbeschrei-
bungen
Kontrollstruktu-
ren
Ausnahmen

755

35. Demo

```
Mini) factorial 1  
1  
Mini) trace := true  
()  
Mini) !trace  
true  
Mini) factorial 1  
call 0  
return 1  
return 1  
1  
Mini) trace := false  
()  
Mini) !trace  
false  
Mini) factorial 1  
1
```

☞ '!' ist keine Funktion im mathematischen Sinne: der gleiche Aufruf, *!trace*, führt zu zwei unterschiedlichen Ergebnissen.

VII Effekte
Ralf Hinze

Ein- und
Ausgabe
Zustand
Motivation
Abstrakte Syntax
Statische Semantik
Dynamische
Semantik
Vertiefung
Blick über den
Tellerand
Listenbeschrei-
bungen
Kontrollstruktu-
ren
Ausnahmen

756

35. Exkursion: Module

☞ Namen zu erfinden ist schwer!

F# erlaubt es, Definitionen in sogenannten Modulen zusammenzufassen, die jeweils einen eigenen „Namensraum“ bilden.

```
module Values.Modules
```

```
module Ann =
```

```
  let name = "Ann"
```

```
module Bob =
```

```
  let name = "Bob"
```

```
let hello = "Hello " ^ Ann.name ^ " and " ^ Bob.name ^ "!"
```

Das Programm definiert das Modul *Values.Modules*, das zwei lokale Module, *Ann* und *Bob*, enthält. Der Bezeichner *name* wird zweimal definiert, lebt aber in zwei unterschiedlichen Namensräumen.

Auf die definierten Werte kann mittels qualifizierter Namen zugegriffen werden: *Ann.name* und *Bob.name*. (Von „außen“ entsprechend: *Values.Modules.Ann.name*.)

VII Effekte
Ralf Hinze

Ein- und
Ausgabe
Zustand
Motivation
Abstrakte Syntax
Statische Semantik
Dynamische
Semantik
Vertiefung
Blick über den
Tellerand
Listenbeschrei-
bungen
Kontrollstruktu-
ren
Ausnahmen

757

35. Exkursion: Module

Die Verwendung qualifizierter Namen ist oft mühselig.

```
module Values.Modules
```

```
module Ann =
```

```
  let name = "Ann"
```

```
module Bob =
```

```
  let name = "Bob"
```

```
open Ann
```

```
let hello = "Hello " ^ name ^ " and " ^ Bob.name ^ "!"
```

Mit **open** werden Namensräume geöffnet.

VII Effekte

Ralf Hinze

Ein- und
Ausgabe

Zustand

Motivation

Abstrakte Syntax

Statische Semantik

Dynamische

Semantik

Vertiefung

Blick über den

Tellerand

Listenbeschrei-

bungen

Kontrollstruktu-

ren

Ausnahmen

758

35. Bankkonto

☞ Speicherzellen können beliebige Werte enthalten:

- ▶ Boolesche Werte,
- ▶ natürliche Zahlen,
- ▶ Funktionen,
- ▶ Adressen anderer Speicherzellen usw.

☞ Eine Speicherzelle, die eine *natürliche* Zahl enthält, kann zum Beispiel verwendet werden, um ein *Bankkonto* zu modellieren: der Inhalt repräsentiert den Kontostand.

VII Effekte

Ralf Hinze

Ein- und
Ausgabe

Zustand

Motivation

Abstrakte Syntax

Statische Semantik

Dynamische

Semantik

Vertiefung

Blick über den

Tellerand

Listenbeschrei-

bungen

Kontrollstruktu-

ren

Ausnahmen

759

35. Bankkonto

```
module Account =
```

```
  let private funds = ref 0
```

```
  let deposit (amount : Nat) =
```

```
    funds := !funds + amount
```

```
  let withdraw (amount : Nat) =
```

```
    let old = !funds
```

```
    funds := !funds ÷ amount
```

```
    old ÷ !funds
```

```
  let balance () = !funds
```

☞ '÷' bezeichnet die Subtraktion auf den natürlichen Zahlen („minus“). Die Differenz zwischen dem alten und dem neuen Kontostand wird zurückgegeben.

VII Effekte

Ralf Hinze

Ein- und
Ausgabe

Zustand

Motivation

Abstrakte Syntax

Statische Semantik

Dynamische

Semantik

Vertiefung

Blick über den

Tellerand

Listenbeschrei-

bungen

Kontrollstruktu-

ren

Ausnahmen

760

35. Kapselung

- ▶ Die Speicherzelle *funds* ist lokal zu *deposit*, *withdraw* und *balance*.
- ▶ Der Zusatz **private** stellt sicher, dass der Bezeichner *funds* nur innerhalb des Moduls sichtbar ist.
- ▶ Man sagt auch, der Zustand ist *gekapselt*; von außen ist nicht sichtbar, dass die Funktionen eine Speicherzelle verwenden.
- ▶ Der Kontostand kann nur mit Hilfe der Funktion *balance* eingesehen werden.
- ▶ Der Kontostand kann nicht direkt über eine Zuweisung verändert werden, sondern nur indirekt mit *deposit* und *withdraw*.
- ▶ Heimliche Kontomanipulationen sind in der Bankenwelt sehr ungern gesehenen.

VII Effekte

Ralf Hinze

Ein- und
Ausgabe

Zustand

Motivation

Abstrakte Syntax

Statische Semantik

Dynamische

Semantik

Vertiefung

Blick über den

Tellerand

Listenbeschrei-

bungen

Kontrollstruktu-

ren

Ausnahmen

761

35. Demo

```
Mini) Account.deposit 4711
()
Mini) Account.withdraw 815
815
Mini) Account.withdraw 2765
2765
Mini) Account.withdraw 2765
1131
Mini) Account.withdraw 2765
0
```

☞ *withdraw* ist keine mathematische Funktion: drei Aufrufe der Form *withdraw 2765*, drei unterschiedliche Funktionsergebnisse.

VII Effekte

Ralf Hinze

Ein- und Ausgabe

Zustand

Motivation

Abstrakte Syntax

Statische Semantik

Dynamische Semantik

Vertiefung

Blick über den Tellerrand

Listenbeschreibungen

Kontrollstrukturen

Ausnahmen

762



Endlich mal was, das einigermaßen bekannt aussieht. Statt $funds := !funds + amount$ würde ich ja schreiben

$funds = funds + amount$

Und ich habe schon gedacht, du hättest C++ vergessen ;-).



Mon dieu, dosch nischt '='. Dasch ischt doch kein Vergleich, sondern eine Zuweisung.

$funds := funds + amount$

Na ja, es *ist* nicht gleich, aber es wird doch gleich *gemacht*.



Aber die Ausdrücke links und rechts haben doch noch nicht einmal den gleichen Typ!

VII Effekte

Ralf Hinze

Ein- und Ausgabe

Zustand

Motivation

Abstrakte Syntax

Statische Semantik

Dynamische Semantik

Vertiefung

Blick über den Tellerrand

Listenbeschreibungen

Kontrollstrukturen

Ausnahmen

763



Wieso das denn nicht? Links *Nat* (na ja, eigentlich *unsigned int*) und rechts *Nat*.

Dann darf ich auch schreiben:

$4711 = 815$

Links *Nat*, rechts *Nat*.



Natürlich nicht! Links muss eine Variable stehen!

Genau. Links muss ein Ausdruck stehen, der zu einer *Adresse* ausgewertet. Der neue *Inhalt* der adressierten Speicherzelle steht dann auf der rechten Seite.



Aber dann ergibt doch $funds + amount$ keinen Sinn: *funds* ist eine Adresse und *amount* eine Zahl.

Genau. Deswegen schreiben wir in Mini-F# ja auch $funds := !funds + amount$.



VII Effekte

Ralf Hinze

Ein- und Ausgabe

Zustand

Motivation

Abstrakte Syntax

Statische Semantik

Dynamische Semantik

Vertiefung

Blick über den Tellerrand

Listenbeschreibungen

Kontrollstrukturen

Ausnahmen

764

35. Abstrakte Syntax

Wir erweitern Mini-F# um Speicheroperationen.

$e ::= \dots$
 $\mid \text{ref } e$
 $\mid !e$
 $\mid e_1 := e_2$

Ausdrücke:

Allokation

Dereferenzierung

Zuweisung

☞ *ref* *e* allokiert eine Speicherzelle und gibt die Adresse der bzw. eine Referenz auf die Speicherzelle zurück. Aus diesem Grund heißt der Zugriff *!e* auch Dereferenzierung.

VII Effekte

Ralf Hinze

Ein- und Ausgabe

Zustand

Motivation

Abstrakte Syntax

Statische Semantik

Dynamische Semantik

Vertiefung

Blick über den Tellerrand

Listenbeschreibungen

Kontrollstrukturen

Ausnahmen

765

35. Statische Semantik

Adressen erhalten einen Referenztyp; dieser ist mit dem Typ des Inhaltes parametrisiert.

$t ::= \dots$
 $| \text{Ref}(t)$ *Typen:*
Referenztyp

Typregeln:

$$\frac{\Sigma \vdash e : t}{\Sigma \vdash \text{ref } e : \text{Ref}(t)}$$

$$\frac{\Sigma \vdash e : \text{Ref}(t)}{\Sigma \vdash !e : t}$$

$$\frac{\Sigma \vdash e_1 : \text{Ref}(t) \quad \Sigma \vdash e_2 : t}{\Sigma \vdash e_1 := e_2 : \text{Unit}}$$

VII Effekte
Ralf Hinze

Ein- und Ausgabe
Zustand
Motivation
Abstrakte Syntax
Statische Semantik
Dynamische Semantik
Vertiefung
Blick über den Tellerrand
Listenbeschreibungen
Kontrollstrukturen
Ausnahmen

766

35. Statische Semantik

Zur Erinnerung: Ausdrücke können beliebig miteinander kombiniert werden.

$\text{let } p = (\text{ref } \text{false}, \text{ref } 0)$

p hat den Typ $\text{Ref}(\text{Bool}) * \text{Ref}(\text{Nat})$.

Alle folgenden Ausdrücke sind zulässig:

$!(\text{fst } p)$
 $\text{snd } p := 4711$
 $\text{fst } (\text{swap } p) := !(\text{snd } p) + 1$

VII Effekte
Ralf Hinze

Ein- und Ausgabe
Zustand
Motivation
Abstrakte Syntax
Statische Semantik
Dynamische Semantik
Vertiefung
Blick über den Tellerrand
Listenbeschreibungen
Kontrollstrukturen
Ausnahmen

767

35. Dynamische Semantik

Die drei neuen Konstrukte manipulieren einen sogenannten (Haupt-) Speicher.

Ein Speicher ist eine endliche Abbildung von Adressen auf Werte.

$a \in \text{Addr}$ *Adressen*
 $\sigma \in \text{Addr} \rightarrow_{\text{fin}} \text{Val}$ *Speicher*

Den Bereich der Adressen lassen wir abstrakt; wir fordern nur, dass es unendlich viele Adressen gibt.

VII Effekte
Ralf Hinze

Ein- und Ausgabe
Zustand
Motivation
Abstrakte Syntax
Statische Semantik
Dynamische Semantik
Vertiefung
Blick über den Tellerrand
Listenbeschreibungen
Kontrollstrukturen
Ausnahmen

768

35. Dynamische Semantik

- Die Auswertung verändert sich mit dem Einzug von Effekten.
- Wie müssen wir die Auswertungsregeln modifizieren, um Speichermodifikationen modellieren zu können?
- Ein Ausdruck hat neben einem Wert zusätzlich einen Effekt. Die dreistellige Relation

$\delta \vdash e \Downarrow \nu$

taugt nicht mehr für dieses Szenario.

- Idee: Wir erweitern die Auswertungsrelation zu einer fünfstelligen Relation

$\delta \vdash \sigma \parallel e \Downarrow \nu \parallel \sigma'$

Der Ausdruck e wertet zu ν aus und bewirkt zusätzlich eine Zustandsänderung: σ ist der Speicher vor der Auswertung von e und σ' ist der Speicher nach der Auswertung.

- (Im Interesse der Lesbarkeit lassen wir die Umgebung δ unter den Tisch fallen — ebenso ignorieren wir Ein- und Ausgaben.)

VII Effekte
Ralf Hinze

Ein- und Ausgabe
Zustand
Motivation
Abstrakte Syntax
Statische Semantik
Dynamische Semantik
Vertiefung
Blick über den Tellerrand
Listenbeschreibungen
Kontrollstrukturen
Ausnahmen

769

35. Vertiefung

☞ Eine Speicherzelle vom Typ $\text{Ref}(\text{Nat})$ können wir verwenden, um zu zählen, wie oft eine bestimmte Funktion aufgerufen wird. (Zum Zwecke der Programmoptimierung.)

```
let counter = ref 0
let rec fibonacci (n : Nat) : Nat =
  counter := !counter + 1;
  if n ≤ 1 then
    n
  else
    fibonacci (n ÷ 1) + fibonacci (n ÷ 2)
```

VII Effekte

Ralf Hinze

Ein- und Ausgabe
Zustand
Motivation
Abstrakte Syntax
Statische Semantik
Dynamische Semantik
Vertiefung
Blick über den Tellerrand
Listenbeschreibungen
Kontrollstrukturen
Ausnahmen

774

35. Demo

Die Berechnung von $\text{fibonacci } n$ ist aufwändig:

```
Mini) counter := 0; let f = fibonacci 10 in (f, !counter)
(55, 177)
Mini) counter := 0; let f = fibonacci 20 in (f, !counter)
(6765, 21891)
```

☞ Die Anzahl der Aufrufe übersteigt den Wert der Fibonaccifunktion in beiden Fällen.

VII Effekte

Ralf Hinze

Ein- und Ausgabe
Zustand
Motivation
Abstrakte Syntax
Statische Semantik
Dynamische Semantik
Vertiefung
Blick über den Tellerrand
Listenbeschreibungen
Kontrollstrukturen
Ausnahmen

775

35. Vertiefung

Wir können auch effektfrei zählen:

```
let rec counting-fibonacci (n : Nat) : Nat * Nat =
  if n ≤ 1 then
    (n, 1)
  else
    let (f1, c1) = counting-fibonacci (n ÷ 1)
    let (f2, c2) = counting-fibonacci (n ÷ 2)
    (f1 + f2, c1 + c2 + 1)
```

☞ Die Zahl c der Aufrufe für die Berechnung von $\text{fibonacci } n$ ist fast doppelt so groß wie der Funktionswert von $\text{fibonacci } (n + 1)$, es gilt: $c = 2 \cdot \text{fibonacci } (n + 1) - 1$.

VII Effekte

Ralf Hinze

Ein- und Ausgabe
Zustand
Motivation
Abstrakte Syntax
Statische Semantik
Dynamische Semantik
Vertiefung
Blick über den Tellerrand
Listenbeschreibungen
Kontrollstrukturen
Ausnahmen

776

35. Memoisierung

Problem: Die gleichen Funktionswerte werden wiederholt berechnet: $\text{fibonacci } 8$ wird bei der Berechnung von $\text{fibonacci } 20$ insgesamt 233 mal (= $\text{fibonacci } 12$) neu ausgerechnet.

Idee: Wir merken uns die Aufrufe und greifen später auf die memorierten Werte zurück.

```
let memo-fibonacci = [| for i in 0..99 → fibonacci i |]
```

☞ Statt $\text{fibonacci } n$ verwenden wir $\text{memo-fibonacci}.[n]$.

VII Effekte

Ralf Hinze

Ein- und Ausgabe
Zustand
Motivation
Abstrakte Syntax
Statische Semantik
Dynamische Semantik
Vertiefung
Blick über den Tellerrand
Listenbeschreibungen
Kontrollstrukturen
Ausnahmen

777

35. Memoisierung

Abstrahieren wir von $0..99$ und von *fibonacci*, erhalten wir

```
let memo (dom : Nat, func : Nat → 'v) : Nat → 'v =
  let
    memo-table = [| for i in 0..dom - 1 → func i |]
  in
    fun (n : Nat) → memo-table.[n]
let memo-fibonacci = memo (100, fibonacci)
```

VII Effekte

Ralf Hinze

Ein- und Ausgabe
Zustand
Motivation
Abstrakte Syntax
Statische Semantik
Dynamische Semantik
Vertiefung
Blick über den Tellerrand
Listenbeschreibungen
Kontrollstrukturen
Ausnahmen

778

35. Demo

fsi Memo.fs

...

☞ Unglücklicherweise dauert die Auswertung von *fibonacci* extrem (!) lange, da zunächst die Tabelle vollständig gefüllt wird.

☞ Eigentlich schwebt uns eine *bedarfsgetriebene* Füllung der Tabelle vor: erst wenn ein Funktionswert angefordert wird, berechnet *memo* den entsprechenden Tabelleneintrag.

VII Effekte

Ralf Hinze

Ein- und Ausgabe
Zustand
Motivation
Abstrakte Syntax
Statische Semantik
Dynamische Semantik
Vertiefung
Blick über den Tellerrand
Listenbeschreibungen
Kontrollstrukturen
Ausnahmen

779

35. Memoisierung

☞ Wenn wir Tabelleneinträge ändern wollen, müssen wir statt einem Array von Zahlen ein Array von Speicherzellen verwenden.

Jede Speicherzelle nimmt einen von zwei möglichen Zuständen an:

- ▶ „der Funktionswert wurde noch nicht berechnet“ oder
- ▶ „der Wert wurde berechnet und er lautet ...“.

☞ Die beiden Zustände können wir mit Elementen des Datentyps *Option* modellieren:

- ▶ *None* bzw.
- ▶ *Some value*, wobei *value* der berechnete Wert ist.

☞ Die Memotabelle hat somit insgesamt den furchteinflößenden Typ *Array <Ref <Option 'a>>* statt *Array 'a* wie bisher.

VII Effekte

Ralf Hinze

Ein- und Ausgabe
Zustand
Motivation
Abstrakte Syntax
Statische Semantik
Dynamische Semantik
Vertiefung
Blick über den Tellerrand
Listenbeschreibungen
Kontrollstrukturen
Ausnahmen

780

35. Memoisierung

```
let memo (dom : Nat, func : Nat → 'v) : Nat → 'v =
  let
    memo-table = [| for i in 0..dom - 1 → ref None |]
  in
    fun (n : Nat) →
      match !memo-table.[n] with
      | None → let v = func n
                do memo-table.[n] := Some v
                v
      | Some v → v
let memo-fibonacci = memo (100, fibonacci)
```

VII Effekte

Ralf Hinze

Ein- und Ausgabe
Zustand
Motivation
Abstrakte Syntax
Statische Semantik
Dynamische Semantik
Vertiefung
Blick über den Tellerrand
Listenbeschreibungen
Kontrollstrukturen
Ausnahmen

781

35. Demo

```
fsi Memo.fs
Mini) memo-fibonacci 99
...
```

☞ Die Auswertung der beiden Definitionen dauert nunmehr keinen Fingerschnips.

☞ Jetzt tritt eine lange (!) Stille ein, wenn wir *memo-fibonacci 99* aufrufen: die *rekursiven* Aufrufe der Fibonacci Funktion werden *nicht* memoisiert.

VII Effekte

Ralf Hinze

Ein- und Ausgabe
Zustand
Motivation
Abstrakte Syntax
Statische Semantik
Dynamische Semantik
Vertiefung
Blick über den Tellerrand
Listenbeschreibungen
Kontrollstrukturen
Ausnahmen

782

35. Memoisierung

Was ist zu tun?

- ▶ Wertedefinitionen dürfen nicht rekursiv sein.
- ▶ Die Wertedefinition in eine Funktionsdefinition zu überführen,

```
let rec fibonacci (n : Nat) : Nat =
  memo (100, fun fib (n : Nat) →
    if n ≤ 1 then n
    else fibonacci (n ÷ 1) + fibonacci (n ÷ 2)) n
```

ist verführerisch, aber nicht sinnvoll: Für jeden rekursiven Aufruf wird eine *neue* Memotabelle angelegt!

- ▶ Wir können die rekursiven Aufrufe memoisieren, indem wir *memo* keine Funktion des Typs $\text{Nat} \rightarrow 'v$ übergeben, sondern eine Funktion des Typs

```
(Nat → 'v) → (Nat → 'v)
```

die über die rekursiven Aufrufe abstrahiert.

VII Effekte

Ralf Hinze

Ein- und Ausgabe
Zustand
Motivation
Abstrakte Syntax
Statische Semantik
Dynamische Semantik
Vertiefung
Blick über den Tellerrand
Listenbeschreibungen
Kontrollstrukturen
Ausnahmen

783

35. Memoisierung

```
let rec-memo (dom : Nat, functional : (Nat → 'v) → (Nat → 'v)) : Nat → 'v =
  let memo-table = [ | for i in 0 .. dom ÷ 1 → ref None | ]
```

```
  let rec memo-f (n : Nat) : 'v =
    match !memo-table.[n] with
    | None → let v = functional memo-f n
              do memo-table.[n] := Some v
            v
    | Some v → v
```

```
  in
  memo-f
```

```
let memo-fibonacci =
  rec-memo (100,
    fun fib → fun n →
      if n ≤ 1
      then n
      else fib (n ÷ 1) + fib (n ÷ 2))
```

VII Effekte

Ralf Hinze

Ein- und Ausgabe
Zustand
Motivation
Abstrakte Syntax
Statische Semantik
Dynamische Semantik
Vertiefung
Blick über den Tellerrand
Listenbeschreibungen
Kontrollstrukturen
Ausnahmen

784

35. Memoisierung

```
fsi Memo.fs
Mini) memo-fibonacci 99
354224848179261915075
```

☞ Der Lohn der Anstrengungen: sowohl die Definition von *memo-fibonacci* als auch alle Aufrufe von *memo-fibonacci* sind in Windeseile ausgerechnet; auch die rekursiven Aufrufe füllen die Tabelle.

VII Effekte

Ralf Hinze

Ein- und Ausgabe
Zustand
Motivation
Abstrakte Syntax
Statische Semantik
Dynamische Semantik
Vertiefung
Blick über den Tellerrand
Listenbeschreibungen
Kontrollstrukturen
Ausnahmen

785