

Script generated by TTT

Title: Seidl: Info2 (12.01.2018)

Date: Fri Jan 12 08:33:04 CET 2018

Duration: 93:04 min

Pages: 26


module T: Sort = Sort

Ein MiniOcaml-Programm ...

```
let rec comp = fun f g x -> f (g x)
and map = fun f list -> match list
  with [] -> []
  | x::xs -> f x :: map f xs
```

Beispiele für Werte ...

```
1
(1, [true; false])
fun x -> 1 + 1
[fun x -> x+1; fun x -> x+2; fun x -> x+3]
```

317

Ein MiniOcaml-Programm ...

```
let rec comp = fun f g x -> f (g x)
and map = fun f list -> match list
  with [] -> []
  | x::xs -> f x :: map f xs
```

Beispiele für Werte ...

```
1
(1, [true; false])
fun x -> 1 + 1
[fun x -> x+1; fun x -> x+2; fun x -> x+3]
```

317

Idee

- Wir definieren eine Relation: $e \Rightarrow v$ zwischen Ausdrücken und ihren Werten \implies **Big-Step operationelle Semantik**.
- Diese Relation definieren wir mit Hilfe von Axiomen und Regeln, die sich an der **Struktur** von e orientieren.
- Offenbar gilt stets: $v \Rightarrow v$ für jeden Wert v .

318

Lokale Definitionen

$$\frac{e_1 \Rightarrow v_1 \quad e_0[v_1/x] \Rightarrow v_0}{\text{let } x = e_1 \text{ in } e_0 \Rightarrow v_0}$$

Funktionsaufrufe

$$\frac{e \Rightarrow \text{fun } x \rightarrow e_0 \quad e_1 \Rightarrow v_1 \quad e_0[v_1/x] \Rightarrow v_0}{e \ e_1 \Rightarrow v_0}$$

320

Tupel

$$\frac{e_1 \Rightarrow v_1 \quad \dots \quad e_k \Rightarrow v_k}{(e_1, \dots, e_k) \Rightarrow (v_1, \dots, v_k)}$$

Listen

$$\frac{e_1 \Rightarrow v_1 \quad e_2 \Rightarrow v_2}{e_1 :: e_2 \Rightarrow v_1 :: v_2}$$

Globale Definitionen

$$\frac{f = e \quad e \Rightarrow v}{f \Rightarrow v}$$

319

Durch mehrfache Anwendung der Regel für Funktionsaufrufe können wir zusätzlich eine Regel für Funktionen mit **mehreren** Argumenten ableiten:

$$\frac{e_0 \Rightarrow \text{fun } x_1 \dots x_k \rightarrow e \quad e_1 \Rightarrow v_1 \dots e_k \Rightarrow v_k \quad e_0[v_1/x_1, \dots, v_k/x_k] \Rightarrow v}{e_0 \ e_1 \dots e_k \Rightarrow v}$$

Diese abgeleitete Regel macht Beweise etwas weniger umständlich.

321

Pattern Matching

$$\frac{e_0 \Rightarrow v' \equiv p_i[v_1/x_1, \dots, v_k/x_k] \quad e_i[v_1/x_1, \dots, v_k/x_k] \Rightarrow v}{\text{match } e_0 \text{ with } p_1 \rightarrow e_1 \mid \dots \mid p_m \rightarrow e_m \Rightarrow v}$$

— sofern v' auf keines der Muster p_1, \dots, p_{i-1} passt ;)

$4 + 7 \Rightarrow 11$

Eingebaute Operatoren

$$\frac{e_1 \Rightarrow v_1 \quad e_2 \Rightarrow v_2 \quad v_1 \text{ op } v_2 \Rightarrow v}{e_1 \text{ op } e_2 \Rightarrow v}$$

Die unären Operatoren behandeln wir analog.

322

Beispiel 2

```
let f = fun x -> x+1
let s = fun y -> y*y
```

$$\frac{\frac{f = \text{fun } x \rightarrow x+1}{f \Rightarrow \text{fun } x \rightarrow x+1} \quad \frac{s = \text{fun } y \rightarrow y*y}{s \Rightarrow \text{fun } y \rightarrow y*y} \quad \frac{16+1 \Rightarrow 17}{f \ 16 \Rightarrow 17} \quad \frac{2*2 \Rightarrow 4}{s \ 2 \Rightarrow 4} \quad 17+4 \Rightarrow 21}{f \ 16 + s \ 2 \Rightarrow 21}$$

// Benutzungen von $v \Rightarrow v$ haben wir i.a. weggelassen

324

Der eingebaute Gleichheits-Operator

$$v = v \Rightarrow \text{true}$$

$$v_1 = v_2 \Rightarrow \text{false}$$

sofern v, v_1, v_2 Werte sind, in denen keine Funktionen vorkommen, und v_1, v_2 syntaktisch verschieden sind.

Beispiel 1

$$\frac{17+4 \Rightarrow 21 \quad \text{21} \Rightarrow \text{21} \quad 21=21 \Rightarrow \text{true}}{17 + 4 = 21 \Rightarrow \text{true}}$$

323

Beispiel 3

```
let rec app = fun x y -> match x
with [] -> y
| h::t -> h :: app t y
```

Behauptung: $\text{app } (1::[]) (2::[]) \Rightarrow 1::2::[]$

325

Beweis

$$\begin{array}{c}
 \frac{\text{app} = \text{fun } x \ y \rightarrow \dots}{\text{app} \Rightarrow \text{fun } x \ y \rightarrow \dots} \quad \frac{2::[] \Rightarrow 2::[]}{\text{match } [] \dots \Rightarrow 2::[]} \\
 \hline
 \text{app } [] \ (2::[]) \Rightarrow 2::[] \\
 \hline
 \frac{\text{app} = \text{fun } x \ y \rightarrow \dots \quad \frac{1::\text{app } [] \ (2::[]) \Rightarrow 1::2::[]}{\text{match } 1::[] \dots \Rightarrow 1::2::[]}}{\text{app } (1::[]) \ (2::[]) \Rightarrow 1::2::[]}
 \end{array}$$

Handwritten annotations: Red arrows point from a_1 to the $1::\text{app } [] \ (2::[])$ part of the bottom rule. Another red arrow points from a_2 to the $1::2::[]$ result.

// Benutzungen von $v \Rightarrow v$ haben wir i.a. weggelassen

Beweis

$$\begin{array}{c}
 \frac{\text{app} = \text{fun } x \ y \rightarrow \dots}{\text{app} \Rightarrow \text{fun } x \ y \rightarrow \dots} \quad \frac{2::[] \Rightarrow 2::[]}{\text{match } [] \dots \Rightarrow 2::[]} \\
 \hline
 \text{app } [] \ (2::[]) \Rightarrow 2::[] \\
 \hline
 \frac{\text{app} = \text{fun } x \ y \rightarrow \dots \quad \frac{1::\text{app } [] \ (2::[]) \Rightarrow 1::2::[]}{\text{match } 1::[] \dots \Rightarrow 1::2::[]}}{\text{app } (1::[]) \ (2::[]) \Rightarrow 1::2::[]}
 \end{array}$$

Handwritten annotations: A red triangle is drawn around the bottom rule. Above it, $h \mapsto 1, t \mapsto []$ is written. A red dot is placed on the $1::\text{app } [] \ (2::[])$ part of the bottom rule.

// Benutzungen von $v \Rightarrow v$ haben wir i.a. weggelassen

Beispiel 3

```

let rec app = fun x y -> match x
with [] -> y
| h::t -> h :: app t y

```

Handwritten annotations: Red arrows point from $\pi::[]$ to the $h::t$ case and from $z::[]$ to the $h::\text{app } t \ y$ result. A red a_1 is written near the match expression.

Behauptung: $\text{app } (1::[]) \ (2::[]) \Rightarrow 1::2::[]$

Beispiel 3

```

let rec app = fun x y -> match x
with [] -> y
| h::t -> h :: app t y

```

Handwritten annotations: A yellow circle highlights the $h::\text{app } t \ y$ part of the match expression. Red arrows point from 1 and $z::[]$ to the h and $\text{app } t \ y$ parts respectively.

Behauptung: $\text{app } (1::[]) \ (2::[]) \Rightarrow 1::2::[]$

Beispiel 3

```
let rec app = fun x y -> match x
  with [] -> y
       | h::t -> h :: app t y
```

Behauptung: $\text{app } (1::[]) (2::[]) \Rightarrow 1::2::[]$

Handwritten:
 $1::[] \text{ app } [] (2::[]) \Rightarrow 1::2::[]$

325

Beispiel 3

```
let rec app = fun x y -> match x
  with [] -> y
       | h::t -> h :: app t y
```

Behauptung: $\text{app } (1::[]) (2::[]) \Rightarrow 1::2::[]$

325

Beweis

$[\] \Rightarrow [\]$

$$\frac{\frac{\text{app} = \text{fun } x \ y \rightarrow \dots}{\text{app} \Rightarrow \text{fun } x \ y \rightarrow \dots} \quad \frac{2::[] \Rightarrow 2::[]}{\text{match } [] \dots \Rightarrow 2::[]}}{\text{app } [] (2::[]) \Rightarrow 2::[]}$$

$$\frac{\text{app} = \text{fun } x \ y \rightarrow \dots \quad \text{app } [] (2::[]) \Rightarrow 2::[]}{1 :: \text{app } [] (2::[]) \Rightarrow 1::2::[]}$$

$$\frac{\text{app} \Rightarrow \text{fun } x \ y \rightarrow \dots \quad \text{match } 1::[] \dots \Rightarrow 1::2::[]}{\text{app } (1::[]) (2::[]) \Rightarrow 1::2::[]}$$

// Benutzungen von $v \Rightarrow v$ haben wir i.a. weggelassen

326

Beweis

$$\frac{\frac{\text{app} = \text{fun } x \ y \rightarrow \dots}{\text{app} \Rightarrow \text{fun } x \ y \rightarrow \dots} \quad \frac{2::[] \Rightarrow 2::[]}{\text{match } [] \dots \Rightarrow 2::[]}}{\text{app } [] (2::[]) \Rightarrow 2::[]}$$

$$\frac{\text{app} = \text{fun } x \ y \rightarrow \dots \quad \text{app } [] (2::[]) \Rightarrow 2::[]}{1 :: \text{app } [] (2::[]) \Rightarrow 1::2::[]}$$

$$\frac{\text{app} \Rightarrow \text{fun } x \ y \rightarrow \dots \quad \text{match } 1::[] \dots \Rightarrow 1::2::[]}{\text{app } (1::[]) (2::[]) \Rightarrow 1::2::[]}$$

// Benutzungen von $v \Rightarrow v$ haben wir i.a. weggelassen

326

Beispiel 3

```
let rec app = fun x y -> match x
  with [] -> y
       | h::t -> h :: app t y
```

Behauptung: $\text{app } (1::[]) (2::[]) \Rightarrow 1::2::[]$

325

Diskussion

- Die **Big-Step operationelle Semantik** ist nicht sehr gut geeignet, um Schritt für Schritt nachzu vollziehen, was ein **MiniOcaml**-Programm macht.
- Wir können damit aber sehr gut nachweisen, dass die Auswertung eine Funktion für bestimmte Argumentwerte stets terminiert:
Dazu muss nur nachgewiesen werden, dass es jeweils einen Wert gibt, zu dem die entsprechende Funktionsanwendung ausgewertet werden kann ...

327

Beweis

Handwritten notes: $\text{match } [] \text{ mit } [] \rightarrow 2::[]$
 $1::t \rightarrow h::\text{app } t (2::[])$

$$\frac{\frac{\text{app} = \text{fun } x \ y \rightarrow \dots}{\text{app} \Rightarrow \text{fun } x \ y \rightarrow \dots} \quad \frac{2::[] \Rightarrow 2::[]}{\text{match } [] \dots \Rightarrow 2::[]}}{\text{app } [] (2::[]) \Rightarrow 2::[]}$$

$$\frac{\frac{\text{app} = \text{fun } x \ y \rightarrow \dots}{\text{app} \Rightarrow \text{fun } x \ y \rightarrow \dots} \quad \frac{1::\text{app } [] (2::[]) \Rightarrow 1::2::[]}{\text{match } 1::[] \dots \Rightarrow 1::2::[]}}{\text{app } (1::[]) (2::[]) \Rightarrow 1::2::[]}$$

// Benutzungen von $v \Rightarrow v$ haben wir i.a. weggelassen

326

Beweis

$$\frac{\frac{\text{app} = \text{fun } x \ y \rightarrow \dots}{\text{app} \Rightarrow \text{fun } x \ y \rightarrow \dots} \quad \frac{2::[] \Rightarrow 2::[]}{\text{match } [] \dots \Rightarrow 2::[]}}{\text{app } [] (2::[]) \Rightarrow 2::[]}$$

$$\frac{\frac{\text{app} = \text{fun } x \ y \rightarrow \dots}{\text{app} \Rightarrow \text{fun } x \ y \rightarrow \dots} \quad \frac{1::\text{app } [] (2::[]) \Rightarrow 1::2::[]}{\text{match } 1::[] \dots \Rightarrow 1::2::[]}}{\text{app } (1::[]) (2::[]) \Rightarrow 1::2::[]}$$

// Benutzungen von $v \Rightarrow v$ haben wir i.a. weggelassen

326

Diskussion

- Die **Big-Step operationelle Semantik** ist nicht sehr gut geeignet, um Schritt für Schritt nachzu vollziehen, was ein **MiniOcaml**-Programm macht.
- Wir können damit aber sehr gut nachweisen, dass die Auswertung eine Funktion für bestimmte Argumentwerte stets terminiert:
Dazu muss nur nachgewiesen werden, dass es jeweils einen Wert gibt, zu dem die entsprechende Funktionsanwendung ausgewertet werden kann ...

327

Beispiel-Behauptung

$\text{app } l_1 l_2$ terminiert für alle Listen-Werte l_1, l_2 .

Beweis

Induktion nach der Länge n der Liste l_1 .

$n = 0$: D.h. $l_1 = []$. Dann gilt:

$$\frac{\text{app} = \text{fun } x \ y \ \rightarrow \dots \quad \text{match } [] \ \text{with } [] \ \rightarrow \ l_2 \ | \ \dots \ \Rightarrow \ l_2}{\text{app } [] \ l_2 \Rightarrow l_2}$$

$[] \Rightarrow [] \quad l_2 \Rightarrow l_2$

328

$n > 0$: D.h. $l_1 = h :: t$.

Insbesondere nehmen wir an, dass die Behauptung bereits für alle kürzeren Listen gilt. Deshalb haben wir:

$$\text{app } t \ l_2 \Rightarrow l$$

für ein geeignetes l . Wir schließen:

$$\frac{\text{app} = \text{fun } x \ y \ \rightarrow \dots \quad \text{app } t \ l_2 \Rightarrow l \quad \text{match } h :: t \ \text{with } \dots \ \Rightarrow \ h :: l}{\text{app } (h :: t) \ l_2 \Rightarrow h :: l}$$

$h :: (\text{app } t \ l_2) \Rightarrow h :: l$

329