

Informatik II Skript

Steffen Lindner

April 23, 2015

Contents

1	Einführung - 14.04.15	3
2	Ausdrücke, Defines, usw. - 16.04.2015	4
3	Signaturen, Testfälle - 21.04.15	6
4	Substitutionsmodell, Fallunterscheidung	8

Chapter 1

Einführung - 14.04.15

Scheme: Ausdrücke, Auswertung und Abstraktion

Dr.Racket: Definitionsfenster (oberer Bereich), Interaktionsfenster (unterer Bereich)

Die Anwendung von Funktionen wird in Scheme ausschließlich in Präfixnotation durchgeführt.

Beispiele

Mathematik	Scheme
$44-2$	<code>(- 44 2)</code>
$f(x,y)$	<code>(f x y)</code>
$\sqrt{81}$	<code>(sqrt 81)</code>
9^2	<code>(expt 9 2)</code>
$3!$	<code>(! 3)</code>

Allgemein: (`< function > < arg1 > < arg2 > ...`)

`(+ 40 2)` und `(odd? 42)` sind Beispiele für Ausdrücke, die bei Auswertung einen Wert liefern. (Notation: \rightsquigarrow)

`(+ 40 2)` \rightsquigarrow 42 (\rightsquigarrow = Auswertng / Reduktion / Evaluation)

`(odd? 42)` \rightsquigarrow #f

Interaktionsfenster: Read \rightarrow Eval \rightarrow Print \rightarrow Read ... (Read-Eval-Print-Loop aka. REPL)

Literale stehen für einen konstanten Wert (auch konstante) und sind nicht weiter reduzierbar.

Literal:

#t, #f (true, false, Wahrheitswerte) (boolean)

"abc", "x", " " (Zeichenkette) (String)

0 1904 42 -2 (ganze Zahlen) (Integer)

0.42 3.1415 (Fließkommazahl) (Real)

1/2, 3/4 (rationale Zahl) (Rational)

(""/_ (Bilder) (Image)

Chapter 2

Ausdrücke, Defines, usw. - 16.04.2015

Auswertung zusammengesetzter Ausdrücke in mehreren Schritten (steps), von "innen nach außen" bis keine Reduktion mehr möglich ist.

$$(+ (+ 20 20) (+ 1 1)) \rightsquigarrow (+ 40 (+ 1 1)) \rightsquigarrow (+ 40 2) \rightsquigarrow 42$$

Achtung: Scheme rundet bei Arithmetik mit Fließkommazahlen (interne Darstellung ist binär).

Bsp.: Auswertung des zusammengesetzten Ausdrucks $0.7 + (1/2)/0.25 - 0.6/0.3$

Arithmetik mit rationalen Zahlen ist exakt.

Ein Wert kann an einen Namen (auch Identifier) gebunden werden, durch

$$(\text{define } \langle id \rangle \langle e \rangle) \quad (\langle id \rangle \text{ Identifier, } \langle e \rangle \text{ Expression})$$

Erlaubt konsistente Wiederverwendung und dient der Selbstdokumentation von Programmen.

Achtung: Dies ist eine sogenannte Spezifikation und kein Ausdruck. Insbesondere besitzt diese Spezialform keinen Wert, sondern einen Effekt: Name $\langle id \rangle$ wird an den Wert von $\langle e \rangle$ gebunden.

Namen können in Scheme fast beliebig gewählt werden, solange:

1. die Zeichen (kommt noch) nicht vorkommen
2. der Name nicht einem numerischen Literal gleicht
3. kein whitespace (Leerzeichen, Tabulatoren, Return) enthalten ist.

Bsp.: $\text{euro} \rightarrow \text{us\$}$

Achtung: Groß-/Kleinschreibung ist in Identifiern nicht relevant.

Eine Lambda-Abstraktion (auch: Funktion, Prozedur) erlaubt die Formulierung von Ausdrücken, die mittels Parametern konkreten Werten abstrahieren:

$$(\text{lambda } (\langle p1 \rangle \langle p2 \rangle \dots) \langle e \rangle), \quad \langle e \rangle \text{ Rumpf}$$

$\langle e \rangle$ enthält Vorkommen der Parameter $\langle p1 \rangle, \langle p2 \rangle \dots$

$(\text{lambda } \dots)$ ist eine Spezialform. Wert der Lambda-Abstraktion ist $\# \langle procedure \rangle$

Anwendung (auch: Applikation/Aufruf) der Lambda-Abstraktion führt zur Ersetzung der vorkommenden Parameter im Rumpf durch die angegebenen Argumente:

(lambda (days) (* days (* 155 min-in-a-day))) \rightsquigarrow (* 365 (* 155 min-in-a-day)) \rightsquigarrow 81468000

In Scheme leitet ein Semikolon einen Kommentar, der bis zum Zeilenende reicht, ein und wird vom System bei der Auswertung ignoriert.

Prozeduren sollten im Programm eine ein-bis zweizeiliger Kurzbeschreibung direkt voran gestellt werden.

Chapter 3

Signaturen, Testfälle - 21.04.15

Eine Signatur prüft, ob ein Name an einen Wert einer angegebenen Sorte (Typ) gebunden wird. Signaturverletzungen werden protokolliert.

$$(: < id > < signatur >)$$

Bereits eingebaute Signaturen:

- natural \mathbb{N}
- integer \mathbb{Z}
- rational \mathbb{Q}
- real \mathbb{R}
- number \mathbb{C}
- boolean
- string
- image

(: ...) ist eine Spezialform ohne Wert, aber Effekt: Signaturprüfung

Prozedur-Signaturen spezifizieren sowohl Signaturen für die Parameter p_1, p_2, \dots, p_n als auch den Ergebniswert der Prozedur:

$$(< signaturp_1 > \dots < signaturp_n > - > < signatur - ergebnis >)$$

Prozedur-Signaturen werden bei jeder Anwendung eine Prozedur auf Verletzung geprüft.

Testfälle dokumentieren das erwartete Ergebnis einer Prozedur für ausgewählte Argumente:

$$(check - expect < e_1 > < e_2 >)$$

Werte Ausdruck $\langle e_1 \rangle$ aus und teste, ob der erhaltene Wert der Erwartung (= der Wert von $\langle e_2 \rangle$) entspricht.

Einer Prozedurdefinition sollten Testfälle direkt vorangestellt werden.

Spezialform: Kein Wert, aber Effekt: Testverletzung protokollieren.

Konstruktionsanleitung für Prozeduren

- ; ... (1) Kurzbeschreibung (1-2 zeiliger Kommentar mit Bezug auf Parameter)
- (: ...) (2) Signatur
- (check-expect ...) (3) Testfälle
- (define (lambda (...) ...) (4) Prozedur + Rumpf

Top-Down-Entwurf (Programmieren durch "Wunschdenken")

Bsp.: Zeichnen Ziffernblatt (Stunden- und Minutenzeiger) zur Uhrzeit H:m auf einer analogen 24h-Uhr

- Minutenzeiger legt $360^\circ/60$ pro Minute zurück ($360/60 * m$)
- Stundenzeiger legt $360^\circ/12$ pro Stunde zurück ($360/12 * h + 360/12 * m/60$)

Chapter 4

Substitutionsmodell, Fallunterscheidung -

23.04.15

Reduktionsregeln für Scheme (Fallunterscheidung je nach Ausdrucksart)

Wiederhole, bis keine Reduktion mehr möglich:

- Literal (1, "abc", #t, ...) $[\text{eval}_{lit}]$

$l \rightsquigarrow l$

- Identifier id (pi, clock-face, ...) $[\text{eval}_{id}]$

$id \rightsquigarrow$ gebundener Wert

- Lambda-Abstraktion

$(\text{lambda } ()) \rightsquigarrow (\text{lambda } ()) [\text{eval}_{\lambda}]$

- Applikation (f, e1, e2)

– (1) f, e1, e2 reduziere, erhalte f', e1', e2'

– (2)

* Operation f' auf e1', e2', ... falls f' primitive Operation (+, *, ...) $[\text{apply}_{prim}]$

* Argumentenwert e1', e2', ... Rumpf von f' einsetzen, dann Rumpf reduzieren, falls f' Lambdaabstraktion $[\text{apply}_{\lambda}]$

Beispiel: Applikation

(+ 40 2)

$\rightsquigarrow (\text{\#< procedure + > 40 2}) \rightsquigarrow 42$

$\text{eval}_{lit} (+)$

$\text{eval}_{lit} (40)$

$\text{eval}_{lit} (2)$

(position-minute-hand 30)

$\rightsquigarrow ((\text{lambda } (m) (* \text{degrees-per-minute } m)) 30)$

$\rightsquigarrow (* \text{degrees-per-minute } 30)$

$\rightsquigarrow (* \text{degress-per-minute } 30)$

$\rightsquigarrow (\#<\text{procedure } * > 360/60 30)$

Bezeichnen $(\text{lambda } (x) (* x x))$ und $(\text{lambda } (r) (* r r))$ die gleiche Prozedur? \Rightarrow Ja!

Achtung: Das hat Einfluss auf das korrekte Einsetzen von Argumenten für Parameter! (s. apply_λ)

Das bindenen Vorkommen eines Identifiers x kann im Programmtext systematisch bestimmt werden: suche strik von "innen nach außen" bis zum ersten

- $(\text{lambda } (x))$
- $(\text{define } x)$

(Prinzip der lexikalischen Bindung)

Übliche Notation in der Mathematik: Fallunterscheidung

$$\text{maximum}(x_1, x_2) = \begin{cases} x_1, & \text{falls } x_1 \geq x_2 \\ x_2, & \text{sonst} \end{cases}$$

Tests auch (Prädikate) sind Funktionen, die einen Wert der Signatur boolean liefern. Typische primitive Tests:

- $(: = (\text{number number} \rightarrow \text{boolean}))$
- $(: < (\text{real real} \rightarrow \text{boolean}))$, auch $>, \leq, \geq$
- $(: \text{string}=? (\text{string string} \rightarrow \text{boolean}))$, auch $\text{string}>?, \text{string}\leq?$
- $(: \text{boolean}? (\text{boolean boolean} \rightarrow \text{boolean}))$
- $(: \text{zero}? (\text{number} \rightarrow \text{boolean}))$
- $\text{odd?}, \text{even?}, \text{positive?}, \text{negative?}, \dots$

Binäre Fallunterscheidung: if

$(\text{if } < t_1 > < e_1 > < e_2 >)$

$$\text{Mathematisch: } \begin{cases} e_1, & \text{falls } t_1 \\ e_2, & \text{sonst} \end{cases}$$