

14.4.2015**Scheme**Ausdrücke , Auswertung und Abstaktion**Dr Racket**

Definitonsfenster

Willkommen bei [DrRacket](#), Version 6.1.1 [3m].

Sprache: Die Macht der Abstraktion; memory limit: 128 MB.

> Interaktionsfenster

Die Anwendung von Funktionen wird in Scheme ausschließlich in Präfixnotation durchgeführt

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 92)</code>
$3!$	<code>(! 3)</code>

Allgemein: (`<funktion><argument1><argument2>...`)


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

(Notation: \rightsquigarrow)`(+ 40 2)` \rightsquigarrow 42`(odd? 42)` $\overset{\text{Reduktion}}{\rightsquigarrow}$ #f

Interaktionsfenster:

$$\underbrace{Read \rightarrow Eval \rightarrow Print \rightarrow Loop}_{REPL}$$

Literale sethen für einen konstanten Wert (auch: Konstante) und sind nicht weiter reduzierbar.

Literal		Sorte, Typ
#f, #t	(true, false, Wahrheitswert)	boolean
"x"	(Zeichenketten)	String
0 1904 42 -2	(ganze Zahl)	Integer
0.42 3.14159	(Fließkommazahl)	real
1\2, 3\4, -1\10	(rationale Zahlen)	rational
	(Bilder)	image

16.4.2015

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

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

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

Beispiel: Auswertung des zusammengesetzten Ausdruckes

```
; Achtung: Arithmetik mit Fliesskommazahlen (real)\
unterliegt Rundung!
(+ 0.7
  (- (/ 1/2 0.25)
    (/ 0.6 0.3)))

(- (+ 0.7
    (/ 1/2 0.25))
  (/ 0.6 0.3))

; Arithmetik mit rationalen Zahlen (rational) ist exakt
(- (+ 7/10
    (/ 1/2 1/4))
  (/ 6/10 3/10))
```

Ein Wer kann an einen Namen (auch Identifizier) gebunden werden, durch
(define <id><e>) <id>Identifizier <e>Ausdruck

Erlaubte konsistente Wiederverwendung, dient der Selbstdokumentation von Programmen

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

Namen können in Scheme beliebig gewählt werden, solange

- (1) Die Zeichen () [] { } “ , ‘ ‘ ; # | \ nicht vorkommen
- (2) der nicht einem numerischen Literal gleicht.
- (3) kein Whitespace (Leerzeichen, Tabulator, Return) enthalten ist.

Beispiel: euro → US\$

Achtung: Groß\Kleinschreibung ist irrelevant

```
; Bindung von Werten an Namen
(define absoluter-nullpunkt -273.15)
(define pi 3.141592653)
(define Gruendungsjahr-SC-Freiburg 1904)
(define top-level-domain-germany "de")
(define minutes-in-a-day (* 24 60))
(define vorwahl-tuebingen (sqrt 1/2))
```

Eine lambda-Abstraktion (auch Funktion, Prozedur) erlaubt die Formatierung von Ausdrücken, in denen mittels Parametern von konkreten Werten abstrahiert wird.

(lambda (<p1><p2>...) <e>)

<e> Rumpf ⇒ enthält Vorkommen der Parameter <p_n>

(lambda(...)) ist eine Spezialform. Wert der lambda-Abstraktion ist #<procedure>

Anwendung (auch : Application) des lambda Aufrufs führt zur Ersetzung aller Vorkommen der Parameter im Rumpf durch die angegebenen Argumente.

```
; Abstraktion: Ausdruck mit "Loch" \odot
(lambda (\odot) (* \odot (* 155 minutes-in-a-day)))

; Zuwachs der Weltbevölkerung innerhalb von days Tagen
(define population-growth-in-days
  (lambda (days) (* days (* 155 minutes-in-a-day))))

(population-growth-in-days 7)
```

```
(lambda (days) (* days (* 155 minutes-in-a-day))) 365) ~~~~
(* 365 (* 155 minutes-in-a-day)) ~~~~81468000
```

In Scheme leitet ein Semikolon einen Kommentar ein, der bis zum Zeilenende reicht und

vom System bei der Auswertung ignoriert wird.

Prozeduren sollten im Programm ein- bis zwei zeilige Kurzbeschreibungen direkt vorangestellt werden

21.4.2015

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

(: <id><signatur>)

Bereits eingebaute Sinaturen

natural \mathbb{N} boolean

integer \mathbb{Z} string

rational \mathbb{Q} image

real \mathbb{R} ...

numver \mathbb{C}

(: ...) ist eine Spezialform und hat keinen Wert, aber einen Effekt: Signaturprüfung

Prozedur Signatur spezifizieren sowohl Signaturen für die Parameter P_1, P_2, \dots, P_n als auch den Ergebniswert der Prozedur

(: <Signatur P_1 >... <Signatur P_n >- > <Signatur Ergebnis>)

Prozedur Signaturen werden bei jeder Anwendung einer Prozedur auf Verletzung geprüft Testfälle dokumentieren das erwartete Ergebnis einer Prozedur für ausgewählte Argumente:

(check-expect < e_1 >< e_2 >)

Werte Ausdruck < e_1 >aus und teste, ob der erhaltene Wert der Erwartung < e_2 >entspricht (= der Wert von < e_2 >) Einer Prozedur sollte Testfälle direkt vorangestellt werden.

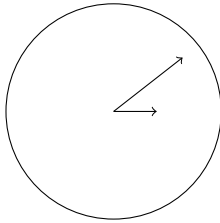
Spezialform: kein Wert, sondern Effekt: Testverletzung protokollieren

Konstruktionsanleitung für Prozeduren:

- (1) Kurzbeschreibung (ein- bis zweizeiliger Kommentar mit Bezug auf Parameternamen)
 - (2) Signaturen
 - (3) Testfälle
 - (4) Prozedurrumpf
-

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

Beispiel:Zeichne Ziffernblatt (Stunden- und Minutenzeiger) zu Uhrzeit auf einer analogen 24 Uhr



Minutenzeiger legt $\frac{360^\circ}{60}$ Grad pro Minute zurück (also $\frac{360}{60} \cdot m$)

Stundenzeiger liegt $\frac{360}{12}$ pro Stunde zurück ($\frac{360}{12} \cdot h + \frac{360}{12} \cdot \frac{m}{60}$)

```
; Grad, die Minutenzeiger pro Minute zuruecklegt
(define degrees-per-minute 360/60)

; Grad, die Stundenzeiger pro voller Stunde zuruecklegt
(define degrees-per-hour 360/12)

; Zeichne Ziffernblatt zur Stunde h und Minute m
(: draw-clock (natural natural -> image))
(check-expect (draw-clock 4 15) (draw-clock 16 15))
(define draw-clock
  (lambda (h m)
    (clock-face (position-hour-hand h m)
                (position-minute-hand m))))

; Winkel (in Grad), den Minutenzeiger zur Minute m einnimmt
(: position-minute-hand (natural -> rational))
(check-expect (position-minute-hand 15) 90)
(check-expect (position-minute-hand 45) 270)
(define position-minute-hand
  (lambda (m)
    (* m degrees-per-minute)))

; Winkel (in Grad), den Stundenzeiger zur Stunde h einnimmt
(: position-hour-hand (natural natural -> rational))
(check-expect (position-hour-hand 3 0) 90)
(check-expect (position-hour-hand 18 30) 195)
(define position-hour-hand
  (lambda (h m)
    (+ (* (modulo h 12) degrees-per-hour)
       ; h mod 12 in {0,1,...,11}
```

```

      (* (/ m 60) degrees-per-hour)))

; Zeichne Ziffernblatt mit Minutenzeiger um dm und
; Stundenzeiger um dh Grad gedreht
(: clock-face (rational rational -> image))
(define clock-face
  (lambda (dh dm)
    (clear-pinhole
     (overlay/pinhole
      (circle 50 "outline" "black")
      (rotate (* -1 dh) (put-pinhole 0 35 (line 0 35 "red"))))
      (rotate (* -1 dm) (put-pinhole 0 45 (line 0 45 "blue"))))))))

```

23.4.2015

Substitutionsmodell

Reduktionsregeln für Scheme (Fallunterscheidung je nach Ausdrücken) wiederhole, bis keine Reduktion mehr möglich

- literal (1, "abc", #t, ...) $l \rightsquigarrow l$ [eval_{lit}]
- Identifier id(pi, clock-face,...) $id \rightsquigarrow$ gebundene Wert [eval_{id}]
- lambda Abstraktion $(\text{lambda } (\dots) \dots) \rightsquigarrow \text{lambda}(\dots) \dots$ [eval_λ]
- Applikationen (f e₁ e₂ ...)

$$f, e_1, e_2 \text{ reduzieren erhalte: } f', e_1', e_2' \quad (1)$$

- (2) $\begin{cases} \text{Operation } f' \text{ auf } e_1' \text{ und } e_2' \text{ [apply}_{\text{prim}}] & \text{falls } f' \text{ primitiv ist} \\ \text{Argumentenwerte in den Rumpf von } f' \text{ einsetzen, dann reduzieren} & \text{falls } f' \text{ lambda Abstraktion} \end{cases}$

Beispiel:

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

$$\begin{aligned}
 (\text{position-minute-hand } 30) & \rightsquigarrow_{\text{eval id}} ((\text{lambda } (m) (* \text{degrees-per-minute } m)) 30) \\
 & \rightsquigarrow_{\text{eval lambda}} (* \text{degrees-per-minute } 30) \\
 & \rightsquigarrow_{\text{eval id}} (\# <\text{procedure } *> \frac{360}{60} 30) \\
 & \rightsquigarrow_{\text{apply prim}} 180
 \end{aligned}$$

Bezeichnen (lambda (x) (* x x)) und lambda (r) (* r r) die gleiche Prozedur? \Rightarrow JA!

Achtung: Das hat Einfluß auf das Korrekte Einsetzen von Argumenten für Prozeduren (siehe apply)

Das bindene Vorkommen eines Identifiers id kann im Programmtext systematisch bestimmt werden: Suche strikt von innen nach außen, bis zum ersten

$$(1) (\text{lambda } (r) <\text{Rumpf}>$$

(2) (define <e>)

Prinzip der lexikalische BindungÜbliche Notation in der Mathematik: Fallunterscheidung

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

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

Typische primitive Tests.

(: = (number number -> boolean))

(: < (real real -> boolean))

auch >, <=, >=

(: String=? (string string -> boolean))

auch string>?, string<=?

(: zero? (number -> boolean))

odd?, even?, positive?, negative?

Binäre Fallunterscheidung if*if*< e₁ > Mathematik:

$$< e_2 > \begin{cases} e_1 & \text{falls } t_1 \\ e_2 & \text{sonst} \end{cases}$$

< e₂ >)

28.4.2015

Die Signatur one of lässt genau einen der ausgewählten Werte zu.(one of <e₁> <e₂> ... <e_n>)

Reduktion von if:

(if t₁ <e₁> <e₂>)

① Reduziere t₁, erhalte t'₁ \rightsquigarrow ② $\begin{cases} < e_1 > & \text{falls } t'_1 = \#t, < e_2 > \text{niemals ausgewertet} \\ < e_2 > & \text{falls } t'_1 = \#f, < e_1 > \text{niemals ausgewertet} \end{cases}$

Spezifikation Fallunterscheidung (conditional expression):

(cond

Mathematik:

$$\begin{array}{l} (<t_1> <e_1>) \\ (<t_2> <e_2>) \\ \dots \\ (<t_n> <e_n>) \\ (\text{else } <e_{n+1}>) \end{array} \left\{ \begin{array}{l} e_1 \text{ falls } t_1 \\ e_2 \text{ falls } t_2 \\ \dots \\ e_n \text{ falls } t_n \\ e_{n+1} \text{ sonst} \end{array} \right.$$

Werte die Tests in den Reihenfolge t₁, t₂, t₃, ..., t_n aus.Sobald t_i # t ergibt, wert Zweig e_i aus. e_i ist Ergebnis der Fallunterscheidung. Wenn t_n # t liefert, dann liefert

$\begin{cases} \text{Fehlermeldung „cond: alle Tests ergaben false“} & \text{falls kein else Zweig} \\ \langle e_{n+1} \rangle & \text{sonst} \end{cases}$

Reduktion von cond [eval_{cond}]

$(\text{cond } (\langle t_1 \rangle \langle e_1 \rangle) (\langle t_2 \rangle \langle e_2 \rangle) \dots (\langle t_n \rangle \langle e_n \rangle))$