

Inf-Prog

18.01.2010

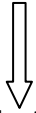
INHALT

Metalinguistische Abstraktion (Kapitel 4)	2
Einführung	2
Entwurf und Realisierung neuer Sprachen	2
Quotierung: Programme als Daten	2
4.3 Ein metazirkulärer Evaluator für (Mini-) Scheme	3

METALINGUISTISCHE ABSTRAKTION (KAPITEL 4)

EINFÜHRUNG

Konkreter Problembereich (z.B. Banken, Studenten)



(vernachlässigen unnötiger Details)

Modell des Problembereichs (z.B. Modellierung der Bankkonten)



(Umsetzung mit konkreter Programmiersprache)

Implementierung es Modells

- angemessen
 - nachvollziehbar
- (man muss das Modell noch wiedererkennen können)



Forderung nach mächtigen
Abstraktionsmechanismen

ENTWURF UND REALISIERUNG NEUER SPRACHEN

- wichtiges Hilfsmittel bei der Realisierung komplexer Anwendungen
- aufwändig
- Eingrenzung des Aufwands durch Metalinguistische Abstraktion
- Idee: realisiere Evaluator (auch Interpreter) für Programmiersprache (PS)
- Evaluator für Sprache PS
 - o Eingabe: Programm der Sprache PS
 - o Ausgabe: Berechnungsergebnis bzgl. der Semantik von PS

Im Folgenden: Evaluator für Scheme in Scheme (metazirkulärer Evaluator)

Vorteile der Herangehensweise:

- präzise Beschreibung des Umgebungsmodells
- Erweiterungen der Sprache möglich
 - o Sonderformen
 - o neue Laufzeitfunktionalität (z.B. Tracing)
 - o Auswertungsstrategien (Normalordnung z.B.)
 - o Neue Sprachkonstrukte (Objekte)

Aus Gründen der Einfachheit: Teilmenge von Scheme => **Mini-Scheme**

QUOTIERUNG: PROGRAMME ALS DATEN

➔ Schonmal gesehen in 2.2.2

Quotieren von Symbolen unterdrückt die Auswertung ('first)

Quotierung ist nicht nur auf Symbole beschränkt, auch Ausdrücke können quotiert werden.

Wenn eine Kombination quotiert wird, dann ist dies äquivalent zu der Liste der quotierten Elemente der Anwendung.

```
| >'(a b 99)
| (list 'a 'b 99)
```

```
>'(define (quadrat x) (* x x))
(list 'define (list 'quadrat 'x) (list '* 'x 'x))
```

Also: Quotierung gibt die Möglichkeit, Scheme-Programme als normale Listen zu verarbeiten.

Für die Eingabeschleife müssen Eingaben des Benutzers in Quotierungen übersetzt werden.

Dazu: Prozedur read

```
>(read)
-- (a b 99)
(list 'a 'b 99)

>(read)
-- (define (quadrat x) (* x x))
(list 'define (list 'quadrat 'x) (list '* 'x 'x))

>(read)
-- '(a b 99)
(list 'quote (list 'a 'b 99))

>(equal? 'first (quote first))
true ;Also: Auch quotierte Ausdrücke können verarbeitet werden.

>(first ``first)
'quote

>(second ``first)
'first
```

4.3 EIN METAZIRKULÄRER EVALUATOR FÜR (MINI-) SCHEME

Syntax von Minischeme und Abstrakte Syntax

Wir lassen in Minischeme folgende Ausdrücke zu:

- Zahlen : Die üblichen Zahlenkonstanten
- Quotierungen: (quote a), wobei a ein Ausdruck ist. Dies entspricht ´a
- Variablen: wie in Scheme üblich, alle möglichen Symbole
- Konditionale: (if b a₁ a₂), b ist Ausdruck, der true/false sein liefert.
- Abstraktionen: (lambda (x₁ ... x_n) a), wobei x₁ ... x_n Parameter, also Variablen sind und a ein Ausdruck ist.
- Definitionen:
 - (define x a), wobei x eine Variable ist, a ein Ausdruck
 - (define (p x₁ ... x_n) a), wobei p ein Symbol ist (Prozedurname), x₁ ... x_n Variablen sind und a der Ausdruck ist
- Zuweisungen: (set! x a), wobei x eine Variable und a ein Ausdruck ist.
- Ausdrucksfolge: (begin a₁ ... a_n), a_i Ausdruck
- Lokale Umgebungen (local (d₁ ... d_n) a), wobei d₁ ... d_n Definitionen sind, a ist ein Ausdruck.
- Applikation: (p a₁ ... a_n), wobei p, a_i Ausdrücke sind.

Was fehlt sind die Konstrukte cond, define-struct

```
(define sample-prog
  `((define sq x) (* x x))
    (define (fac n) (if = n 0) 1 (* n (fac (- n 1))))))
(define (konstr-zaehler)
  (local ((define x 0))
    (lambda () (begin (set! x (+ x 1)) x ...))))
```