Makros in Scheme

Marek Kubica

TU München

7. Juni 2010

Über Scheme

Unterschiede zu Common Lisp

- Wesentlich kleinerer Standard (161 vs. 1029 Seiten)
- Scheme „akademisches“ Lisp, CL „Industrie“-Lisp
- Scheme eher funktional, CL hat u.a. Objektsystem
- nur ein Namespace für Funktionen, Variablen, etc.

Hygienisches Makrosystem

- lange Zeit nicht standardisiert
- nutzt Pattern-Matching
- zwei verschiedene Varianten: `syntax-rules` & `syntax-case`

Wozu Makros?

Vorteile

- Anpassung der Sprache an Anforderungen
- Assimilation von neuen Konzepten
  - OOP
  - Transactional Memory (STM)
  - Actors

Nachteile

- Bildung von Dialekten
- Schwer zu Debuggen

Ein Beispielmakro

Nachbau von `for` aus Python

Aufruf mit einem Namen

```
for x in [1, 2, 3]:
    print x
```

```
(py-for x in '(1 2 3)
        (display x))
```

Aufruf mit einer Liste von Namen

```
for x, y in [('a', 'b'), ('A', 'B'), (1, 2)]:
    print x, y
```

```
(py-for (x y) in '((a b) (A B) (1 2))
        (display x) (display y))
```

Implementation von py-for

```
1  (define-syntax py-for
2    (syntax-rules (in)
3      ((_ (?bindings ...) in ?collection body ...)
4       (let ((code (lambda (?bindings ...) body ...)))
5         (let loop ((collect ?collection))
6           (unless (null? collect)
7             (apply code (car collect))
8             (loop (cdr collect))))))
9      ((_ ?binding in ?collection body ...)
10      (py-for (?binding) in (map list ?collection)
11              body ...))))
```

Unhygienische Makros

Das Problem

```
(swap! x y) ;; expandiert zu
(let ((VALUE x))
  (set! x y)
  (set! y VALUE))
```

Umbenennen

```
(let ((value-name (gensym)))
  (let ((value-name x))
    (set! x y)
    (set! y value-name)))
```

Lösung: Hygienische Makros

Lexikalisches Scoping für Makros

- Makros können nur auf die Identifier zugreifen, die zur *Definitionszeit* verfügbar sind
- Die Implementation kümmert sich um die Details
- analog wie bei Funktionen

Vorteile

- Robust
- Einfach(er) zu schreiben

Metalevel

Das Problem

- Makros werden vor der Laufzeit ausgeführt
- Makros haben einen eigenen „Namensraum“
- Makros können zu Makros expandieren
- Was ist im Namensraum von geschachtelten Makros?
  $\Rightarrow$ Notwendigkeit Identifier richtig zu importieren

Lösungen in R^{6}RS-Scheme

1. Explizites Phasing: Der Programmierer gibt an, welche Identifier in welcher Phase verwendet werden
2. Implizites Phasing: Die Implementation inferiert diese Information

Beispiel für Metalevel

```
1  (library (static-map)
2           (export static-map)
3           (import (rnrs)
4                   (for (rnrs) (meta -1))))
5
6  (define-syntax static-map
7    (syntax-rules ()
8      ((_ (name value) ...)
9       (syntax-rules (<names> name ...)
10        ((_ <names>) '(name ...))
11        ((_ name) value) ...)))))
```

Damenproblem

Schelog

- Prolog eingebettet in Scheme
- ermöglicht Kombination funktionaler mit logischer Programmierung
- hat mich Prolog gelehrt ☺

Algorithmus

- Datenstruktur: Liste. Position stellt Zeile dar, Zahl stellt Spalte dar
- Bildung von Permutationen der Liste
- Prüfen ob die Permutationen gültige Lösungen darstellen

Prolog vs. Schelog

Prolog

```
do_insert(X,Y,[X|Y]).
do_insert(X,[Y1|Y2],[Y1|Z]) :- do_insert(X,Y2,Z).
```

Schelog

```
(define do-insert
  (%rel (x y y1 y2 z)
        ((x y (cons x y)))
        ((x (cons y1 y2) (cons y1 z))
         (do-insert x y2 z))))
```

Ressourcen

Standards

- R^6RS: aktueller Standard, komplizierter, geringe Verbreitung
- R^5RS: vorheriger Standard, einige wichtige Elemente nicht spezifiziert

Nützliche Ressourcen

- The Scheme Programming Language, 4^{th} Edition
- The Adventures of a Pythonista in Schemeland
- Meine Ausarbeitung ☺

Danke fürs Zuhören.