Einführung in die Programmiertechnik

Funktionale Programmierung: LISP

© 2005 Martin v. Löwis

Grundlagen funktionaler Programmierung

- Idee: Zu lösendes Problem wird als mathematische Funktion formuliert
  - Beispiel Rechtschreibprüfung: Gegeben das zu prüfende Programm, bestimme die Menge aller Schreibfehler (leere Menge: Dokument ist fehlerfrei)
  - Beispiel Wettervorhersage: Gegeben das aktuelle Wetter, bestimme das Wetter in 24h
  - Beispiel automatisches Übersetzen: Gegeben einen englischen Text, ermittle einen entsprechenden russischen Text
- Definition der Funktion erfolgt nicht durch Aufzählung von Rechenschritten, sondern durch Zusammensetzung aus vorhandenen Funktionen
  - Beispiel Rechtschreibprüfung: Teilfunktion könnte z.B. “Gegeben ein Wort, bestimme den Wortstamm” sein

© 2006 Martin v. Löwis

Programmelemente

- Verzicht auf explizite Programmsteuerung (Schleifen, bedingte Anweisungen)
- Statt dessen:
  - Funktion
  - Ausdruck
  - Wert
- getypte oder ungetypte Sprachen
  - ungetypte Sprachen: Alle Werte gehören zu einem Typ
  - auch: statisch oder dynamisch getypte Sprachen
- Funktion i.d.R. partiell für Parametertypen
  - formal spezieller Wert $\perp$ für “kein Wert”
- Datentypen

© 2006 Martin v. Löwis

Striktheit

- Auswertungsreihenfolge: Werden für eine Funktion erst die Argumente ausgewertet und dann die Funktionsdefinition angewendet?
  - nicht-strikte Sprachen: Auswertungsreihenfolge ist nicht festgelegt; Funktionsdefinition wird u.U. angewendet, ohne alle Argumente berechnet zu haben
- Beispiel: (6*5 < 20) & (8/4 < 2)
  - Zur Ermittlung des Gesamtergebnisses (F) reicht die Ermittlung des ersten Arguments von &
- Bewertungsstrategien (*evaluation strategies*):
  - Regeln, wann Argumente ausgewertet werden
  - z.B. parallel Auswertung, *lazy evaluation*, ...

© 2006 Martin v. Löwis

Striktheit (2)

- Striktheit und ⊥:
  - strikte Auswertung: Falls ein Argument ⊥ ist, ist das Ergebnis ⊥
  - nicht-strikte Auswertung: Falls ein Argument ⊥ ist, kann das Ergebnis trotzdem wohl-definiert sein
    - Beispiel: (20 > 5) | (10/0 < 7) kann als T ausgewertet werden, falls Definition lautet:
      X & Y ist T falls X den Wert T hat oder Y den Wert T hat
- Striktheit und Terminierung
  - falls die Auswertung eines Arguments nicht terminiert, terminiert bei strikter Auswertung auch das Ergebnis nicht
  - bei nicht-strikter Auswertung hängt es von der Strategie ab
  - Nicht-terminierende Berechnungen werden oft als gleich zu ⊥ betrachtet

© 2006 Martin v. Löwis

LISP

- *LIst Processing Language*
- Erfunden 1958 von McCarthy am MIT
  - CACM, “Recursive Functions of Symbolic Epressions and Their Computation by Machine”
- Erste Implementierung auf IBM 704
  - 36-bit Rechner (erster Rechner mit Gleitkommahardware)
  - Hauptspeicher auf Basis von Ferritkernen (*core memory*)
  - 40 000 Anweisungen pro Sekunde
  - zwischen 1955 und 1960 wurden 123 Geräte verkauft
- viele Sprachversionen
- 1994 Definition von ANSI Common Lisp (CL)

© 2006 Martin v. Löwis

Common Lisp

- strike Sprache
- Syntax basiert auf S-Ausdrücken (*S-expressions*)
- feste Menge vordefinierter Datentypen
  - Skalare: Zahlen (ganze, Gleitkomma, ...), Zeichen, Symbole
  - Folgen: Listen, Vektoren, Strings, ...
  - Funktionen
  - ...
- große Zahl vordefinierter Funktionen
- imperative Konstrukte (keine “reine” funktionale Sprache)

© 2006 Martin v. Löwis

S-Ausdrücke

- “symbolic expressions”
- textuelle Notation für Datenstrukturen
- Zwei Formen:
  - ATOM
  - (sexp . sexp)
  - Teile heißen car und cdr
- Atome: Zahlen, Symbole, ...
  - Symbole: Namen, ohne Unterscheidung von Groß- und Kleinschreibung
  - Spezielles Symbol: NIL
- Beispiel: (1 . 2)
- Beispiel: (a . (b . (c . NIL)))
  - Kurzform falls “letzter” cdr NIL ist: (a b c)

© 2006 Martin v. Löwis

GNU Common Lisp

- entwickelt von Bruno Haible
  - clisp.cons.org
- Interaktiver Modus: clisp
  - read-eval-print loop (*repl*)
- Interpreter-Modus: clisp *quelldatei*
  - -repl geht nach Einlesen/Ausführung in den interaktiven Modus
  - -x <ausdruck>: Auswertung und Ausgabe von Ausdrücken
  - Dateiendung für Quelldateien: .lisp
  - Hilfe: :h
  - interaktiver Debugger
- Compiler-Modus: clisp -c *quelldatei*
  - Ergebnis: *dateiname*.fas (clisp-Bytecode)

© 2006 Martin v. Löwis

Ausdrücke

- Syntax: (funktion argumente)
  - (+ 3 4)
  - (+ 10 6 3 9 5)
  - (> 20 5)
    - Logische Wahrheitswerte: T und NIL
  - (max 6 3 9 10 -4)
  - (print (/ 100 3))
    - print ist Funktion mit Seiteneffekt
- Bedingter Operator: Funktion if (nicht strikt)
  - (if (> 10 4)
    (+ 3 5)
    (- 9 2))
  - Standardwert für den else-Fall: NIL

© 2006 Martin v. Löwis

Funktionen

- Funktion “defun” definiert neue Funktionen
  - (defun *name* (*parameter*) *koerper*)

```
(defun fib (n)
  (if (< n 2)
      1
      (+ (fib (- n 1)) (fib (- n 2)))))
```

© 2006 Martin v. Löwis

Variablen

- keine Variablen im eigentlichen Sinn
  - Zuweisung an Variable ist Seiteneffekt, würde funktionales Prinzip verletzen
    - Common Lisp bietet zusätzlich auch “richtige” Variablen (setq)
- Namen (Symbole) erhalten Werte bei ihrer Definition; Bindung des Namens nachträglich nicht änderbar
  - Funktionsparameter erhalten Wert bei Funktionsruf
  - let-Konstrukt hilft, weitere Symbole als Abkürzungen einzuführen:
  - (let ((a (fib 10)) (b 8))
    (+ a b))

© 2006 Martin v. Löwis

Listen

- Bestehend aus cons-Zellen (car . cdr)
  - cdr ist wieder Liste
  - Ende der Liste: spezieller Wert NIL
- Zahlreiche vordefinierte Funktionen
  - (first L) liefert erstes Element
    - ehemals: (car L)
  - (rest L) liefert Restliste
    - ehemals: (cdr L)
  - (nth N L) liefert N-tes Element (Zählung beginnt mit 0)
    - Spezialfälle: second, third, ... ninth
  - (length L) liefert Länge der Liste
  - (last L Z) liefert letzte Z Elemente (Defaultwert für Z: 1)
  - (cons V L) liefert Liste, die mit V anfängt und mit L
  - (append L1 L2) liefert Liste, die alle Elemente aus L1 und L2 enthält

© 2006 Martin v. Löwis

quote

- (x y z) bedeutet i.d.R.: rufe Funktion x mit Argumenten y und z
  - x muss Funktionsname sein (oder an Funktion gebundenes Symbol)
  - y und z müssen Symbole sein, die an Werte gebunden sind
- quote: Verwendung von Listen und Symbolen “direkt”
  - (quote (1 2 3))
  - (quote (x y z))
- Kurzform: ‘(x y z)
- Auch für Symbole: ‘x ist Symbol X, nicht Wert von X
  - (list ‘A ‘(B C) (+ 2 3))
- Zugriff auf Funktionsobjecte: (function *funktion*)
  - Kurzform: #’funktion

© 2006 Martin v. Löwis

Prädikate

- Funktionen, die t oder nil liefern
- Konvention: Funktionsname endet mit p
  - Falls Bindestriche im Funktionsnamen vorkommen, mit -p
- Typtests: listp, numberp, integerp, floatp
  - Allgemeiner Typtest: typep:
  - (typep ‘(a b c) ‘integer)
- Tests für Zahlen: evenp, oddp, zerop, plusp
- Tests für Zeichen: lower-case-p, digit-char-p, char-less-p
- Interaktives Prädikat: y-or-n-p

© 2006 Martin v. Löwis

Strukturen

- Typen mit einer festen Menge benannter Felder
- Definition mithilfe von defstruct
  - (defstruct point x y z)
  - Optional: Initialwerte für Felder, Funktion für Textrepräsentation, ...
- Strukturdefinition definiert implizite Funktionen
- Konstruktor: (make-*struktur werte*)
  - Liste aus Feldnamen und -werten
  - (make-point :x 10 :y 7 :z 4)
- Zugriffsfunktionen: (*struktur-feld wert*)
  - (point-z p) ; p muss point sein
- Typtest: *struktur*-p

© 2006 Martin v. Löwis

Funktionen höherer Ordnung

- Funktionen als Parameter von Funktionen
- Beispiel: Gegeben sei Funktion
  - (defun verdoppeln (n) (* 2 n))
- Anwenden der Funktion auf alle Elemente einer Liste:
  - (map1 (function verdoppeln) '(3 5 10))
- Definition der Funktion map1:

```
(defun map1 (f L)
      (if L
            (cons (apply f (first L)) (map1 f (rest L))
            ())))
```

- Funktion apply: (apply *f argumente*)
  - Aufruf von Funktion f mit angegebenen Argumenten

© 2006 Martin v. Löwis

Vordefinierte Funktionen höherer Ordnung

- (mapcar *funktion argumentlisten*)
  - Anwendung der funktion auf alle Elemente der Argumentlisten
  - (mapcar #'+ '(1 2 3) '(4 5 6))
- (some *predicate argumentlisten*)
  - Anwendung des Prädikats auf alle Elemente, bis Prädikat T ergibt
    - falls alle Rufe des Prädikats nil ergeben, liefert some nil
  - (some #'evenp '(1 2 4 8 16 17))
- (every *predicate argumentlisten*)
  - Anwendung des Prädikats auf alle Elemente, bis Prädikat nil ergibt
    - falls kein Ruf nil ergibt, liefert every T
  - (every #'> '(10 20 30) '(1 2 3))
- (find-if *predicate liste*)
  - Liefert erstes Element der Liste, für das das Prädikat erfüllt ist
  - Optionale Argumente :from-end, :start, :end, ...

© 2006 Martin v. Löwis

Lambda-Ausdrücke

- Anonyme Funktionen
- (lambda (*argumente*) *ausdruck*)
  - In Common Lisp nur in der Form #'(lambda ...) erlaubt
- Beispiel:
  - (mapcar #'(lambda (n) (* n 2)) '(1 2 3))

© 2006 Martin v. Löwis

Imperative Konzepte in Lisp

- Seiteneffekte
  - Globale Variablen
  - Variablenzuweisungen
  - Änderung von Datenstrukturen
  - Ein/Ausgabe
- Kontrollfluss
  - Sequentielle Ausführung (progn, let, defun, ...)
  - Schleifen (loop, do, dolist, ...

© 2006 Martin v. Löwis

Globale Variablen

- (defvar *variablenname initialwert dokumentation*)
  - Initialwert und Dokumentation sind optional
  - (defvar passwd)
- Zuweisung: (set *variablenname wert*)
  - Variablenname muss quoted sein:
    - (set (quote passwd) (read-passwd “/etc/passwd”))
- Kurzform: setq
  - (setq passwd (read-passwd “/etc/passwd”))

© 2006 Martin v. Löwis

Ein-/Ausgabe

- Öffnen von Dateien: (open “dateiname”)
- Automatisches Schließen am Ende der Bearbeitung: with-open-file
  - (with-open-file (*variable dateiname options*) *verarbeitung*)
  - options: :direction *dir,* :if-exists *action*
  - dir: :input, :output, :io (Standard :input)
  - action: :error, :rename, :overwrite, :append, ...

© 2006 Martin v. Löwis

Weitere Konzepte

- *reader* und *printer*
- Makros, Trennung in Compile-Zeitpunkt, Lade-Zeitpunkt, und Auswertungszeitpunkt
- *arrays*
- *association lists*
- *hashtables*
- *conditions* und Fehlerbehandlung
- CLOS: Common Lisp Object System

© 2006 Martin v. Löwis