Grundlagen wissenschaftlichen Arbeitens

COMMON LISP

Jürgen Prömer, Matr.Nr.: 0426423
Nr. 9, Gruppe E, Leiter: Prof. Franz Puntigam
WS 2004

Abstract

Zunächst gebe ich einen kurzen historischen Überblick über die Programmiersprache Lisp und führe die Gründe an, die zur Entwicklung und Implementierung von Common Lisp führten. Danach folgt eine kurze Einführung in die Syntax und Semantik von Common Lisp, mit einigen Beispielen.
Im nächsten Kapitel weise ich auf Besonderheiten der Programmiersprache Lisp selbst im Vergleich zu anderen Sprachen hin. In der Zusammenfassung liste ich Stärken, Schwächen und Anwendungsgebiete von Lisp auf.

Motivation zur Entwicklung der Sprache

Lisp wurde 1959 von John McCarthy am MIT (Massachussetts Institute of Technology) entwickelt und ist eine der ältesten sogenannten “High-Level” Programmiersprachen, die auch heute noch verwendet wird (nur Fortran ist um 1 Jahr älter). Seine Erweiterbarkeit erlaubt es ihr mit den Entwicklungen, die in Sprach-Design, in strukturierter Programmierung und in der Softwaretechnik gemacht werden, schrittzuhalten.

Common Lisp wurde im Jahr 1981 vom DARPA-Konsortium aus der Sprache LISP entwickelt und implementiert. Es ist ein Abkömmling der MACLISP Familie der Lisp-Dialekte.
Hauptgrund war, dass sich die einzelnen Dialekte von Lisp, die in den Jahren zuvor entwickelt wurden, sehr stark unterschieden. Es musste ein Standard geschaffen werden.
Andere moderne Dialekte von Lisp sind zum Beispiel Scheme, LeLisp oder EuLisp.

Mit der Entwicklung von Common Lisp wurde versucht, einen Sprachstandard zu setzen, der sich dann auch in den nächsten Jahren – vor allem in den USA – weitgehend durchsetzte und schliesslich auch in Europa und Japan immer grössere Akzeptanz gewann.

Ende der 80er Jahre begannen eine amerikanische ANSI-Kommission, die europäische EuLISP-Kommission, eine französische AFNOR-Kommission und eine deutsche DIN-Kommission Vorschläge zu einer weiteren Standardisierung von LISP auf der Basis von Common Lisp zu erarbeiten, die schliesslich im Dezember 1994 zu der Verabschiedung eines ANSI-LISP Standards führten. Es ist zu erwarten, dass dieser Standard in den kommenden Jahren die Grundlage für künftige Lisp-Dialekte sein wird.

Das unbestrittene Standard-Nachschlagewerk für Common Lisp, das aufgrund seines Umfangs allerdings nur von denen sinnvoll genutzt werden kann, die schon über grundlegende Lisp-Kenntnisse vefügen, ist:
Guy L. Steele, Common Lisp. The Language., Digital Press, 2. AuAE. 1990.
Er gab im Jahre 1982 einen Überblick über Common Lisp auf dem “1982 ACM Symposium on LISP and Functional Programming”.

Kurzüberblick über Syntax und Semantik mit Beispielen

(Die linke runde Klammer ist der Beginn einer Liste von Objekten. Diese Liste kann eine beliebige Anzahl von Objekten enthalten (auch 0 Objekte) und auch verschachtelt sein.
Zum Beispiel, (cons (car x) (cdr y)) ist eine Liste von 3 Objekten, von denen die beiden letzten wieder Listen sind.

) Die rechte runde Klammer beendet eine Liste von Objekten.

' Das einfache Anführungszeichen gefolgt von einem Ausdruck form ist eine Abkürzung für (quote form).
Folglich bedeutet 'foo (quote foo) und '(cons 'a 'b) bedeutet (quote (cons (quote a) (quote b))).

; Der Strichpunkt ist das Zeichen für ein Kommentar. Der Strichpunkt selbst und alle Zeichen bis zum Ende der Zeile werden nicht berücksichtigt.

“ Doppelte Anführungszeichen umgeben Zeichenketten (character strings): “This is a thirty-nine-character string.”

\ Backslash is das sogenannte Fluchtsymbol. Es hat zur Folge, dass das folgende Zeichen behandelt wird, wie für seinen üblichen syntaktischen Zweck.
Zum Beispiel, A\(B bezeichnet eine Zeichenkette, die aus den 3 Zeichen A, (und B besteht. Ebenso, “\”” bezeichnet eine Zeichenkette bestehend aus einem Zeichen, nämlich dem doppelten Anführungszeichen. Das Backslash bewirkt, dass das zweite Anführungszeichen wörtlich genommen wird und verhindert, dass es als Ende der Zeichenkette interpretiert wird.

| Der vertikale Balken wird paarweise verwendet um eine Zeichenkette (oder einen Teil davon), die aus vielen speziellen Zeichen besteht, einzuschließen. Es ist ungefähr das gleiche als wenn man vor jedes Zeichen ein Backslash schreiben würde, das jetzt von vertikalen Balken umschlossen wird.
Zum Beispiel, |A(B)|, A|(|B|)| und A\(B\) bestehen alle aus den Z Zeichen A, (, B und).

Das Rautezeichen signalisiert den Beginn einer komplizierten syntaktischen Struktur. Das Zeichen danach kennzeichnet welche konkrete Syntax folgt.
Zum Beispiel, #o105 heißt 105 in Oktalschreibweise; #x105 heißt 105 in Hexadezimalschreibweise; #b1011 heißt 1011 in Binärschreibweise; #\L bezeichnet ein Objekt für das Zeichen L; #(a b c) bezeichnet einen Vektor mit 3 Elementen a, b und c.
Ein besonders wichtiger Fall ist, dass #'fn heißt (function fn),

analog zu 'form heißt (quote form).

: Der Doppelpunkt zeigt an zu welchem Paket eine Variable gehört. Zum Beispiel, network:reset bedeutet eine Variable reset im Paket network.
Ein anführender Doppelpunkt vor einem Symbol zeigt ein festes Schlüsselwort an.

Variablen (Symbols)

Variablen sind entweder einzelne Buchstaben oder Ketten von Buchstaben, Zahlen und Bindestrichen.

```
> (setq a 5)   ;speichert den Wert einer Zahl in einer Variablen
5
> a   ;nimmt den Wert einer Variable
5
> (+ a 6)   ;verwendet Wert einer Variablen als Argument einer Funktion
11
```

Es existieren 2 spezielle Variablen, **t** und **nil**. t hat immer den festen Wert t und nil den Wert nil, was in Lisp soviel bedeutet wie **true** und **false**.

```
> (if t 5 6)
5
> (if nil 5 6)
6
> (if 4 5 6)
5
```

Variablen mit einem “:” als erstes Zeichen nennt man Schlüsselwörter.

```
> :this-is-a-keyword
:THIS-IS-A-KEYWORD
```

Zahlen (Numbers)

Ganze Zahlen sind Ketten von Ziffern, die optional mit einem – oder + beginnen. Reelle Zahlen haben einen Dezimalpunkt und können auch in wissenschaftlicher Notation geschrieben werden. Rationale Zahlen sehen aus wie 2 ganze Zahlen mit einem “/” dazwischen. Lisp unterstützt auch komplexe Zahlen, die in dieser Form angegeben werden: #c(r i).
Die standardmäßigen arithmetischen Funktionen sind alle vorhanden:
+, -, *, /, floor, ceiling, mod, sin, cos, tan, sqrt, exp, expt usw.

```
> (+ 3 3/4)   ;addiert 2 Zahlen
15/4
> (exp 1)   ;e
2.7182817
> (exp 3)   ;e*e*e
20.085537
> (expt 3 4.2)   ;Exponent nicht mit der Basis e
100.90418
```

Conses

Conses sind Datensätze aus 2 Feldern. Diese Felder werden aus historischen Gründen car und cdr ("contents of address register" und "contents of decrement register") genannt. Sie sind sehr einfach zu verwenden.

```
> (cons 4 5)   ;definiert ein Cons, setzt car auf 4 und cdr auf 5
(4 . 5)
> (cons (cons 4 5) 6)
((4 . 5) . 6)

> (car (cons 4 5))
4
> (cdr (cons 4 5))
5
```

Listen (Lists)

Aus Conses kann man sehr viele Strukturen bauen. Die einfachste ist eine Liste. Man kann sie mit der list Funktion erzeugen.

```
> (list 4 5 6)
(4 5 6)
```

Funktionen (Functions)

Eine Funktion habe wir eben besprochen. Es gibt aber natürlich noch viele andere.

```
> (+ 3 4 5 6)   ;diese Funktion nimmt beliebig viele Argumente
18
> (+ (+ 3 4) (+ (+ 4 5) 6))
22

> (defun foo (x y) (+ x y 5))   ;definiert eine Funktion
FOO
> (foo 5 0)   ;ruft die Funktion auf
10

> (defun fact (x)   ;eine rekursive Funktion
    (if (> x 0)
      (* x (fact (- x 1)))
      1
  ) )
FACT
> (fact 5)
120
```

Ausgabe (Printing)

Einige Funktionen können Ausgaben verursachen. Die einfachste ist print, das sein Argument auf den Bildschirm schreibt und danach den

Wert zurückgibt.

```
> (print 3)
3
3
```

Special forms

Es gibt einige special forms, die wie Funktionen aussehen, aber keine sind. Dazu gehören Kontroll-Konstrukte wie if-Abfragen oder do-Schleifen.
Eine sehr nützliche special form ist die quote form. quote verhindert, dass sein Argument ausgeführt wird.

```
> (setq a 3)
3
> a
3
> (quote a)
A
> 'a   ;'a ist eine Abkürzung für (quote a)
A
```

Besonderheiten

Lisp ist eine Sprache, deren vorrangige Datenstruktur eine Liste mit Symbolen war. Eigentlich sind Programme selbst Listen von Symbolen, denn Lisp stellt sowohl Daten als auch Programme als Listen dar.
Eine einheitliche Darstellung von Daten und Programmen wie diese gibt es nur in Lisp.
In Lisp kann ein Programm ein anderes Programm (eine Liste) genau wie einen Wert erstellen und dann ausführen.

Aufgrund der unterschiedlichen Syntax verhielt sich Lisp ursprünglich wie alle herkömmlichen Sprachen, da die Programmierung eine Reihe von Nebeneffekten produzierte. Lisp besaß jedoch einen konditionalen Aufbau und Funktionen. Die Bedeutung rekursiver Funktionen als Programmierungsmuster wurde mehr und mehr erkannt.
Funktionen ohne Nebeneffekte können einfach entwickelt und nachvollzogen werden.

Sprachen, die sich ausschließlich auf Funktionen und deren Anwendungen verlassen, werden als funktionale Sprachen bezeichnet. Lisp Sprachen spielten eine prominente Rolle als -- unter anderem -- erste funktionale Programmiersprache.

Allerdings war Lisp immer auch eine "praktische" Sprache. Heutzutage eine Sprache mit der alle Programmierparadigmen (imperative, funktionale, objektorientierte und auch logische mit Erweiterungen) abgedeckt werden können.

Die Sprache Lisp wurde wurde im Lauf der Zeit verfeinert, genormt, und man einigte sich auf einige der zahlreichen Erweiterungen, die Lisp bis dahin in viele Dialekte aufgeteilt hatten.

ANSI Common Lisp war die erste ANSI standardisierte Objekt-orientierte Programmiersprache.

Zusammenfassung

Die weltweite Verbreitung von Lisp in den Jahren nach 1966, ist durch zwei Tendenzen gekennzeichnet:

1. LISP wird auf immer neuen Rechnertypen implementiert und in neue Umgebungen eingebunden.
2. Es verstärkt sich die Tendenz zur Entwicklung neuer Lisp-Dialekte mit zum Teil sehr unterschiedlichen Merkmalen und damit sehr unterschiedlichen Anwendungsgebieten.

Stärken:

- Einfache Syntax, einfache Semantik, hochentwickeltes Makrosystem, hochentwickeltes Objektsystem
- Ausgereift. Technologisch sehr weit fortgeschritten
- Umfangreiche Standardbibliothek
- Garbage Collection
- Gute Bücher

Schwächen:

- Schlechtes Image (AI, GC, Klammern)
- Weniger Programmierer

Typische Anwendungsgebiete:

- Alles schwierigere und teurere, bei denen Einzelkämpfer noch durchkommen, wie bei der NASA, Banken, Medizin, CAD, ...
- Dynamische Probleme, Robotik, Internet, AI
- Allgemeine Anwendungsentwicklung
- Immer noch ein Tool erster Wahl für "explorative" Programmierung. (Explorativ meint um neue Ideen auszuprobieren oder überhaupt erst mal eine Lösung finden)
- Spiele Programmierung
- Nicht zu vergessen: emacs, xemacs.

Literaturliste

Guy L. Steele jr., “An overview of Common Lisp”
Guy L. Steele jr., “Common Lisp. The Language 2nd edition.”
Guy L. Steele jr., Richard P. Gabriel, “The Evolution of Lisp”
Geoffrey J. Gordon, “Common Lisp Hints”
David B. Lamkins, “Successful Lisp: How to Understand and Use Common Lisp”
Horst Zuse, “Geschichte der Programmiersprachen”