Common LISP - Eine allgemeine Übersicht

Martin Elshuber*

24. November 2004, Wien

Abstract

Common LISP ist eine sehr vielseitige Programmiersprache. Dieser Artikel soll einen kurzen Überblick über Common LISP liefern. Zuerst gebe ich einen Überblick über die Geschichte von Common Lisp. Als nächsten Punkt beschreibe ich die Syntax von Common LISP sowie die wichtigsten Befehle. Abschließend werden die Besonderheiten von Common LISP beschrieben, sowie einige Anwendungsbeispiele erwähnt.

1 Geschichte und Motivation

Common LISP entstand in den achtziger Jahren aus Nachfolgern der Programmiersprache LISP. LISP wurde von John McCarthy in den 50 Jahren erstmals definiert. Er wollte eine Sprache definieren die durch folgende Ideen charakterisiert wird: rechnen mit symbolischen Ausdrücken und nicht mit Zahlen, Repräsentation von Symbolischen Ausdrücken und anderen Informationen als Listenstruktur im Speicher eines Computers, eine kleine Anzahl von Selektions- und Konstruktionsoperationen, Kompositionsfunktionen als Werkzeug um komplexere Funktionen zu bilden, Bedingungsausdrücke um Verzweigungen in Funktionsdefinitionen zu erlauben, rekursive Benutzung von Bedingungsausdrücken sollen ausreichend sein um Funktionen berechenbar zu machen, λ-Ausdrücke zur Funktionsbenennung, Repräsentation von LISP-Programmen als LISP-Daten, die LISP-Funktion `EVAL` dient sowohl als formale Definition als auch als Interpreter[3].

Aus LISP haben sich nun viele andere Dialekte entwickelt, von den die größten sicherlich MacLISP und InterLISP waren. Weiters existierten noch Franz LISP und Spice Lisp.

Im April 1981 lud ARPA alle Enwicklungsgruppen von LISP zum "Lisp Community Meeting" ein, um die Zukunft von LISP zu diskutieren. Man kam zum Entschluß daß ein neuer Dialekt "Common LISP" entwickelt werden sollte. Dieser neue Dialekt sollte folgende Ziele erfüllen.

- **Allgemeinheit**: Common LISP soll als allgemeiner Dialekt gelten. In der Vergangenheit führten Unterschiede zwischen einzelnen MacLISP Dialekten oft zu Inkompatibilitäten.
- **Portabilität**: Common LISP sollte keine Features enthalten die nicht einfach auf den meisten Hardwaresystemen implementiert werden können.
- **Konsistenz**: Die meisten LISP Implementationen waren inkonsistent. Das heißt, das der Übersetzer und der Interpreter einem korrekten Code unterschiedliche Semantik zuweisten. In Common LISP soll dieses Problem behoben werden.

*MatNr. 9825286, e-mail: martin@elshuber.com

- **Ausdruckskraft**: Konstrukte die sich in alten Implementationen bewährt haben sollen beibehalten werden und schlechte Konstrukte sollen aus gesiebt werden.
- **Kompatibilität**: Common LISP soll kompatibel sein mit MacLISP und InterLISP etwa in dieser Reihenfolge.
- **Effizienz**: Es sollte möglich sein ein optimierenden Compiler für Common LISP zu schreiben.
- **Power**: Common LISP sollte es ermöglichen Pakete zu erstellen und einzubinden aber keine bereitstellen.
- **Stabilität**: Nachdem Common LISP einmal definiert wurde, sollte es sich nur noch langsam und mit bedächtig ändern

Definitiv keine Ziele sind[2]

- **Grafik**: Grafik Programme tendieren dazu sehr Umgebungs abhängig zu sein. Darum spezifiziert Common LISP keine Grafik Befehle.
- **Multiprocessing**: Obwohl einige geplante Implementationen Hilfsmittel bereitstellen werden, ist Common LISP nicht für Multiprocessing vorgesehen.
- **Objekt orientiertes Programmieren**: Objekt orientiertes Programmieren ist für Common LISP nicht vorgesehen

2 Syntax uns Semantik

2.1 Allgemeines

In den folgenden Absätzen werde ich versuchen die wichtigsten Aspekte der Syntax und Semantik von Common LISP dar zu stellen. Ausführliche Information zur Syntax und Semantik können im Buch **Common Lisp the Language**[5] nachgeschlagen werden.

Obwohl es so aussieht als sei Common LISP nicht Case-Sensitiv ist das nicht ganz korrekt. Intern sind alle Funktionen groß geschrieben definiert (zum Beispiel `CDR`, `CONS`, ...). Man kann sie allerdings auch mit `(cdr '(a b))` aufrufen. Das liegt daran, daß der Interpreter alle Zeichen in Großbuchstaben umwandelt, es sei den sie sind besonders gekennzeichnet. Mehr dazu im Abschnitt Symbole.

Variablen in LISP sind grundstzlich typenlos. Man kann ihnen allerdings Typen zuweisen, was speziell dem Übersetzer hilft besseren Code zu erzeugen.

2.2 Basistypen

Ich werde in den nächsten Absätzen die wichtigsten Datentypen die Common LISP verwendet kurz erläutern. In Common LISP gibt es folgende *Basistypen*.

- `integer`
- `ratio`
- `short-float`
- `single-float`
- `double-float`
- `long-float`
- `float` standard Gleitkommatyp, üblicherweise gleich `single-float`
- `complex`
- `character`

```
;integer Zahlen
0         ;Zero
-0        ;das gleiche
#16rff    ;255
#16rFf    ;255 Common LISP ist
   nicht Case-Sensitive
#12r2a    ;-(2*10^12+11*10^0) = 34
```

```
;ratios
2/3      ;
#2r10/11          ;2/3
;floats
-0.      ;0.0 standard float
2.5s1    ;25.0 shoart-float
;complex
#C(3.6421-2 -2.435d24)   ;complexe
    zahl
;character
#\n      ;Zeichen 'n'
#\SPACE  ;Zeichen ' '
```

2.3 Conses, Listen und S-Expressions

Ein `cons` besteht aus zwei Elementen die `car` und `cdr` genannt werden und wird notiert als `(<car> . <cdr>)` (die beiden Leerzeichen vor und nach dem Punkt sind relevant). Eine Liste wird Rekursiv definiert. Sie ist entweder eine leere Liste `nil` oder ein `cons` dessen `cdr` eine Liste ist. Also ist eine Liste eine Kette von `cons`en mit einem terminierenden `nil`[4]. Eine Liste wird mit einer umkammerten durch Leerzeichen separierte Liste notiert.

```
(4 . 2) ;ein cons mit car=4 und
    cdr=2
(4 . (2 . nil)) ;das gleiche wie
    (4 2) also eine Liste
```

Alle in Common LISP darstellbaren Ausdrücke sind sind S-Expressions. Eine S-Expression ist entweder ein `atom` oder ein `cons` dessen `car` und `cdr` wieder eine S-Expression darstellen.

2.4 Symbole

Symbole dienen in LISP als Namen für Variablen, Funktionen, Makros .... Notiert werden sie einfach durch den Symbolnamen. Soll ein Symbolname ein Sonderzeichen oder einen Kleinbuchstaben enthalten, so ist ein '\' vorzustellen. Weiters kann man ein einen Symbolnamen auch durch zwei '|' umklammern. Das bedeutet das die umklammerten Zeichen genau so bleiben wie sie sind.

```
abcdefg ;Das Symbol mit dem Namen
    "ABCDEFG"
AbCdEfG ;das Gleiche
\abcdefg        ;Das Symbol mit
    dem Namen "aBCDEFG"
|abcdefg|       ;Das Symbol mit
    dem Namen "abcdefg"
```

2.5 Die Symbole NIL und T

Das Symbol `NIL` beschreibt eine Leere Liste. Weiters wird es benutzt um boolsche Ausdrücke zu evaluieren `(= 0 1)` ⇒ `NIL`. Das Symbol `T` ist definiert als `(not nil)` ⇒ `T`

```
nil      ;=>NIL
t        ;=>T
(not nil)        ;=>T
(not 1) ;=>NIL
(= 1 2) ;=>NIL
```

2.6 Programmstruktur

Ein Programm in Common LISP besteht aus einer Folge von `form`en. Ein `form` ist einfach ein Datenobjekt das evaluiert werden soll. `2` ⇒ `2` und `(* 3 4)` ⇒ `12` (⇒ bedeutet evaluiert zu). Symbole werden in Common LISP als Namen für Variablen und ähnliches verwendet. Nachdem man `(setq v 5)` ausgeführt hat evaluiert `v` ⇒ `5`. Ist das Datenobjekt eine Liste so notiert das erste Element einen Funktionsnamen und die Restlichen die Funktionsparameter.

2.7 Wichtige Funktionen

- `quote`: Eine der am häufigsten verwendeten Funktion ist `(quote x)` ⇒ `x`. Diese Funktion evaluiert einfach zu `x` ohne das `x` ausgewertet wird. Da `quote` sehr häufig

verwendet wird kann man es durch `'x` abkürzen.

```
(quote x)                    ;=> x
(setq a 10) '(a 65)          ;=> (a
    65)
(setq a 10) (list a 65) ;=>
   (10 65)
```

- `list [a1 [a2 [...]]]`: Evaluiert nach der Liste die die Parameter als Elemente hat. `(list 1 2 3 4)` ⇒ `(1 2 3 4)`

- `car <cons> / cdr <cons>`: Diese beiden Funktionen liefern das `car` beziehungsweise des `cdr` eines `cons`. Da eine Liste durch eine Verkettung `cons` dargestellt wird evaluiert `(car '(a1 a2 ... an))` mit `a1` und `(cdr '(a1 a2 ... an))` mit `(a2 ... an)`.

- `cons <car> <cdr>`: `cons` evaluiert zum Cons `(<car> . <cdr>)`

- `setq <Symbol> <Ausdruck>`: Bindet das eine Variable an einen Wert. `(setq a 13)` bindet das Symbol `A` an den Ausdruck `13`.

- `defun <Name> (<Parameter>) <Rumpf>`: Definiert eine neu Funktion. Der Name kann ein beliebiges Symbol sein, die Parameter sind eine Liste aus Symbolen. Der Rumpf ist dann eine Folge von `form`en. Wobei das Ergebnis der `form` gleichzeitig das Ergebnis der Funktion ist.

- `if <Bedingung> <Rumpf1> <Rumpf2>`: Wenn die Bedingung ungleich `NIL` ist dann evaluiert dieses Form zu Rumpf 1 ansonsten zu Rumpf2.

- `cond ((<b1> <R1>) ... (<bn> <Rn>))`: Die Bedingungen werden der Reihe nach ausgewertet. und der Erste Zweig der nicht `NIL` ausgeführt.

- `loop <rumpf>`: Führt eine Endlosschleife aus die mit `return` unterbrochen werden kann.

3 Anwendungen

Common LISP wurde in mehreren wirtschaftlich orientierten Projekten erfolgreich angewandt allen voran die Yahoo! Store web-commerce Internetseite. Weiters sind erwähnenswert:

- Mirai: Izware LLC's 2D/3D Graphic Programm

- Xanalys Corp's Investiagtions-Software: Diese Programmlinie wird weltweit von Polizei und Sicherheitsdiensten verwendet.

- ICAD: Eine der führenden Softwarepakte für mechanisches Design

Auch in der open-source Gemeinde wird Common LISP verwendet.

- Maxima: Ein Computer Algebra System

- Compo ist eine Sprache in der man komplexe musikalische Strukturen natürlich beschreiben kann

Die NASA verwendet Common LISP Programme wie zum Beispiel:

- SPIKE: Das Zeitplanungs-System für das Hubble Weltraumteleskop

- Remote Agent: Gewinner der NASA Software of the Year Auszeichnung 1999

4 Besonderheiten

- **typenlose Variablen**: Eine der Besonderheiten von Common LISP ist, daß es typenlose Variablen verwendet. Es ist allerdings möglich eine Variable auf einen

speziellen Typ zu restriktieren. Dies führt im allgemeinen zu einer Performacesteigerung des Programms, speziell wird es dem Übersetzer erleichtert Optimierungen vorzunehmen.

- **komplexe Zahlentypen**: Common Lisp definiert Datentypen wie `bignum` oder `ratio` die es ermöglicht mit komplexen Zahlenformaten effektiv zu rechnen. Ein Programm wie zum Beispiel

```
(defun factorial(x)
 (if (= x 0)
  1
  (* x (factorial (- x 1)))
))
(fac 1000)
```

  stellt für Common LISP kein Problem dar. Auch das exakte rechnen mit Brüchen ist möglich `(+ 1/7 1/3)` ⇒ `10/21`

- **Symbolisches Auswerten**: Dies ist der Grund warum Lisp sehr oft für Anwendungen in der AI verwendet wird.

- **S-Expressions**: Einheitliche Repräsentation aller beliebigen Datentypen sowie auch der Code durch S-Expressions[4]

5 Zusammenfassung

Common LISP durchaus eine Alternative zu den allgemein bekannten Sprachen wie C, Java oder Pascal. Die vielen Klammern stellen für Einsteiger sicher ein großes Problem dar und wenn man Common LISP verwenden will muß man damit rechenen (gerade als Einsteiger) einiges der Debugzeit in Klammerzählen zu investieren.

Dieser Nachteil ist aber auch gleichzeitig ein Vorteil. Dadurch das die Syntax sehr einfach ist - ein Programm ist ja in Wirklichkeit auch eine S-Expression - kann man leichter effizientere Übersetzer und Interpreter zu schreiben. In einem Vergleichstest zwischen LISP, Java, C und C++ schneidet LISP sehr gut ab[1].

Zahlentypen wie `ratio` oder `bignum` machen Common LISP praktikabel für allerlei numerische Rechenaufgaben.

Literatur

[1] Erann Gat. Point of view: Lisp as an alternative to java. *Intelligence*, 11(4):21–24, 2000.

[2] Jr. Guy L. Steele. An overview of common lisp. In *Proceedings of the 1982 ACM symposium on LISP and functional programming*, pages 98–107. ACM Press, 1982.

[3] Jr. Guy L. Steele and Richard P. Gabriel. The evolution of lisp. In *The second ACM SIGPLAN conference on History of programming languages*, pages 231–270. ACM Press, 1993.

[4] John McCarthy. Lisp: A programming system for symbolic manipulations. *Preprints of papers presented at the 14th national meeting of the Association for Computing Machinery*, 1959.

[5] Guy L. Steele. *Common Lisp the Language, 2nd edition.* 1990.