Bedienung CMU Lisp

Abteilung Intelligente Systeme
Leiter: Prof. Dr. E. Lehmann
B. Bohnet, S. Klatt

Die Übungsaufgaben enthalten sowohl Rechner- als auch Papierübungen. Da für die Rechnerübungen außer der Sprache Common Lisp Daten und Übungssoftware benötigt wird, ist eine Durchführung auf den Linux-Rechnern des Grundstudiums-Pools vorgesehen. Dort ist CMU Common Lisp (CMU-CL) installiert. Dieses Blatt erklärt die Bedienung der Arbeitsumgebung CMU-CL im Emacs-Editor.
Abkürzungen für Emacs-Tastenkombinationen:

C-<Buchstabe>	Controltaste + <Buchstabe> gleichzeitig
M-<Buchstabe>	Meta-Taste (Alt, oder Esc) + <Buchstabe> nacheinander
Sh-<Buchstabe>	Shift-Taste + <Buchstabe> gleichzeitig

1 CMU-CL Lisp im Emacs

CMU-CL ist eine Common Lisp-Implementierung der Carnegie Mellon Universität. Die Programmierumgebung in CMU-CL besteht aus dem Image (dem Lisp-Interpreter) der in einer Shell bzw. einer Shell im Emacs gestartet werden kann.

Prinzipielle Vorgehensweise :

1. Emacs starten
2. Shell starten mit `M-x shell`
3. Eingabe des Kommandos `lisp`
4. Liegt die Programmdatei bereits vor, mit dem Emacs-Befehl "Open File" [C-x C-f] öffnen, sonst in einem neuen Puffer (nicht dem `Lisp-Shell` Puffer !) den Programmcode erstellen, editieren und abspeichern.

   WICHTIG: nach jeder Änderung am Programmcode diesen speichern. CMU-CL lädt sonst die alte Version der Datei und gibt dann wieder den gleichen Fehler zurück wie vor der Änderung, was meist längeres Suchen nach der Ursache zu Folge hat.
5. Datei in den `Lisp-Shell` Puffer laden : (load ''<dateiname>'')
   In <dateiname> muß zusätzlich auch noch der Pfad angegeben werden, sofern sich die Datei nicht im Hauptverzeichnis befindet.

   WICHTIG : jedes geänderte Programm wieder mit (load ''<dateiname>'') in den CL-Puffer laden, sonst kommt es zu dem gleichen Phänomen wie oben beschrieben.
6. Beim Auftauchen von Fehlern (beim Laden oder beim Ausführen einer Prozedur) Programmcode verändern (im jeweiligen Puffer), abspeichern, laden,... (zur Fehlerbehandlung siehe Kap. 5)
7. Lisp verlassen mit Kommando `(quit)`.
8. Shell verlassen mit Kommando `exit`.
9. Emacs verlassen mit `C-x C-c`.

2 Dateibefehle im Emacs-Lispmodus

Die folgenden Befehle werden in dem Puffer, in dem sich die Programmdatei befindet, ausgeführt. Man muß kein Emacs-Profi sein, um mit CMU-CL arbeiten zu können. Gewisse Tastenkombinationen können sich beim Debugging jedoch als sehr wertvoll erweisen. Hier nun eine Auswahl der wichtigsten :

TAB	Rückt den Code sinnvoll ein. Code sollte **immer** eingerückt werden, denn die Einrückung ist leichter zu prüfen als Klammern zu zählen.
C-g	Quit. Bricht Ausführung des gegenwärtigen Emacs-Befehls ab.
)	Beim Schließen einer Klammer springt der Cursor kurz zur entsprechenden öffnenden zurück. Zur Überprüfung der Klammerung entfernt man die `)` und beobachtet beim erneuten Einfügen den Cursor.

Pufferbefehle

C-x b	zu Puffer wechseln
C-x f	Datei in Puffer laden
C-x s	Puffer in Datei speichern
C-x w	Puffer speichern unter ...
C-x i	Datei an Cursorposition einfügen
C-x 1	Nur ein Puffer – der aktive – wird im Fenster dargestellt
C-x 2	Das Fenster wird horizontal in zwei Hälften geteilt. Oben erscheint der aktive Puffer, unten der in der Pufferliste folgende. Sinnvolle Kombination: Programmdatei und `Lisp-Shell` Puffer

Search / Replacebefehle

C-s <Ausdruck>	<Ausdruck> wird im Puffer gesucht. Suchrichtung : vorwärts. Erneutes Betätigen von C-s führt zum nächsten Vorkommen von <Ausdruck> im Puffer.
C-r <Ausdruck>	wie C-s, aber Suchrichtung rückwärts
M-x replace-string	ersetzt im gesamten Puffer `<Ausdr1>` durch `<Ausdr2>`

3 Befehle im Lisp-Shell Puffer

Mit den CMU-CL-Befehlen kann man im `Lisp-Shell`-Puffer das CMU-CL-Image steuern.

C-c C-c	momentan ausgeführte Funktion abbrechen. Dann unbedingt mit 1 (`abort`) fortfahren. Bei Fortsetzung der Funktionsabarbeitung mit 0 (`continue`) hilft danach nur noch der Notausstieg (Puffer schließen mit `C-x k` und Lisp-Shell Puffer neu anlegen, wie unter Kapitel 1 beschrieben).
C-$\uparrow$, C-$\downarrow$	Tastaturpuffer. Durch C-$\uparrow$ erscheint die letzte Eingabe wieder am Prompt. Mehrmaliges Betätigen führt zur vorletzten,... Eingabe. Mit C-$\downarrow$ kehrt man diese Suchrichtung um.

4 CMU-CL Top-Level Befehle

Die Top-Level-Schleife (read-eval-print-loop) ist die Hauptschnittstelle zwischen Benutzer und CMU-CL-Image. Sie liest die Eingabe des Benutzers, wertet sie aus und zeigt das Ergebnis an.
Treten Fehler bei der Abarbeitung eines Programms auf, geht CMU-CL in den debug-Modus. Dies ist insofern von Bedeutung, als daß man in diesem Modus nur die Befehle zum Stack-Debugging und zum Tracing ausführt und dann mit meistens `0` in den Top-Level zurückkehrt.

Datei- und Verzeichnisbefehle

`(load ''<Dateiname>'')` angegebene Datei(en) laden.

`(compile-file ''<Dateiname>'')` angegebene Datei(en) compilieren. Ergebnis: Dateien gleichen Namens, aber mit dem Suffix '.x86f'. Compilierter Code läuft schneller als interpretierter.

`(quit)` beendet den Lisp-Prozess.

Stack-Debugging-Befehle

Der Stack ist der Aufrufstapel der ausgeführten Funktionen. Bei einem Fehler erkennt man am Stack, welche Funktion beim Abbruch gerade ausgeführt wurden. Diesen Stack kann man durchwandern und lokale Variablenwerte u.ä. überprüfen (bei CMU-CL derzeit ein Mysterium).

Tracing

`(trace` <Funktionsname>`)` Nimmt die Funktion in die Liste der zu protokollierenden Funktionen auf. Jeder Aufruf solch einer Funktion wird am Bildschirm angezeigt.

`(trace)` zeigt die Liste aller zu protokollierenden Funktionen an.

`(untrace` <Funktionsname>`)` entfernt die Funktion aus der Liste. Wird <Funktionsname> nicht angegeben, wird die gesamte Liste gelöscht.

5 Fehler und Warnungen

Beim Laden des Programms

`End-of-File on #<Stream for file ...>`

Tritt dann auf, wenn eine schließende Klammer oder ein Anführungszeichen fehlt und der Lisp-Evaluator diese bis zum Dateiende sucht, aber natürlich nicht findet.
Andere Ladefehler liegen meist daran, daß der Dateiname falsch geschrieben wurde oder die Datei an falschen Stellen die Zeichen "." bzw. "," enthält und sind selbsterklärend.

Während des Programmlaufs

`Warning: Declaring <...> special.`

Geschieht z. B. bei erstmaliger Variablenzuweisung mit `setq`. Am besten Ignorieren.

```
Error : the variable <...> is unbound
Error : the function <...> is undefined
```

Wird meist durch einen Tippfehler im Variablen- bzw. Funktionsnamen ausgelöst.
Die meisten anderen Fehlermeldungen entstehen durch Klammerfehler, z.B.

```
Error: Wrong argument count, wanted 1 and got 2
Error: Illegal function call
```

Deshalb : Klammerung überprüfen und immer sauber einrücken.
Nach der Fehlermeldung erscheint ein Menü, aus dem man bestimmte Aktionen auswählen kann, die den Grund des Fehlers beheben sollen. Die Auswahl der Optionen erfolgt durch Eingabe von `:continue`, gefolgt von der entsprechenden Nummer. In CMU-CL ist meistens nur eine Rückkehr zum Top-Level möglich.