Bedienung Allegro Lisp

Abteilung Intelligente Systeme
Leiter: Prof. Dr. E. Lehmann
M. Löthe, A. Lang

Die Übungsaufgaben enthalten sowohl Rechner- als auch Papierübungen. Da für die Rechnerübungen außer der Sprache Common Lisp Daten und Übsoftware benötigt wird, ist eine Durchführung auf den Rechnern des CIP-Pools (tick, trick und track) vorgesehen. Dort ist Allegro Common Lisp installiert. Dieses Blatt erklärt die Bedienung der Arbeitsumgebung Allegro Common Lisp 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 Allegro Common Lisp im Emacs

Allegro Common Lisp ist eine Common Lisp-Implementierung, der Firma Franz Allegro. Die Programmierumgebung in Allegro CL besteht aus dem Image (dem Lisp-Interpreter) und der Schnittstelle zum Emacs-Editor. Allegro CL benutzt Emacs-Funktionalität, um wichtige Befehle für die Entwicklungsumgebung zur Verfügung zu stellen, wie Laden, Editieren, Debugging, Programmanalyse.
Das Allegro CL Image wird im Emacs gestartet und läuft dann in einem (meist mit *common-lisp* bezeichneten) Emacs-Puffer. Lisp-Dateien werden in einem separaten Emacs-Puffer editiert und dann in den CL-Puffer geladen.
Prinzipielle Vorgehensweise :

1. Emacs starten

2. Im Emacs Allegro CL aufrufen mit M-x allegro und angebotenes Default-Image bestätigen

   Wenn dieser Befehl nicht funktioniert, in der Datei .emacs die Zeile (load "allegro") einfügen.

   Anschließend wieder bei 1. beginnen.

3. Liegt die Programmdatei bereits vor, mit dem Emacs-Befehl "Open File" [C-x C-f] öffnen, sonst in einem neuen Puffer (nicht dem *common-lisp* Puffer !) den Programmcode erstellen, editieren und abspeichern.

   WICHTIG: nach jeder Änderung am Programmcode diesen speichern. Allegro 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.

4. Datei in den *common-lisp* Puffer laden : :ld <dateiname>
   :ld ist ein sog. Allegro-Top-Level-Befehl. Alle diese Befehle müssen mit vorangestelltem ':' (kein Leerzeichen freilassen) aufgerufen werden. In <dateiname> muß zusätzlich auch noch der Pfad angegeben werden, sofern sich die Datei nicht im Hauptverzeichnis befindet.

WICHTIG : jedes geänderte Programm wieder mit `:ld` in den CL-Puffer laden, sonst kommt es zu dem gleichen Phänomen wie oben beschrieben.

5. Beim Auftauchen von Fehlern (beim Laden oder beim Ausführen einer Prozedur) Programmcode verändern (im jeweiligen Puffer), abspeichern, laden,... (zur Fehlerbehandlung siehe Kap. 5)

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 Allegro 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.

Evaluierbefehle
Der angegebene Bereich wird zur Auswertung an Common Lisp übergeben. Dadurch liegt er auch ohne Verwendung des `:ld` Kommandos im CL-Puffer vor.

C-c C-s	den Klammerausdruck *vor* dem Cursor evaluieren
C-c C-b	den gesamten Puffer evaluieren
C-M-x	Funktiondefinition, in der der Cursor ist, evaluieren

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 `*common-lisp*`

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 Allegro-Puffer

Mit den Allegro-Befehlen kann man im `*common-lisp*`-Puffer das Allegro-Image steuern.

C-c C-c	momentan ausgeführte Funktion abbrechen
C-p, C-n	Tastaturpuffer. Durch C-p(revious) erscheint die letzte Eingabe wieder am Prompt. Mehrmaliges Betätigen führt zur vorletzten,... Eingabe. Mit C-n(ext) kehrt man diese Suchrichtung wieder um.
C-c .	Definition finden. Nach Eingabe des Prozedurnamens werden alle Dateien, in denen eine Funktion solchen Namens definiert wird, angezeigt.
C-c ?	zeigt die Parameterliste aller Funktionen an, in denen die Eingabe erscheint

4 Allegro Top-Level Befehle

Die Top-Level-Schleife (read-eval-print-loop) ist die Hauptschnittstelle zwischen Benutzer und Allegro CL-Image. Sie liest die Eingabe des Benutzers, wertet sie aus und zeigt das Ergebnis an. Anstelle von Ausdrücken kann man nach dem Prompt auch Top-Level-Befehle eingeben. Sie beginnen mit einem ':'
Treten Fehler bei der Abarbeitung eines Programms auf, geht Allegro 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 `:reset` in den Top-Level zurückkehrt.

Datei- und Verzeichnisbefehle

`:cd`	change directory
`:cf` <files>	die angegebenen Dateien compilieren. Ergebnis: Dateien gleichen Namens, aber mit dem Suffix '.fasl'. Compilierter Code läuft schneller als interpretierter.
`:cload` <files>	Datei(en) compilieren und dann laden
`:ld` <files>	Datei(en) laden (compilierte Fassung wird vorgezogen, auch wenn sie veraltet ist !)
`:pwd`	momentanes Arbeitsverzeichnis ausgeben
`:exit`	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.

`:zoom`	gibt den oberen Teil des Stack zurück.
`:reset`	kehrt nach einem Fehler wieder zum Top-Level zurück
`:current`	zeigt den Ausdruck an, auf dem der Stack-Pointer gerade steht
`:dn` <Zahl>	bewegt den Stack-Pointer um <Zahl> nach unten bzw. oben.

:up <Zahl>	Wird <Zahl> nicht angegeben, wird 1 angenommen
:local <Variable>	gibt den Wert von <Variable> im momentanen Stack-frame aus

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

```
Warning : <Funktion> has been defined in <Datei1> and is now being defined in
<Datei2>
```

Passiert meistens, wenn man ein geändertes Programm unter neuem Namen abspeichert und dieses dann lädt. Lisp überschreibt die alten Definitionen mit den neuen und gibt für jede obige Warnung aus. Meist ignorierbar.

```
Error: eof encountered on stream #<...>
```

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

```
Error : unbound variable <...>
Error : unbound function <...>
```

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

```
Error: <Funktion> got 1 args, wanted at most 0.
Error: Illegal Function object
Error: Malformed <...> statement
```

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.
Meist nicht sehr hilfreich bei der Fehlersuche, da hier auch noch Optionen von vorangehenden Fehlern angeboten werden. Dagegen ist es recht sinnvoll, sich mit Hilfe der Stack-Debugging-Befehle (siehe 4) die letzten Evaluierschritte anzuschauen, da hier fehlerhafte Berechnungen, Parameterübergaben usw. recht schnell sichtbar werden. Danach kehrt man mit `:reset` wieder in den Top-Level zurück.