Technische Universität München
Institut für Informatik
Prof. Michael Beetz, PhD
Lorenz Mösenlechner, Thomas Rühr

WS 2011/2012
Aufgabenblatt 1
25.10.2011

Übungen zur Vorlesung Praktikum KI-Basierte Robotersteuerung

Functional Programming in Common Lisp

The purpose of this exercise is to familiarize with the Lisp syntax.

In this course, we will use the Lisp compiler SBCL (`http://sbcl.org`). On Ubuntu you can easily install it with `sudo apt-get install sbcl`.

The next step is to install SLIME `http://common-lisp.net/project/slime/`, the Lisp development environment. The easiest way is to make sure that Emacs is installed and to download and extract the file `https://ias.in.tum.de/_media/teaching/ws2011/240952262/slime-repl.tar.bz2` from the course homepage. Finally, `make` has to be called in directory *slime-repl*. By executing `./repl` a Lisp prompt should be started up.

To load a lisp file, execute *(load "filename")*.

If an error occurs, clicking on the "restart" `abort` switches back to the REPL.

1. (a) Which of the following expressions are atoms, a list or none of the two. Why?

   i. ATOM
   ii. (THIS IS AN ATOM)
   iii. (THIS IS AN EXPRESSION)
   iv. ((A B) (C D)) 3 (3)
   v. (list 3)
   vi. (/ (+ 3 1) (- 3 1))
   vii.)(
   viii. ((()))
   ix. ())(
   x. ((ABC

   (b) Which of the following expressions can be evaluated according to Lisp's evaluation rules?

   i. (+ 2 3 4 5 6)
   ii. (+ (+ 1 2) (+ (+ 1 2) (+ 3 6)))
   iii. ((+ 3 4) 7 8 9)

   Please give an explanation when an expression cannot be evaluated.

   (c) How many elements do the following lists have?

   i. (a b c)
   ii. (a b cc)
   iii. (a (b c d) e)
   iv. ((a b c))
   v. ()
   vi. (())
   vii. ((()))
   viii. ((() ()))
   ix. (() ())
   x. (a ((d)))
   xi. ((a b) (c d))

   (d) Transform the following expressions to valid Lisp expressions:

   i. $(25 * 3) - 120 + 68$
   ii. $\frac{93+178-12}{(7-5)*(12+4)}$
   iii. $(1234 + 4321 + 2222)/101$

iv. $(9000 + 900 + 90 + 9) - (5000 + 500 + 50 + 5)$

2. In the following exercise, helper functions can be written. These should be declared locally with *labels* or *flet* if they are used only once.

   (a) Implement a few functions for List processing. All of these functions are already part of Common Lisp. Please don't use the built-in functions but only *first, rest* and *cons* for your solution.

      i. *reverse-list* reverses the list, i.e. generate a list with the same elements in inverse order.

      ii. *append-lists-2* concatenate two lists.

      iii. *append-lists* concatenate an arbitrary number of lists.

      iv. *(delete-from-list* obj list count*)* removes the element *obj* from the list *list* at most *count* times.

      v. Extend the function *delete-from-list* to also support the keywords *:all*, *:first* and *:last* as *count*.

   (b) Lists can also be used to represent sets. In contrast to lists, elements only appear once in a set. Common Lisp already has many functions to work with sets which should *not* be used in this exercise. The following functions should again only be implemented based on the previously defined functions and *first, rest, cons*.

      i. *emptyp* checks if a set is empty.

      ii. *setp* checks if a list is a set.

      iii. *set-add* extends the set by one element, keeping the set property the list.

      iv. *set-union* unifies two sets.

      v. *(set-diff* s1 s2*)* calculates the difference set of *s1* and *s2*, i,e, returns all elements which are in *s1* but not in *s2*.

   (c) Implement a function that calculates the Euclidean distance between two vectors of arbitrary dimension.

$$edist(\vec{v}, \vec{w}) = \sqrt{\sum (v_i - w_i)^2}$$

   (d) Implement the function *occurrences* which returns the number of all elements inside the list, sorted by number of occurrences.

   Example:
   (occurrences '(a b a d a c d c a))
   ⇒ ((A . 4) (C . 2) (D . 2) (B . 1))