Programming Language Design
Mandatory Assignment 1 (of 4)

Torben Mogensen

February 4, 2020

This assignment is *individual*, so you are not allowed to discuss it with other students. All questions should be addressed to teachers and TAs. If you use material from the Internet or books, cite the sources. Plagiarism *will* be reported.

Assignment 1 counts 15% of the grade for the course, but you are required to get at least 33% of the possible score for every assignment, so you can not count on passing by the later assignments only. You are expected to use around 12 hours in total on this assignment, including the resubmission (if any).

The deadline for the assignment is Friday February 14 at 16:00 (4:00 PM). Feedback will be given by your TA no later than February 21. The individual exercises below are given percentages that provide a rough idea how much they count (and how much time you are expected to use on them). These percentages do *not* translate directly to a grade, but will give a rough idea of how much each exercise counts.

We strongly recommend you to resubmit your answer after you have used the feedback to improve it. You can do so until February 26 at 16:00 (4:00 PM). Note that resubmission is made as a separate mandatory assignment on Absalon. If you resubmit, the resubmission is used for grading, otherwise your first submission will be used for grading.

The assignments consists of several exercises, some from the notes and some specified in the text below. You should hand in a single PDF file with your answers. Hand-in is through Absalon. The assignment must be written in English.

The exercises

A1.1) (33%) Download `LISP.zip` from the Code folder on Absalon. Read the document `PLD-LISP.pdf`. Run the example shown at the end of `PLD-LISP.pdf` to check that the interpreter works. You may need to recompile, see `PLD-LISP.pdf` for details.

a. Make a file `assignment1.le` containing the following definitions:

- A function `reverse` that reverses a list.
- A function `sort` that sorts a list of numbers.
- Any number of helper functions that you use to define `rev` and `sort`.

You can assume that `listfunctions.le` has been loaded before your functions are called. You can ensure this by adding the line

```
(load listfunctions)
```

to the start of `assignment1.le`.

Show the contents of `assignment1.le` in your assignment report.

b. Show in the report a LISP-session that shows examples of using your functions – both on empty lists, lists of one element, and lists of at least five elements.

c. Reflect on which elements of PLD-LISP that you found easy to learn and use and which elements that you found difficult. For example, did it take long to learn the syntax? Did you find it easy to express what you wanted in the language?

A1.2) (33%) Solve Exercise 3.3 from the notes.

A1.3) (33%)

If we ignore closures, values in PLD LISP are numbers, symbols, Nil, or pairs. We will now look at how these can be represented in memory:

- A number n is represented as a 64-bit word with the value $4n + 2$ (so the two last bits are 10). This limits the range of numbers to 62-bit integers.
- Nil is represented as a 64-bit value with the value 0, stored at index 0 in the heap. You can assume that only index 0 can have the value 0.
- Index 1 is not used, and can contain any non-zero value.
- A pair `(`a `.` b`)` is represented as a 64-bit value of the following form: 31 bits for the index $i \neq 1$ into the heap where a is stored, followed by 31 bits for the index $j \neq 1$ into the heap where b is stored, followed by two 0-bits. So the word contains the value $2^{33}i + 4j$. Indices must be less than 2^{31} and can not be equal to 1.
- A symbol s is represented as a 64-bit word with the value $4j + 2^{33}$, where j is an index into an array of characters (stored outside the heap). Characters from index j up to (but not including) the next null character form the symbol. Identical symbols point to the same location. Indices are less than 2^{31}. Note that we use that index 1 is unused, so we represent a symbol as a value that would otherwise be a pair where the first element is at index 1.

Note that all values are represented as 64-bit values. The representation is designed so the last bit is always 0. This allows mark-sweep collection to use this bit for marking.

a. Write pseudocode for a function `kind` that given the representation of a value returns 0 if the value is Nil, 1 if the value is a number, 2 if the value is a symbol, and 3 if the value is a pair. Use C-style notation for extracting bits.

b. Write pseudocode for a function `equal` that by looking at their representation determines if two values are equal. Recall that PLD LISP uses deep equality, so two pairs are equal if they are constructed from the same elements, even if these elements are not stored at the same indices in the heap. Use C-style notation for extracting bits.

c. Given that mark-sweep collection is used for reusing heap elements, what forms of fragmentation can occur?

d. Garbage collection in the heap can leave some indices into the symbol array unused. If new symbols can be added (e.g., by reading s-expressions from a file), this can mean that the symbol array can fill up even if there are unused indices. Consider how the mark-sweep collector can be modified to also garbage-collect the symbol array. Note that garbage collection of the symbol array are expected to occur relatively seldom, so you can assume that an unmodified collector is used most of the time, so the overhead of collecting the symbol array is only incurred rarely, so efficiency is not essential.