Programming Language Design
Assignment 1 2024

Torben Mogensen and Hans Hüttel

February 5, 2024

This assignment is a group assignmentwith groups of up to 3 people. Do not seek help from other groups, nor provide them with help.. This will be considered plagiarism. If you use material from the Internet or books, cite the sources. Plagiarism *will* be reported. If you use ChatGPT or similar large language models, you should include the entire dialogue with the AI in your report (otherwise, it is plagiarism), and you should discuss the relevance and correctness of the answers you get (otherwise, you will get no points whatsoever).

Assignment 1 is the first of three mandatory assignments. You are required to get at least 40% of the possible score for every mandatory assignment, so you can not count on passing by the later assignments only. If you know what you are doing, you can solve the assignment in an afternoon, but if you have not or actively participated in the plenary and classroom activities, you should expect to use *considerably* more time.

We advise you not to split the questions between the members of your group. You will get much more benefit if you work on all questions together, and this will prepare you better for the exam assignment.

The deadline for the assignment is **Friday February 16 at 16:00 (4:00 PM)**. The 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).

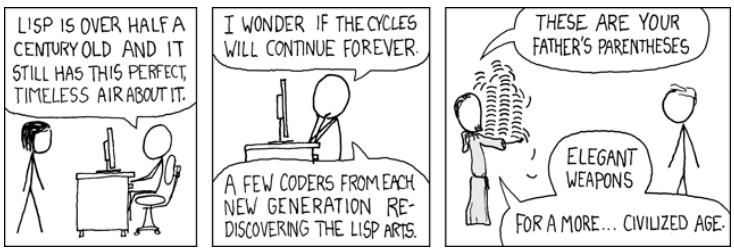
In normal circumstances, feedback will be given by your TA no later than February 23. Note that, even if the TAs find no errors in your submission, this is no guarantee that it is perfect, so we strongly recommend that you take time to improve your submission for resubmission regardless of the feedback you get. You can do so until **March 4 at 16:00 (4:00 PM)**. Note that resubmission is made as a separate mandatory assignment on Absalon.

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 and a zip-file with your code. Hand-in is through Absalon. Your submission must be written in English.

A1.1) (10%) **Lambda calculus (History)**

a. Show reduction of the lambda term $((\lambda x\,.\,(\lambda x\,.\,(x\,.\,x)))\ y)$. Show all steps in the substitution as well as the single reduction step.

A1.2) (50%) **PLD LISP (History and Implementation)**



If you have not already done so, install PLD LISP as explained in `PLD-LISP.pdf`. Then do the following

a. On page 9 in the book, a function called `ff` for finding the leftmost symbol in an S-expression is shown using both M-expression syntax, old LISP S-expression syntax, and Scheme syntax. Write the equivalent function in PLD LISP.

You can assume the argument to `ff` is build entirely from symbols and pairs (i.e., no numbers nor ()). You can use the functions defined in `listfunctions.le` (remember to load this first).

b. By modifying `RunLISP.fsx`, extend PLD LISP with exceptions. An exception throws a value that can be caught by a handler that can do pattern-matching on the value, similar to how the F# try-with expression (`https://learn.microsoft.com/en-us/dotnet/fsharp/language-reference/exception-handling/the-try-with-expression`) can catch an expression thrown by the raise expression (`https://learn.microsoft.com/en-us/dotnet/fsharp/language-reference/exception-handli` `the-raise-function`), though without the `reraise` part.

You can decide on the concrete syntax, but you should use a LISP-like format.

Hints: you can use exceptions in F# to implement exceptions in PLD LISP. You can use a function as argument to the handler instead of implementing pattern matching directly in the handler.

c. In question A1.2)a, we assumed that the input to `ff` is built only from symbols and pairs. Now, we allow both numbers and () in the input. Write, using the exception mechanism you designed in question A1.2)b, write a version of `ff` that can handle this. Show execution of `(ff '(2 4 z 3))` and `(f 7)`.

A1.3) (10%) **Bratman diagrams (Implementation)**

You have the following components at your disposal:

- A machine running ARM machine code
- An interpreter for .NET written in ARM machine code
- An compiler from C# to .NET written in .NET

a. Show Bratman diagrams for these components.

b. Show with diagrams how you, given these components, can compile and run a C# program.

A1.4) (30%) **Syntax**

The textbook has a couple of short comments about the use of indentation on p. 62-63. This problem asks you to dig deeper into this choice of syntax and assess the pros and cons of it in two different guises.

In Fortran, the indentation rules guiding the syntax are exemplified in Figure 1.4 in the textbook. A much later language, namely Haskell, also has special layout rules.

If you try to load a file with the following content into the Haskell interpreter GHCI:

```
fun bingo x = 484000 + k
where k = 0
```

it will tell you the following:

```
Prelude> :load "/Users/hans/Undervisning/DIKU-undervisning/PLD 2024/bingo.hs"
[1 of 1] Compiling Main

/Users/hans/Undervisning/DIKU-undervisning/PLD 2024/bingo.hs:2:1: error:
    parse error on input `where'
  |
2 | where k = 0
  | ^^^^^
Failed, no modules loaded.
Prelude>
```

a. Find out what the layout rules say for Haskell and explain precisely what caused the above error message.

b. Are the rationales behind the indentation rules in Fortran and the layout rules in Haskell the same? If not, what are the essential differences? Is the rationale behind the layout rules in Haskell?

c. Discuss the advantages and disadvantages of having layout rules from a writeability/readability point of view and from an implementation-based point of view.