Programming Language Design
Assignment 4 2022

Torben Mogensen and Hans Hüttel

March 15, 2022

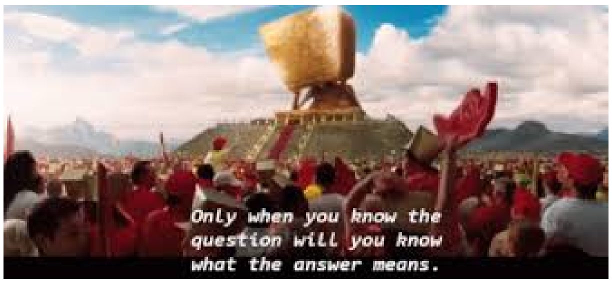
This assignment is *individual*, so you are not allowed to discuss it with other students. All questions should be addressed to teachers and TAs. If you use material from the Internet or books, cite the sources. Plagiarism *will* be reported.

Assignment 4 counts 40% 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 32 hours in total on this assignment. Note that this estimate assumes that you have solved the exercises suggested in slides/videos and participated in the plenary and TA sessions. If not, you should expect to use considerably more time for this assignment.

The deadline for the assignment is **Friday April 1 at 16:00 (4:00 PM)**. 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.

You will get feedback from your lecturers as they grade the assignments.

The assignments consists of several exercises, 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.



A4.1) (20%) **Domain-Specific Languages**

We will look at a specific way of rolling dice and getting a single value from these: You roll 12 6-sided dice, each with faces numbered 1 to 6 (i.e., standard d6s). You add up the values to a single sum. But if the sum divides evenly by 7, you reroll all dice to get a new sum. If that also divides evenly by 7, you reroll again, and so on until you get a sum between 12 and 72 that does not divide evenly by 7.

a. Implement this method in Troll and use this find both the average value of the sum and the probability (in percent) of the sum being greater than 42, and the probability (in percent) of the sum being greater than 50, all three to at least six digits of precision.
b. Implement the same dice roll method in any general-purpose language (GPL) of your choice. The calculations of average value and the probability of rolling more than 42 and 50 should agree with the Troll calculation to at least six digits of precision. Include your code in the zip-file. Show a screenshot of running your code.
c. Compare your GPL-program to the Troll program in terms of effort to write, readability, size of the code, and (approximate) time for execution.

A4.2) (13%) **Limits of Computation**

Consider a simple imperative language with the following syntax

$$
\begin{array}{lcl}
S & \rightarrow & \textbf{var} := E; \\
S & \rightarrow & \texttt{local}\ \textbf{var}\ \texttt{end} \\
S & \rightarrow & \texttt{repeat}\ E\ S\ \texttt{end} \\
S & \rightarrow & S\ S \\
\\
E & \rightarrow & \textbf{num} \\
E & \rightarrow & \textbf{var} \\
E & \rightarrow & E\ \textbf{binop}\ E
\end{array}
$$

S denotes statements that can be assignments to variables, local variable declarations, loops, and statements followed by statements.

A statement of the form `local` y s `end` creates a local variable y that is initialised to 0 and accessible only inside the body statement s.

A loop of the form `repeat` e s evaluates e to a number n and repeats execution of the statement s n times. If $n \leq 0$, s is not executed.

E denotes expressions, which can be number constants, variables, or binary operators applied to expressions. Binary operators are +, -, and *.

A program is a statement and is executed by setting the global variable `x` to the input, executing the statement, and then outputting the value of `x`.

a. Write a program in this language that given input n outputs $n!$ (the factorial of n).

b. A property of a program is whether or not it terminates for all inputs (The Halting Problem). Is this property trivial, decidable or undecidable for this language? Justify your answer.

c. Is the language Turing complete? Justify your answer.

A4.3) (10%) **Scopes**

PLD LISP has both `lambda` (for statically scoped functions) and `lambdaD` (for dynamically scoped functions).

a. Write, using `lambda`, an expression in PLD LISP that evaluates to 42 such that if you replace one or more occurrences of `lambda` by `lambdaD` in the expression it evaluates to 24 instead. No other modifications of the expression is allowed.

A4.4) (12%) **Semantics of Programming Languages**

Let us extend our imperative programming language with a new loop construct of the form `loopenter` x_1 S `loopexit` x_2.

The informal interpretation of `loopenter` x_1 S `loopexit` x_2 is that the loop is entered if the value of x_1 is non-zero. Otherwise, nothing happens. If the loop is entered, the body S is executed. If after this, the value of the variable x_2 is non-zero, the loop terminates. Otherwise, the loop `loopenter` x_1 S `loopexit` x_2 is entered again.

Extend the big-step semantics of the imperative programming language with transition rules that describe this behaviour. The rules must be *structural* like the existing transition rules. That is, they must be syntax-directed and cannot make explicit reference to e.g. while-loops.

A4.5) (15%) **Languages at the Limit**

Consider the loop construct `loopenter` x_1 S `loopexit` x_2 as introduced above and assume that S is a reversible statement as shown in Figure 13.4 in the notes.

a. Draw a flowchart for this loop construct in the style of Figure 13.2 (a) and Figure 13.3 (a) in the notes.

b. What additional restrictions (if any) are required on x_1, S, and x_2 to make the loop construct reversible, and why are they needed? **Hint:** Look at the join points in the flowchart.

c. Draw a flowchart for the modified loop construct in the style of Figure 13.2 (b) and Figure 13.3 (b) in the notes.

d. Given these additional restrictions, what is the inverse of `loopenter` x_1 S `loopexit` x_2? You can use $R(S)$ to denote the inverse of S as in Figure 13.5 in the notes.

A4.6) (15%) **Types (and semantics)**

Let us extend our functional programming language with the construct $\textsf{provide}\ x = e_1\ \textsf{in}\ e_2$. The intention is that x has the value of e_1 in e_2 but *only when* there is no binding of a variable with the same name outside the construct. If there is such an outer binding, we keep that binding when evaluating e_2.

This means that the value of

$$\textsf{provide}\ x = 7\ \textsf{in}\ \textsf{provide}\ x = 8\ \textsf{in}\ x + 15$$

should be 22. On the other hand, the value of

$$\textsf{provide}\ x = 7\ \textsf{in}\ \textsf{provide}\ y = 8\ \textsf{in}\ x + y + 15$$

should be 30.

a) Define a transition rule that extends the transition rules of the functional programming language and captures the informal semantics of the new construct.

b) Define a type rule for the new construct and argue why you have defined it the way you have.

A4.7) (15%) **Logic programming**

In this exercise, it is a good idea to use the list syntax used in SWI-Prolog: this allows us the write lists such as [1,2,3,4] and to have list patterns such as [H|T] where H is the head and T is the tail.

In a political discussion, the prime minister can only say yes and maybe, whereas the leader of the opposition can only say no.

a) Write a collection of Prolog clauses that define this.

b) A political discussion is a list of utterances made by the prime minister and the leader of the opposition. The winner of the discussion is the politician that got the last word.

   For instance, in the political discussion

   ```
   [yes , no , no , no , yes , yes , no , yes , no , maybe]
   ```

   the prime minister won.

   Write a collection of Prolog clauses that define a predicate winner that tells us who ended up winning a political discussion.