Exercises for Computability and Complexity, Spring 2014, Sheet 7

Please return your solutions in the Friday lecture on April 4.

Exercise 1 Define three λ-terms **a**, **b**, **c** and another λ-term **L** such that **Laa** = **Lbb** = **Lcc** = **Lba** = **Lca** = **Lcb** = **false**, and **Lab** = **Lac** = **Lbc** = **true**. (You may think of **L** as a "properly less than" ordering of **a**, **b**, **c**). Hint: use some of the λ-terms from the lecture notes (Booleans, list operators) in the makeup of **a**, **b**, **c** and **L**.

Exercise 2 Design a λ-expression LISTSUM, which applied to a list whose entries are Church numerals returns the sum of the list elements, and returns $\underline{0}$ if the list is empty.

Notice. Exercises 3 and 4 are L5 programming exercises. The L5 interpreter at http://www.csse.monash.edu.au/~lloyd/tildeFP/Lambda/Ch/ (or at the mirror http://www.allisons.org/ll/FP/Lambda/) has a peculiarity that one should know about. If within a `let rec` environment some `funname = lambda x` ... is introduced, and later this `funname` is used, e.g. as in `funname a::b`, then `funname` binds stronger than later bindings, that is, `funname a::b` would be interpreted as `(funname a)::b` and not as `funname (a::b)`. Be generous with brackets!

Exercise 3 Express your solution term from the previous exercise as a L5 program, run it at http://www.csse.monash.edu.au/~lloyd/tildeFP/Lambda/Ch/ on the input list [1, 2, 5], print (or copy manually) the program and the output, and deliver this.

Exercise 4 (optional) Write an L5 program that carries out differentiation of polynomials of the kind $(a_1x^{b_1} + ... + a_nx^{b_n})' \rightarrow a_1\, b_1\, x^{b_1-1} + ... + a_n\, b_nx^{b_n-1}$, where the a_i, b_i are integers (the a_i may be negative). A core part of this exercise is to first design a coding of such terms in a format suitable for L5 – I would suggest a list-based coding. Deliverables: a description of your coding in words, plus a printout of your L5 program and of the result on inputting a (coded version of) $2x^3 - 1 + x^3 - 3x$. The output should represent polynomial terms in ascending order w.r.t. to the exponent, and there should be no multiple instances of terms with the same exponent. Thus, on input $2x^3 - 1 + x^3 - 3x$, the output should be a coded version of $-3 + 9x^2$. *Remark:* the first symbolic math programs for automated algebraic transformations (such as differentiation) were all written in LISP. In general, functional programming languages are perfectly suited to tasks that have a symbolic-transformational, "calculus" flavour – not only mathematical algebra systems, but also, for instance, linguistic grammar processing, theorem proving, or knowledge representation in AI.