Homework set 16: Combinatory reduction — due Monday 9 April

Total number of points available on this homework is 300. Full credit is equivalent to 100 points.

In this homework we look at a way to evaluate λ-expressions efficiently; this is the essence of the implementation strategy for functional programming languages with lazy evaluation, Haskell included. Previously we defined several combinators as terms in the λ-calculus, and looked at examples of reduction of combinators within λ-calculus. It turns out that λ-expressions can be expressed as applications of combinators, and that β-reductions can be broken down into reductions of combinators. To evaluate λ-expressions, one can first translate to combinatory expressions and then use a virtual machine to effect combinatory reduction.

1. (50 pts.) Read the handout on combinators. Augment your λ-calculus code with combinatory expressions over combinator bases $\{\mathbf{S},\mathbf{K},\mathbf{I}\}$ and $\{\mathbf{S},\mathbf{K},\mathbf{I},\mathbf{B},\mathbf{C}\}$. Implement the translation of λ-expressions to combinatory expressions, with and without **B** and **C**.

2. (20 pts.) Test on the translation of the λ-expression for **Y**, as well as a few other λ-expressions.

3. (30 pts.) How is the size of the resultant combinatory expression related to the size of the input λ-expression? Answer should be different for the two combinator bases.

4. (80 pts. extra credit) Implement graph reduction as outlined in the handout. Test on various λ-programs you have already written, including the merge-sort. Report how fast the evaluator works for different inputs or input sizes. *Careful: have you truly achieved the sharing of graph nodes of the kind depicted in the handout? If not, what is the obstacle? What are the repercussions for performance of graph reduction?*

5. (120 pts. extra credit) Implement string reduction (as in Project 1) and graph reduction (as above) using an imperative language from the set {C, Pascal, Java} as your implementation language.