Lab: Deriving recursive programs
CSC 312, “Programming language implementation”
Department of Computer Science · Grinnell College
August 31, 2015

Today’s pair-programming partnerships are:
• Ms. Liddell and Ms. Kyaruzi
• Mr. Mulhall and Mr. Schlager
• Mr. Dewey and Ms. DeWitt
• Mr. Boateng Assante and Mr. Mataire
• Mr. Champagne and Ms. Jervis
• Ms. Byrne and Mr. Nordin

In each pair, the partner listed first should log in and manage the keyboard and screen.

Exercises

A. For each of the pairs of rules in Exercise 1.2 of the textbook (pages 5 and 6), write a concise description in English prose of the set $S$ that that pair of rules defines. (As an example: I would describe the set defined by the rules following Definition 1.1.2 on page 3 of the text as “the set of natural numbers divisible by 3.”)

B. Prove by structural induction that, if $e \in L_{cExp}$, the number of left parentheses in $e$ is equal to the number of right parentheses in $e$. (This is Exercise 1.5 of the textbook, page 12.)

C. In Exercise 1.7, the “more informative” error message that the authors have in mind seems to include the original values of the lst and n parameters — that is, the values supplied by an external caller — even if the error is detected only when the procedure has been invoked recursively, perhaps more than once. This is an instance of a common problem: The interface that makes the recursive procedure simple and efficient is different from the interface that the external caller expects. Show how to accommodate this mismatch by using a named let-expression (or a letrec-expression) to define the recursive “kernel” simply and efficiently while presenting a slightly different interface to the external caller.

D. In developing the mutually recursive procedures subst and subst-in-s-exp, the authors eliminated the Kleene star from the grammar that defines the $S$-list data type, replacing it with two rules, one for empty s-lists and one for non-empty s-lists. In the resulting code, these two rules correspond to separate subexpressions, one (the consequent of the if-expression) specifying how to perform a substitution in an empty s-list, the other (the alternative of that if-expression) specifying how to perform it in a non-empty s-list. But Scheme has a procedure named map that abstracts out this common design pattern and allows the programmer to write code based directly on the Kleene-star definition of the datatype. Use map to write a definition of subst in which the structure of the code reflects the structure of the original definition of $S$-list. (This is exercise 1.13 of the textbook, page 22.)