Functional Programming
Carsten Schürmann
Date: September 16, 2002

Homework 2

Due: Monday, September 23, 2002.

Guidelines

While we acknowledge that beauty is in the eye of the beholder, you should nonetheless strive for elegance in your code. Not every program which runs deserves full credit. Make sure to state invariants in comments which are sometimes implicit in the informal presentation of an exercise. If auxiliary functions are required, describe concisely what they implement. Do not reinvent wheels, and try to make your functions small and easy to understand. Use tasteful layout and avoid long winded and contorted code. None of the problems requires more than a few lines of code.

Exercise 1: Extend our current design of the programming language by a type for lists. Follow the standard steps: First add the type, then the value, then the expressions, then the typing rules, and finally the evaluation rules. Consider both possibilities of making the language eager and making it lazy.

Exercise 2: Prove that your design is correct, by showing that the language preserves types. You can simply take the theorem we have proved in class, and extend it by the necessary new cases. You don't have to extend the substitution lemma.

Exercise 3: In the fragment of the functional programming language we have now finished, program the quicksort algorithm that maps a list of integers to a list of integers. Program the function in both languages, SML and Haskell.

```
quicksort : int list -> int list
quicksort : [Int] -> [Int]
```

Pay close attention to only use the constructs that we have defined in class. State the relevant invariants clearly, and prove that they are maintained during evaluation.