Dan Scarafoni

Prolog week 1 write up.

This project's purpose was to familiarize students with the prolog language. It significantly different from other programming languages, most notably c. It operates entirely on logic and has no control structure (no for or while loops). There is no type casting, either. The language is very heavily based off of recursion

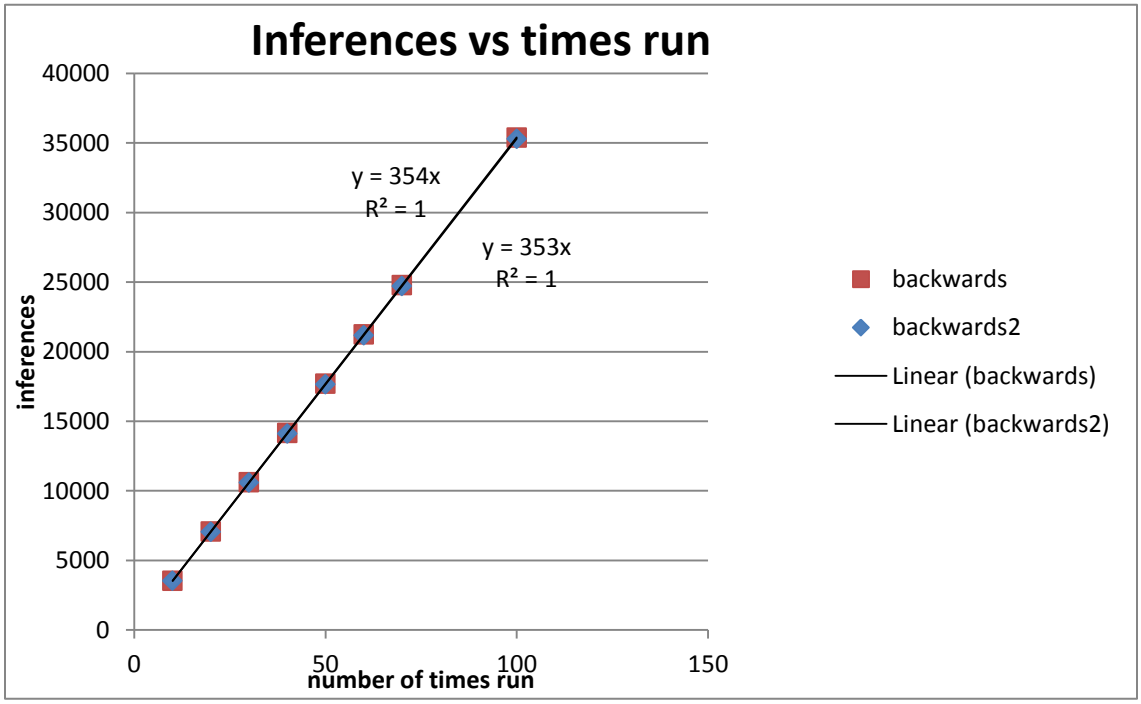
The first part of the project was simply testing and and familiarization with the syntax and basic structure of prolog. The second part involved programming a function that shuffled two lists into one list. The last part of the project involved creating a program that checked whether two lists were reverses of one another (called backwards), copying a function that did the same (backwards2). The runtimes of the two functions were compared.

Backwards2 uses an accumulator to keep track of the current value, which at the end of the program will be returned. It works by placing L1 and L into the first and third arguments of revpaste. Revpaste splits the first argument into the head and tail of the list, and passes the tail as the first argument of a recursive call. The middle value is used to hold the reversed version of L1. The program continues the recursive call until the end condition is reached, where there are no values in the first argument and the second and third arguments are identical. If the second and third arguments are not identical, the program returns false

Backwards, however, calls itself recursively, and works by separating the head from the tail of the list, and appending it to the end using paste. The append function operates in a similar fashion to

the revpaste; however it uses the end value as the accumulator. It continuously adds the head of argument one as the new head for argument 3, and then makes a recursive call.

The two functions were compared in terms of their runtime. Each was timed as to how many inferences were needed to confirm that the first and second arguments were reverses of each other. The following graph shows the results.



The results of this graph indicate that the two programs have almost identical runtime efficiency. The two both take linear time to run, since both can be represented by linear equations with and r^2 or 1 each. It can be inferred that they are each bigO(n), where n is the size of the input. Thus, it can be inferred also that they do approximately the same number of inferences per amount of data entered. Thus the equation mentioned on the instruction page (sum = ½(n^2+3n+2)) most likely holds true for backwards and backwards2.