LIN 177 Final Project Write Up

Shayan Mandegarian

Prolog Model of the Persian Number System

For my final project, the linguistic phenomenon I am modeling in Prolog is the number system of the Persian language. Specifically, all of the spoken form of numbers from 0-1000 in Persian. For example, the number 421 in Persian would be pronounced “chaharsad bist o yek”, and so that would be one of the 1001 outputs from the model, which is produced from the lexicon via a series of concatenations. My main reason for choosing this topic, besides the fact that my first idea was not accepted, is because my own background is Persian, and so this is something related to me personally. Despite being Persian, I have very little knowledge of the language besides very basic formalities, but I have been exposed to it my whole life. So, this topic doubled as a good choice for a prolog model, and a reason to learn more about a part of my heritage. This topic relates more to the topics discussed in the later part of this course, because it is all about how the numbers are put together and formed, rather than anything to do with phones or other more basic elements of linguistics. Because of that, I think it is most comparable to the fourth homework assignment.

The program comes with two prolog files, numbers.pl and project.pl. The numbers.pl file is the file that stores all the lexicon entries required to form the Persian numbers, while the project.pl file contains the predicates that puts those lexicon entries together. My lexicon mainly consists of number facts that contain the parts of the numbers themselves, ie “yek” (which is 1 in Persian), and a list of identifiers that go along with that number. For the most part, each number has only one identifier, as in with the example of “yek” which only has the identifier ‘ones,’ which signifies that yek is a ones place digit and nothing more. However, there are some entries that require multiple identifiers, like with the example of “sad.” “Sad” is not only the Persian form of the number 100, but it also serves as a common root for the rest of the hundreds place numbers except for 200, ie “sisad” which is 300. Thus, “sad” needs to have identifiers that mark it as being a hundreds place number, and being the root for other hundreds place numbers. In addition, like the aforementioned 200 which doesn’t follow the pattern, it needs another identifier that tells the model not to follow the standard pattern of putting “sad” at the end of a hundreds place number, else it would output “sadsad.” To query my project and get the intended output, consult the project.pl file and use the persian/1 predicate. You can either do `?- persian(X).` which will output all possible Persian numbers from 0-1000, or you can replace X with a persian number, ie ‘bist o yek’ (with the quotes), and it will output true if the persian number is correct. In addition, since the persian/1 predicate is built using the persianOne/1, persianTeen/1, persianTen/1, persianHundred/1, and persianThousand/1 predicates, each of which models types of persian numbers that follow distinct patterns, you can query each of those predicates to get specific types of numbers rather than all of them at once.

In my opinion, the greatest challenge I faced during the development of my implementation was the order in which the output is generated. While the output is not random my any means, it does not spit out each number from 0-1000 in the exact numerical order. Rather, because of the fact that some lexicon entries, ie chahar, contain multiple identifiers, the order in which the teens and hundreds are generated seems odd because 14 will be generated before the other teen values, and so on. That is because since chahar is first listed as part of the ones digits numbers, while the rest of the teens numbers come next, but it contains the identifier for being a teen before the rest are reached, prolog will generate it before the others. A similar ordering issue happens in the persianTen and persianHundred predicates, where the “base” tens place and hundreds place numbers, ie 20, 30, … 400, 500, are generated before the compound numbers, ie 24, 405. While it would be possible for me to correct this issue, I believe it would possibly make my approach less principled because I would likely need to add multiple lexicon entries for the same numbers, and handcraft each number individually rather than generating them through a principled manner. The second challenge that I faced was making the form in which the output is generated look nice. While in class before it has been allowed to have output in the form of a list, I decided that I wanted my output to be in the form of single streams, for example like “si o do” rather than [si, o, do]. To accomplish this, rather than using append in my project I use atom_concat, which essentially works the same but uses strings rather than lists. If need be, I could easily switch from strings to lists, but I found strings made the output more readable. I do believe that my project is principled relative to what I learned in class, because it follows a similar pattern as the one in homework assignment 4, in that it contains a lexicon with numbers and their identifiers, and predicates which put them in all of the correct possible orders based on their identifiers.