Maria Volpe

Interfacing with C in OCaml

My project was interfacing with C in OCaml, specifically interfacing the cJSON library, a JSON implementation and parser in C, with OCaml code that builds cJSON and compares it against an OCaml implementation of JSON.

My original project goal was to achieve this by writing the type conversion from C to OCaml myself. The first step was doing research, so I found resources online that had tutorials on how to achieve. My original understanding was that the tutorials I found in OCaml used ctypes, but the tutorials I found that in C would contain information about to implement the type conversion myself. The plan I was going to follow would be that I would learn to use ctypes, and learn about how interfacing works on a low level, and then implement an OCaml program that interfaces with a C library using ctypes. My next step would be to remove ctypes from the program and then replace it with my own functionality. That did not work out, for reasons I will detail later.

My next step was to learn how to use ctypes. There appeared to be an abundance of ctypes tutorials online, however it turned out that all of them were actually the same tutorial, from a chapter from the book *Real World OCaml*. The good news is that the version of the tutorial in the ctypes GitHub repository is written in simpler English and is easier to understand than the other tutorials. I followed that at first, and used my knowledge to interface with the Math and String C standard libraries as practice with the library. Those library interfaces made excellent examples for my presentation, in which my aim was to teach the audience how to use ctypes. Then I started interfacing the cJSON library. Here, I returned to other versions of the

ctypes tutorials, because although they were harder to understand, they had more advanced examples and contained more information, which I needed for my project when I came to the more complicated bits.

Partway through interfacing the cJSON library, I started to think about my original project goals. I had already spent quite a bit of time understanding the ctypes library, and hadn't been thinking about implementing the types myself. I looked back at the tutorials I had in C, and found I was actually mistaken about them providing information on doing the type conversion myself. The tutorials in C actually contained instructions for using the MLValues library, a C library that implements the type translation to OCaml. This was sort of the complement to what I wanted to do, in that it allowed interfacing with C in OCaml code, but the type conversion happened in the C code, unlike in ctypes, where the conversion happens in the OCaml code. I did take a look at how MLValues was implemented, but it was largely unrelated to my project, and also very complicated and hard to read as a person who is not incredibly skilled with C at the moment.

After (finally) reaching out for (much needed) help, I realized it was best to keep my project entirely using ctypes. So the scope of my project became to use ctypes to interface a subset of the cJSON library. I wrote an implementation of JSON in OCaml, as a OCaml variant type, where the JSON type was a list of nodes, and a node was a tuple, styled as a key value pair. The key, the name field, could be a string (denoting it was an object in JSON), or an integer (denoting it was an array in JSON). The value could be of type string, of type float (as all numbers in JavaScript are floats under the hood), of type boolean, the variant type null, or the variant types "child" and "array", both of type json. Although in cJSON both arrays and object

children are simply called children with no differentiation, I differentiated the two to make representing them more intuitive and easier.

Then I set out to use the cJSON library to read from a file using cJSON functions and to translate the resulting cJSON structure into my OCaml JSON type. The short of it is that that worked out horribly because it's extremely hard to deal with getting values out of a cJSON pointer object when working in OCaml, which I discovered midway through the key algorithm involved in the translation. After having a good talk with the professor about what types of programs are best suited for interfacing with C, I instead tried to do the reverse--build OCaml JSON from scratch and translate it into cJSON, and then use cJSON functions on it to print it out. This was substantially easier and what interfacing is well suited for: leveraging side effects in C, where you call C functions in OCaml that produce results that are "stored" entirely behind the curtain of C. The role OCaml plays is essentially pulling levers from the outside, telling C to manipulate that C data, rather than acting directly on the data, because the conversion needed to do that gets messy fast.

What I learned from this is how interfacing with another language works, and why, and about how OCaml works under the hood. In the future I am likely to do more projects with OCaml, and also likely to interface with C again--likely when working with Python, for running calculations that would otherwise be slow in Python.