CMSC631 Project Writeup

Austin Bourgerie

7 December 2017

1 Original Project Goals

My goal in selecting a project was to broaden my formal understanding of type systems. To facilitate that, I selected six chapters from Robert Harper's *Practical Foundations for Programming Languages* to read and for which to complete all the exercises. Four of the six chapters were to be done on paper, and the remaining two were to be mechanized due to the exercises' heavy reliance on proofs. The book offers shallow dips into a wide range of topics in programming languages, making it seem an ideal candidate for fostering a breadth of understanding.

2 Mechanized Proofs in System T

As expected, the exercises presented in the chapter on System T were good candidates for representations as Coq proofs. Although I was able to strengthen my grasp on Coq by building the language model, proving lemmas, and completing the majority of exercises, technical limitations in Coq's type system stopped me from completing the final steps.

2.1 Successes

After a good amount of toying with Coq to arrive at a satisfactory language model, I was able to make solid progress on many of the proof exercises and confront many of the ideas from *Software Foundations* in an augmented context. For example, I had to represent and prove theorems such as progress, preservation, and substitution, along with their auxiliary lemmas, for System T. That process led me to some interesting realizations about the process of proof writing. Phrasing the new theorems in a way that was usable for Coq was more important than ever, as my starting point wasn't an unproved theorem, but no theorem at all. Doing the proving process from scratch helped me to generalize and solidify my understanding of Coq and the intricacies of the properties I set out to prove.

2.2 Shortcomings

The overarching issue that I ran into in completing the mechanized proofs was the presentation of the exercises. Rather than presenting simpler lemmas first then working towards the larger goal of proving termination, the exercises instead lead by asking for a proof of termination, but suffixed the question to ask why the proof was impossible. That cycle went down a few layers until I reached a lemma on hereditary termination, that is the notion that a term is terminating at a value of consistent type. Unfortunately, the way the idea of hereditary termination was phrased introduced a negative type operator in an inductive proposition and pushed me into a technical limitation of Coq itself, which can only represent strictly positive inductive types. Although there is certainly a way to encode the termination property without this lemma, it dead-ended my attempts to follow this chapter's exercises to the end. Despite that, I feel like I learned a lot about the theory and the mechanization process by working through this chapter's exercises.

3 Paper Exercises

For the remaining chapters, my goal was to complete all of the exercises on paper. By and large this process went well and produced answers that I am happy with, with some exceptions.

3.1 Successes

For all of my selected chapters except for System FPC, I either completed all of the exercises or made a single omission (which will be discussed in the next subsection).

The chapter on sum types was the least time consuming, as it is a concept that I was already familiar with and just had to work my way around Harper's particular formalization for. A good portion of the time spent on this chapter was on trying to understand exercise 11.5, which I ultimately decided was a good candidate for omission. The same can be said for System F which, while still interesting to work through, wasn't entirely unknown to me.

The chapter on generic programming dealt with the functors (though not in those words) - the notion of using a type as a template for generalizing the application of a function to many different types is one I have seen before (e.g. OCaml's module system supports functors), but never actually spent the time to understand. This chapter was probably one of my favorites because of how practical it is in general, and how new it was to me. Coincidentally, understanding this chapter led me to understand the limitation in Coq that hampered my termination proof for System T. Namely, admitting type operators that aren't strictly positive is sufficient to implement general recursion, whereas Coq limits its types to inductive types, which encode the notion of "syntactically decreasing" that Coq relies on as an implicit proof of termination for all programs.

As a side note on generic programming, my initial appraisal when I was choosing chapters led me to believe that this chapter would be a good candidate for Coq proofs just based on the language of the exercises, but the proofs were mainly about properties of substitution, which aren't particularly complicated on paper, but would be incredibly painful to do in Coq just due to the nature of working with environments, substitution, and capture avoidance.

The chapter on inductive and coinductive types is definitely the one that caused me the most heartache and, despite the brevity of the ultimate exercise solutions, took by far the most time of any of the paper exercises. The notion of a primitive recursor for inductive types that he presents is ultimately quite simple, but the way that it manipulated the data, that is by using a functor to inductively apply the recursor to recursive points in the type, took a while to understand. The same goes for coinduction, but with respect to the `unfold` operator. This chapter relied heavily on a discussion of greatest and least solutions to type equations, which I was only able to gain a sufficient understanding of by searching through numerous additional resources online; I wish that he had taken the time to discuss what it means to be a solution to a type equation, as that may have made the whole process much more approachable. The answers I did come up with, however, I'm confident in. This one was very difficult for me, but I think the takeaway was a much better understanding of induction, and a general (if rudimentary) sense of coinduction that I didn't have before.

3.2 Omissions

With the exception of System FPC, all of my omissions fell into two distinct categories: RS latches and the SK combinator.

The first RS latch question came up in the chapter on sum types, which led me down a rabbit hole of YouTube videos explaining what an SR latch is (yes, SR not RS - I could hardly find an occurrence of the acronym in his order outside the book itself, but they referred to the same thing). I found this question to be confusing on a few levels - first of all, his definition of an RS latch is simply "a digital circuit with two input signals and two output signals". However, in reality, it has vary particular properties outside of that, and his notion of varying the two inputs independently by time actually leads to nondeterminism in all the models I could find. It was at this point that I realized there were absolutely no occurrences of sum types anywhere in the question, and that I'm not an electrical engineer. Unfortunately, a variation on this same question came up again in the chapter on inductive types, and once again I omitted it there.

The SK combinator question came up in the chapter on System F. It asked for polymorphic types and definitions for `s` and `k` as defined in another exercise for chapter 3. After flipping back to chapter 3, which I read for background information but did not do exercises for, I found some equational semantics for the two combinators but no real description of what they were. I eventually realized that he was referring to a subset of the SKI combinator calculus, but I ultimately decided to omit the question.

3.3 Shortcomings

The major shortcoming for this project was my inability to complete the chapter on System FPC, which is Harper's augmentation of generic programming to admit all type operators, and thereby general recursion with no ad-hoc recursor. When I started the chapter, I realized that three of the four exercises were of the same form as the ones I omitted in previous chapters, which was not a great start. The remaining exercise was to encode the recursor and generator from chapter 15, that is the inductive elimination form and the coinductive introduction form, in terms of general recursion. In the end, I ran out of time and was not able to complete the exercises for the chapter.

When I got to the chapter on inductive and coinductive types, it took me the better part of five long days to wrap my head around Harper's definitions at a level that I could begin to apply his definitions to problems, then actually complete the exercises. Just by virtue of the fact that System FPC came last, underestimating how long the chapter would take left me without a buffer to seek out help or really puzzle it over as I did with induction and coinduction. I also found the chapter's approach to the topic to be really strange - in the six pages dedicated to the topic, more than a page was spent talking about lazy evaluation precluding the existence of inductive types, but there was really no mention of the mechanics of general recursion under the type system. He introduces an entire second syntax for self-referential types separate from recursive types, then relates the two, but never goes further than talking about unrolling a level of recursion, which I didn't find very useful in understanding how to implement general recursion. In the exercises, he explicitly says general recursion is necessary to implement the previously ad-hoc inductive and coinductive constructs in System FPC. I wish he had spent time talking about fixpoint combinators in the context of his definitions to lead towards that exercise. I'm sure I would have been able to line it all up if I'd managed time better at the end of the project, but I also didn't feel like the exercises followed logically from the chapter, which was frustrating.

4 Conclusion

In the end, the purpose of this project was to broaden my understanding of type systems. Although I wasn't able to bring every part of the original proposal to completion, the dozens of hours I spent on the project over the last month helped me solidify my understanding of types, Coq, and even gave me a reason to learn some LaTeX. Most of the challenges I faced during the project had to do with a seeming disconnect between the chapters and exercises, which left me on my own to fill in the blanks; If I were to do the project again, I would choose to do it with a book more suited to deep exploration of types in an introductory capacity, such as *Types and Programming Languages* by Benjamin Pierce.