CS6180: Lecture 6

Robert Constable

September 7, 2017

1 Lecture Summary

The first part of the lecture will discuss the topic of *ancestral logic*, AL [1, 2], an extension of first-order, FOL. AL uses the *transitive closure operator* to provide an abstract form of induction natural to pure logic without introducing the type of natural numbers. It is quite interesting that we can express a rich form of induction without introducing the natural numbers. Dr. Liron Cohen will give this part of the lecture. She is one of the top authorities on AL, and spent last year with us as a Fulbright Fellow and is now a Schmidt Fellow. She has new results on Brouwer's intuitionistic type theory which I hope she will present to us later in the course.

In a second part of the lecture we will begin discussing the abstract notion of a *constructive type*. We will look for the essential characteristics of constructive types and how they relate to types in programming languages and types in Russell's type theory [6, 7]. We will see that types called *universes*, denoted U_i reveal a direct connection to Russell's theory of types.

We will continue to give examples of proofs in *refinement logic*. Refinement proofs are structured as trees using Frege's turnstile symbol, $\vdash$, separating hypotheses H from the goal G to be proved, $H \vdash G$, as we build up the proof. The hypotheses H are arranged as a *list* of labeled formulas and variable declarations, and the goal is a single formula. Such expressions are called *sequents*. In the Nuprl book they are written as $H >> G$. In these notes we use $x_1 : A_1, ..., x_n : A_n \vdash G$. Some formalisms, such as tableaux, allow multiple goals, say $G_1, ..., G_m$, but Nuprl does not.

2 Types

We have already seen the type of natural numbers,$\mathbb{N}$. We have discussed the type as Kleene presents it, where the elements are $0, S(0), S(S(0)), S(S(S(0))), ...$, the *unary numbers*. We might think of these values such as $S(0)$ as the *constants*, this one being the decimal number 1. As mentioned in the previous lecture, it is also sensible to take the natural numbers to be given by their decimal representations as "Big Nums," say 0,1,2,3,4,...99,100,101,102,103,...,199, 200, ..., 999,1000,1001,1002,...,9999,10000, ... These are the numbers that Nuprl uses, and we depend on

highly reliable implementations of them. There are research centers that have taken on the task of creating highly reliable arithmetic packages based on these Big Nums decimal numbers.[1]

To define a type, the first requirement is to choose a standard name, e.g. such as $\mathbb{N}$. Then we must describe the constants or *canonical elements* of the type and say when they are equal. This provides the concrete data for computation on the type. The data format must support the operations that characterize the type, such as addition of natural numbers. These operations give rise to *non canonical expressions*, such as $12 + 5$. There must be computation rules showing how to reduce these non canonical expressions to canonical data, e.g. rules for reducing $12 + 5$ to 17. It would be a good exercise to analyze Kleene's account of the natural numbers and relate it to the standard type-theoretic account. We will do this together in class.

Consider next the type of *ordered pairs* of natural numbers. We name the type this way, $\mathbb{N} \times \mathbb{N}$. The canonical elements are either denoted $pair(n; m)$ or $< n, m >$ for n and m natural numbers. We need operations that select the components of these ordered pairs. Type names for such operators are fst and snd for *first* and *second*. Alternative operator names are $1of$ and $2of$ which select the first and second components respectively. This step leads to *reduction rules* or *computation rules* of the form:

- $fstpair(l; r) = l.$
- $sndpair(l; r) = r.$

References

[1] Liron Cohen. Ancestral Logic and Equivalent Systems. Master's thesis, School of Mathematical Sciences, Tel Aviv University, 2010.

[2] Liron Cohen and Robert Constable. Intuitionistic Ancestral Logic. *Journal of Logic and Computation: exv073v1-exv073*, October 2015.

[3] Thierry Coquand and G. P. Huet. Constructions: A higher order proof system for mechanizing mathematics. In *EUROCAL '85*, volume 203 of *Lecture Notes in Computer Science*. Springer-Verlag, 1985.

[4] Cristina Cornes, Judicaël Courant, Jean-Christophe Filliâtre, Gérard Huet, Pascal Manoury, Christine Paulin-Mohring, César Muñoz, Chetan Murthy, Catherine Parent, Amokrane Saïbi, and Benjamin Werner. The Coq Proof Assistant reference manual. Technical report, INRIA, 1995.

[5] G. Dowek, A. Felty, H. Herbelin, G. Huet, C. Murthy, C. Parent, C. Paulin-Mohring, and B. Werner. *The Coq Proof Assistant User's Guide*. INRIA, Version 5.8, 1993.

[1] I mentioned in lecture that the first versions of Coq used unary numbers, much to our astonishment at Cornell. This limited the examples they could demonstrate in the early Coq systems [3]. Things improved with newer versions of Coq [5, 4].

[6] Bertrand Russell. Mathematical logic as based on a theory of types. *Am. J. Math.*, 30:222–62, 1908.

[7] Bertrand Russell. *The Principles of Mathematics*. Cambridge University Press, Cambridge, 1908.