CS6180: Lecture 8

Robert Constable

September 14, 2017

1 Lecture Summary

In this lecture we will further expand our coverage of the basic types of constructive type theory. Eventually we will add types that are characteristic of Brouwer type theory, the most recent theory implemented by Nuprl. Recall that we have previously stressed in Lecture 6 that to define a type, we need to give it a name, then specify its *canonical members* and the notion of *equality* on them. Other sources of information about these types are the articles from 2006 [1] and these earlier articles about Nuprl [3, 2].

We will use the *refinement style* to present the rules of type theory. Recall that we use Frege's turnstile symbol, $\vdash$, separating the list of hypotheses H from the single goal G to be proved. Sequents, $H \vdash G$, are used to express rules. The goal is a single formula. Some sequent calculi allow multiple goals, but the Nuprl style does not. In the Nuprl book sequents are written as $H >> G$. We use $x_1 : A_1, ..., x_n : A_n \vdash G$.

1.1 Plans for the week

Lecture 7 introduced: *products* (x,&), *function* ($\rightarrow$,$\forall$), *disjoint union* ($+$,$\vee$,$\cup$). In this lecture we will cover *universes* (U_i), *quotient* (A/=), *set types* $\{x : A|B(x)\}$, *intersection* ($\cap$, $A \cap B$), *dependent intersections* ($x : A \cap B(x)$) or $x : A \cap B(x)$. *partial types* $\bar{A}$. We might briefly discuss the older impredicative *recursive types* and *dependent records*.

In Lecture 9 next week we will introduce an approach to *Euclidean Geometry* (**EG**) in constructive type theory. Ariel Kellison, a research assistant in the PRL group, will present Nuprl proofs of three or four propositions and discuss the constructive proof of Euclid Prop 2 that she and Mark Bickford discovered despite claims by other experts that this proposition is not constructively true. We might also try to introduce a *synthetic approach* to the continuum by introducing a representation of the natural numbers $\mathbb{N}$ into EG. Having a representation of the numbers will allow us to express *continuity properties* of Euclidean objects such as lines, circles, and curves.

1.2 Suggested problems and questions

Here is a repetition of suggested exercises from the previous lecture. I would appreciate seeing these by a week from today so that I can provide feedback and see how the class is progressing on basic type theory concepts.

Problem 1: What were Brouwer's reasons for changing the meaning of the logical operations and denying the law of excluded middle? Give an example of the way in which his approach leads to theorems that do not arise naturally in classical logic but which are computationally very interesting. You might use the next problem to accomplish this.

Problem 2: Write an informal constructive proof of some simple theorem about integers or natural numbers and *extract* a functional program from the proof of the theorem, e.g. every positive natural number has a least prime factor. Another good example is from PLCV, if a prime p divides the product of two positive natural numbers, say a and b, then it divides either a or b.

Reading: Please look over the article posted with this lecture: *The Structure of Nuprl's Type Theory.*

2 Types in Constructive Type Theory (CTT) Continued

In Lecture 7 we discussed the critical idea of *canonical elements* of a type and canonical names for types. *It is essential to say when two canonical elements are considered equal.* Some types are abstract such as types for algebraic structures such as groups, rings, and fields. In these cases we do not specify what the equality relation is, but we assume there is an equality on the underlying type of objects that form the algebraic structure. The canonical names with a precise equality provide the basis for computation on the type. The data format must support the operations that characterize the type, such as addition for natural numbers. These operations give rise to *non-canonical expressions*, such as (12 + 5) or (17 x 15) and so forth.

2.1 Universes

One of the key features of Russell's type theory as expressed in *Principia Mathematica* is the notion of a *universe hierarchy,* denoted $\mathbb{U}_i$ for i ranging over the positive natural numbers. The universe hierarchy is cumulative, thus if $i < j$, then $\mathbb{U}_i$ belongs to $\mathbb{U}_j$.

2.2 The void type

The type $Void$ belongs to every universe and has no elements.

If we are in a context where we are assuming that some element v belongs to $Void$, e.g. the hypotheses have the assumption $v : Void$, then we can conclude that $any(v)$ belongs to any well formed type A in any universe.

2.3 Product types

Type constructor: If A and B are types, then $A \times B$ is a type, the product of A and B. The product type is in the *largest of the universes* containing A and B.

Canonical elements: If $a \in A$ and $b \in B$, then $pair(a; b)$ is a canonical element of $A \times B$.

Equality: If $(a = a') \in A$ and $(b = b') \in B$, then $(pair(a, b) = pair(a', b')) \in (A \times B)$.

Reduction rules:

$((fst\ pair(a; b)) \ = \ a) \in A$.

$((snd\ pair(a; b)) \ = \ b) \in B$.

2.4 Dependent products

If we have a family of types indexed by elements of a type A in $\mathbb{U}_i$, that is, for each a in type A, $B(a)$ is a type, say in $\mathbb{U}_j$, then the product $x : A \times B(x)$ is a type in the maximum of the two universes. A pair $pair(a; b)$ belongs to this type if for all $a \in A$, $B(a)$ is a type, and $b \in B(a)$.

This leads to a type that I like to call the "billion dollar data type." This name made sense at the time, in 1999, with Fed chairman Greenspan, was very worried about the Y2K problem. He thought that people would hoard cash, and others had so little confidence in the software industry that he injected a billion dollars into the economy to cope with the anticipated disruptions caused by faulty software. It turns out that there were very few software problems. The dependent product data type could have guaranteed that there were even fewer. It is easy to imagine this type if we name the components, e.g. $c : Century \rightarrow y : Year \rightarrow m : Month \rightarrow Day(c, y, m)$. To implement day correctly requires a great deal of knowledge about how we keep track of the date. Handling the change to the 21st century was not so hard.

2.5 Disjoint union

Type constructor: If A and B are types, then $A + B$ is a type.

Canonical elements: If $a \in A$ then $inl(a) \in A + B$, and if $b \in B$ then $inr(b) \in A + B$.

Reduction rule: If $d \in (A + B)$ and if $u \in A$ implies $lft(u) \in G$ and $v \in B$ implies $rt(v) \in G$, then $decide(d; u.lft(u); v.rt(v)) \in G$.

2.6 Function space

Type constructor: If A and B are types, then $A \to B$ is a type.

Canonical elements: If for any $x \in A$, $b(x) \in B$, then $\lambda(x.b(x)) \in A \to B$.

Reduction rule: If $f \in (A \to B)$ and $a \in A$, then $ap(f; a) \in B$.

2.7 Dependent function space

Type constructor: If A is a type in $\mathbb{U}_i$ and for each a in A, $B(a)$ is a type in $\mathbb{U}_j$, then $x : A \to B(x)$ is a type in the maximum of the universes.

Canonical elements: If for any $x \in A$, $b(x) \in B(x)$, then $\lambda(x.b(x)) \in x : A \to B(x)$.

Reduction rule: If $f \in (x : A \to B(x))$ and $a \in A$, then $ap(f; a) \in B(a)$.

2.8 Set types

Nuprl introduced important new types into constructive type theory. One of them is the set type denoted $\{x : A|B(x)\}$. Intuitively these are the elements of type A that have property $B(x)$, but unlike in the dependent products, the type does not keep the evidence that the property B holds.

2.9 Quotient types

Another new type introduced in Nuprl is the quotient type, denoted $A//R$. This type simply changes the equality relation on its base type A. A good example is to consider that we mean constructively by the set of integers with a designated equality, e.g. the integers modulo 3 for example. What are they in set theory for example? Why is the set theoretic definition no so appropriate for computer science?

2.10 Partial types

Next week we will look at one of the most innovative features of the type theory implemented by Nuprl, namely the partial types. These types include expressions which might diverge when we attempt to evaluate them, e.g. they might run forever. We introduced these in the 1980's [5, 4]. They help in giving the semantics of programming languages. These types have interesting properties and lead us to basic results in recursive function theory.

References

[1] Stuart Allen, Mark Bickford, Robert Constable, Richard Eaton, Christoph Kreitz, Lori Lorigo, and Evan Moran. Innovations in computational type theory using Nuprl. *Journal of Applied Logic*, 4(4):428–469, 2006.

[2] R. L. Constable. The structure of Nuprl's type theory. In Helmut Schwichtenberg, editor, *Logic of Computation*, volume 157 of *Series F: Computer and Systems Sciences*, pages 123–156, Berlin, 1997. NATO Advanced Study Institute, International Summer School held in Marktoberdorf, Germany, July 25–August 6, 1995, Springer.

[3] Robert L. Constable and Jason Hickey. Nuprl's class theory and its applications. In Friedrich L. Bauer and Ralf Steinbrueggen, editors, *Foundations of Secure Computation*, NATO ASI Series, Series F: Computer & System Sciences, pages 91–116. IOS Press, 2000.

[4] Robert L. Constable and Scott F. Smith. Computational foundations of basic recursive function theory. In *Proceedings of the 3rd IEEE Symposium on Logic in Computer Science*, pages 360–371, Edinburgh, UK, 1988. IEEE Computer Society Press. (Cornell TR 88-904).

[5] Robert L. Constable and Scott Fraser Smith. Partial objects in constructive type theory. In D. Gries, editor, *Proceedings of the 2nd IEEE Symposium on Logic in Computer Science*, pages 183–193. IEEE Computer Society Press, June 1987.