Lecture 26 – Constructive Type Theory Continued

Abstract

This lecture is a continuation of Lecture 24 which began our focus on type theory. Here we examine features of the theory implemented by the Nuprl proof assistant, calling it *Constructive Type Theory* (CTT). One of the novel and important features of this theory is that it allows us to provide a constructive semantics for classical first-order logic using the notion of *virtual evidence*.

Another unique feature of this theory is its formalization and implementation of two of the fundamental principles of Brouwer's intuitionistic mathematics, *bar induction* and the *continuity principle*. We will not have time in this course to explore in detail these two fundamental features of *intuitionistic mathematics*,but it is interesting to know that they have now been implemented and are being used in constructive analysis. As far as we know, Nuprl is the only proof assistant that implements these important and very useful principles. In due course the theory might be known as the first implementation of fully intuitionistic mathematics.

1 Background

Constructive Type Theory was formulated at Cornell by members of the PRL group starting in 1980's. The Nuprl system was implemented in 1984, and the book *Implementing Mathematics* was written by several members of the implementation team and published in 1986 [13]. There have been many additions to the theory since then including recent formulations of Brouwer's principle of Bar Induction and his Continuity Principle.

The CTT theory was also implemented and explored at Edinburgh University [7] in the AI Department and in the Potsdam University computer science department in Germany. CTT was also implemented in the MetaPRL proof assistant [21] at CalTech. There is a new effort to implement the theory in the RedPRL system at CMU. There is a semantic model for CTT in the Coq proof assistant [1, 2] which has been used to prove most of the CTT rules correct.

The authors and later contributors shared the goal of seeking a logical foundation suitable for both computer science and mathematics. To some extent this goal is shared by the research groups that implemented the Agda [6], Coq [3], and Nuprl and MetaPRL proof assistants. All four proof assistants implement type theory. They all share fundamental ideas put forward by Dana Scott [29], Per Martin-Löf [23, 24, 25], N. G. de Bruijn [14, 15], Errett Bishop [5], and the author [11, 12].

At the core of CTT is Church's lambda calculus. Church conceived of this as a foundational theory for mathematics. His core idea was very productive [8, 9, 10]. The lambda calculus lies at the core of all of the logical foundations mentioned above. It is also the core of all functional programming languages such as Lisp, SML, OCaml, Haskell, $F^\sharp$. Church also recognized the importance of the type theoretic approach to foundations, which was first presented in work of Bertrand Russell [28, 33].

2 Computational Core

The constructive type theories start with a type free computational core, the idea embodied in Lisp. The notion is that computation rules are used to define and understand types. The canonical forms of a type free programming language are the basic constants of the theory. As we have noted at several points in the course, the canonical forms are those syntactic expressions that are irreducible under the computation rules. For example, $\lambda(x.x)$ is a canonical form for the identity function. The numerical constants, 0,1,-1,2,-2,... are the canonical forms for integers. Pairs of integers, $pair(2,-2)$, are canonical forms. The discriminators such as $inl(0)$ and $inr(-2)$ are canonical forms.

The non-canonical forms can be reduced by computation rules, e.g. 352 x 17 reduces to the canonical form 5,984. The term $spread(pair(352,17); l, r.lxr$ is a non-canonical integer, namely 5,984.

The term $decide(v; l.l + l; r.rxr)$ is non-canonical, and if v is a variable, it does not denote a mathematical value. However, $decide(inr(17); l.l + l; r.r \times r)$ evaluates to the canonical value 289.

3 Type System

How do we introduce types into the theory? Do we decorate the terms with types, as in $\lambda(x^\alpha.x)$? Or are type more like sets, ways of collecting certain canonical forms into a collection? We want them to be more like sets, but there is a major and fundamental difference. In set theory, there is one notion of equality only; two sets are the same if they have the same elements. In type theory, types are collections of canonical values, but the equality depends on the kind of value it is. There is one equality for integers, another one for pairs, another one for functions, another one for disjoint unions and so forth. *So a type is distinguished by its equality relation as well as its elements.*

4 Summary of the constructive semantics of classical logic

In a sense, mathematics has been constructive since Euclid, and so called "classical mathematics" is a 19th century trend, say from 1880 onwards. There are topics in mainstream mathematics that have always been constructive. Also, given a choice, mathematicians generally favor constructive proofs. Edwards [17] gives a modern look at this constructive thread as does Henrici [19] and other articles too numerous to mention. Intuitionistic mathematics is an approach to constructive mathematics developed by L.E.J. Brouwer starting with his doctoral thesis in 1907 and developed over his lifetime in a series of deeply novel and fundamental mathematical results [20, 31, 30].[1] Moreover, Kolmogorov suggested a unification of intuitionistic and classical logics based on his *double negation embedding* of classical logics into intuitionistic ones. Glivenko [18] proved the two versions are equally expressive for propositional logic, and Kuroda [22] extended the result to first-order logic. They used proof theoretic techniques. Murthy [26] implemented this Kolmogorov embedding in Nuprl and used it in his automated formal transformation of a classical proof of Higman's Lemma into a constructive proof. Kolmogorov's approach would also allow mixing of classical and constructive logics in which the classical theorems used double negations to translate the logical operators into constructive versions. We accomplish the translation by semantics means.

In his article about Kolmogorov's work [16], Drago explores the use of *doubly negated sentences* (DNS) to give a constructive meaning to results in scientific papers. He shows that many scientific claims have the form of a DNS.[2] This article provides a semantic explanation for these results using concepts from constructive type theory that were not available until the 1980's.

The Key Idea The new idea presented here involves two steps. The **first step** is to hide (or suppress) the evidence for a proposition P using the *refinement type*. The classical meaning of P is based on the constructive refinement type $\{Unit|P\}$ where $Unit$ is the type with precisely one element, $\star$. To know $\{Unit|P\}$ constructively, one needs to know evidence p for P, but it is not included with the evidence type. We can say that the evidence is discarded or forgotten. It becomes *virtual evidence* or *imaginary evidence* when we talk about $\{Unit|P\}$. Only $\star$ is an element of this type when it is non empty. The evidence p is virtual in this setting. We also call this refinement type *squashed* P. It is clear that there is evidence for $P \Rightarrow \{Unit|P\}$, so we know this implication constructively. However, we do not know $\{Unit|P\} \Rightarrow P$ in general because we are only assuming that there is virtual evidence for P, and even if we had known evidence for P and hidden it, to prove this proposition, we would need to reconstruct that evidence. Clearly we do not have explicit evidence for $\{Unit|P \vee \sim P\} \Rightarrow (P \vee \sim P)$ for all propositions P.

Virtual evidence We will refer frequently to explicit evidence or concrete evidence or computational evidence for the evidence in the *full types*, these are all the types except for refinement types

[1] Brouwer called his work *modern mathematics*, see [30]. Kolmogorov appears to have been the first logician to axiomatize Brouwer's ideas for *intuitionistic* mathematics. Brouwer is also considered one of the founders of topology and is well known for his classical fixed point theorems.

[2] Mark Bickford and I have found similar applications in understanding statements about distributed systems [4].

$\{x : A|B(x)\}$ also called *squashed types*. Part of the evidence for refinement types is virtual, the evidence on the right hand side of the separator, i.e. the evidence that is squashed or hidden.

Another way to explain the meaning of the refinement type $\{Unit|P\}$ is in terms of a general refinement type of the form $\{x : A|B(x)\}$, which consists of those elements a of type A such that evidence b for $B(a)$ is known. Thus an element belongs to $\{Unit|P\}$ if and only if it belongs to $Unit$ and evidence p for P is known. So if constructive evidence p for P is known, then $\star$ belongs to $\{Unit|P\}$. One can interpret this type as "squashing" the evidence p which might be complex, down to a single point, $\star$. The constructive details are hidden, but this can only be done once evidence is known in the first place or built up from assumptions that have this classical character. This "hiding operation" was introduced in "Constructive Mathematics as a Programming Logic I: Some Principles of Theory" [12] and used extensively since. This step does not take us beyond constructive logic, but it introduces the idea that there is a type that is designed to hide evidence. Now we look at properties of this type in preparation for the second step.

If we know $\sim P$ constructively, then we know that $\{Unit|P\}$ is definitely empty – there is nothing to hide. When we know constructively that there is no evidence for $\sim P$, then we know that $\{Unit|\sim\sim P\}$ *cannot be empty.* We also know that it can have only one possible unhidden element, namely $\star$. Inspired by Kolmogorov's observation that the classical axiom for Double Negation Elimination, $\sim\sim P \Rightarrow P$, gives us full classical logic, we naturally consider whether we can make constructive sense of the idea that $\sim\sim P \Rightarrow \{Unit|P\}$. *If we don't actually have evidence for P, we can't constructively inhabit the type $\{Unit|P\}$.* To do so requires evidence for P, even though once we have that evidence, we hide it.[3] We call this hidden evidence *virtual* because we can use $\{Unit|P\}$ *as if we had evidence for P* when we are trying to prove other refinement types. That is how the refinement rules work.

For classical reasoning, constructive evidence for P is not required. Classical reasoners are contemplating the possibility that there might be evidence available in ways yet unknown. Perhaps an *Omniscient Assistant* (OA) knows convincing evidence. Once a deeper understanding is achieved using virtual evidence, it might be possible to find constructive evidence. This has often happened in the past.

In light of the above observations, the **second step** is adding the following new axiom of **Classical Introduction (CI)**:

$$\sim\sim P \Rightarrow \{Unit|P\}.$$

The required constructive evidence to know that $\star$ belongs to this type is knowing specific evidence p for P.[4] We might not have such evidence since we only assume we know $\sim\sim P$. In particular, we do not have it when we consider $\sim\sim (P \vee \sim P) \Rightarrow \{Unit|(P \vee \sim P)\}$ even though we have constructive evidence for $\sim\sim (P \vee \sim P)$.

Our proposal is to forego including evidence for P when realizing the new axiom $\sim\sim P \Rightarrow$

[3] In proof assistants such as Nuprl and MetaPRL, the evidence can be found in the proof object, but it is not part of the refinement type.

[4] We might also want to name this Constructive DNE.

$\{Unit|P\}$. In doing this, we give up evidence semantics, but not entirely. We can justify this rule under the interpretation that propositions justified by virtual evidence cannot be disproved. We can prove in a constructive metatheory such as the one formalized by Anand and Rahli [2] that the augmented theory cannot prove $False$. We know for every rule, that if the hypotheses are not false, then the conclusion is not false. *This approach agrees with the classical notion that evidence is not required for axioms.* It agrees with the constructive notion that axioms should be realizable in that the constant function whose value is $\star$ *is an adequate realizer for the only evidence that must be visible.* The missing evidence is not carried along with the type, and when the type occurs in assumptions, only the virtual evidence is available. That virtual constructive evidence is all that is necessary for classical reasoning. The only explicit evidence needed for computation is $\star$. Since constructively $\{Unit|P\}$ can not be empty, and since it can only have $\star$ as a member, this must be the member. *So this axiom is constructive in the sense that we have not lost the required computational meaning of the type.*

This simple rule narrows the difference between constructive and classical reasoning down to a very small point. Moreover, because our explanation is given in a constructive meta-theory, we preserve a constructive meaning for the rules, albeit for some rules we use virtual evidence knowing that it *cannot lead to constructively false conclusions.* My colleague Mark Bickford notes that this axiom allows us to think of classical mathematics as the minus one slice of constructive mathematics as seen from homotopy type theory [27, 32].

To recapitulate, if there is constructive evidence for $\sim\sim P$, call it nnp, then we can provide a function to give us the *accessible evidence* for $\{Unit|P\}$, namely $\star$. That is all we are allowed to use constructively from the type. The constant function whose value is $\star$ is the realizer for this axiom. *There is no loss of classical content* when we make this positive assertion that we have trivial evidence $\star$ for P. Thus we can provide *computational evidence* for the implication $\sim\sim P \Rightarrow \{Unit|P\}$, that is, evidence for $\sim\sim P \Rightarrow \{P\}$. With this new axiom we can prove:

$$\{Unit|P\} \Leftrightarrow \sim\sim P.$$

The Classical Introduction axiom reveals the essential difference in evidence requirements between classical and constructive logic, and it allows us to conduct classical reasoning with refinement types in a way that can be explained constructively without assuming that we have an oracle that can provide enough evidence to justify the proposition constructively, yet *can provide computational evidence that is sufficient for classical reasoning.* By mixing the classical and constructive modes, we allow a natural style of classical reasoning that supports extracting programs from proofs with some classical steps.

In this context, one way of understanding $\{P\}$ is that by proving this squashed proposition, one knows that it is *classically consistent.* In some cases there is constructive evidence in hand for P. In other cases the constructive status of P might be entirely unknown, yet to be discovered or perhaps forgotten or only a partly scribbled note in the margin or perhaps evidence known only to an Omniscient Assistant that might be revealed one day or even discovered by a proof assistant that knows $\{P\}$ and is set the goal to keep looking for a proof of P. In any case, establishing $\{P\}$, raises the possibility of the stronger result P. Many constructive results have been discovered by

exploiting this possibility.

What we also know is that the classical reasoning mode is semantically consistent with constructive reasoning. This will not be proved here. It can be established in the semantic model of CTT and could thus be formally proved in Coq.

References

[1] A. Anand and V. Rahli. Verifying Nuprl rules in Coq. Technical report, Cornell University, 2013.

[2] Abhishek Anand and Vincent Rahli. Towards a formally verified proof assistant. In *ITP*, pages 27–44, 2014.

[3] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development; Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer-Verlag, 2004.

[4] Mark Bickford and Robert L. Constable. Formal foundations of computer security. In *Formal Logical Methods for System Security and Correctness*, volume 14, pages 29–52, 2008.

[5] E. Bishop. *Foundations of Constructive Analysis*. McGraw Hill, NY, 1967.

[6] Ana Bove, Peter Dybjer, and Ulf Norell. A brief overview of Agda – a functional language with dependent types. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *LNCS 5674, Theorem Proving in Higher Order Logics*, pages 73–78. Springer, 2009.

[7] Alan Bundy, Frank van Harmelen, Christian Horn, and Alan Smaill. The oyster-clam system. In *Proceedings of the 10th International Conference on Automated Deduction*, pages 647–648, London, UK, 1990. Springer-Verlag.

[8] Alonzo Church. A set of postulates for the foundation of logic. *Annals of mathematics, second series*, 33:346–366, 1932.

[9] Alonzo Church. A formulation of the simple theory of types. *The Journal of Symbolic Logic*, 5:55–68, 1940.

[10] Alonzo Church. *The Calculi of Lambda-Conversion*, volume 6 of *Annals of Mathematical Studies*. Princeton University Press, Princeton, 1941.

[11] Robert L. Constable. Constructive mathematics and automatic program writers. In *Proceedings of the IFIP Congress*, pages 229–233. North-Holland, 1971.

[12] Robert L. Constable. Constructive mathematics as a programming logic I: Some principles of theory. In *Annals of Mathematics*, volume 24, pages 21–37. Elsevier Science Publishers, B.V. (North-Holland), 1985. Reprinted from *Topics in the Theory of Computation*, Selected Papers of the International Conference on Foundations of Computation Theory, FCT '83.

[13] Robert L. Constable, Stuart F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, Douglas J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, James T. Sasaki, and Scott F. Smith. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice-Hall, NJ, 1986.

[14] N. G. de Bruijn. A survey of the project Automath. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus, and Formalism*, pages 589–606. Academic Press, 1980.

[15] N. G. de Bruijn. The mathematical vernacular, a language for mathematics with typed sets. In R. P. Nederpelt, J. H. Geuvers, and R. C. De Vrijer, editors, *Selected Papers on Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics*, pages 865–935. Elsevier, Amsterdam, 1994.

[16] Antonio Drago. *A.N. Kolmogorov and the Relevance of the Double Negation Law in Science*, pages 57–81. Polimetrica, International Scientific Publisher, Monza, Italy, 2007.

[17] Harold M. Edwards. Introduction to my book "essays in constructive mathematics". In Thierry Coquand, Henri Lombardi, and Marie-Françoise Roy, editors, *Mathematics, Algorithms, Proofs*, number 05021 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2006. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[18] V. Glivenko. Sur quelques points de la logique de m. brouwer. *Bulletins de la classe des sciences*, 15(2):183–188, 1929.

[19] Peter Henrici. *Applied and Computational Complex Analysis*, volume 1–3. John Wiley and Sons, New York, 1988.

[20] A. Heyting, editor. *L. E. J. Brouwer. Collected Works*, volume 1. North-Holland, Amsterdam, 1975. (see On the foundations of mathematics 11-98.).

[21] Jason Hickey, Aleksey Nogin, Robert L. Constable, Brian E. Aydemir, Eli Barzilay, Yegor Bryukhov, Richard Eaton, Adam Granicz, Alexei Kopylov, Christoph Kreitz, Vladimir N. Krupski, Lori Lorigo, Stephan Schmitt, Carl Witty, and Xin Yu. MetaPRL — A modular logical environment. volume 2758 of *Lecture Notes in Computer Science*, pages 287–303. Springer-Verlag, 2003.

[22] S. Kuroda. Intuitionistische Untersuchungen der formalistchen Logik. *Nagoya Mathematical Journal*, 2:35–47, 1951.

[23] Per Martin-Löf. Constructive mathematics and computer programming. In *Proceedings of the Sixth International Congress for Logic, Methodology, and Philosophy of Science*, pages 153–175, Amsterdam, 1982. North Holland.

[24] Per Martin-Löf. *Intuitionistic Type Theory*. Number 1 in Studies in Proof Theory, Lecture Notes. Bibliopolis, Napoli, 1984.

[25] Per Martin-Löf. An intuitionistic theory of types. In Giovanni Sambin and Jan M. Smith, editors, *Twenty-Five Years of Constructive Type Theory*, volume 36 of *Oxford Logic Guides*, pages 127–172, Oxford, 1998. Clarendon Press.

[26] Chetan Murthy. An evaluation semantics for classical proofs. In *Proceedings of the 6th Symposium on Logic in Computer Science*, pages 96–109, Vrije University, Amsterdam, The Netherlands, July 1991. IEEE Computer Society Press.

[27] Univalent Foundations Program. *Homotopy Type Theory.* Univalent Foundations Program, 2013.

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

[29] D. Scott. Constructive validity. In D. Lacombe M. Laudelt, editor, *Symposium on Automatic Demonstration*, volume 5(3) of *Lecture Notes in Mathematics*, pages 237–275. Springer-Verlag, NY, 1970.

[30] Mark van Atten. *On Brouwer.* Wadsworth Philosophers Series. Thompson/Wadsworth, Toronto, Canada, 2004.

[31] Walter P. van Stigt. *Brouwer's Intuitionism.* North-Holland, Amsterdam, 1990.

[32] Valdimir Voevodsky. Notes on type systems. School of Math, IAS, Princeton, NJ, 2011.

[33] A.N. Whitehead and B. Russell. *Principia Mathematica*, volume 1, 2, 3. Cambridge University Press, 2nd edition, 1925–27.