Review for HPL 2007

Paul Gilmore, **Logicism Renewed: Logical Foundations for Mathematics and Computer Science.** Lecture Notes in Logic, Association of Symbolic Logic. 2005. 250 pages. $69. ISBN 1-56881-275-2.

Reviewed by Fairouz Kamareddine, School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, EH8 4SQ, UK.
E-mail: fairouz@macs.hw.ac.uk

The first chapter introduces the so-called elementary logic (EL), its syntax and semantics and its consistency and completeness proofs. First, the author incrementally defines the types where each stage develops a particular aspect of what the author intends to incorporate. As in Church's Simple Type Theory (STT), types are built from two basic types (roughly, the type of individuals and the type of propositions), yet at this stage, these types remain a reasonably restricted form of Church's types. In the next chapter, these types will be extended further (but will remain a proper subset of Church's types in STT). The syntax of constants, variables, logical connectives, quantification, λ-abstraction, terms and formulae is given as well as the domains, valuations and models. EL plays the role of both a logic and a λ-calculus and hence the syntax and semantics deal with both aspects. Substitution, β- and η-contractions are given as well as the proof theory for EL (based on sequents). The author takes sequents to be of the form $F_1, \ldots F_m \vdash G_1 \ldots G_n$ where $m, n \geq 0$ and the antecedent $\{F_1, \ldots F_m\}$ and the succedent $\{G_1, \ldots G_n\}$ are sets rather than ordered sequences or multisets. Satisfaction and validity of sequents as well as the concept of derivation are defined. For example, $\Gamma \vdash \Theta$ is satisfied by a valuation Φ if $\Phi(F)$ is $-$ (i.e. false) for some $F \in \Gamma$ or $\Phi(F)$ is $+$ (i.e. true) for some $F \in \Theta$. Throughout, one writes $+F$ if $\Phi(F)$ is $+$ and $-F$ if $\Phi(F)$ is $-$ where Φ is a conjectured valuation. The semantic rules for EL include the cut rule $\frac{}{+F \quad -F}$. A number of properties and examples of derivations are given. An alternative proof theory is presented and the author presents a variant ELG of Gentzen's formulation of sequent calculus and shows that ELG and EL are equivalent and that a derivation in EL is an abbreviation of a derivation in ELG. The proof given for the consistency of EL is semantic and prepares the reader for the semantic proof of the consistency of the logic of ITT to be presented later (for ITT, no syntactic proof of the eliminability of cuts is known). Next, the author establishes the completeness of EL without cut and as an immediate corollary (using the consistency of EL), one obtains the cut-eliminability result for EL: Any derivable

sequent in EL, can be derived without any use of the cut rule.

The second chapter introduces the type theory TT where the types given in Chapter 1 are extended to the following:

- 1 is a type and [ ] is a predicate type and a type;
- If $\tau_1, \ldots, \tau_n$ is a sequence of types where $n \geq 1$ then $[\tau_1, \ldots, \tau_n]$ is a predicate type and a type.

Note that these types remain a proper subset of Church's types in his STT. Throughout, $\overline{\tau}$ is used as an abbreviation for $\tau_1, \ldots \tau_n$ where $0 \leq n$. For the benefit of the reader, we include below the inductive definition of terms of TT:

- A constant or a variable cv is a term of type t[cv].
- If $P : [\tau, \overline{\tau}]$ and $Q : \tau$ then $(PQ) : [\overline{\tau}]$.
- If $P[\overline{\tau}]$ and v is a variable of type τ then $(\lambda v.P) : [\tau, \overline{\tau}]$.

As in EL, formulae of TT are terms of type [] but the author gives an alternative definition for formulae in TT which he proves later on in this chapter to be equivalent to the earlier definition. The alternative definition is given in terms of the constants $\downarrow$: [[ ], [ ]], $\exists_\tau : [[\tau]]$ and $=_\tau$: $[\tau, \tau]$ (note also that joint denial $\downarrow$ is the primitive logical connective of EL and $\exists_\tau$ is the primitive quantifier of EL). Next, type variables $\alpha, \beta, \ldots$ are introduced and type expressions are defined using both types and type variables. A unification algorithm which finds the most general unifier of a sequence of pairs of type expressions is given and its correctness is shown. This unification between type expressions is used to give a polytyping of TT which is shown to be correct. The $\beta\eta$-contraction relation is shown to be weak Church-Rosser and it is shown that every term of TT has a unique normal form (modulo the well-known α-renaming of variables). In TT, the semantic rules for $\downarrow$, $\exists$ and cut are (almost) those of EL. The rules for λ are given in terms of the contraction relation and there are new rules for the new constant $=$ which was not part of EL. Other connectives (e.g., $\forall$, $\rightarrow$, etc. are assumed to be built from $\downarrow$ and $\exists$ in the usual way). Next, the author discusses alternatives to the logical constant $=$ and gives two alternative definitions, one as the intensional identity $= \stackrel{df}{=} \lambda x, y.\forall Z.[Z(x) \rightarrow Z(y)]$ and the other as the extensional identity $=_e \stackrel{df}{=} \lambda x, y.\forall \overline{u}[x(\overline{u}) \leftrightarrow y(\overline{u})]$ and shows that the sequent $\vdash \forall x, y.[x = y \rightarrow x =_e y]$ is derivable but that the converse sequent $\vdash \forall x, y.[x =_e y \rightarrow x = y]$ is not. Next, the author defines the extensionality predicate $Ext \stackrel{df}{=} \lambda z.\forall x, y.[x =_e y \rightarrow [z(x) \rightarrow z(y)]]$ and shows that the sequents $\vdash Ext(\exists)$ and $\vdash Ext(Ext)$ are derivable. Finally an extensional semantics of TT is given where all predicates are assumed to be extensional and the author gives a particular extensional model Φ such that $\Phi(\forall x, y.[x =_e y \rightarrow x = y])$ is $+$ concluding that extensional semantics is not the appropriate semantics for TT. Hence, the author gives an intensional semantics of TT and asserts (without a proof) that TT is complete for this semantics. The author also asserts that

the proof of completeness for ITT (to be presented in the next chapter) adapts directly to a proof for TT.

In the third chapter, the author introduces an intensional type theory ITT whose types are those of TT but whose terms extend those of TT to include the so-called secondary typing which enables one to distinguish between "use" and "mention" occurrences of a predicate name. The secondary typing of predicate names is obtained by adding a new clause to the definition of terms which assigns the secondary type 1 to terms P of type $[\overline{\tau}]$ on the condition that no variable of type other than 1 has a free occurrence in P. The author shows that without this restriction, ITT would be inconsistent since one can derive both $\vdash R(R)$ and $R(R) \vdash$ where $R \stackrel{df}{=} \lambda x.\exists X.[\neg X(u) \wedge u = X]$. Like TT, ITT enjoys weak Church-Rosser and every ITT-term has a unique $\beta\eta$-normal form. The presence of secondary typing complicates the polytyping for ITT. To deal with this, the author adds the notation $str : \tau \cap 1$ as a shorthand for both statements $str : \tau$ and $str : 1$ and the unification of type expressions takes this notation into account. Like TT, a polytyping algorithm is given for ITT and is shown to be correct. Similarly, an intensional semantics is defined for ITT as well as its proof theory. The semantic rules of ITT are those of TT plus two new rules $+Int$ and $-Int$ which deal with equality between terms P and Q which have the same primary type τ and the same secondary type 1. The semantic proof of the consistency of ITT is left as an exercise and is said to be similar to the semantic proof of the consistency of EL. The completeness proof of ITT, similar to that of the completeness of EL is then given. In the final section of this chapter, the author presents the so-called strongly impredicative type theory SITT which is an extension of ITT which allows impredicative predicates like the Russell set R which is the extension of the predicate $\lambda u.\neg u(u)$. This extension is achieved by extending the ITT types to include the clause:

- If τ is a predicate type then $\tau \cap 1$ is an intersection type and a type but not a predicate type.

Variables are then divided into quantification variables and abstraction variables. The terms of SITT extend those of ITT and are given as follows:

- A constant or a variable cv is a term of type t$[cv]$.
- If $P : [\tau, \overline{\tau}]$ and $Q : \tau$ then $(PQ) : [\overline{\tau}]$.
- If $P : [\overline{\tau}]$ and $x : \tau$ then $(\lambda v.P) : [\tau, \overline{\tau}]$ for a quantification variable x.
  If $P : [\overline{\tau}]$ and $v : \tau$ then $(\lambda v.P) : [\tau \cap 1, \overline{\tau}]$ for an abstraction variable v.
- If $P : [\overline{\tau}]$ then $P : 1$ provided that no quantification variable of type other than 1 has a free occurrence of P.
- If $P : [\overline{\tau}]$ and $P : 1$ then $P : [\overline{\tau}] \cap 1$.

The author explains how following the steps of ITT, the semantics and the proof theory of SITT can be defined and how its consistency can be established. The author shows how R is indeed a term of SITT whereas $R(R)$ is not.

In the fourth chapter, the author is concerned with recursion. Since ITT is a logic of predicates and not functions, the author takes the so-called recursion generators as the fundamental concept for recursion theory. The chapter starts by defining recursion generators. A recursion generator with parameters of type $\overline{\rho}$ is any term of ITT of type $[\overline{\rho}, [\overline{\tau}], \overline{\tau}]$ without any free occurrences of variables, where $\overline{\rho}$ can be empty but $\overline{\tau}$ cannot. The chapter defines recursive predicates from recursion generators using the so-called least and greatest predicate operators of type $[[[\overline{\tau}], \overline{\tau}], \overline{\tau}]$. The chapter first defines least and greatest predicate operators Lt and Gt for predicates of type $[\tau]$ as follows:

$Lt \stackrel{df}{=} \lambda wg, u. \forall Z. [\forall x. [wg(Z, x) \rightarrow Z(x)] \rightarrow Z(u)]$

$Gt \stackrel{df}{=} \lambda wg, u. \exists Z. [\forall x. [Z(x) \rightarrow wg(Z, x)] \wedge Z(u)]$

Note that these Lt and Gt have the type $[[[\tau], \tau], \tau]$ (where $u, x : \tau, Z : [\tau]$ and wg has the type $[[\tau], \tau]$ of a recursion generator without parameters). The author discusses how the above definitions can be generalised to a recursion generator of any type, and derives sequents which express a general form of induction for $Lt(wg)$ and $Gt(wg)$ (where $wg : [[\tau], \tau]$). The author also discusses the following rules that are said to be related to co-induction and fixed-point induction (note that $RG : [\overline{\rho}, [\tau], \tau]$ and $\overline{r} : \overline{\rho}$):

$$\text{LtInd.} \qquad \frac{-\forall x. [Lt(RG(\overline{r}))(x) \rightarrow P(x)]}{-\forall x. [RG(\overline{r})(P, x) \rightarrow P(x)]}$$

$$\text{GtInd.} \qquad \frac{-\forall x. [P(x) \rightarrow Gt(RG(\overline{r}))(x)]}{-\forall x. [P(x) \rightarrow RG(\overline{r})(P, x)]}$$

Next, the author defines monotonic recursion generators and derives some of their properties. Recursion generators are then used to define some known recursive predicates. From 0, the successor S and the recursion generator $RN \stackrel{def}{=} \lambda Z, x. [x = 0 \vee \exists u. [Z(u) \wedge x = S(u)]]$ of type $[[1], 1]$, the predicate for natural numbers is defined by $N \stackrel{def}{=} Lt(RN)$. The five Peano axioms are derived, and the author shows how projection and pairing can be defined in ITT, and reflects on $Gt(RN)$ and $Lt(RN)$ showing that the former is not well-founded whereas the latter is. Continuous recursion generators are defined next using two predicates $Con\exists$ and $Con\forall$. Conditions for useful properties of $Lt(RG)$ and $Gt(RG)$ are given for monotonic and $\exists$- or $\forall$-continuous recursion generators RG and a number of results are established. For example, positive and e-positive occurrences of variable Z in a formula H are defined, as are positive and e-positive recursion generators. The author proves that certain properties must hold if Z is a positive (respectively e-positive) variable of H and gives a similar result for $\neg H$. Moreover, the author shows that a positive recursion generator is monotonic and an e-positive recursion generator is $\exists$- and $\forall$-continuous. Next, a Horn sequent defined by a recursion generator $RG : [\overline{\rho}, [\overline{\sigma}], \overline{\sigma}]$ is given to be a sequent of the form: $RG(\overline{w})(C(\overline{w}), \overline{x}) \vdash C(\overline{w})(\overline{x})$ where $C : [\overline{\rho}, \overline{\sigma}]$ is a constant which does not occur in RG, $\overline{w} : \overline{\rho}$ and $\overline{x} : \overline{\sigma}$. The author shows that if $RG(\overline{r})$ is a monotonic recursion generator then the Horn sequent $RG(\overline{r})(C(\overline{r}), \overline{x}) \vdash C(\overline{r})(\overline{x})$ and

its converse (where C is defined by either $\lambda\overline{w}.Lt(RG(\overline{w}))$ or $\lambda\overline{w}.Gt(RG(\overline{w}))$) are derivable. The author also defines simultaneous Horn sequents and establishes a similar result to the one established for Horn sequents and gives an example and a number of exercises on (simultaneous) Horn sequents. Furthermore, the author discusses again the predicates $Lt(RG)$ and $Gt(RG)$ and shows that they could be defined by iteration provided that RG is monotonic and respectively $\exists$-continuous and $\forall$-continuous. The final section of the chapter starts with a discussion of the need for potentially infinite domains (e.g., the potentially infinite tape of a Turing machine), and the need of treating these domains differently than truly infinite domains (e.g., the domain of the natural numbers), and gives a number of definitions necessary for the study of potentially infinite domains.

It was already said that the types of ITT can be seen as a proper subset of the types of Church's STT. In ITT, the type of function values can only be [ ] whereas in STT, this is not the case. In the fifth chapter, the author addresses this issue and defines in ITT, a notation for partial functions with intentionally defined values. In order to deal with non-denoting terms (i.e., terms that cannot be given a value by a valuation), the author introduces the choice (indefinite description) operator ϵ. For a choice term ϵM to be denoting, it suffices that $\exists v.M(v)$ be satisfied. But, since different occurrences of ϵM may denote different members of the extension of M, the sequent $\vdash \exists v.M(v) \rightarrow \epsilon M = \epsilon M$ may not be derivable whereas the sequent $\vdash \exists v.M(v) \rightarrow (\lambda v.v = v)\epsilon M$ is. This means that contractions of the form $(\lambda v.P)\epsilon M > [\epsilon M/v]P$ must be excluded. The chapter introduces the new type theory ITTϵ which is the same as ITT except for the following:

- terms are extended with choice terms: "If M is a predicate term of type $[\tau]$ then ϵM is a choice term and term of type τ but not a predicate term";
- the scope of occurrences of choice terms plays an important role;
- formulae of ITTϵ are predicate terms of type [ ] (choice terms excluded);
- substitution $[Q/v]P$ is only defined when Q is not a choice term;
- the contraction relation $>_\epsilon$ includes the known η-rule and:
  - (β) $(\lambda v.P)Q$ contracts to $[Q/v]P$ when Q is not a choice term.
  - (ϵ) $P\epsilon M$ contracts to $\lambda\overline{u}.[\exists v : M].P(v, \overline{u})$ where $\overline{u} : \overline{\sigma}$ and $v : \tau$ are fresh distinct variables, $P : [\tau, \overline{\sigma}]$ and $M : [\tau]$.

Example derivations in ITTϵ are given and the semantics of ITTϵ is studied in terms of ITT through a partial mapping from ITTϵ to ITT which takes a non choice term P of ITTϵ to a term $P^\ddagger$ of ITT with the same type and free variables, and takes $P\epsilon M$ (where P is a predicate term) to a term of the form $\lambda\overline{u}.[\exists v : M^\ddagger].P^\ddagger(v, \overline{u})$. The author shows that ITTϵ is a conservative extension of ITT and is hence consistent. The given semantics implies that if $\Gamma^\ddagger \vdash \Theta^\ddagger$ is valid/derivable in ITT, then $\Gamma \vdash \Theta$ is valid /derivable in ITTϵ, ITTϵ is complete

and enjoys cut elimination. Using the ϵ operator, the author very briefly defines function terms ιM, gives a number of exercises and discusses partial functions and dependent types.

The sixth chapter deals with intuitionistic formulations of ITT. First, the author takes Gentzen's sequent calculus ELG which was shown in chapter 1 to be equivalent to the author's elementary logic EL, and adapts it to give ITTG, a sequent calculus formulation of ITT. As in EL, the author takes sequents $\Gamma \vdash \Theta$ where Γ, Θ are finite sets rather than ordered sequences. Moreover, instead of taking $\downarrow$ and $\exists$ to be primitives (as in ITT), the author takes $\exists, \wedge$ and $\neg$ as primitives in order to smooth transition to HITTG, the intuitionistic version of ITTG. Similarly to proving in chapter 1 that EL and ELG are equivalent, one can prove here that ITT and ITTG are equivalent. The extension of ITT to ITTϵ can be followed here too to extend ITTG into ITTGϵ. Classical $\vee, \rightarrow$ and $\forall$ are defined and named $\vee_c, \rightarrow_c$ and $\forall_c$. For HITTG, $\neg, \wedge, \vee, \rightarrow, \forall$ and $\exists$ are all taken to be primitive ($\neg, \wedge$ and $\exists$ are shared with ITTG, and the intuitionistic $\vee, \rightarrow, \forall$ are distinct from $\vee_c, \rightarrow_c, \forall_c$) and the succedent Θ in a sequent $\Gamma \vdash \Theta$ of HITTG, is always assumed to be either empty or a single formula. The derivability relationship between ITTG and HITTG is established and it is also shown that HITTG fits in the intuitionistic/constructive philosophy in the sense that if $\Gamma \vdash F \vee G$ then either $\Gamma \vdash F$ or $\Gamma \vdash G$; and if $\Gamma \vdash \exists R$ then there is a term t such that $\Gamma \vdash R(t)$. So far, we have the equivalence between the semantic tree formulation EL (resp. ITT) and the sequent calculus formulation ELG (resp. ITTG). Moreover, a derivation in EL (resp. ITT) is an abbreviation of a derivation in ELG (resp. ITTG). In order to study whether some results obtained in ITT also hold in HITTG, the author gives the semantic tree formulation HITT of HITTG, studies derivations in HITT and establishes the equivalence between HITT and HITTG. With the help of the so-called forests of semantic trees, examples are given which illustrate how the logic of HITT differs from that of ITT. The author reflects on the fact that since recursion theory as developed in ITT in chapter 4 can also be developed in HITT, then the fundamental logic for recursion theory is intuitionistic.

In chapter 7, the author proposes that it is worthwhile restating some old positions on the relationship between logic and mathematics and starts with the position that mathematics is applied logic. The author discusses Feferman's simplification of MacLane's example of an argument used in category theory but which cannot be formalised in traditional set theories/logics (because of self-membership). The simplified argument centres around the statement that "the set of Abelian semi-groups is itself an Abelian semi-group under Cartesian product and isomorphism". The author shows how this example can be formalised in ITTϵ. Next, the author discusses formalising Zermelo-Fraenkel set theory with the axiom of choice (ZFC) in ITT. The nine non logical axioms of ZFC are all expressed in ITT and the conjunction of these axioms in ITT is called Axioms. For each closed formula of ZF, a closed formula of ITT is obtained and is called a ZF-formula. A predicate ZFC is defined in ITT by $ZFC \stackrel{df}{=} \lambda Z.[\text{Axioms} \rightarrow Z]$. The author points out that it is easy to show

that $\vdash ZFC(F)$ for each ZF-formula F which translates a theorem in ZFC but that it is a challenge to show that $\{F \mid F$ is a ZF-formula and $\vdash ZFC(F)\}$ is consistent. Finally, the author develops Cantor's diagonal argument in ITT.

In the eigth chapter, the author aims to demonstrate that the domains needed for the semantics of programming languages can be defined in ITT using recursion generators. After reflecting on the difference between definitions and computations, the author uses a flowchart language example to show that a semantics for recursive commands of a programming language can be given in ITT. The syntax of the simple flowchart language as well as the semantics of expressions and commands are given in ITT. Simultaneous Horn sequents are used in the definition of command semantics. A number of derivations that illustrate this semantics are given. Finally, the author shows how recursive domains can be defined in ITT. First, a number of primitive domains is given. This is followed by a number of domain constructors. Recursive domains are recursively defined by domain equations using primitive domains and domain constructors. The author explains why only finite functions are allowed as domain constructors and solves (using simulatenous Horn sequents and simultaneous recursion generators) an example of domain equations taken from Milner and Tofte.