Extracting Programs from Constructive HOL Proofs via IZF Set-Theoretic Semantics

Robert Constable and Wojciech Moczydłowski *

{rc, wojtek}@cs.cornell.edu
Department of Computer Science, Cornell University, Ithaca, NY 14853, USA

Abstract. Church's Higher Order Logic is a basis for proof assistants — HOL and PVS. Church's logic has a simple set-theoretic semantics, making it trustworthy and extensible. We factor HOL into a constructive core plus axioms of excluded middle and choice. We similarly factor standard set theory, ZFC, into a constructive core, IZF, and axioms of excluded middle and choice. Then we provide the standard set-theoretic semantics in such a way that the constructive core of HOL is mapped into IZF. We use the disjunction, numerical existence and term existence properties of IZF to provide a program extraction capability from proofs in the constructive core.

We can implement the disjunction and numerical existence properties in two different ways: one modifying Rathjen's realizability for CZF and the other using a new direct weak normalization result for intensional IZF by Moczydłowski. The latter can also be used for the term existence property.

1 Introduction

Church's Higher-Order logic [1] has been remarkably successful at capturing the intuitive reasoning of mathematicians. It was distilled from *Principia Mathematica*, and is sometimes called the Simple Theory of Types based on that legacy. It incorporates the λ calculus as its notation for functions, including propositional functions, thus interfacing well with computer science.

One of the reasons Higher-Order logic is successful is that its axiomatic basis is very small, and it has a clean set-theoretic semantics at a low level of the cummulative hierarchy of sets (up to $\omega + \omega$) and can thus be formalized in a small fragment of ZFC set theory . This means it interfaces well with standard mathematics and provides a strong basis for trust. Moreover, the set theory semantics is the basis for many extensions of the core logic; for example, it is straightforward to add arrays, recursive data types, and records to the logic.

Church's theory is the logical basis of two of the most successful interactive provers used in hardware and software verification, HOL and PVS. This is due in part to the two characteristics mentioned above in addition to its elegant automation based on Milner's tactic mechanism and its elegant formulation in the ML metalanguage.

* This research was supported by NSF grants DUE-0333526 and 0430161.

Until recently, one of the few drawbacks of HOL was that its logical base did not allow a way to express a constructive subset of the logic. This issue was considered by Harrison for HOL-light [2], and recently Berghofer implemented a constructive version of HOL in the Isabelle implementation [3, 4] in large part to enable the extraction of programs from constructive proofs. This raises the question of finding a semantics for HOL that justifies this intuitively sound extraction.

The standard justification for program extraction is based on logics that embedded extraction deeply into their semantics; this is the case for the Calculus of Inductive Constructions (CIC) [5, 6], Minlog [7], Computational Type Theory (CTT) [8, 9] or the closely related Intuitionistic Type Theory (ITT) [10, 11]. The mechanism of extraction is built deeply into logic and the provers based on it, e.g. Agda [12] on ITT, Coq [13] on CIC, MetaPRL [14] and Nuprl [15] on CTT.

In this paper we show that there is a way to provide a clean set-theoretic semantics for HOL and in the same stroke use it to semantically justify program extraction. The idea is to first factor HOL into its constructive core, say Constructive HOL, plus the axioms of excluded middle and choice. The semantics for this language can be given in ZFC set theory, and if that logic is factored into its constructive core, called IZF, plus excluded middle and choice (choice is sufficient to give excluded middle), then in the standard semantics, IZF provides the semantics for Constructive HOL. Moreover, we can base program extraction on the IZF semantics.

The constructive content of IZF is not as transparent as in the constructive set theories of Aczel, introduced in [16], however, in these set theories it is not possible to express the impredicative nature of Higher-Order Logic. Also, IZF is not as expressive as Howe's ZFC [17, 18] with inaccessible cardinals and computational primitives, but this makes IZF a more standard theory.

Our semantics is appealing not only because it factors so elegantly, but also because the computational issues and program extraction can be reduced to the standard constructive properties of IZF — the disjunction, numerical existence and term existence properties.

We can implement the disjunction and numerical existence properties in two different ways: one modifying Rathjen's realizability for CZF [19] and the other using a new direct weak normalization result for intensional IZF by Moczydłowski [20]. The latter can also be used for the term existence property.

In this paper, we provide a set-theoretic semantics for HOL which has the following properties:

- It is as simple as the standard semantics, presented in Gordon and Melham's [21].
- It works in constructive set-theory.
- It provides a semantical basis for program extraction.
- It can be applied to the constructive version of HOL recently implemented in Isabelle-HOL as a means of using constructive HOL proofs as programs.

This paper is organized as follows. In section 2 we present a a version of HOL. In section 3 we define set-theoretic semantics. Section 4 defines constructive set theory IZF and states its main properties. We show how these properties can be used for program extraction in section 5.

2 Higher-order logic

In this section, we present in detail higher-order logic. There are two syntactic categories: *terms* and *types*. The types are generated by the following abstract grammar:

$$\tau ::= nat \mid bool \mid prop \mid \tau \to \tau \mid (\tau, \tau)$$

The distinction between *bool* and *prop* corresponds to the distinction between the two-element type and the type of propositions in type theory, or between the two-element object and the subobject classifier in category theory or, as we shall see, between 2 and the set of all subsets of 1 in constructive set theory.

The terms of HOL are generated by the following abstract grammar:

$$t ::= x_\tau \mid c_\tau \mid (t_{\tau\to\sigma}\ u_\tau)_\sigma \mid (\lambda x_\tau.\ t_\sigma)_{\tau\to\sigma} \mid (t_\tau, s_\sigma)_{(\tau,\sigma)}$$

Thus each term t_α in HOL is annotated with a type α, which we call *the type of t*. We will often skip annotating of terms with types, this practice should not lead to confusion, as the implicit type system is very simple. Terms of type *prop* are called *formulas*.

The free variables of a term t are denoted by $FV(t)$ and defined as usual. We consider α-equivalent terms equal. Our version of HOL has a set of builtin constants. To increase readability, we write $c : \tau$ instead of c_τ to provide the information about the type of c. If the type of a constant involves α, it is a constant *schema*, there is one constant for each type τ substituted for α. There are thus constants $=_{bool}$, $=_{nat}$ and so on.

$$\bot : prop \qquad \top : prop \qquad =_\alpha : (\alpha, \alpha) \to prop$$

$$\to : (prop, prop) \to prop \qquad \land : (prop, prop) \to prop \qquad \lor : (prop, prop) \to prop$$

$$\forall_\alpha : (\alpha \to prop) \to prop \qquad \exists_\alpha : (\alpha \to prop) \to prop \qquad \varepsilon_\alpha : (\alpha \to prop) \to \alpha$$

$$0 : nat \qquad S : nat \to nat \qquad false : bool \qquad true : bool$$

We present the proof rules for HOL in a sequent-based natural deduction style. A *sequent* is a pair (Γ, t), where Γ is a list of formulas and t is a formula. Free variables of a context are the free variables of all its formulas. A sequent (Γ, t) is written as $\Gamma \vdash t$. We write binary constants (equality, implication, etc.) using infix notation. We use standard abbreviations for quantifiers: $\forall a : \tau.\ \phi$ abbreviates $\forall_\tau(\lambda a_\tau.\ \phi)$, similarly with $\exists a : \tau.\ \phi$. The proof rules for HOL are:

$$\frac{}{\Gamma \vdash t}\ t \in \Gamma \qquad \frac{}{\Gamma \vdash t = t} \qquad \frac{\Gamma \vdash t = s}{\Gamma \vdash \lambda x_\tau.\ t = \lambda x_\tau.\ s}\ x_\tau \notin FV(\Gamma)$$

$$\frac{\Gamma \vdash t \quad \Gamma \vdash s}{\Gamma \vdash t \land s} \qquad \frac{\Gamma \vdash t \land s}{\Gamma \vdash t} \qquad \frac{\Gamma \vdash t \land s}{\Gamma \vdash s} \qquad \frac{}{\Gamma \vdash \top}$$

$$\frac{\Gamma \vdash t}{\Gamma \vdash t \vee s} \qquad \frac{\Gamma \vdash s}{\Gamma \vdash t \vee s} \qquad \frac{\Gamma \vdash t \vee s \quad \Gamma, t \vdash u \quad \Gamma, s \vdash u}{\Gamma \vdash u}$$

$$\frac{\Gamma, t \vdash s}{\Gamma \vdash t \to s} \qquad \frac{\Gamma \vdash s \to t \quad \Gamma \vdash s}{\Gamma \vdash t} \qquad \frac{\Gamma \vdash s = u \quad \Gamma \vdash t[u]}{\Gamma \vdash t[s]}$$

$$\frac{\Gamma \vdash f_{\alpha \to prop}\ t_\alpha}{\Gamma \vdash \exists_\alpha(f_{\alpha \to prop})} \qquad \frac{\Gamma \vdash \exists_\alpha(f_{\alpha \to prop}) \quad \Gamma, f_{\alpha \to prop}\ x_\alpha \vdash u}{\Gamma \vdash u}\ x_\alpha \text{ new}$$

Finally, we list HOL axioms.

1. (FALSE) $\bot = \forall b : prop.\ b$.
2. (FALSENOTTRUE) $false = true \to \bot$.
3. (BETA) $(\lambda x_\tau.\ t_\sigma)s_\tau = t_\sigma[x_\tau := s_\tau]$.
4. (ETA) $(\lambda x_\tau.\ f_{\tau \to \sigma}\ x_\tau) = f_{\tau \to \sigma}$.
5. (FORALL) $\forall_\alpha = \lambda P_{\alpha \to prop}.\ (P = \lambda x_\alpha.\ \top)$.
6. (P3) $\forall n : nat.\ (0 = S(n)) \to \bot$.
7. (P4) $\forall n, m : nat.\ S(n) = S(m) \to n = m$.
8. (P5) $\forall P : nat \to prop.\ P(0) \wedge (\forall n : nat.\ P(n) \to P(S(n))) \to \forall n : nat.\ P(n)$.
9. (BOOL) $\forall x : bool.\ (x = false) \vee (x = true)$.
10. (EM) $\forall x : prop.\ (x = \bot) \vee (x = \top)$.
11. (CHOICE) $\forall P : \alpha \to prop.\ \forall x : \alpha.\ P\ x \to P(\varepsilon_{(\alpha \to prop) \to \alpha}(P))$.

Our choice of rules and axioms is redundant. Propositional connectives, for example, could be defined in terms of quantifiers and *bool*. However, we believe that this makes the account of the semantics clearer and shows how easy it is to define a sound semantics for such system.

The theory CHOL (Constructive HOL) arises by taking away from HOL axioms (CHOICE) and (EM).

We write $\vdash_H \phi$ and $\vdash_C \phi$ to denote that HOL and CHOL, respectively, proves ϕ. We will generally use letters P, Q to denote proof trees. A notation $P \vdash_C \phi$ means that P is a proof tree in CHOL of ϕ.

3 Semantics

3.1 Set theory

The set-theoretic semantics needs a small part of cumulative hierarchy — $R_{\omega+\omega}$ is sufficient to carry out all the constructions. The Axiom of Choice is necessary in order to define the meaning of the ε constant. For this purpose, C will denote a[1] blatantly non-constructive function such that for any $X, Y \in R_{\omega+\omega}$, if X is non-empty, then $C(X, Y) \in X$, and if X is empty then $C(X, Y) = Y$.

Recall that in the world of set theory, $0 = \emptyset$, $1 = \{0\}$ and $2 = \{0, 1\}$. Classically $P(1)$, the set of all subsets of 1, is equal to 2. This is not the case constructively; there is no uniform way of transforming an arbitrary subset A of 1 into an element of 2.

The following helpful lemma, however, does hold in a constructive world:

[1] Note that if we want to pinpoint C, we need to assume more than AC, as the existence of a definable choice function for $R_{\omega+\omega}$ is not provable in ZFC.

Lemma 1. *If $A \in P(1)$, then $A = 1$ iff $0 \in A$.*

We will use lambda notation in set theory to define functions: $\lambda x \in A.\ B(x)$ means $\{(x, B(x)) \mid x \in A\}$.

3.2 The definition of the semantics

We first define a meaning $[\![\tau]\!]$ of a type τ by structural induction on τ.

- $[\![nat]\!] = \mathbb{N}$.
- $[\![bool]\!] = 2$.
- $[\![prop]\!] = P(1)$.
- $[\![(\tau, \sigma)]\!] = [\![\tau]\!] \times [\![\sigma]\!]$, where $A \times B$ denotes the cartesian product of sets A and B.
- $[\![\tau_1 \to \tau_2]\!] = [\![\tau_1]\!] \to [\![\tau_2]\!]$, where $A \to B$ denotes the set of all functions from A to B.

The meaning of a constant c_α is denoted by $[\![c_\alpha]\!]$ and is defined as follows.

- $[\![=_\alpha]\!] = \lambda(x_1, x_2) \in ([\![\alpha]\!] \times [\![\alpha]\!]).\ \{x \in 1 \mid x_1 = x_2\}$.
- $[\![\to]\!] = \lambda(b_1, b_2) \in [\![prop]\!] \times [\![prop]\!].\ \{x \in 1 \mid x \in b_1 \to x \in b_2\}$.
- $[\![\vee]\!] = \lambda(b_1, b_2) \in [\![prop]\!] \times [\![prop]\!].\ b_1 \cup b_2$.
- $[\![\wedge]\!] = \lambda(b_1, b_2) \in [\![prop]\!] \times [\![prop]\!].\ b_1 \cap b_2$.
- $[\![false]\!] = [\![\bot]\!] = 0$.
- $[\![true]\!] = [\![\top]\!] = 1$.
- $[\![\forall_\alpha]\!] = \lambda f \in [\![\alpha]\!] \to [\![prop]\!].\ \bigcap_{a \in [\![\alpha]\!]} f(a)$.
- $[\![\exists_\alpha]\!] = \lambda f \in [\![\alpha]\!] \to [\![prop]\!].\ \bigcup_{a \in [\![\alpha]\!]} f(a)$.
- $[\![\varepsilon_\alpha]\!] = \lambda P \in [\![\alpha]\!] \to [\![prop]\!].\ C(P^{-1}(1), [\![\alpha]\!])$.
- $[\![0]\!] = 0$.
- $[\![S]\!] = \lambda n \in \mathbb{N}.\ n + 1$

Standard semantics, presented for example by Gordon and Melham in [21], uses a truth table approach — implication $\phi \to \psi$ is false iff ϕ is true and ψ is false etc. It is easy to see that with excluded middle, our semantics is equivalent to the standard one.[2]

To present the rest of the semantics, we need to introduce environments. An *environment* is a partial function from HOL variables to sets such that $\rho(x_\tau) \in [\![\tau]\!]$. We will use the symbol ρ exclusively for environments. The meaning $[\![t]\!]_\rho$ of a term t is parameterized by an environment ρ and defined by structural induction on t:

- $[\![c_\tau]\!]_\rho = [\![c_\tau]\!]$.
- $[\![x_\tau]\!]_\rho = \rho(x_\tau)$
- $[\![s\ u]\!]_\rho = App([\![s]\!]_\rho, [\![u]\!]_\rho)$.
- $[\![\lambda x_\tau.\ u]\!] = \{(a, [\![u]\!]_{\rho[x_\tau := a]}) \mid a \in [\![\tau]\!]\}$.
- $[\![(s, u)]\!]_\rho = ([\![s]\!]_\rho, [\![u]\!]_\rho)$.

[2] For the interested reader, our definition of the meaning of logical constants is essentially a combination of the fact that any complete lattice with pseudo-complements is a model for higher-order logic and that $P(1)$ is a complete lattice with pseudo-complement defined in the clause for $\to$.

3.3 Properties

There are several standard properties of the semantics we have defined. The following two lemmas are proved by induction on t:

Lemma 2 (Substitution lemma). *For any terms t, s and environments ρ, $\llbracket t \rrbracket_{\rho[x:=\llbracket s \rrbracket_\rho]} = \llbracket t[x := s] \rrbracket_\rho$.*

Lemma 3. *For any ρ, $\llbracket t_\alpha \rrbracket_\rho \in \llbracket \alpha \rrbracket$.*

In particular, this implies that for any formula t, $\llbracket t \rrbracket \subseteq 1$. So if we want to prove that $\llbracket t \rrbracket = 1$, then by Lemma 1 it suffices to show that $0 \in \llbracket t \rrbracket$.

3.4 Soundness

The soundness theorem establishes validity of the proof rules and axioms with respect to the semantics.

Definition 1. *$\rho \models \Gamma \vdash t$ means that ρ is defined for $x_\tau \in FV(\Gamma) \cup FV(t)$.*

By the definition of environments, if $\rho \models \Gamma \vdash \bar{t}$, then for all $x_\tau \in FV(\Gamma) \cup FV(t)$, $\rho(x_\tau) \in \llbracket \tau \rrbracket$.

Definition 2. *We write $\llbracket \Gamma \rrbracket_\rho = 1$ if $\llbracket t_1 \rrbracket_\rho = 1, \ldots, \llbracket t_n \rrbracket_\rho = 1$, where $\Gamma = t_1, t_2, \ldots, t_n$.*

Theorem 1 (Soundness). *If $\Gamma \vdash t$, then for all $\rho \models \Gamma \vdash t$, if $\llbracket \Gamma \rrbracket_\rho = 1$, then $\llbracket t \rrbracket = 1$.*

Proof. Straightforward induction on $\Gamma \vdash t$. We show some interesting cases. Case $\Gamma \vdash t$ of:

- $$\frac{\Gamma \vdash t = s}{\Gamma \vdash \lambda x_\tau .\, t = \lambda x_\tau .\, s}$$
  Take any $\rho \models \Gamma \vdash \lambda x_\tau .\, t = \lambda x_\tau .\, s$. We need to show that $\{(a, \llbracket t \rrbracket_{\rho[x_\tau := a]}) \mid a \in \llbracket \tau \rrbracket\} = \{(a, \llbracket s \rrbracket_{\rho[x_\tau := a]} \mid a \in \llbracket \tau \rrbracket\}$. That is, that for any $a \in \llbracket \tau \rrbracket$, $\llbracket t \rrbracket_{\rho[x_\tau := a]} = \llbracket s \rrbracket_{\rho[x_\tau := a]}$. Let $\rho' = \rho[x_\tau := a]$. Since $\rho' \models \Gamma \vdash t = s$, by the inductive hypothesis we get the claim.
- $$\frac{\Gamma, t \vdash s}{\Gamma \vdash t \to s}$$
  Suppose $\llbracket \Gamma \rrbracket_\rho = 1$. We need to show that $0 \in \{x \in 1 \mid x \in \llbracket t \rrbracket_\rho \to x \in \llbracket s \rrbracket_\rho\}$. Since $0 \in 1$, assume $0 \in \llbracket t \rrbracket_\rho$. Then $\llbracket \Gamma, t \rrbracket_\rho = 1$, so by the inductive hypothesis $\llbracket s \rrbracket_\rho = 1$ and $0 \in \llbracket s \rrbracket_\rho$.
- $$\frac{\Gamma \vdash t \to s \quad \Gamma \vdash t}{\Gamma \vdash s}$$
  Suppose $\llbracket \Gamma \rrbracket_\rho = 1$. By the inductive hypothesis, $0 \in \{x \in 1 \mid x \in \llbracket t \rrbracket_\rho \to x \in \llbracket s \rrbracket_\rho\}$ and $0 \in \llbracket t \rrbracket_\rho$, so easily $0 \in \llbracket s \rrbracket_\rho$.

- $$\frac{\mathit{\Gamma} \vdash s = u \quad \mathit{\Gamma} \vdash t[x := u]}{\mathit{\Gamma} \vdash t[x := s]}$$

  The proof is straightforward, using the Substitution Lemma.

- $$\frac{\mathit{\Gamma} \vdash f\ t_\alpha}{\mathit{\Gamma} \vdash \exists_\alpha(f_{\alpha \to prop})}$$

  Assume $[\![\mathit{\Gamma}]\!]_\rho = 1$. We have to show that $0 \in \bigcup_{a \in [\![\alpha]\!]} [\![f]\!]_\rho(a)$, so that there is $a \in [\![\alpha]\!]$ such that $0 \in f(a)$. By Lemma 3, $[\![t_\alpha]\!]_\rho \in [\![\alpha]\!]$, so taking $a = [\![t_\alpha]\!]_\rho$ we get the claim by the inductive hypothesis.

- $$\frac{\mathit{\Gamma} \vdash \exists_\alpha(f_{\alpha \to prop}) \quad \mathit{\Gamma}, f\ x_\alpha \vdash u}{\mathit{\Gamma} \vdash u}\ x_\alpha \text{ new}$$

  Suppose $[\![\mathit{\Gamma}]\!]_\rho = 1$. By the inductive hypothesis, there is $a \in [\![\alpha]\!]$ such that $0 \in [\![f]\!]_\rho(a)$. Let $\rho' = \rho[x_\alpha := a]$. Then $\rho' \models \mathit{\Gamma}, f\ x_\alpha \vdash u$, so by the inductive hypothesis we get $0 \in [\![u]\!]_\rho$, which is what we want.

Having verified the soundness of the HOL proof rules, we proceed to verify the soundness of the axioms.

Theorem 2. *For any axiom t of HOL and any ρ, $0 \in [\![t]\!]_\rho$.*

Proof. We proceed axiom by axiom and sketch the respective proofs.

- (FALSE) $[\![\bot]\!]_\rho = \emptyset = \bigcap_{a \in P(1)} a = [\![\forall b : prop.\ b]\!]_\rho$. The second equality follows by $0 \in P(1)$.
- (BETA) Follows by the Substitution Lemma.
- (ETA) Follows by the fact that functions in set theory are represented by their graphs.
- (FORALL) We have:

  $$[\![\forall_\alpha]\!]_\rho = \{(P, \bigcap_{a \in [\![\alpha]\!]} P(a)) \mid P \in [\![\alpha]\!] \to P(1)\}$$

  Also:

  $$[\![(\lambda P_{\alpha \to prop}.\, P = \lambda x_\alpha.\, \top)]\!]_\rho = \{(P, \{x \in 1 \mid P = \lambda x \in [\![\alpha]\!].\, 1\}) \mid P \in [\![\alpha]\!] \to P(1)\}$$

  Take any $P \in [\![\alpha]\!] \to P(1)$. It suffices to show that $\bigcap_{a \in [\![\alpha]\!]} P(a) = \{x \in 1 \mid P = \lambda y \in [\![\alpha]\!].\, 1\}$. But $x \in \bigcap_{a \in [\![\alpha]\!]} P(a)$ iff for all $a \in [\![\alpha]\!]$, $x \in P(a)$ and $x = 0$. This happens if and only if $x = 0$ and for all $a \in [\![\alpha]\!]$, $P(a) = 1$ which is equivalent to $x \in \{x \in 1 \mid P = \lambda y \in [\![\alpha]\!].\, 1\}$. The sets in question are therefore equal.
- The axioms $P3, P4, P5$ follow by the fact that natural numbers satisfy the respective Peano axioms.

- (BOOL) We need to show that $[\![\forall_{bool}.\ (\lambda x_{bool}.\ x = false \vee x = true)]\!]_\rho = 1$. Unwinding the definition, this is equivalent to $\bigcap_{x \in 2}(\{z \in 1 \mid x = 0\} \cup \{z \in 1 \mid x = 1\}) = 1$. and furthermore to: for all $x \in 2$, $x \in \{z \in 1 \mid x = 0\} \cup \{z \in 1 \mid x = 1\}$. If $x \in 2$, then either $x = 0$ or $x = 1$. In the former case, $0 \in \{z \in 1 \mid x = 0\}$, in the latter $0 \in \{z \in 1 \mid x = 1\}$.
- (EM) We need to show that $[\![\forall_{prop}.\ (\lambda x_{prop}.\ x = \bot \vee x = \top)]\!]_\rho = 1$. Reasoning as in the case of (BOOL), we find that this is equivalent to: for all $x \in P(1)$, $x \in \{z \in 1 \mid x = 0\} \cup \{z \in 1 \mid x = 1\}$. Suppose $x \in P(1)$. At this point, it is impossible to proceed further constructively, all we know is that x is a subset of 1, which doesn't provide enough information to decide whether $x = 0$ or $x = 1$. However, classically, using the rule of excluded middle, $P(1) = 2$ and we proceed as in the previous case.
- (CHOICE) Straightforward.

Corollary 1. *HOL is consistent: it is not the case that $\vdash_H \bot$.*

Proof. Otherwise we would have $[\![\bot]\!] = [\![\top]\!]$, that is $0 = 1$.

4 IZF

The essential advantage of the semantics in the previous section over a standard one is that for the constructive part of HOL this semantics can be defined in constructive set theory IZF.

An obvious approach to creating a constructive version of ZFC set theory is to replace the underlying first-order logic with intuitionistic first-order logic. As many authors have explained [22–24], the ZF axioms need to be reformulated so that they don't imply the law of excluded middle.

In a nutshell, to get IZF from ZFC, the Axiom of Choice and Excluded Middle are taken away and Foundation is reformulated as $\in$-induction. The axioms of IZF are thus Extensionality, Union, Infinitiy, Power Set, Separation, Replacement or Collection[3] and $\in$-Induction. The detailed account of the theory can be found for example in Friedman's [25]. Besoon's book [23] contains a lot of information on metamathematical properties of IZF and related set theories. For convenience, we assume that the first-order logic has built-in terms and bounded quantifiers.

The properties of IZF important for us, proven for the first time by Myhill in [22], are:

- Disjunction Property (DP) : If IZF $\vdash \phi \vee \psi$, then IZF $\vdash \phi$ or IZF $\vdash \psi$.
- Numerical Existence Property (NEP) : If IZF $\vdash \exists x \in \mathbb{N}.\ \phi(x)$, then there is a natural number n such that IZF $\vdash \phi(\overline{n})$, where $\overline{n} = S(S(\ldots(0)))$ and $S(x) = x \cup \{x\}$.
- Term Existence Property (TEP) : If IZF $\vdash \exists x.\ \phi(x)$, then for some term t, IZF $\vdash \phi(t)$.

[3] There is a difference, in particular the version with Collection doesn't satisfy TEP. A concerned reader can replace IZFz with IZF$_R$ whenever TEP is used.

Moreover, the semantics and the soundness theorem for CHOL work in IZF, as neither Choice nor Excluded Middle were necessary to carry out these developments. Note that the existence of $P(1)$ is crucial for the semantics.

All the properties are constructive — there is a recursive procedure extracting a natural number, a disjunct or a term from a proof. A trivial one is to look through all the proofs for the correct one. For example, if IZF $\vdash \phi \vee \psi$, a procedure could enumerate all theorems of IZF looking for either ϕ or ψ; its termination would be ensured by DP. We discuss more efficient alternatives in section 5.3.

5 Extraction

We will show that the semantics we have defined can serve as a basis for program extraction for proofs. All that is necessary for program extraction from constructive HOL proofs is provided by the semantics and the soundness proof. Therefore, if one wants to provide an extraction mechanism for the constructive part of the logic, it may be sufficient to carefully define set-theoretic semantics, prove the soundness theorem and the extraction mechanism for IZF would take care of the rest. We speculate on practical uses of this approach in section 6.

5.1 IZF Extraction

We first describe extraction from IZF proofs. To facilitate the description, we will use a very simple fragment of type theory, which we call TT^0.

The *types* of TT^0 are generated by the following abstract grammar. They should not be confused with HOL types; the context will make it clear which types we refer to.

$$\tau ::= * \mid P_\phi \mid \mathit{nat} \mid \mathit{bool} \mid (\tau, \tau) \mid \tau + \tau \mid \tau \to \tau$$

We associate with each type τ of TT^0 a set of its elements, which are finitistic objects. The set of elements of τ is denoted by $El(\tau)$ and defined by structural induction on τ:

- $El(*) = \{*\}$.
- $El(P_\phi)$ is the set of all IZF proofs of formula ϕ.
- $El(\mathit{nat}) = \mathbb{N}$, the set of natural numbers.
- $El(\mathit{bool}) = \{true, false\}$.
- $M \in El((\tau_1, \tau_2)) = El(\tau_1) \times El(\tau_2)$.
- $M \in El(\tau_1 + \tau_2)$ iff either $M = \mathit{inl}(M_1)$ and $M_1 \in El(\tau_1)$ or $M = \mathit{inr}(M_1)$ and $M_1 \in El(\tau_2)$.
- $M \in El(\tau_1 \to \tau_2)$ iff M is a method which given any element of $El(\tau_1)$ returns an element of $El(\tau_2)$.

In the last clause, we use an abstract notion of “method”. It will not be necessary to formalize this notion, but for the interested reader, all “methods“ we use are functions provably recursive in $ZF + Con(ZF)$.

The notation $M : \tau$ means that $M \in El(\tau)$.

We call a TT^0 type *pure* if it doesn't contain $*$ and P_ϕ. There is a natural mapping of pure types TT^0 to sets. It is so similar to the meaning of the HOL types that we will use the same notation, $\llbracket \tau \rrbracket$:

- $\llbracket nat \rrbracket = \mathbb{N}$.
- $\llbracket bool \rrbracket = 2$.
- $\llbracket (\tau, \sigma) \rrbracket = \llbracket \tau \rrbracket \times \llbracket \sigma \rrbracket$.
- $\llbracket \tau + \sigma \rrbracket = \llbracket \tau \rrbracket + \llbracket \sigma \rrbracket$, the disjoint union of $\llbracket \tau \rrbracket$ and $\llbracket \sigma \rrbracket$.
- $\llbracket \tau \to \sigma \rrbracket = \llbracket \tau \rrbracket \to \llbracket \sigma \rrbracket$.

If a set (and a corresponding IZF term) is in a codomain of the map above, we call it *type-like*. If a set A is type-like, then there is a unique pure type τ such that $\llbracket \tau \rrbracket = A$. We denote this type $Type(A)$.

Before we proceed further, let us extend TT^0 with a new type Q_τ, where τ is any pure type of TT^0. The members of $El(Q_\tau)$ are pairs $(t, \mathcal{P})$ such that $\mathcal{P} \vdash_{IZF} t \in \llbracket \tau \rrbracket$ ($\mathcal{P}$ is an IZF proof of $t \in \llbracket \tau \rrbracket$). Note that there is a natural mapping from HOL terms M of type τ into Q_τ — it is easy to construct using Lemma 3 a proof $\mathcal{P}$ of the fact that $\llbracket M \rrbracket_\emptyset \in \llbracket \tau \rrbracket$, so the pair $(\llbracket M \rrbracket_\emptyset, P) : Q_\tau$. In particular, any natural number n can be injected into Q_{nat}. The set of pure types stays unchanged.

We first define a helper function T, which takes a pure type τ and returns another type. Intuitively, $T(\tau)$ is the type of the extract from a statement $\exists x \in \llbracket \tau \rrbracket$. T is defined by induction on τ:

- $T(bool) = bool$.
- $T(nat) = nat$.
- $T((\tau, \sigma)) = (T(\tau), T(\sigma))$
- $T(\tau + \sigma) = T(\tau) + T(\sigma)$.
- $T(\tau \to \sigma) = Q_\tau \to T(\sigma)$ (in order to utilize an IZF function from $\llbracket \tau \rrbracket$ to $\llbracket \sigma \rrbracket$ we need to supply an element of a set $\llbracket \tau \rrbracket$, that is an element of Q_τ)

Now we assign to each formula ϕ of IZF a TT^0 type $\overline{\phi}$, which intuitively describes the *computational content* of an IZF proof of ϕ. We do it by induction on ϕ:

- $\overline{a \in b} = *$.
- $\overline{a = b} = *$ (atomic formulas carry no useful computational content).
- $\overline{\phi_1 \vee \phi_2} = \overline{\phi_1} + \overline{\phi_2}$.
- $\overline{\phi_1 \wedge \phi_2} = (\overline{\phi_1}, \overline{\phi_2})$.
- $\overline{\phi_1 \to \phi_2} = P_{\phi_1} \to \overline{\phi_2}$.
- $\overline{\exists a \in A.\ \phi_1} - (T(Type(A)), \overline{\phi_1})$, if A is type-like.
- $\overline{\exists a \in A.\ \phi_1} = *$, if A is not type-like.
- $\overline{\exists a.\ \phi_1} = *$.
- $\overline{\forall a \in A.\ \phi_1} = Q_{Type(A)} \to \overline{\phi_1}$, if A is type-like.
- $\overline{\forall a \in A.\ \phi_1} = *$, if A is not type-like.
- $\overline{\forall a.\ \phi_1} = *$.

The definition is tailored for HOL logic and could be extended to allow meaningful extraction from a larger class of formulas, i.e. we could extract a term from $\exists a.\ \phi_1$ using TEP. We present some natural examples of this translation in action:

1. $\overline{\exists x \in \mathbb{N}.\ x = x} = \langle nat, * \rangle$.
2. $\overline{\forall x \in \mathbb{N} \exists y \in \mathbb{N}.\ \phi} = Q_{nat} \to \langle nat, \overline{\phi} \rangle$.
3. $\overline{\forall f \in \mathbb{N} \to \mathbb{N} \exists x \in \mathbb{N}.\ f(x) = 0} = Q_{nat \to nat} \to \langle nat, * \rangle$.

These types are richer than what we intuitively would expect — nat in the first case, $nat \to nat$ in the second and $(nat \to nat) \to nat$ in the third, because any HOL term of type nat or $nat \to nat$ can be injected into Q_{nat} or $Q_{nat \to nat}$. The extra $*$ can be easily discarded from types (and extracts).

Lemma 4. *For any natural number n, $\overline{\phi[a := \overline{n}]} = \overline{\phi}$.*

Proof. Straightforward induction on ϕ.

Lemma 5 (IZF). $(\exists a \in 2.\ \phi(a))$ *iff* $\phi(0) \vee \phi(1)$.

We are now ready to describe the extraction function E, which takes an IZF proof $\mathcal{P}$ of a formula ϕ and returns an object of TT^0 type $\overline{\phi}$. We do it by induction on ϕ, checking on the way that the object returned is of type $\overline{\phi}$. Recall that DP, TEP and NEP denote Disjunction, Term and Numerical Existence Property, respectively. Case ϕ of:

- $a \in b$ — return $*$. We have $* : *$.
- $a = b$ — return $*$. We have $* : *$, too.
- $\phi_1 \vee \phi_2$. Apply DP to $\mathcal{P}$ to get a proof $\mathcal{P}_1$ of either ϕ_1 or ϕ_2. In the former case return $inl(E(\mathcal{P}_1))$, in the latter return $inr(E(\mathcal{P}_1))$. By the inductive hypothesis, $E(\mathcal{P}_1) : \overline{\phi_1}$ (or $E(\mathcal{P}_1) : \overline{\phi_2}$), so $E(\mathcal{P}) : \overline{\phi}$ follows.
- $\phi_1 \wedge \phi_2$. Then there are proofs $\mathcal{P}_1$ and $\mathcal{P}_2$ such that $\mathcal{P}_1 \vdash \phi_1$ and $\mathcal{P}_2 \vdash \phi_2$. Return a pair $(E(\mathcal{P}_1), E(\mathcal{P}_2))$. By the inductive hypothesis, $E(\mathcal{P}_1) : \overline{\phi_1}$ and $E(\mathcal{P}_2) : \overline{\phi_2}$, so $(E(\mathcal{P}_1), E(\mathcal{P}_2)) : \overline{\phi_1 \wedge \phi_2}$.
- $\phi_1 \to \phi_2$. Return a function G which takes an IZF proof $\mathcal{Q}$ of ϕ_1, applies $\mathcal{P}$ to $\mathcal{Q}$ (using the modus-ponens rule of the first-order logic) to get a proof $\mathcal{R}$ of ϕ_2 and returns $E(\mathcal{R})$. By the inductive hypothesis, any such $E(\mathcal{R})$ is in $El(\overline{\phi_2})$, so $G : P_{\phi_1} \to \overline{\phi_2}$.
- $\exists a \in A.\ \phi_1(a)$, where A is type-like. Let $T = Type(A)$. We proceed by induction on T, case T of:
  - $bool$. By Lemma 5, we have $\phi_1(0) \vee \phi_1(1)$. Apply DP to get a proof $\mathcal{Q}$ of either $\phi_1(0)$ or $\phi_1(1)$. Let b be $false$ or $true$, respectively. Return a pair $(b, E(\mathcal{Q}))$. By the inductive hypothesis, $E(\mathcal{Q}) : \overline{\phi_1([\![b]\!])}$. By Lemma 4, $E(\mathcal{Q}) : \overline{\phi_1}$.
  - nat. Apply NEP to $\mathcal{P}$ to get a natural number n and a proof $\mathcal{Q}$ of $\phi_1(\overline{n})$. Return a pair $(n, E(\mathcal{Q}))$. By the inductive hypothesis, $E(\mathcal{Q}) : \overline{\phi_1(\overline{n})}$, by Lemma 4, $E(\mathcal{Q}) : \overline{\phi_1}$, so $(n, E(\mathcal{Q})) : (nat, \overline{\phi_1})$.

  - (τ, σ). Construct a proof $\mathcal{Q}$ of $\exists a_1 \in [\![\tau]\!] \exists a_2 \in [\![\sigma]\!].\ a = \langle a_1, a_2 \rangle \wedge \phi_1$. Let $M = E(\mathcal{Q})$. By the inductive hypothesis M is a pair $\langle M_1, M_2 \rangle$ such that $M_1 : T(\tau)$ and $M_2 : \overline{\exists a_2 \in [\![\sigma]\!].\ a = \langle a_1, a_2 \rangle \wedge \phi_1}$. Therefore M_2 is a pair $\langle M_{21}, M_{22} \rangle$, $M_{21} : T(\sigma)$ and $M_{22} : \overline{a = \langle a_1, a_2 \rangle \wedge \phi_1}$. Therefore M_{22} is a pair $\langle N, O \rangle$, where $O : \overline{\phi_1}$. Therefore $\langle M_1, M_{21} \rangle : T((\tau, \sigma))$, so $\langle \langle M_1, M_{21} \rangle, O \rangle : (T((\tau, \sigma)), \overline{\phi_1})$ and we are justified to return $\langle \langle M_1, M_{21} \rangle, O \rangle$.
  - $\tau + \sigma$. Construct a proof $\mathcal{Q}$ of $(\exists a_1 \in [\![\tau]\!].\ \phi_1) \vee (\exists a_1 \in [\![\sigma]\!].\ \phi_1)$. Apply DP to get the proof $\mathcal{Q}_1$ of (without loss of generality) $\exists a_1 \in [\![\tau]\!].\ \phi_1$. Let $M = E(\mathcal{Q}_1)$. By the inductive hypothesis, $M = \langle M_1, M_2 \rangle$, where $M_1 : T(\tau)$ and $M_2 : \overline{\phi_1}$. Return $\langle inl(M_1), M_2 \rangle$, which is of type $(T(\tau + \sigma), \overline{\phi_1})$.
  - $\tau \to \sigma$. Use TEP to get a term f such that $(f \in [\![\tau]\!] \to [\![\sigma]\!]) \wedge \phi_1(f)$. Construct proofs $\mathcal{Q}_1$ of $\forall x \in [\![\tau]\!] \exists y \in [\![\sigma]\!]. f(x) = y$ and $\mathcal{Q}_2$ of $\phi_1(f)$. By the inductive hypothesis and Lemma 4, $E(\mathcal{Q}_2) : \overline{\phi}$. Let G be a function which works as follows: G takes a pair $t, \mathcal{R}$ such that $\mathcal{R} \vdash t \in [\![\tau]\!]$, applies $\mathcal{Q}_1$ to $t, \mathcal{R}$ to get a proof $\mathcal{R}_1$ of $\exists y \in [\![\sigma]\!].\ f(t) = y \wedge \phi_1(f)$ and calls $E(\mathcal{R}_1)$ to get a term M. By inductive hypothesis, $M : \exists y \in [\![\sigma]\!].\ f(t) = y$, so $M = \langle M_1, M_2 \rangle$, where $M_1 : T(\sigma)$. G returns M_1. Our extraction procedure $E(\mathcal{P})$ returns $\langle G, E(\mathcal{Q}_2) \rangle$. The type of $\langle G, E(\mathcal{Q}_2) \rangle$ is $\langle \mathcal{Q}_\tau \to T(\sigma), \overline{\phi_1} \rangle$ which is equal to $\langle T(\tau \to \sigma), \overline{\phi_1} \rangle$.

- $\exists a \in A.\ \phi_1(a)$, where A is not type-like. Return $*$.
- $\exists a.\ \phi_1(a)$, $\forall a.\ \phi_1(a)$. Return $*$.
- $\forall a \in A.\ \phi_1(a)$, where A is type-like. Return a function G which takes an element $(t, \mathcal{Q})$ of $Q_{Type(A)}$, applies $\mathcal{P}$ to t and $\mathcal{Q}$ to get a proof R of $\phi_1(t)$, and returns $E(\mathcal{R})$. By the inductive hypothesis and Lemma 4, $E(\mathcal{R}) : \overline{\phi_1}$, so $G : Q_{Type(A)} \to \overline{\phi_1}$.
- $\forall a \in A.\ \phi_1(a)$, where A is not type-like. Return $*$.

5.2 HOL extraction

As in case of IZF, we will show how to do extraction from a subclass of CHOL proofs. The choice of the subclass is largely arbitrary, our choice illustrates the method and can be easily extended.

We say that a CHOL formula is *extractable* if it is generated by the following abstract grammar, where τ varies over pure TT^0 types and $\oplus \in \{\wedge, \vee, \to\}$.

$$\phi ::= \forall x : \tau.\ \phi \mid \exists x : \tau.\ \phi \mid \phi \oplus \phi \mid \bot \mid t = t,$$

We will define extraction for CHOL proofs of extractable formulas. By Theorem 2, if CHOL $\vdash \phi$, then $IZF \vdash 0 \in [\![\phi]\!]$. We need to slightly transform this IZF proof in order to come up with a valid input to E from the previous section. To this means, for any extractable ϕ (with possibly free variables) we define a formula ϕ' such that $IZF \vdash 0 \in [\![\phi]\!] \leftrightarrow \phi'$. The formula ϕ' is essentially ϕ with type membership information replaced by set membership information. We define ϕ' by induction on ϕ. The correctness follows trivially in each case. In all the cases we work in IZF. Case ϕ of:

- $\bot$. Then $\phi' = 0 \in [\![\bot]\!]$.

- $t = s$. Then $\phi' = 0 \in [\![t = s]\!]$.
- $\phi_1 \vee \phi_2$. $0 \in [\![\phi_1 \vee \phi_2]\!]$ iff $0 \in [\![\phi_1]\!]$ or $0 \in [\![\phi_2]\!]$. By the inductive hypothesis we get ϕ_1' and ϕ_2' such that $0 \in [\![\phi_1]\!] \leftrightarrow \phi_1'$ and $0 \in [\![\phi_2]\!] \leftrightarrow \phi_2'$. Take $\phi' = \phi_1' \vee \phi_2'$.
- $\phi_1 \wedge \phi_2$. Then $0 \in [\![\phi]\!]$ iff $0 \in [\![\phi_1]\!]$ and $0 \in [\![\phi_2]\!]$. Take ϕ_1' and ϕ_2' from the inductive hypothesis and set $\phi' = \phi_1' \wedge \phi_2'$.
- $\phi_1 \to \phi_2$. Then $0 \in [\![\phi_1 \to \phi_2]\!]$ iff $0 \in \{x \in 1 \mid x \in [\![\phi_1]\!] \to x \in [\![\phi_2]\!]\}$ iff $0 \in [\![\phi_2]\!] \to 0 \in [\![\phi_2]\!]$. By the inductive hypothesis get ϕ_1' such that $0 \in [\![\phi_1]\!] \leftrightarrow \phi_1'$ and ϕ_2' such that $0 \in [\![\phi_2]\!] \leftrightarrow \phi_2'$. Set $\phi' = \phi_1' \to \phi_2'$.
- $\forall a : \tau.\ \phi_1$. Then $0 \in [\![\phi]\!]$ iff for all $A \in [\![\tau]\!]$, $0 \in App([\![\lambda a : \tau.\ \phi_1]\!], A)$ iff for all $A \in [\![\tau]\!]$, $0 \in App(\{(x, [\![\phi_1]\!]_{\rho[a:=x]}) \mid x \in [\![\tau]\!]\}, A)$ iff for all $A \in [\![\tau]\!]$ $0 \in [\![\phi_1]\!]_{\rho[a:=A]}$ iff, by the Substitution Lemma, for all $A \in [\![\tau]\!]$, $0 \in [\![\phi_1[a := A]]\!]$ iff for all $A \in [\![\tau]\!]$, $0 \in [\![\phi_1]\!]$. Take ϕ_1' from the inductive hypothesis and set $\phi' = \forall a \in [\![\tau]\!].\ 0 \in \phi_1'$.
- $\exists a : \tau.\ \phi_1$. Then $0 \in [\![\phi]\!]$ iff $A \in [\![\tau]\!]$ iff $0 \in [\![\phi_1[a := A]]\!]$. Just as in the previous case, get ϕ_1' from the inductive hypothesis and set $\phi' = \exists a \in [\![\tau]\!].\ \phi_1'$.

Now we can finally define the extraction process. Suppose CHOL $\vdash \phi$, where ϕ is extractable. Using the soundness theorem, construct an IZF proof P that $0 \in [\![\phi]\!]$. Use the definition above to get ϕ' such that IZF $\vdash 0 \in [\![\phi]\!] \leftrightarrow \phi'$ and using P obtain a proof R of ϕ'. Finally, apply the extraction function E to R to get the computational extract.

5.3 Implementation issues

The extraction process is parameterized by the implementation of NEP, DP and TEP for IZF. Obviously, searching through all IZF proofs to get a witnessing natural number, term or a disjunct would not be a very effective method. We discuss two alternative approaches.

The first approach is based on realizability. Rathjen defines a realizability relation in [19] for weaker, predicative constructive set theory CZF. For any CZF proof of a formula ϕ, there is a realizer e such that the realizability relation $e \Vdash \phi$ holds, moreover, this realizer can be found constructively from the proof. Realizers provide the information for DP and NEP — which of the disjuncts holds and the witnessing number. They could be implemented using lambda terms. Adapting these results to IZF should be a straightforward matter and according to [19] the proof will appear in the forecoming paper. This approach has the drawback of not providing the proof of TEP, which would restrict the extraction process from statements of the form $\exists x \in [\![\tau]\!].\ \phi$ to atomic types τ. Moreover, the gap between the existing theoretical result and possible implementation is quite wide.

The second, more direct approach is based on Moczydłowski's proof in [20] of weak normalization of the lambda calculus λZ corresponding to proofs in IZF. The normalization is used to prove NEP, DP and TEP for the theory and the necessary information is extracted from the normal form of the lambda term corresponding to the IZF proof. Thus in order to provide the implementation of DP, NEP and TEP for IZF, it would suffice to implement λZ, which is specified completely in [20].

6 Conclusion

We have presented a computational semantics for HOL via standard interpretation in intuitionistic set theory. The semantics is clean, simple and agrees with the standard one.

The advantage of this approach is that the extraction mechanism is completely external to Constructive HOL. Using only the semantics, we can take any constructive HOL proof and extract from it computational information. No enrichment of the logic in the normalizing proof terms is necessary.

The separation of the extraction mechanism from the logic makes the logic very easily extendable. For example, inductive datatypes and subtyping have clean set-theoretic semantics, so can easily be added to HOL preserving consistency, as witnessed in PVS. As the semantics would work constructively, the extraction mechanisms from section 5 could be easily adapted to incorporate them. Similarly, one could define a set-theoretic semantics for the constructive version of HOL implemented in Isabelle ([3, 4]) in the same spirit, with the same advantages.

The modularity of our approach and the fact that it is much easier to give set-theoretic semantics for the logic than to prove normalization, could make the development of new trustworthy provers with extraction capabilities much easier and faster.

We would like to thank anonymous reviewers for their helpful comments.

References

1. Church, A.: A formulation of the simple theory of types. The Journal of Symbolic Logic **5** (1940) 55–68
2. Harrison, J.: HOL Light: A tutorial introduction. In: Formal Methods in Computer-Aided Design (FMCAD'96). Volume 1166 of Lecture Notes in Computer Science., Springer (1996) 265–269
3. Berghofer, S.: Proofs, Programs and Executable Specifications in Higher Order Logic. PhD thesis, Technische Universität München (2004)
4. Berghofer, S., Nipkow, T.: Executing higher order logic. In Callaghan, P., Luo, Z., McKinna, J., Pollack, R., eds.: Types for Proofs and Programs: TYPES'2000. Volume 2277 of Lecture Notes in Computer Science., Springer-Verlag (2002)
5. Coquand, T., Paulin-Mohring, C.: Inductively defined types, preliminary version. In: COLOG '88, International Conference on Computer Logic. Volume 417 of Lecture Notes in Computer Science., Springer, Berlin (1990) 50–66
6. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development; Coq'Art: The Calculus of Inductive Constructions. Springer-Verlag (2004)
7. Benl, H., Berger, U., Schwichtenberg, H., et al.: Proof theory at work: Program development in the Minlog system. In Bibel, W., Schmitt, P.G., eds.: Automated Deduction. Volume II. Kluwer (1998)
8. Allen, S.F., et al.: Innovations in computational type theory using Nuprl. To appear in 2006 (2006)
9. Constable, R.L., et al.: Implementing Mathematics with the Nuprl Proof Development System. Prentice-Hall, NJ (1986)

10. Martin-Löf, P.: Constructive mathematics and computer programming. In: Proceedings of the Sixth International Congress for Logic, Methodology, and Philosophy of Science, Amsterdam, North Holland (1982) 153–175
11. Nordström, B., Petersson, K., Smith, J.M.: Programming in Martin-Löf's Type Theory. Oxford Sciences Publication, Oxford (1990)
12. Augustsson, L., Coquand, T., Nordström, B.: A short description of another logical framework. In: Proceedings of the First Annual Workshop on Logical Frameworks, Sophia-Antipolis, France (1990) 39–42
13. The Coq Development Team: The Coq Proof Assistant Reference Manual – Version V8.0. (2004) http://coq.inria.fr.
14. Hickey, J., et al.: MetaPRL — A modular logical environment. In Basin, D., Wolff, B., eds.: Proceedings of the TPHOLs 2003. Volume 2758 of Lecture Notes in Computer Science., Springer-Verlag (2003) 287–303
15. Allen, S., et al.: The Nuprl open logical environment. In McAllester, D., ed.: Proceedings of the 17^{th} International Conference on Automated Deduction. Volume 1831 of Lecture Notes in Artificial Intelligence., Springer Verlag (2000) 170–176
16. Aczel, P.: The type theoretic interpretation of constructive set theory. In MacIntyre, A., Pacholski, L., Paris, J., eds.: Logic Colloquium '77, North Holland (1978)
17. Howe, D.J.: Semantic foundations for embedding HOL in Nuprl. In Wirsing, M., Nivat, M., eds.: Algebraic Methodology and Software Technology. Volume 1101 of Lecture Notes in Computer Science. Springer-Verlag, Berlin (1996) 85–101
18. Howe, D.J.: Toward sharing libraries of mathematics between theorem provers. In: Frontiers of Combining Systems, FroCoS'98, ILLC, Kluwer Academic Publishers (1998)
19. Rathjen, M.: The disjunction and related properties for constructive Zermelo-Fraenkel set theory. Journal of Symbolic Logic **70** (2005) 1233–1254
20. Moczydłowski, W.: Normalization of IZF with Replacement. Technical Report 2006-2024, Computer Science Department, Cornell University (2006)
21. Gordon, M., Melham, T.: Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic. Cambridge University Press, Cambridge (1993)
22. Myhill, J.: Some properties of intuitionistic Zermelo-Fraenkel set theory. In: Cambridge Summer School in Mathematical Logic. Volume 29., Springer (1973) 206–231
23. Beeson, M.J.: Foundations of Constructive Mathematics. Springer-Verlag (1985)
24. McCarty, D.: Realizability and recursive set theory. Journal of Pure and Applied Logic **32** (1986) 153–183
25. Friedman, H.: The consistency of classical set theory relative to a set theory with intuitionistic logic. The Journal of Symbolic Logic (1973) 315–319