Exercising Nuprl's open-endedness

Rahli, Vincent

DOI:
10.1007/978-3-319-42432-3_3

License:
Other (please specify with Rights Statement)

Document Version
Peer reviewed version

Citation for published version (Harvard):
Rahli, V 2016, Exercising Nuprl's open-endedness. in G-M Greuel, T Koch, P Paule & A Sommese (eds), *Mathematical Software - ICMS 2016 : 5th International Conference, Berlin, Germany, July 11-14, 2016, Proceedings.* Lecture Notes in Computer Science, vol. 9725, Springer, pp. 18-27, The 5th International Congress on Mathematical Software, ICMS 2016, Berlin, Germany, 11/07/16. https://doi.org/10.1007/978-3-319-42432-3_3

Exercising Nuprl's Open-Endedness

Vincent Rahli*

SnT, University of Luxembourg, Luxembourg
vincent.rahli@gmail.com

Abstract. Nuprl is an interactive theorem prover that implements an extensional constructive type theory, where types are interpreted as partial equivalence relations on closed terms. Nuprl is both computationally and type-theoretically open-ended in the sense that both its computation system and its type theory can be extended as needed by checking a handful of conditions. For example, Doug Howe characterized the computations that can be added to Nuprl in order to preserve the congruence of its computational equivalence relation. We have implemented Nuprl's computation and type systems in Coq, and we have showed among other things that it is consistent. Using our Coq framework we can now easily and rigorously add new computations and types to Nuprl by mechanically verifying that all the necessary conditions still hold. We have recently exercised Nuprl's open-endedness by adding nominal features to Nuprl in order to prove a version of Brouwer's continuity principle, as well as choice sequences in order to prove truncated versions of the axiom of choice and of Brouwer's bar induction principle. This paper illustrates the process of extending Nuprl with versions of the axiom of choice.

Keywords: Nuprl, Coq, Semantics, Open-Endedness, Axiom of Choice, Choice Sequences, Bar Induction, Continuity

1 Introduction

Nuprl. The Nuprl interactive theorem prover [13, 3] implements a dependent type theory called *Constructive Type Theory* (CTT), which is based on an untyped functional programming language. It has a rich type theory including identity (or equality) types, a hierarchy of universes, W types, quotient types [14], set types, union and (dependent) intersection types [28], image types [32], partial equivalence relation types [4], approximation and computational equivalence types [35], and partial types [38, 16]. CTT "mostly" differs from other similar constructive type theories such as the ones implemented by Agda [10, 1], Coq [8, 15], or Idris [11, 26], in the sense that CTT is an *extensional* theory (i.e., propositional and definitional equality are identified [20]) with types of partial functions [38, 16]. For example, the fixpoint $\texttt{fix}(\lambda x.x)$ diverges. It is nonetheless a member of many types such as $\overline{\mathbb{Z}}$, which is the type of integers and diverging terms (essentially the integer type of ML-like programming languages such as OCaml).

* This work was partially supported by the SnT and by the National Research Fund Luxembourg (FNR), through PEARL grant FNR/P14/8149128.

In Nuprl, type checking is undecidable but in practice this is mitigated by type inference and type checking heuristics implemented as tactics.

Formalization of Nuprl's metatheory in Coq. Following Allen's semantics [2], CTT types are interpreted as Partial Equivalence Relations (PERs) on closed terms. We have formalized Nuprl's metatheory in Coq [6, 5]. Our implementation includes: (1) an implementation of Nuprl's computation system; (2) an implementation of Howe's computational equivalence relation [22], and a proof that it is a congruence; (3) a definition of Allen's PER semantics of CTT [2, 16]; (4) definitions of Nuprl's derivation rules, and proofs that these rules are valid w.r.t. Allen's PER semantics; (5) and a proof of Nuprl's consistency [6, 5]. Our implementation is available at `https://github.com/vrahli/NuprlInCoq`, and additional information can be found at `http://www.nuprl.org/html/Nuprl2Coq/`.

Exploring type theory. Using our implementation of CTT in Coq, we are exploring type theory. For example, (1) we are reformulating CTT using a smaller core of primitive types, by allowing the theory to directly represent PERs as types [4]. We conjecture that dependent product and sum types will be definable in this new theory. (2) We have proved the validity of truncated (or squashed—see Sec. 2.2 for a discussion of truncation/squashing) versions of Brouwer's continuity principle [27, 17, 40, 12, 41, 44, 37]. w.r.t. Nuprl's PER semantics [34]. For that, following Longley's method [31], we used named exceptions as a probing mechanism to compute the modulus of continuity of a function. (3) We have proved the validity of versions of Brouwer's bar induction principle [27, 21, 17, 12, 41, 37, 43], which we are using to build parametrized families of W types from parametrized families of co-W types [9]. For that we added "choice sequences" to Nuprl's term language [36]. A choice sequence of type T is a Coq function from natural numbers to terms of type T. They are similar to the infinite sequences in [7], which are used to prove that the negative translation of the Axiom of Choice (AC) is realizable. They are also similar to Howe's set-theoretical functions in [24, 25, 23], which he used to provide a set-theoretical semantics of both Nuprl and HOL, allowing the shallow embedding of HOL in Nuprl. (4) We have proved the validity of truncated versions of AC [34, Sec.5.3].

Open-endedness. Because Nuprl was designed to be open-ended, i.e., theorems about computations and types "hold for a broad class of extensions to the system" [19], adding new features to the system often did not require much modifications to the existing properties of its computation system or to the statements and proofs of its inference rules. We illustrate this here with AC. Sec. 4 presents true and false versions of AC, which we have proved by extending CTT's computation and type systems.

2 Background on Nuprl

2.1 Constructive Type Theory

Nuprl's programming language is an untyped (à la Curry), lazy and applied (with pairs, injections, a fixpoint operator,...) λ-calculus. Its term language is open-ended in the sense that it contains all possible terms that follow a given structure described, for example, in [22], and mentioned below. Therefore, most terms do not have any operational semantics according to Nuprl's computation system. A type is a value of the computation system. Among other things, Allen's PER semantics associates PERs to these values, i.e., types are interpreted as partial equivalence relations (PERs) on closed terms [2]. As illustrated in [6, Fig.6], Allen's PER semantics can be seen as an inductive-recursive definition of: (1) an inductive relation $T_1 \equiv T_2$ that expresses type equality; and (2) a recursive function $a \equiv b \in T$ that expresses equality in a type.

Type equality is mostly intentional in the sense that two types that are interpreted by the same PER are not necessarily equal. Most notably, identity types of the form $a =_T b$, which expresses that a and b are equal members of the type T, can only be inhabited by the constant $\star$, i.e., they do not have any computational content as opposed to HoTT [42]. However, the two types $0 =_{\mathbb{N}} 0$ and $1 =_{\mathbb{N}} 1$ are not equal, even though they have the same PER. Note that Uniqueness of Identify Proofs (UIP) is true by definition in Nuprl. As mentioned above, because of this treatment of equality, Nuprl is also extensional in the sense that propositional and definitional equality are identified [20]. Also, function extensionality is true by definition of dependent product (function) types.

It turns out that Nuprl's type system is not only closed under computation but more generally under Howe's computational equivalence $\sim$, which he proved to be a congruence [22]. For example, one can prove that $\lambda x.(x+1)+0 \sim \lambda x.x+1$ without requiring one to infer a type for x. In any context C, when $t \sim t'$ we can rewrite t into t' without having to prove anything about types. We rely on this relation to prove equalities between programs (bisimulations) without concern for typing [35]. See Sec. 2.3 below for more details.

As mentioned above, we have implemented Nuprl's term language, its computation system, Howe's $\sim$ relation, and Allen's PER semantics in Coq [6, 5]. We have also showed that Nuprl is consistent by (1) proving that Nuprl's inference rules are valid w.r.t. Allen's PER semantics, and (2) proving that `False` is not inhabited. Using these two facts, we derive that there cannot be a proof derivation of `False`, i.e., Nuprl is consistent (see [6, 5, 33, Appx.A] for more details). We are using our Coq formalization to prove the validity of all the inference rules of Nuprl, and have already verified a large number of them.

2.2 Squashing

It is sometimes necessary to *truncate* or *squash* types for them to be true or consistent with Martin-Löf-like type theories such as Nuprl. For example, the non-truncated version of Brouwer's continuity principle, i.e., where the existential quantifier is interpreted constructively, is false in type theories such as Agda or Nuprl [29, 39, 18, 34, 36], while its truncated version is true in Nuprl [34]. Similar results hold about AC as discussed in Sec. 4, as well as about Brouwer's bar induction principle [36].

In Nuprl, there are various ways of *squashing* or *truncating* a type. The most widely used squashing operator in Nuprl throws away the evidence that a type is inhabited and squashes it down to a single inhabitant using, e.g., set types:

$\downarrow T = \{\texttt{Unit} \mid T\}$ (as defined in [13, pp.60]). The only member of this type is the constant $\star$, which is Unit's single inhabitant, and which is similar to `()` in languages such as OCaml, Haskell or SML. The constant $\star$ inhabits $\downarrow T$ if T is true/inhabited, but we do not keep the proof that it is true. See [33, Appx.F] for more information regarding squashing. Using the HoTT terminology, we also sometimes truncate types at the *propositional level* [42, pp.117]. In Nuprl propositional truncation corresponds to *squashing* a type down to a single equivalence class, i.e. all inhabitants are equal, using, e.g., quotient types [14]: $\lfloor T = T//\texttt{True}$. $\lfloor T$ is a proof-irrelevant type. Its members are the members of T, and they are all equal to each other in $\lfloor T$ because if $x, y \in T$ then $(x =_T y \iff \texttt{True})$. Note that the implication $\lfloor T \rightarrow \downarrow T$ is true because it is inhabited by $\lambda x.\star$, but we cannot prove the converse because to prove $\lfloor T$ we have to exhibit an inhabitant of T, which $\downarrow T$ does not give us because only $\star$ inhabits $\downarrow T$.

2.3 Howe's Computational Equivalence

Howe's computational equivalence is defined on closed terms as follows: $t \sim u$ if $t \preccurlyeq u \ \wedge \ u \preccurlyeq t$. Howe coinductively defines the approximation (or simulation) relation $\preccurlyeq$ as the largest relation R on closed terms such that $R \subset [R]$, where $[\cdot]$ is the following closure operator (also defined on closed terms): $t\ [R]\ u$ if whenever t computes to a value $\theta(\overline{b})$, then u also computes to a value $\theta(\overline{b'})$ such that $\overline{b}\ R\ \overline{b'}$. We write $\theta(\overline{b})$ for the term with outer operator θ and subterms $\overline{b}$, where each subterm is essentially a pair of a list of binding variables and a term. For example $\lambda x.x$ has one subterm that has one binding variable x. See [22, 6, 5] for details. By definition, one can derive, e.g., that $\bot \preccurlyeq t$ for all closed term t.

To prove that $\sim$ is a congruence, Howe first proves that $\preccurlyeq$ is a congruence [22]. Unfortunately, this is not easy to prove directly. Howe's "trick" was to define another inductive relation $\preccurlyeq^*$, which is a congruence and contains $\preccurlyeq$ by definition. To prove that $\preccurlyeq^*$ and $\preccurlyeq$ are equivalent and therefore that $\preccurlyeq$ and $\sim$ are congruences, it suffices to prove that $\preccurlyeq^*$ respects computation, i.e., given that $t \preccurlyeq^* u$, if t computes to a value of the form $\theta(\overline{b})$ then u also computes to a value $\theta(\overline{b'})$ such that $\overline{b} \preccurlyeq^* \overline{b'}$. Howe defined a condition called *extensionality* [22, Def.5] that non-canonical (i.e., non-values) operators of lazy computation systems have to satisfy for $\preccurlyeq^*$ to imply $\preccurlyeq$. Essentially, a non-canonical operator is extensional if it never reduces a term by making a decision based on non-canonical subterms, which is the case about Nuprl's non-canonical operators. For example, the following "bad" non-canonical operator is not extensional: if we allow $\mathrm{bad}(f(a))$, where $f(a)$ is the application of f to a, to reduce to a, and $\mathrm{bad}(v)$, where v is a value, to reduce to v, then $\mathrm{bad}((\lambda x.x+1)\ 1)$ would reduce to 1, and $(\lambda x.x+1)\ 1$ would reduce to 2, while $\mathrm{bad}(2)$ would reduce to 2, which is different from 1.

3 Open-Endedness and Exploration

One can extend CTT by either adding new computations, new types, or new inference rules. Typically, to add a new computation, one simply has to prove that it satisfies various preservation properties such as: if a term t_1 computes to a term t_2 then the free variables of t_2 are included in the free variables of t_1 and $t_1[x\backslash u]$ (the substitution of x for u in t_1) computes to $t_2[x\backslash u]$. Also, in the case of non-canonical operators, one has to prove that they are extensional.

We have recently added several operators to Nuprl: (1) named exceptions [34]; (2) a try/catch operator [34]; (3) a fresh operator to generate fresh names [34]; (4) choice sequences; (5) an eager application operator; as well as (6) various values denoting types such as our PER types [4]. In the case of exceptions, which are some sorts of values in the sense that they do not compute further, we had to modify one inference rule [33, Appx.C]. The proofs that our try/catch and eager application operators are extensional were standard. However, proving that our fresh operator is extensional required modifying the definition of $\preccurlyeq^*$ as discussed in [34, Sec.4.2]. Adding choice sequences made us loose the decidability of several relations such as α-equality or even syntactic equality, which it turned out we did not need [36, Sec.4.1]. We also had to modify one inference rule. Because types are values of the computation system, when adding a type we usually do not have to modify theorems and proofs about computations. However, we have to (1) provide an interpretation for the type: essentially a PER. (2) Then, because a type system has to satisfy some properties, as explained in [6, Sec.6.3], such as: PERs respect computation, we have to prove that the new type constructor satisfies these properties. We can then start stating and proving the validity of type inference rules regarding the new type constructor.

4 The Axiom of Choice

We now illustrate the process of extending Nuprl's computation system and type theory in order to validate axiom of choice type inference rules.

4.1 Squashed or Non-squashed?

The axiom of choice (where A and B are types, and P is of type $A \to B \to \mathbb{P}$)

$$\boldsymbol{\Pi} a{:}A.\boldsymbol{\Sigma} b{:}B.P\ a\ b \ \Rightarrow\ \boldsymbol{\Sigma} f{:}B^A.\boldsymbol{\Pi} a{:}A.P\ a\ f(a)$$

follows from the usual inference rules of the universal (dependent product) and existential (dependent sum) quantifiers [34, Sec.5.3]. However, this non-squashed version of AC is not always enough because existential quantifiers cannot always be interpreted as $\boldsymbol{\Sigma}$ but sometimes as truncated $\boldsymbol{\Sigma}$'s (see for example [18, 34, 36]). Therefore, we sometimes need instances of AC where $\boldsymbol{\Sigma}$ is either $\downharpoonleft$-squashed or $\downarrow$-squashed. In that case it is not obvious anymore which instances of AC are consistent with or provable in Nuprl. This section provides some answers.

We showed in [34, Sec.5.3] that we can directly prove in Nuprl the following $\downharpoonleft$-squashed versions of $\mathrm{AC}_{0,\mathrm{B}}$ and $\mathrm{AC}_{1,\mathrm{B}}$, where $\mathcal{B} = \mathbb{N}^{\mathbb{N}}$:

$$\boldsymbol{\Pi} n{:}\mathbb{N}.\downharpoonleft\boldsymbol{\Sigma} f{:}B.P\ n\ f \ \Rightarrow\ \downharpoonleft\boldsymbol{\Sigma} f{:}B^{\mathbb{N}}.\boldsymbol{\Pi} n{:}\mathbb{N}.P\ n\ f(n)$$

$$\boldsymbol{\Pi} n{:}\mathcal{B}.\downharpoonleft\boldsymbol{\Sigma} f{:}B.P\ n\ f \ \Rightarrow\ \downharpoonleft\boldsymbol{\Sigma} f{:}B^{\mathcal{B}}.\boldsymbol{\Pi} n{:}\mathcal{B}.P\ n\ f(n)$$

We mentioned in [36, Appx.B] that we have proved the validity of the following $\downarrow$-squashed version of $\mathrm{AC}_{0,\mathtt{NBase}}$ in our Coq framework using classical logic and choice sequences of terms of type $\mathtt{NBase} = \{t : \mathtt{Base} \mid (t : \mathtt{Base})\#\}$, where $(t : T)\#$ says that the term t is in the type T and does not contain any name, and `Base` is the type of closed terms with $\sim$ as its equality:

$$\boldsymbol{\Pi} n{:}\mathbb{N}.\downarrow\boldsymbol{\Sigma} f{:}\mathtt{NBase}.P\ n\ f \ \Rightarrow\ \downarrow\boldsymbol{\Sigma} f{:}\mathtt{NBase}^{\mathbb{N}}.\boldsymbol{\Pi} n{:}\mathbb{N}.P\ n\ f(n)$$

We showed in [36, Appx.B] that the $\downharpoonleft$-squashed version of Brouwer's weak continuity principle (`WCP`) and the negation of its unsquashed version, which are provable in Nuprl, imply the negation of the following $\downharpoonleft$-version of $\mathrm{AC}_{2,0}$ (where T is a non-empty type):

$$\begin{array}{l}\boldsymbol{\Pi} P{:}\mathbb{N}^{\mathcal{B}} \to T \to \mathbb{P}.\\ \quad(\boldsymbol{\Pi} f{:}\mathbb{N}^{\mathcal{B}}.\downharpoonleft\boldsymbol{\Sigma} n{:}\mathbb{N}.\ P\ f\ n)\ \Rightarrow\ \downharpoonleft\boldsymbol{\Sigma} N{:}\mathbb{N}^{\mathcal{B}} \to T.\ \boldsymbol{\Pi} f{:}\mathbb{N}^{\mathcal{B}}.P\ f\ (N\ f)\end{array}$$

Let us repeat the proof here. It suffices to prove that $\mathbb{N}^{\mathcal{B}}$ does not have the following choice principle (while $\mathcal{B}$ and $\mathbb{N}$ do):

$$\mathtt{ChoicePrinciple}(T) = \boldsymbol{\Pi} P{:}T \to \mathbb{P}.(\boldsymbol{\Pi} t{:}T.\downharpoonleft P(t)) \iff (\downharpoonleft\boldsymbol{\Pi} t{:}T.P(t))$$

which follows easily from both the facts that the $\downharpoonleft$-version of `WCP` is true in Nuprl and its unsquashed version is false. Let us prove $\neg\mathtt{ChoicePrinciple}(\mathbb{N}^{\mathcal{B}})$, i.e., assuming the hypothesis $\mathtt{ChoicePrinciple}(\mathbb{N}^{\mathcal{B}})$ we have to prove `False`. We instantiate this hypothesis with the following function, which we call C (where $\mathbb{N}_k$ is the type of natural numbers strictly less than k):

$$\lambda F.\boldsymbol{\Sigma} M{:}\mathbb{N}^{\mathcal{B}}.\boldsymbol{\Pi} f, g{:}\mathcal{B}.f =_{(\mathbb{N}_{M(f)} \to \mathbb{N})} g \to F(f) =_{\mathbb{N}} F(g)$$

We now get to assume $(\boldsymbol{\Pi} t{:}\mathbb{N}^{\mathcal{B}}.\downharpoonleft C(t)) \iff (\downharpoonleft\boldsymbol{\Pi} t{:}\mathbb{N}^{\mathcal{B}}.C(t))$. Because the $\downharpoonleft$-squashed version of `WCP` is true in Nuprl, we also get to assume $\boldsymbol{\Pi} t{:}\mathbb{N}^{\mathcal{B}}.\downharpoonleft C(t)$. From the above double implication, we obtain $\downharpoonleft\boldsymbol{\Pi} t{:}\mathbb{N}^{\mathcal{B}}.C(t)$. Because we are proving `False`, we can unsquash this new hypothesis, i.e., we get to assume $\boldsymbol{\Pi} t{:}\mathbb{N}^{\mathcal{B}}.C(t)$, which is the unsquashed version of `WCP`, which is false.

4.2 Choice Sequences

As mentioned above, in order to prove the validity of $\mathrm{AC}_{0,\mathtt{NBase}}$, a $\downarrow$-squashed version of AC, we added choice sequences of terms to Nuprl's term syntax:

```
Inductive Term := vterm (v : Var) | sterm (s : nat → Term) | oterm (op : Opid) (bs : list BTerm)
with BTerm := bterm (vs : list Var) (t : Term).
```

Our choice sequences are Coq functions from natural numbers to Nuprl terms. Additionally, we require that such choice sequences do not contain free variables or names [36, Sec.4.2]. This addition had interesting consequences such as: most relations on terms became undecidable such as syntactic equality and α-equality. Also, because of this additional limit constructor in the definition of terms, the return types of functions that are recursively applied to choice sequences had to be turned into W-like types with limit constructors. For example, we had to change the statement of our general induction principle on terms. Originally, this lemma went by induction on the size of term, which was simply a natural number. With choice sequences, the size of a term is now an ordinal number with a limit operator for the size of sequences:

```
Fixpoint osize (t : Term) : ord :=
  match t with
  | vterm _ ⇒ OS OZ
  | sterm f ⇒ OS (OL (fun x ⇒ osize (f x)))
  | oterm op bterms ⇒ OS (oaddl (map osize_bterm bterms))
  end
with osize_bterm (bt : BTerm) : ord := match bt with bterm lv nt ⇒ osize nt end.
```

where oaddl is an addition operation on lists of ordinals, which are defined as follows: `Inductive ord := OZ | OS (o : ord) | OL (s : nat → ord)`. We also had proved equalities between terms, where now in order to still prove an equality, we need to use in addition the function extensionality axiom to prove that two choice sequences are equal. We leave for future work the investigation of whether we can do without additional axioms using custom equality relations.

5 Conclusion

Much remains to be done to bridge the gap between our Coq implementation and Nuprl's current implementation. Following the footsteps of [30], we would like to synthesize a version of Nuprl from our implementation. Nevertheless, our Coq implementation of CTT already turned out to be very useful to investigate extensional type theory on a large scale, which often was made relatively easy by the fact that Nuprl is open-ended in many ways. However, we are sometimes making decisions that limit Nuprl's open-endedness. For example, because we have now added exceptions to Nuprl, it is not clear how or even whether we could add a parallel operator to Nuprl as mentioned in [34, Sec.8]. As another example, we have often used the fact that Nuprl's computation system is deterministic in our implementation, which will prevent us from adding non-deterministic operators. It is not clear yet whether we can do without this property, and most importantly it is not clear how to avoid such accidental limitations.

References

[1] *The Agda Wiki*. `http://wiki.portal.chalmers.se/agda/pmwiki.php`.

[2] Stuart F. Allen. “A Non-Type-Theoretic Semantics for Type-Theoretic Language”. PhD thesis. Cornell University, 1987.

[3] Stuart F. Allen, Mark Bickford, Robert L. Constable, Richard Eaton, Christoph Kreitz, Lori Lorigo, and Evan Moran. “Innovations in computational type theory using Nuprl”. In: *J. Applied Logic* 4.4 (2006). `http://www.nuprl.org/`, pp. 428–469.

[4] Abhishek Anand, Mark Bickford, Robert L. Constable, and Vincent Rahli. “A Type Theory with Partial Equivalence Relations as Types”. Presented at TYPES 2014. 2014.

[5] Abhishek Anand and Vincent Rahli. *Towards a Formally Verified Proof Assistant*. Tech. rep. `http://www.nuprl.org/html/Nuprl2Coq/`. Cornell University, 2014.

[6] Abhishek Anand and Vincent Rahli. “Towards a Formally Verified Proof Assistant”. In: *ITP 2014*. Vol. 8558. LNCS. Springer, 2014, pp. 27–44.

[7] Stefano Berardi, Marc Bezem, and Thierry Coquand. "On the Computational Content of the Axiom of Choice". In: *J. Symb. Log.* 63.2 (1998), pp. 600–622.

[8] Yves Bertot and Pierre Casteran. *Interactive Theorem Proving and Program Development.* `http://www.labri.fr/perso/casteran/CoqArt`. SpringerVerlag, 2004.

[9] Mark Bickford and Robert Constable. "Inductive Construction in Nuprl Type Theory Using Bar Induction". Presented at TYPES 2014 `http://nuprl.org/KB/show.php?ID=723`. 2014.

[10] Ana Bove, Peter Dybjer, and Ulf Norell. "A Brief Overview of Agda - A Functional Language with Dependent Types". In: *TPHOLs 2009.* Vol. 5674. LNCS. `http://wiki.portal.chalmers.se/agda/pmwiki.php`. Springer, 2009, pp. 73–78.

[11] Edwin Brady. "IDRIS —: systems programming meets full dependent types". In: *PLPV 2011.* ACM, 2011, pp. 43–54.

[12] Douglas Bridges and Fred Richman. *Varieties of Constructive Mathematics.* London Mathematical Society Lecture Notes Series. Cambridge University Press, 1987.

[13] R.L. Constable, S.F. Allen, H.M. Bromley, W.R. Cleaveland, J.F. Cremer, R.W. Harper, D.J. Howe, T.B. Knoblock, N.P. Mendler, P.Panangaden, J.T. Sasaki, and S.F. Smith. *Implementing mathematics with the Nuprl proof development system.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1986.

[14] Robert L. Constable. "Constructive Mathematics as a Programming Logic I: Some Principles of Theory". In: *Fundamentals of Computation Theory, Proceedings of the 1983 International.* Vol. 158. LNCS. Springer, 1983, pp. 64–77.

[15] *The Coq Proof Assistant.* `http://coq.inria.fr/`.

[16] Karl Crary. "Type-Theoretic Methodology for Practical Programming Languages". PhD thesis. Ithaca, NY: Cornell University, Aug. 1998.

[17] Michael A. E. Dummett. *Elements of Intuitionism.* Second. Clarendon Press, 2000.

[18] Martín Hötzel Escardó and Chuangjie Xu. "The Inconsistency of a Brouwerian Continuity Principle with the Curry-Howard Interpretation". In: *TLCA 2015.* Vol. 38. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015, pp. 153–164.

[19] Stuart F. Allen, Robert L. Constable, and Douglas J. Howe. "Reflecting the Open-Ended Computation System of Constructive Type Theory". In: *Logic, Algebra and Computation.* Vol. F79. NATO ASI Series. Springer-Verlag, 1990, pp. 265–280.

[20] Martin Hofmann. "Extensional concepts in intensional type theory". PhD thesis. University of Edinburgh, 1995.

[21] William A. Howard and Georg Kreisel. "Transfinite Induction and Bar Induction of Types Zero and One, and the Role of Continuity in Intuitionistic Analysis". In: *J. Symb. Log.* 31.3 (1966), pp. 325–358.

[22] Douglas J. Howe. "Equality in Lazy Computation Systems". In: *LICS 1989.* IEEE Computer Society, 1989, pp. 198–203.

[23] Douglas J. Howe. "Importing Mathematics from HOL into Nuprl". In: *Theorem Proving in Higher Order Logics.* Vol. 1125. LNCS. Berlin: Springer-Verlag, 1996, pp. 267–282.

[24] Douglas J. Howe. "On Computational Open-Endedness in Martin-Löf's Type Theory". In: *LICS '91.* IEEE Computer Society, 1991, pp. 162–172.

[25] Douglas J. Howe. "Semantic Foundations for Embedding HOL in Nuprl". In: *Algebraic Methodology and Software Technology.* Vol. 1101. LNCS. Berlin: Springer-Verlag, 1996, pp. 85–101.

[26] *Idris.* `http://www.idris-lang.org/`.

[27] S.C. Kleene and R.E. Vesley. *The Foundations of Intuitionistic Mathematics, especially in relation to recursive functions.* North-Holland Publishing Company, 1965.

[28] Alexei Kopylov. "Type Theoretical Foundations for Data Structures, Classes, and Objects". PhD thesis. Ithaca, NY: Cornell University, 2004.

[29] Georg Kreisel. "On weak completeness of intuitionistic predicate logic". In: *The Journal of Symbolic Logic* 27.2 (1962), pp. 139–158.

[30] Ramana Kumar, Rob Arthan, Magnus O. Myreen, and Scott Owens. "Self-Formalisation of Higher-Order Logic - Semantics, Soundness, and a Verified Implementation". In: *J. Autom. Reasoning* 56.3 (2016), pp. 221–259.

[31] John Longley. "When is a Functional Program Not a Functional Program?" In: *ICFP'99.* ACM, 1999, pp. 1–7.

[32] Aleksey Nogin and Alexei Kopylov. "Formalizing Type Operations Using the "Image" Type Constructor". In: *Electr. Notes Theor. Comput. Sci.* 165 (2006), pp. 121–132.

[33] Vincent Rahli and Mark Bickford. "A Nominal Exploration of Intuitionism". Extended version of our CPP 2016 paper: `http://www.nuprl.org/html/Nuprl2Coq/continuity-long.pdf`. 2015.

[34] Vincent Rahli and Mark Bickford. "A nominal exploration of intuitionism". In: *CPP 2016*. ACM, 2016, pp. 130–141.

[35] Vincent Rahli, Mark Bickford, and Abhishek Anand. "Formal Program Optimization in Nuprl Using Computational Equivalence and Partial Types". In: *ITP'13*. Vol. 7998. LNCS. Springer, 2013, pp. 261–278.

[36] Vincent Rahli, Mark Bickford, and Robert L. Constable. "A story of Bar Induction In Nuprl". Extended version avaible at `http://www.nuprl.org/html/Nuprl2Coq/bar-induction-long.pdf`. 2015.

[37] Michael Rathjen. "Constructive Set Theory and Brouwerian Principles". In: *J. UCS* 11.12 (2005), pp. 2008–2033.

[38] Scott F. Smith. "Partial Objects in Type Theory". PhD thesis. Ithaca, NY: Cornell University, 1989.

[39] A.S. Troelstra. "A Note on Non-Extensional Operations in Connection With Continuity and Recursiveness". In: *Indagationes Mathematicae* 39.5 (1977), pp. 455–462.

[40] A.S. Troelstra. "Aspects of Constructive Mathematics". In: *Handbook of Mathematical Logic*. North-Holland Publishing Company, 1977, pp. 973–1052.

[41] A.S. Troelstra and D. van Dalen. *Constructivism in Mathematics An Introduction*. Vol. 121. Studies in Logic and the Foundations of Mathematics. Elsevier, 1988.

[42] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: `http://homotopytypetheory.org/book`, 2013.

[43] Wim Veldman. "Brouwer's real thesis on bars". In: *Philosophia Scientiæ* CS6 (2006), pp. 21–42.

[44] Wim Veldman. "Understanding and Using Brouwer's Continuity Principle". English. In: *Reuniting the Antipodes — Constructive and Nonstandard Views of the Continuum*. Vol. 306. Synthese Library. Springer Netherlands, 2001, pp. 285–302.