A Story of Bar Induction in Nuprl

Vincent Rahli\textsuperscript{1*}, Mark Bickford\textsuperscript{2}, and Robert L. Constable\textsuperscript{2}

\textsuperscript{1} SnT, University of Luxembourg, Luxembourg  
\textsuperscript{2} Cornell University, USA

Abstract. In order to turn Nuprl's logic into a fully intuitionistic logic, we are currently experimenting with versions of Brouwer's bar induction principle. Using our formalization of Nuprl's metatheory in Coq, we have proved the classical validity of two such principles: one for sequences of numbers that involved little changes to the system, and a more general one for sequences of closed terms without names that involved adding a limit constructor to Nuprl's term syntax. We have also proved that these additions preserve Nuprl's key metatheoretical properties such as consistency. Finally, we show some new insights on bar induction that were made possible thanks to Nuprl's quotient types.

1 Introduction

Nuprl. The Nuprl interactive theorem prover \cite{Nuprl} implements a type theory called Constructive Type Theory (CTT), which is is a dependent type theory, in the spirit of Martin-Löf's extensional theory \cite{ML}, 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 \cite{Quotient}, set types, union and (dependent) intersection types \cite{Intersection}, image types \cite{Image}, PER types \cite{PER}, approximation and computational equivalence types \cite{Approximation}, and partial types \cite{Partial}. CTT “mostly” differs from other similar constructive type theories such as the ones implemented by Agda \cite{Agda}, Coq \cite{Coq}, or Idris \cite{Idris}, in the sense that CTT is an \textit{extensional} type theory (i.e., propositional and definitional equality are identified \cite{Extensional}) with types of partial functions \cite{Partial}. For example, the fixpoint \texttt{fix}(\texttt{\lambda}x.x) diverges. It is nonetheless a member of, among others, the type \textit{\mathbb{Z}}, which is the type of integers and diverging terms. The type \textit{\mathbb{Z}} can be seen as the integer type of ML-like programming languages such as OCaml. 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 \cite{Allen, Allen2}, CTT types are interpreted as Partial Equivalence Relations (PERs) on closed terms. We have formalized Nuprl's metatheory in Coq \cite{Coq, Coq2}. Our implementation includes: (1) an implementation of Nuprl's computation system; (2) an implementation of Howe's computational equivalence relation \cite{Howe}, and a proof

\textsuperscript{*} This work was partially supported by the SnT and by the National Research Fund Luxembourg (FNR), through PEARL grant FNR/P14/8149128.
that it is a congruence; (3) a definition of Allen’s PER semantics of CTT [1, 27]; (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 [8]. Our implementation is available at: https://github.com/vrahi/NuprlInCoq, and additional information can be found at: http://www.nuprl.org/html/Nuprl2Coq/.

**Inductive types.** Until recently, Nuprl was using Mendler’s inductive types [22] to build inductive types such as Coq’s [47]. Mendler provides proofs of the validity of inference rules for (co-)inductive types in his thesis. Unfortunately, his proof does not hold “as is” anymore for Nuprl’s current version because it relies on the fact that Nuprl was terminating at the time Mendler wrote his thesis [43]. This is not true anymore for several reasons, such as: Nuprl has now types of partial functions [53, 22, 27]. Also, Nuprl’s current function/pi and intersection types allow one to assign types to diverging elements. For example, the type Top of all terms such that all terms are equal in that type, can be defined as Void ⇒ Void (or as Void Void using intersection types), where Void is an uninhabited empty type. By definition of function types all terms inhabit Top, even diverging terms such as fix(λx.x)—this was not the case in [22], where only λ-terms were allowed to inhabit function types. To fix this issue of Nuprl’s inductive types we can either (1) fix Mendler’s proof or (2) provide another way to define inductive types. We leave (1) for future work. We can achieve (2) simply by adding primitive W types to Nuprl, which were already present in Allen’s system [1, 27], and are part of our Coq implementation [8, 9]. The second author of this paper came up with another solution using bar induction as mentioned below.

**Intuitionism.** There are two principles that distinguish Brouwer’s intuitionistic mathematics [22, 20, 9] from other constructive mathematics, namely, bar induction and a continuity principle for numbers [22, 33, 28, 11, 13, 63, 64, 10, 55, 52, 67, 68, 50], which among other things Brouwer used to prove that all real-valued functions on the unit interval are uniformly continuous [21]. A central concept in intuitionistic logic is the notion of a choice sequence [61], which is a never finished sequence of objects created over time by a creating (or creative) subject [28, Sec. 6.3]. Choice sequences can be lawlike in the sense that they are determined by an algorithm, or lawless in the sense that they are not subject to any law (e.g., generated by throwing a dice), or a combination of both. Brouwer developed a notion of intuitionistic continuum by defining real numbers as choice sequences, and, as mentioned above, proved that all real-valued functions on the unit interval are uniformly continuous using his continuity principle for numbers, which roughly speaking says that a decision on a choice sequence can only be made according to an initial segment of the sequence. To prove this uniform continuity principle, Brouwer also used a reasoning principle for choice sequences called the Fan Theorem (FT), which he derived from his bar induction principle.

**Bar induction.** We have recently proved the validity of squashed (or truncated—see Sec. 6.1 for a discussion of squashing/truncation) versions of Brouwer’s continuity principle [50, 51] w.r.t. Nuprl’s PER semantics. These past few years we have also been experimenting with versions of Brouwer’s Bar Induction principle (BI), and in [16], we showed how to build parametrized families of W types.
from parametrized families of co-W types using a variant of BI. Brouwer used BI to, among other things, justify his so-called (decidable) fan theorem, which says that every decidable (or detachable) bar is uniform (this will be made more precise below)—see [64], Ch.7,Sec.7, [28], Sec.3.2, [52], Appx A]. BI is an induction principle on barred universal spreads, where a spread, as Dummett defines it [28], Sec.3.2] “is essentially a tree, with the restriction that every path is infinite, and that we can effectively construct any subtree consisting of initial segments of finitely many paths”. The universal spread is the type of choice sequences of numbers (denoted B below). A fan is a finitary or finitely branching spread.

We first state below a “general” unconstrained version of BI, i.e. where the bar is not constrained, which is not true in constructive mathematics [38], Sec.7.14, [28], Sec.3.4, [54], Rem.3.3, [67], Sec.2]—Kleene [33], Sec.7.14,Lem.*27.23] showed that it contradicts continuity. However, BI is often accepted by intuitionists when bars are restricted to decidable or monotone bars [33, 28, 67]. Also, as proved by Kleene [33, Lem.9.8], functions on numbers (such as B’s members) are not and cannot be restricted to general recursive functions for FT (and therefore BI) to be true (see also [64], pp.223, [28], pp.52–53, [33], Sec.4).

First some notation (Sec.2 discusses Nuprl’s syntax and semantics in more details): We write B (the Baire space) for the function space \( \mathbb{N} \rightarrow \mathbb{N} \), which we also write as \( \mathbb{N}^\mathbb{N} \). We write \( B_k \) for \( \mathbb{N}^\mathbb{N} \), where \( k \) is a natural number and \( \mathbb{N}_k \) is the type of natural numbers strictly less than \( k \). We use \( \Pi \) and \( \Sigma \) for the constructive logical quantifiers \( \forall \) and \( \exists \). We sometimes write \( \Sigma x_1 : T_1 . \cdots . \Sigma x_n : T_n . P \) as \( \Sigma (x_1 : T_1) \cdots (x_n : T_n).P \), and similarly for \( \Pi \) types. In the context of types, we use the symbols + and \( \vee \) for the disjoint union type. The type \( t =_T u \) (also written \( t = u \in T \)) expresses that \( t \) and \( u \) are equal members of the type \( T \). Let \( \text{False} \) (also called Void) be \( 0 =_\mathbb{N} 1 \), and \( \text{True} \) (also called Unit) be \( 0 =_\mathbb{N} 0 \). As usual, \( \neg T \) is defined as \( T \rightarrow \text{Void} \). \( U_i \) is the universe type at level \( i \). We often omit levels and write either Type or \( \mathbb{P} \) for \( U_i \)—as opposed to Coq, there is no distinction between types and propositions in Nuprl.

We now formally state BI. A term \( Q \) is a predicate on finite sequences (of numbers) if it is a member of the type \( \Pi n : \mathbb{N}.B_n \rightarrow \mathbb{P} \). A predicate on finite sequences \( P \) is a subset of another predicate on finite sequences \( Q \) if for all \( n \in \mathbb{N} \) and \( s \in B_n \), \( (P \ n \ s) \Rightarrow (Q \ n \ s) \). A bar is a predicate on finite sequences \( B \), such that \( \Pi s : B.\Sigma n : \mathbb{N}.B \ n \ s \)—we will see below that the \( \Sigma \) type in this formula can sometimes be truncated. A bar \( B \) is decidable if for all \( n \in \mathbb{N} \) and \( s \in B_n \), \( (B \ n \ s) \lor \neg (B \ n \ s) \). A bar \( B \) is monotone if for all \( n, m \in \mathbb{N} \) and \( s \in B_n \), if \( B \ n \ s \) then \( B \ (n+1) \ (s \mathbin{\oplus}_n m) \), where \( s \mathbin{\oplus}_n m = \lambda x.\text{if } x = n \text{ then } m \text{ else } s(x) \). A predicate \( Q \) on finite sequences is inductive if for all \( n \in \mathbb{N} \) and \( s \in B_n \), if \( \Pi m : \mathbb{N}.(Q \ (n+1) \ (s \mathbin{\oplus}_n m)) \) then \( Q \ n \ s \). The unconstrained BI principle says that if \( Q \) is an inductive predicate on finite sequences, and \( B \) is a bar and a subset of \( Q \), then for any term \( t, (Q \ 0 \ t) \), i.e. \( Q \) is true about the empty sequence. Bar Induction on Decidable bars (BID) also assumes that \( B \) is decidable, and Bar Induction on Monotone bars (BIM) assumes that \( B \) is monotone.

In essence, Brouwer’s argument regarding the validity of BI turned a “canonical proof” that a spread is barred by \( B \) into a “canonical proof” that \( Q \) is true.
about the empty sequence \[ \text{Sec. 3.4, 64, Sec. 8.18, 67, Sec. 1} \]. Brouwer came up with the notion of a canonical proof by analyzing how one can prove that a spread is barred. A canonical proof is an infinitely branching proof-tree such that each of its branch is finite, and which is built-up from three kinds of inference steps: monotone (also called upward \[ 62 \], backward \[ 63 \], and \( \zeta \)-inferences \[ 28 \]) and inductive (also called downward \[ 62 \], forward \[ 60 \], and \( f \)-inferences \[ 21 \], \[ 28 \]) steps corresponding to the monotone and inductive predicates introduced above, as well as immediate steps \[ 61 \] (also called opening statements \[ 64 \] or \( \eta \)-inferences \[ 28 \]) to derive that individual sequences are barred. Unsurprisingly, these proof-trees correspond to the trees built by bar recursion operators such as Howard’s \( W \) operator \[ 32 \], which realizes BIM (see Sec. 3.6). As explained for example by Dummett \[ 28 \], Sec. 3.4, Brouwer might have believed that the monotone \( \zeta \)-steps were not necessary in canonical proofs, which was then refuted by Kleene \[ 23 \], Sec. 7.14, Lemma 27.23. As explained for example by Troelstra and van Dalen \[ 54 \], pp. 233, monotone \( \zeta \)-steps can only be eliminated when the bar is monotone or decidable. As explained below in more details, monotone steps are not necessary when proving \emph{squashed} propositions.

\textbf{Roadmap.} \textup{Sec. 3.1-3.6} introduce the \( \ast \)-squashed (see Sec. 3.4) unconstrained BI inference rule that we have proved to be valid using our Coq formalization, and present the versions of BID and BIM that we can derive using bar recursion operators. \textup{Sec. 3.7} presents a new and more general version of BIM. \textup{Sec. 3.8} proves that both this general principle and the standard BIM principle are false in Nuprl when not \( \ast \)-squashed. \textup{Sec. 4.1} proves the validity of a BI inference rule for sequences of numbers. As mentioned above, functions on numbers cannot be restricted to general recursive functions for BI to be true. Consequently, in order to prove the validity of this rule we added all Coq functions from numbers to numbers, to Nuprl’s term language, even those that make use of axioms, and are therefore not computable. Our choice sequences are similar to the choice sequences in \[ 12 \] and are introduced for a similar reason. Finally, \textup{Sec. 4.2} discusses a generalization of this result to sequences of closed terms without names \[ 51 \] ("names" as in "nominal logic" \[ 48 \]—we also sometimes call names, \emph{unguessable atoms}).

The results presented in this paper have either been formalized in Coq: \url{https://github.com/vrahli/NuprlInCoq} or they have been formalized in Nuprl: \url{http://www.nuprl.org/LibrarySnapshots/Published/Version2/Standard/continuity/index.html}—these can be accessed by clicking the green hyperlinks or alternatively the reader can search in our continuity library for the lemmas named as the hyperlinks.

We first start by presenting some key aspects of Nuprl’s syntax and semantics.

\section{Background on Nuprl}

\subsection{Nuprl’s Computation System}

\textup{Fig. 4} presents a subset of Nuprl’s syntax and small-step operational semantics \[ 3, 7 \]. Nuprl’s programming language is an untyped (à la Curry), lazy and
Fig. 1 Syntax (top) and operational semantics (bottom) of a subset of Nuprl

<table>
<thead>
<tr>
<th>`v ∈ Value ::= vt (type)</th>
<th>inl(t) (left injection)</th>
<th>* (as-im)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>t</code> (lambda)</td>
<td>inr(t) (right injection)</td>
</tr>
<tr>
<td>`ut ∈ Type ::= t (type)</td>
<td>Hxt1,t2 (product)</td>
<td>t1 = t2 ∈ t (equality)</td>
</tr>
<tr>
<td></td>
<td>Σxt1,t2 (sum)</td>
<td>t1 + t2 (disjoint union)</td>
</tr>
<tr>
<td></td>
<td>U (universe)</td>
<td>t1 ⊆ t2 (simulation)</td>
</tr>
<tr>
<td></td>
<td>W(x:t1,t2) (W)</td>
<td>t1 ⊑ t2 (bisimulation)</td>
</tr>
<tr>
<td>`t ∈ Term ::= x (variable)</td>
<td>let x := t in t2 (call-by-value)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>v (value)</td>
<td>(spread)</td>
</tr>
<tr>
<td></td>
<td>tt `t2 (application)</td>
<td>fix(t) (fixpoint)</td>
</tr>
<tr>
<td></td>
<td>x (fresh)</td>
<td>if t1 &lt; t2 then t3 else t4 (less than)</td>
</tr>
<tr>
<td></td>
<td>if t1 = t2 then t3 else t4 (integer equality)</td>
<td>case (\text{inl}(x)) of (\text{inr}(y)) ⇒ (t3) (decide)</td>
</tr>
</tbody>
</table>

\[(\lambda x.F)\ a \quad \Rightarrow \quad F[x\backslash a] \quad \text{if } i_1 = i_2 \text{ then } t_1 \text{ else } t_2 \Rightarrow t_3 \quad \text{if } i_1 \neq i_2 \]

\[\text{fix}(v) \quad \Rightarrow \quad v \text{ fix}(v) \quad \text{if } i_1 = i_2 \text{ then } t_1 \text{ else } t_2 \Rightarrow t_2 \quad \text{if } i_1 \neq i_2 \]

\[\text{let } x = v \text{ in } t \quad \Rightarrow \quad t[x\backslash v] \quad \text{if } i_1 < i_2 \text{ then } t_1 \text{ else } t_2 \Rightarrow t_1 \quad \text{if } i_1 > i_2 \]

\[\text{case inl}(t) \text{ of } \text{inl}(x) \Rightarrow F \quad \text{if } \text{inr}(y) \Rightarrow G \Rightarrow F[y \backslash t] \quad \text{case inr}(t) \text{ of } \text{inl}(x) \Rightarrow F \quad \text{if } \text{inr}(y) \Rightarrow G \Rightarrow G[y \backslash t] \]

applied (with pairs, injections, a fixpoint operator, . . . ) λ-calculus. For efficiency, integers are primitive and Nuprl provides operations on integers such as addition, subtraction, . . . , a test for equality and a less than operator.

A term is either a variable, a value (or canonical term), or a non-canonical term. Non-canonical terms have one or two principal arguments (marked using boxes in Fig. 1). A principal argument of a term \(t\) is a term that has to be evaluated to a canonical form before checking whether \(t\) can be reduced further. For example the application \(f(a)\) diverges if \(f\) diverges.

Fig. 1 also shows part of Nuprl’s small-step operational semantics. We omit the rules that reduce principal arguments such as: if \(t_1 \rightarrow t_2\) then \(t_1 \rightarrow t_2\). As usual, \(\Rightarrow^*\) is the reflexive and transitive closure of \(\Rightarrow\), and \(t_1 \rightarrow^k t_2\) is defined inductively on \(k\): \(t \rightarrow^0 t\); and \(t_1 \rightarrow t_2\) if \(t_1 \rightarrow t\) and \(t \rightarrow^k t_2\) for some term \(t\).

We now define a few abstractions that will be used below:

\[\perp = \text{fix}(\lambda x.x)\]
\[tt = \text{inl}(\_\_)\]
\[ff = \text{inr}(\_\_)\]
\[\pi_1(a) = \text{let } x, y = a \text{ in } x\]
\[\pi_2(a) = \text{let } x, y = a \text{ in } y\]
\[a \leq b \quad \Rightarrow \quad \text{if } a < b \text{ then } tt \text{ else if } a = b \text{ then } tt \text{ else } ff\]

Also, we write: \(a \equiv r\) for the type \(a : b \in T\); \(b\) for \(\text{if } b\ \text{then } \text{Unit}\ \text{else } \text{Void}\); \((\text{let } x_1, \ldots, x_n = p \text{ in } t)\) for \(\text{let } x_1, z = p \text{ in } \ldots \text{let } x_{n-1}, x_n = z \in t\), where \(z\) is a fresh variable w.r.t. \(x_1, \ldots, x_n, t\); and \(\lambda x_1, \ldots, x_n.t\) for \(\lambda x_1.\ldots.\lambda x_n.t\).

2.2 Nuprl’s Type System

Following Allen’s PER semantics, Nuprl’s types are defined as partial equivalence relations (PERs) on closed terms [18]. Allen’s PER semantics can be seen as an inductive-recursive definition of: (1) an inductive relation \(T_1 ≡ T_2\) that expresses type equality; and (2) a recursive function \(a ≡ b ∈ T\) that expresses equality in a
type. Among other things, it follows that the theoretical proposition \(a = b \in T\) is true (inhabited by \(\star\)) iff \(a \equiv b \in T\) holds in the metatheory (see \(\llbracket 3, 4 \rrbracket\)).

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 \(\llbracket 3, 4 \rrbracket\). 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 \(\llbracket 3, 4 \rrbracket\).

The top part of Fig. \(\Delta\) lists some of NuPRL’s types. Among these, \texttt{Base} is the type of all closed terms of the computation system with \(\sim\) as its equality. Names \(\llbracket 3, 4 \rrbracket, \llbracket 3, 5 \rrbracket\) come with two operations: a fresh operator \(\nu\) to generate fresh names, and a test for equality (not shown here). We use names to, among other things, validate a continuity inference rule \(\llbracket 50 \rrbracket\).

As mentioned above, we have implemented NuPRL’s term language, its computation system, Howe’s \(\sim\) relation, and Allen’s PER semantics in Coq \(\llbracket 3, 4 \rrbracket\). We have also shown 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 \texttt{False} is not inhabited. Using these two facts, we derive that there cannot be a proof derivation of \texttt{False}, i.e., NuPRL is consistent (see \(\llbracket 3, 4 \rrbracket, \llbracket 5 \rrbracket\), 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.

3 Squashing and Bootstrapping BI

This section presents an unconstrained squashed BI principle; explains how we can derive versions of BID and BIM from this squashed BI principle using bar recursion operators; and proves the negation of a non-[ ]-squashed version of BIM.

3.1 Squashing

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 \(\llbracket 22 \rrbracket\), pp.60). The only member of this type is the constant \(\star\), which is \texttt{unit}’s single inhabitant, and which is similar to \(\mathcal{O}\) 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 \(\llbracket 51 \rrbracket\), Appx F] for more information regarding squashing. Using the HoTT terminology, we also sometimes truncate types at the propositional level \(\llbracket 53 \rrbracket\), 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 \(\llbracket 24 \rrbracket\): \(\downarrow T = T/\text{true}\). \(\downarrow T\) is a proof-irrelevant type. Its members are the members of \(T\), and they are all equal to each other in \(\downarrow T\) because if \(x, y \in T\) then \((x =_T y) \iff \text{true}\). Note that the implication \(\downarrow T \to \downarrow T\) is true because it is inhabited by \(\lambda x.\star\), but we cannot prove the converse because to prove \(\downarrow T\) we have to exhibit an inhabitant of \(T\), which \(\downarrow T\) does not give us because only \(\star\) inhabits \(\downarrow T\).
3.2 Squashed Unconstrained Bar Induction

As mentioned above, the unconstrained unsquashed BI principle is not consistent with constructive mathematics. However, it is consistent when proving ↓-squashed propositions. We prove in this paper the validity of inference rules of the following form (which we call [BarInduction]):

\[
\begin{align*}
\text{(ufd)} & \quad H, n : \mathbb{N}, s : T^{\mathbb{N}} \vdash B \quad n \ s \in \text{Type} \\
\text{(bar)} & \quad H, s : T^{\mathbb{N}} \vdash \text{Σ}_n N \ B \quad n \ s \\
\text{(base)} & \quad H, n : \mathbb{N}, s : T^{\mathbb{N}}, b : B \quad n \ s \vdash P \quad n \ s \\
\text{(ind)} & \quad H, n : \mathbb{N}, s : T^{\mathbb{N}}, i : (\Pi m : T, P \ (n + 1) \ (s \oplus_n m)) \vdash P \quad n \ s
\end{align*}
\]

\[H \vdash \downarrow(P \ 0 \ \text{norm}(c, 0))\]

where \(T\) is \(\mathbb{N}\) in Sec. 4.1 and the type of closed terms without names in Sec. 4.2.

The conclusion of the bar hypothesis is ↓-squashed because the bar is sometimes only used for termination, as in BID, and in that case it does not contribute to the extract, i.e., to its computational content.

Also, let \(\text{norm}(s, n) = \lambda x.\text{if } x < 0 \text{ then } \bot \text{ else if } x < n \text{ then } s(x) \text{ else } \bot\). This normalization operator returns \(s(x)\) for \(x \in \{0, \ldots, n\}\), and otherwise returns \(\bot\). This is to avoid requiring one to also prove that \(P\) is a well-formed predicate on finite sequences, i.e., of type \(\Pi n : \mathbb{N}. T^{\mathbb{N}} \rightarrow \text{Type}\). This is undesirable because we sometimes want to use this rule to prove that \(P\) is indeed well-formed.

For example, we derive below BI principles for non-↓-squashed propositions by proving that some bar recursion operator \(br\) inherits some proposition \(Q\), i.e., \(br \in Q\), using our ↓-squashed BI principle. The proposition \(br \in Q\) is ↓-squashed because Nuprl’s equality types can only be inhabited by the constant \(*\). We might not be able to prove that \(br \in Q\) is well-formed because we might again need to use bar induction. However, we require the bar \(B\) to be well-formed as stated by the subgoal called uf.

\(P\) being a well-formed predicate on finite sequences would allow us, in the proof that [BarInduction] is a valid rule, to sometimes replace a sequence \(s_1\) of type \(T^{\mathbb{N}}\) by another sequence \(s_2\) of type \(T^{\mathbb{N}}\) in an expression of the form \((P \ n \ s_1)\), given that \(s_1 = s_2 \in T^{\mathbb{N}}\). However, when sequences are normalized using norm, two sequences such as \(\text{norm}(s_1, n)\) and \(\text{norm}(s_2, n)\) are then computationally equivalent (see Sec. 22), which allows us to substitute \(\text{norm}(s_1, n)\) for \(\text{norm}(s_2, n)\) in \((P \ n \ \text{norm}(s_1, n))\) without having to prove, e.g., that \(P\) is a well-formed predicate on finite sequences.

Let us now introduce a few variable names that will be used below to define bar recursion operators, and which correspond to the hypotheses of BID and BIM. We provide a list of such terms along with their types:

\[
\begin{align*}
\text{base} & \quad \Pi n : \mathbb{N}. \Pi s : T^{\mathbb{N}}, B \quad n \ s \rightarrow P \quad n \ s \\
\text{bar} & \quad \Pi s : T^{\mathbb{N}}. \text{Σ}\ n : \mathbb{N}. \ B \quad n \ s \\
\text{ind} & \quad \Pi n : \mathbb{N}. \Pi s : T^{\mathbb{N}}. (\Pi m : T, P \ (n + 1) \ (s \oplus_n m)) \rightarrow P \quad n \ s \\
\text{dec} & \quad \Pi n : \mathbb{N}. \Pi s : T^{\mathbb{N}}. (B \quad n \ s) \lor \neg (B \quad n \ s) \\
\text{mon} & \quad \Pi n : \mathbb{N}. \Pi s : T^{\mathbb{N}}. \Pi t : T. (B \quad n \ s) \rightarrow (B \quad (n + 1) \ s \oplus_n t) \\
\text{mon’} & \quad \Pi n : \mathbb{N}. \Pi m : \mathbb{N}. \Pi s : T^{\mathbb{N}}. (B \quad m \ s) \rightarrow (B \quad n \ s)
\end{align*}
\]
Note that the $\Sigma$ type in bar’s type is $\bot$-squashed and not $\bot$-squashed as in [BarInduction] because in Sec. 3.6 we need the bar hypothesis to have some computational content to build a realizer for BIM.

### 3.3 Spector’s Parametrized Bar Recursion Operator

Spector first introduced a bar recursion operator, called SBR here, in order to provide a consistency proof of classical analysis relative to system T extended with this bar recursion operator. Spector mentioned some relation between SBR and BID, and later Howard showed that his W operator (see pp.111), which can be reduced to SBR, realizes BIM (see Sec. 4.4). SBR can be defined as the following parametrized operator (Spector’s operator uses $\leq$ instead of $\leq_\bot$):

$$SBR(Y, G, H, n, s) = \begin{cases} Y n s \leq n \text{ then } G n s \\ H n s \ (\lambda t. SBR(Y, G, H, n + 1, s \oplus n, t) \end{cases}$$

Spector used a restricted form of SBR to interpret the double-negation shift, which he used in his consistency proof. Oliva and Powell later proved that this restricted form of SBR is in fact as general as SBR. Informally, given a barred spread, $Y$ tells us whether we have reached the bar. If we have reached the bar, then we can use the base operator $G$, and otherwise we use the inductive operator $H$. See Sec. 4.4, pp.9 for an explanation of why continuity implies that the recursion terminates. Also, note that this implies that checking whether we have reached the bar has to be decidable. As mentioned in [52, pp.9, Footnote 6], and as further explained in Sec. 4.6 this can be ensured by the fact that we can compute the modulus of continuity of the bar.

### 3.4 Nuprl’s Decidable Bar Recursion Operator

Using an instance of SBR we now prove a BID principle, which is both more general than the one presented in Sec. 3.2 in the sense that it is for non-squashed propositions, and less general because the bar has to be decidable. We prove this principle directly in Nuprl by proving that it is realized by the following decidable bar recursion operator, parametrized by a $n \in \mathbb{N}$ and a $s \in T^{\mathbb{N}}$:

$$DBR(n, s) = \begin{cases} \text{case } dec n s \text{ of inl}(r) \Rightarrow base n s r \\ \text{linl( ) } \Rightarrow ind n s (\lambda t. DBR(n + 1, s \oplus n, t) \end{cases}$$

More precisely, it inhabits $(P \ n \ \text{norm}(s, n))$, for some $n \in \mathbb{N}$ and finite sequence $s \in T^{\mathbb{N}}$, assuming the hypotheses of [BarInduction] and assuming that the bar is decidable. As mentioned above, $dec$ inhabits the proposition that says that the bar is decidable. Given a finite sequence provided by a number $n$ and a sequence $s$, if $(dec n s)$ returns inl($r$), then $r$ is a proof that $(B n s)$ is true. In that case, we use our base hypothesis base. If $(dec n s)$ returns inl($r$) then we are not at the bar and in that case we use our induction hypothesis ind.

As mentioned above, DBR is an instance of SBR:

$$DBR(n, s) = SBR(\lambda n, s. if \ dec n s \ then \ 0 \ else \ n, base, ind, n, s)$$
3.5 Continuity

Brouwer's continuity principle is used below to, among other things, define an instance of Howard's W operator in Sec. 3.6. We use the following variant of Brouwer's continuity principle, which is sometimes called the strong continuity principle for numbers [53], which we have proved to be valid w.r.t. Nuprl's semantics, and which can easily be derived from the one presented in [50].

\[ \text{SCP} = \Pi P: (B \to \mathbb{N} \to \mathbb{P}). \]

\[ \Pi f : B. \Sigma n : \mathbb{N}. P f n \]

\[ \Rightarrow | \Sigma M: (\Pi n : \mathbb{N}. B_n \to (\mathbb{N}_n + \text{Unit})). \]

\[ \Pi f : B. \Sigma n : \mathbb{N}. \Sigma k : \mathbb{N}_n. P f k \land M n f =_{\mathbb{N}_n + \text{Unit}} \text{inl}(k) \land \Pi m : \text{Nisl}(M m f) \to m =_{\mathbb{N} n} \]

SCP says that there is a uniform way (called \( M \) in the formula—such a function is often called a neighborhood function [64, pp. 212]) to decide whether \( n \) is the modulus of continuity of \( P \) at \( f \), and if so returns the only number \( n \) such that \( (P f n) \) is less than \( n \). This version of SCP differs from the one in [50] as follows: (1) here we assume the existence of a predicate that relates numbers and sequences using a \( \mathbb{N} \)-squashed \( \Sigma \) type, while in [50] we assume the existence of a function; and (2) here \( M \) is of type \((\Pi n : \mathbb{N}. B_n \to (\mathbb{N}_n + \text{Unit})) \) instead of \((\Pi n : \mathbb{N}. B_n \to (\mathbb{N} + \text{Unit})) \) in [50], i.e., we are guaranteed that the modulus of continuity \( n \) of \( P \) at \( f \) that \( M \) returns will be larger than the value \( k \) such that \( (P f k) \) is true (or taking \( P \) as a function as in [50], that \( P f < n \))—this is useful to define \text{HBR} in Sec. 3.6. As mentioned by Bridges and Richman [19, pp. 119], SCP is equivalent to a "principle of continuous choice", which they divide into a continuous part, namely the axiom of choice often referred to as \text{AC}_{1,0}, which we have proved to be true in Nuprl (see Nuprl lemma [axiom-choice-0X-quot]):

\[ \text{WCP} = \Pi F : \mathbb{N}^B. \Pi f : B. \Sigma n : \mathbb{N}. \Pi g : B. f =_{B_n} g \rightarrow F(f) =_{\mathbb{N}} F(g) \]

\[ \text{AC}_{0,1} = \Pi n : \mathbb{N}. \Sigma f : B. P n f \Rightarrow (\Sigma f : B^B). \Pi n : \mathbb{N}. P n f (n) \]

As first shown by Kreisel in [41, pp. 154], continuity is not an extensional property in the sense that it does not map equal arguments to equal values. Therefore, the existence of \( M \) in SCP has to be truncated. Troelstra later showed in [54, Thm IIIA] that \( \text{N-HA}^\omega \) (a "neutral" version of \( \text{HA}^\omega \) that "permits extensional as well as intensional interpretations of equality at higher types" [53]) extended with (1) Brouwer's continuity principle, (2) a function extensionality axiom, and (3) a version of the axiom of choice \( \text{AC}_{2,0} \), is inconsistent. Escardó and Xu [29] proved in Agda, without using function extensionality but allowing reductions under \( \lambda \), that the non-truncated version of \text{WCP} is false in a Martin-Löf-like type theory such as Nuprl.

3.6 Howard’s Monotone Bar Recursion Operator

A few years after Spector [59] introduced his bar recursion operator, Howard [32] showed that some instance of it, which he called W, realizes BIM, and of which we present an instance here called \text{HBR}. Let the parameter \( T \) from Sec. 3.2 be
\[ \mathbb{N} \] here, i.e., we only consider sequences of numbers. Our setting is less general than Howard’s because the continuity principle presented in Sec. 3.5 is only for sequences of numbers. Howard does not explicitly mention continuity. However, Spector mentions continuity in [59, pp.9,Footnote 6], where the modulus of continuity of the bar ensures that each infinite sequence has an initial segment that is long enough so that we can check where the sequence is barred. Because SCP is \( | \)-squashed, assuming that the proposition we are proving by bar induction is \( | \)-squashed too, then SCP gives us a \( M \in \Pi n : \mathbb{N}.B_n \to (\mathbb{N} + \text{Unit}) \) such that:

\[
 F \in \Pi f : B. \Sigma n : \mathbb{N}. \Sigma k : \mathbb{N}. B f k \land M n f =_{\mathbb{N} + \text{Unit}} \text{inl}(k) \\
\land \forall m : \mathbb{N}. \text{isl}(M m f) \to m =_{\mathbb{N}} n
\]

If \( B \) is a well-formed predicate on finite sequences, i.e., of type \( \Pi n : \mathbb{N}. T^{n}_m \to \text{Type} \), then HBR can be stated as follows:

\[
\text{HBR}(n, s) = \text{let } x, z, p, q = F(s \uparrow^0_n) \text{ in} \\
if x \leq n \text{ then base } n s (\text{mon } z n (s \uparrow^0_n) p) \\
else \text{ind } n s (\lambda t. \text{HBR}(n+1, s \oplus_n t))
\]

where \( s \uparrow^0_n = \lambda x. \text{if } x < n \text{ then } s(x) \text{ else } k \). As mentioned above HBR is an instance of SBR: \( \text{HBR}(n, s) = \text{SBR}(\lambda n, s. \pi_1(F(s \uparrow^0_n)), \text{hbase}, \text{ind}, n, s) \), where \( \text{hbase} = \lambda n, s. \text{let } x, z, p, q = F(s \uparrow^0_n) \text{ in base } n s (\text{mon } z n (s \uparrow^0_n) p) \).

### 3.7 Bootstrapping

As mentioned in [10], using the \( \downarrow \)-squashed BI principle presented in Sec. 3.2 we can prove that DBR inhabits an unsquashed BID principle. However, using HBR we cannot prove an unsquashed BIM principle because it uses SCP which is \( | \)-squashed. We can only prove that HBR inhabits a \( | \)-squashed BIM principle. Does that mean that, using BIM, one can only prove \( | \)-squashed propositions? We partly answer this question below.

First, we show that we can derive a slightly more general BIM principle than the standard one, which is only for \( | \)-squashed propositions. This principle, which we call BIM\(_{\text{up}}\), is inspired by the way Howard’s W operator works (see Nuprl lemma [20.2b.1.17.8]):

\[
\Pi P : (\Pi n : \mathbb{N} . B_n \to \mathbb{P}) \\
(\Pi s : B . \Sigma n : \mathbb{N} . \Pi m : \{ n \ldots \} . \text{P m s}) \\
\Rightarrow (\Pi n : \mathbb{N} . \Pi s : \mathbb{N}^{\mathbb{N}} . (\Pi m : \mathbb{N} . \text{P (n + 1) (s \oplus_n m)}) \Rightarrow \text{P n s}) \\
\Rightarrow \Pi s : \text{Top} . \text{P 0 s}
\]

where we write \( \{ n \ldots \} \) for the type \( \{ k : \mathbb{N} \mid n \leq k \} \). Note that here we assume that \( P \) is a well-formed predicate on finite sequences. Let us mention the differences with a more “standard” version of BIM. BIM is usually stated using two predicates on finite sequences: a predicate \( B \) that represents the bar; and a predicate \( P \) which we are proving by induction. Here, we do not have the predicate \( B \) that represents the bar, because \( P \) itself represents the bar. Also, here \( P \) has to be true at the bar and above the bar, whereas in the “standard” BIM principle the bar predicate \( B \) has to be true at the bar and monotone below, at, and above the bar. We can easily prove that BIM\(_{\text{up}}\) implies the more “standard”
BIM principle stated for well-formed predicates on finite sequences (see Nuprl lemma \[\text{monotone-bar-induction8-implies-3}\]), which we simply call BIM here:

\[
\begin{aligned}
\Pi B, P : (\Pi n : \text{N} . B_n \to P).
\sigma : B, \quad (\Pi n : \text{N} . B n s)
\Rightarrow (\Pi n : \text{N} . B_n s) . (\Pi m : \text{N} . P (n + 1) (s \oplus n m)) \Rightarrow P n s
\Rightarrow (\Pi n, m : \text{N} . \Pi s : B_n s . B n \Rightarrow B (n + 1) (s \oplus n m))
\Rightarrow (\Pi n : \text{N} . \Pi s : B_n s . B n \Rightarrow P n s)
\Rightarrow \Pi s : \text{Top} . P 0 s
\end{aligned}
\]

3.8 Negation of Unsquashed BIM

We can also prove that the following unsquashed version of BIM, which we call uBIM, is false (see Nuprl lemma \[\text{unsquashed-monotone-bar-induction3-implies-3}\]), which, as mentioned in Sec. 3.5, we can prove to be false in Nuprl, i.e.:

\[
\neg \Pi F : \text{N}^\text{R} . \Pi f : B . \Sigma n : \text{N} . \Pi g : \text{B} . f =_{\text{B}_n} g \Rightarrow F f =_{\text{N}} F(g)
\]

Because the unsquashed version of BIM$_{\text{up}}$ implies uBIM, which in turn implies the unsquashed continuity principle, and because the unsquashed continuity principle is false, we get that both the unsquashed version of BIM$_{\text{up}}$ and uBIM are also false. The proof that uBIM implies an unsquashed version of WCP goes as follows: we assume that $F \in \text{N}^\text{R}$ and $f \in \text{B}$, and we have to prove: $\Sigma n : \text{N} . \Pi g : \text{B} . f =_{\text{B}_n} g \Rightarrow F f =_{\text{N}} F(g)$. Finally, we instantiate uBIM with:

\[
\begin{aligned}
B &= \lambda n, s . \Pi g : \text{B} . (s \oplus n f) =_{\text{B}_n} g \Rightarrow F(s \oplus n f) =_{\text{N}} F(g)
\text{P} &= \lambda n, s . \Sigma m : \{ \ldots \} . \Pi g : \text{B} . (s \oplus n f) =_{\text{B}_n} g \Rightarrow F(s \oplus n f) =_{\text{N}} F(g)
\end{aligned}
\]

where $s \oplus n f = \lambda x . \text{if } x < n \text{ then } s(n) \text{ else } f(x)$. We can then easily prove the hypotheses of uBIM. For example, the bar hypothesis follows from the |-squashed WCP principle, which is why we require it to be |-squashed.

One question that now remains open is: can we prove the validity of an unsquashed version of BIM$_{\text{up}}$ or of the “standard” BIM principle, where the bar is not squashed? This is left for future work.

4 Verifying the Validity of Bar Induction Inference Rules

In this section we prove the validity of BI inference rules w.r.t. Nuprl’s PER semantics (the formal proofs are available at \[\text{https://github.com/vrahl/NuprlInCoq}\]). Sec. 4.1 proves that our \[\text{[BarInduction]}\] rule is valid when $T = \text{N}$. Sec. 4.2 proves the validity of this rule for sequences of closed terms without names.
4.1 Bar Induction for Sequences of Natural Numbers

Following the Standard Classical Proof. We have proved that the above mentioned rule is true in our impredicative metatheory, i.e., in Prop, following Dummett’s standard classical proof [28, pp. 55], which uses the law of excluded middle and the axiom of choice. His proof goes as follows: first we assume the negation of the conclusion using the law of excluded middle, i.e., the Coq axiom \texttt{classic} (available at \url{https://coq.inria.fr/library/Coq.Logic.Classical_Prop.html}). Then, we contrapose our \texttt{(ind)} hypothesis, and using the axiom of choice \texttt{FunctionalChoice_on} (available at \url{https://coq.inria.fr/library/Coq.Logic.ChoiceFacts.html}) we obtain a function \( F \) that, for all \( n \in \mathbb{N}, s \in B_n, \) and proof of \( \neg(P \ n \ s), \) returns a natural number \( m \) such that \( \neg(P \ (n + 1) \ (s \oplus_n m)) \). Because \( \neg(P \ 0 \ \texttt{norm}(c, 0)) \), this gives us a sequence \( \alpha \in B \) such that for all \( n \in \mathbb{N}, \neg(P \ n \ \texttt{norm}(\alpha, n)) \). We now instantiate our \texttt{(bar)} hypothesis with \( \alpha \) to get a number \( k \) such that \( (B \ k \ \alpha) \). Because \( B \) is a well-formed predicate on finite sequences (hypothesis \texttt{(wfd)}), we get a proof of \( (B \ k \ \texttt{norm}(\alpha, k)) \). Finally, using our \texttt{(base)} hypothesis, we get a proof of \( (P \ k \ \texttt{norm}(\alpha, k)) \), which contradicts that for all \( n \in \mathbb{N}, \neg(P \ n \ \texttt{norm}(\alpha, n)) \).

Adding Coq Sequences to Nuprl. How did we construct the sequence \( \alpha \)? \( F \) gives us a Coq function from numbers to numbers, but our proof needs a Nuprl term in the Nuprl type \( B \). To remedy that we added all Coq functions from numbers to numbers to Nuprl’s computation system, even those that make use of axioms such as \texttt{classic} and \texttt{FunctionalChoice_on}, and which are therefore not computable. This coincide with the fact that functions on numbers should not be restricted to general recursive functions for BI to be true [33, Lem.9.8]. Our choice sequences are here Coq’s functions from numbers to numbers. They are similar to the infinite sequences in [12] denoted \( \lambda x. M_x \), where \( M_1, M_2, \ldots \), is an infinite sequence of terms, which are used, in a similar fashion as above, to prove that some bar recursion operator denoted \( \Phi \) realizes the negative translation of the axiom of choice. Our choice sequences are also similar to the set-theoretical functions in [34, 37, 38] (also called “oracles”), which are used to provide a set-theoretical semantics of both Nuprl (extended with set-theoretical terms) and HOL, therefore allowing the shallow embedding of HOL in Nuprl. Therefore, we extend Nuprl’s term syntax presented in Sec 9 with sequences, as well as an \texttt{eager} application operator:

\[
v ::= \cdots \mid \texttt{seq}(f) \quad \text{(choice sequences)}
\]

\[
t ::= \cdots \mid \texttt{seq}(f) \circ t \quad \text{(eager application)}
\]

where \( f \) is a Coq function from numbers to numbers. We also add the following reduction steps to compute with sequences:

\[
\texttt{seq}(f) \circ t \mapsto \texttt{seq}(f) \circ t
\]

i.e., the \texttt{lazy} application of a sequence \( s \) to a term \( t \) computes in one step to the \texttt{eager} application of \( s \) to \( t \). Eager applications compute as follows:

\[
t_1 \circ t_2 \mapsto t_2 \circ t_1 \quad \text{if} \quad t_1 \mapsto t_2 \quad \text{(Ax.b)} @ v \mapsto b [x \mapsto v]
\]

\[
v \circ t_1 \mapsto v \circ t_2 \quad \text{if} \quad t_1 \mapsto t_2 \quad \text{seq}(f) \circ i \mapsto f(i) \quad \text{if} \quad 0 \leq i
\]

where \( f \) is a Coq function from numbers to numbers, \( i \) is a Nuprl integer, and \( v \) is a value. In the last computation step above, we write \( f(i) \) for the computation
that extracts a Coq natural number \( n \) from the positive integer \( i \), then applies \( f \) to \( n \), and finally builds a Nuprl integer from the Coq natural number \( f(n) \).

**A Note on Decidability.** Adding such choice sequences to Nuprl’s terms does have interesting consequences such as: many properties become undecidable. For example, syntactic equality or \( \alpha \)-equality are now undecidable in general. However, it turns out that even though we had proved and used these properties in our Coq development, they are not necessary and we managed to not use them.

**Consistency.** Adding Coq sequences of numbers to Nuprl’s terms also affected Nuprl’s consistency, i.e., some inference rules had to be modified—one to be exact. The rule in question is the following [ApplyCases] rule:

\[
\frac{H \vdash \text{halts}(f(a)) \quad H \vdash f \in \text{Base}}{H \vdash f \simeq \lambda x.f(x)}
\]

where \( \text{halts}(t) = \star \preceq (\text{let } x := t \text{ in } \star) \) asserts that \( t \) computes to a value. This rule says that \( f \) is computationally equivalent to its \( \eta \)-expansion \( \lambda x.f(x) \) (i.e. \( f \) is a function) if \( f(a) \) computes to a value, for some term \( a \). Before adding Coq sequences of numbers to Nuprl’s terms, the only way \( f(a) \) could compute to a value was if \( f \) would compute to a \( \lambda \)-term. This is not true anymore after adding choice sequences to Nuprl’s terms. We chose to restate [ApplyCases] as follows:

\[
\frac{H \vdash \text{halts}(f(a)) \quad H \vdash f \in \text{Base}}{H \vdash f \simeq \lambda x.f(x) \lor \forall x: \text{Base.} \lor \exists x: \text{halts}(x) \cdot \text{ifint}(x, \text{True}, f(x) \preceq \bot) \cdot \text{extend iflam}(f, \text{tt}, \text{ff})}
\]

where \( x \) and \( z \) are distinct variables that do not occur free in \( f \). Only the conclusion of the rule has changed. This rule says that if \( f(a) \) computes to a value then either (1) \( f \) computes to a \( \lambda \)-term (as before), or (2) it computes to a choice sequence, and therefore \( f(x) \) will be computationally equivalent to \( \bot \) when \( x \) is not an integer, i.e., it will either get stuck or diverge. This rule also says that the conclusion, which is a \( \lor \), is realized by \( \text{iflam}(f, \text{tt}, \text{ff}) \), which checks whether \( f \) computes to a \( \lambda \)-term: if it does then the conclusion is realized by \( \text{tt} \), i.e. \( \text{inl}(\star) \), because \( \star \) realizes the left-hand-side of the \( \lor \); otherwise, the conclusion is realized by \( \text{ff} \), i.e \( \text{inr}(\star) \), because \( \star \) realizes the right-hand-side of the \( \lor \). Using this new valid rule, we were able to replay the entire Nuprl library.

### 4.2 Bar Induction For Sequences of Closed Terms Without Names

Intuitively a similar proof as the one presented at the beginning of Sec. 1.1 could be used at least when \( T \) is Base (defined in Sec 2.2). Following the same scheme as in Sec. 1.1 we want to add all Coq functions from natural numbers to closed terms, to the collection of Nuprl terms. However, this modification does not play nicely with our “fresh" \( \nu \) operator. We explain this issue here in more details.

**Banning Names From Choice Sequences** Let us assume that we change our choice sequence operator seq(f) so that \( f \) can be a Coq function from numbers to closed Nuprl terms. The Coq function \( \text{fun } n \Rightarrow a \), where \( a \) is a name, is such a function. In general we cannot compute the collection of all names occurring in such functions. Therefore, unless we somehow tag this function with \( a \), we have no way of knowing that it mentions \( a \). Now, the way our \( \nu \) operator works,
as explained in \[^{50}\], is that to compute \(\nu x.t\), if \(t \rightarrow u\), we first pick a fresh name \(b\) w.r.t. \(t\) (i.e., if \(b\) occurs in \(t\) then it only occur in a sequence). Then, we compute \(t[x\backslash b]\) to \(w\) in one computation step, and finally return \(\nu x.u[b\backslash x]\), where \(t[a\backslash u]\) is a capture avoiding substitution function on names similar to the usual substitution operation on variables. Therefore, if \(t\) contains \(\text{seq}(\text{fun} \: n \Rightarrow a,\) we have to make sure that we do not pick \(a\). Otherwise, when computing \(\nu x.(\text{seq}(\text{fun} \: n \Rightarrow a) \: 0)\), we could pick \(a\) as our fresh name, reduce \(\text{seq}(\text{fun} \: n \Rightarrow a) \: 0\) to \(a\), perform the substitution \(a[a\backslash x] = x\), and finally return \(\nu x.x\), which would not be correct because the two \(a\)s are supposed to be different.

We avoid this here by precluding names from occurring in sequences, and change our choice sequence operator \(\text{seq}(f)\) so that \(f\) is a Coq function from numbers to closed Nuprl terms without names. This means that the Coq type of Nuprl terms is now an ordinal with a limit constructor for such sequences. However, because sequences cannot contain free variables or names, most operations on terms do not change because our two substitution operators do not change. Using these sequences, we have proved the validity of \([\text{BarInduction}]\) when the parameter \(T\) is the type of closed terms without names closed under \(\sim\).

**Can We Add Back Names to Choice Sequences?** We suggest here a possible solution, whose study is left for future work. It consists in introspecting computations. When performing a computation step on a term of the form \(\nu x.t\), we first pick a fresh name \(a\) w.r.t. \(t\) by not looking inside sequences, then we reduce \(t[x\backslash a]\) to \(u\) in one computation step, and we compute a new fresh name \(b\) w.r.t. \(u\). This is to ensure that if the computation step applies a sequence to a term and “reveals” new names, then \(b\) is not one of these names. Finally, we compute \(\nu x.t\) using \(b\) as our fresh name.

## 5 Related Work

Howard and Kreisel studied Brouwer’s bar induction and continuity principles in \[^{32}\] and showed among other things the equivalence between the axiom of transfinite induction (TI, also sometimes called the bar rule \[^{50}\]) and BIM, assuming the strong continuity principle. They also showed without assuming continuity that TI for decidable relations is equivalent to BID. TI says that one can use the transfinite induction principle on well-founded relations. In Coq, TI is simply a lemma called \texttt{well} \_\texttt{founded} \_\texttt{ind} for Prop and \texttt{well} \_\texttt{founded} \_\texttt{induction} \_\texttt{type} for Type; see the Coq library \url{https://coq.inria.fr/library/Coq.Init.Wf.html}.

The bar recursion operators mentioned in Sec. 4 and some of their variants have been extensively studied \[^{54, 12, 13, 14, 34, 30}\]. However, to the best of our knowledge, it has not been studied whether these variants (such as Berger and Oliva’s modified bar recursion operator \[^{13}\]) lead to new BI principles.

Treustra provides a list of uses of BI in \[^{62}\ pp. 114\], such as to prove strong normalization theorems for the terms of systems such as N-HA\(^\omega\). Veldman and Bezerra proved (an intuitionistically valid reformulation of) Ramsey’s theorem in \[^{21}\] using BIM (see also \[^{68}\]). In \[^{71}\], the authors proved similar results using directly Coq’s inductive types rather than BI.
6 Conclusion, Current and Future Work.

Using our Coq framework, we have recently proved a $\mid$-squashed version of Brouwer's continuity principle for numbers [51]. We have now also proved the validity of a $\downarrow$-squashed BI inference rule for sequences of closed terms without names. From this $\downarrow$-squashed BI rule, we have derived an unsquashed version of BID for sequences closed terms without names, as well as a $\mid$-version of BIM for sequences of numbers (because our version of SCP is only for sequences of numbers). We have also shown that this BIM principle is not true in general for non-$\mid$-squashed propositions. Several questions remain open such as: (1) Can we generalize our $\mid$-squashed continuity principle to sequences of terms? (2) Can we generalize our $\downarrow$-squashed BI principle to sequences of terms (with names)? (3) What is the proof-theoretical strength of Nuprl? Is it stronger than before adding choice sequences or bar induction?

Acknowledgements We thank Evan Moran for his helpful criticism.

References


