Bar Induction: The Good, the Bad, and the Ugly

Vincent Rahli
SnT, University of Luxembourg
Email: vincent.rahli@gmail.com

Mark Bickford
Cornell University, USA
Email: markb@cs.cornell.edu

Robert L. Constable
Cornell University, USA
Email: rc@cs.cornell.edu

***Abstract*—We present an extension of the computation system and logic of the Nuprl proof assistant with intuitionistic principles, namely versions of Brouwer's bar induction principle, which is equivalent to transfinite induction. We have substantially extended the formalization of Nuprl's type theory within the Coq proof assistant to show that two such bar induction principles are valid w.r.t. Nuprl's semantics (the Good): one for sequences of numbers that involved only minor changes to the system, and a more general one for sequences of name-free (the Ugly) closed terms that involved adding a limit constructor to Nuprl's term syntax in our model of Nuprl's logic. We have proved that these additions preserve Nuprl's key metatheoretical properties such as consistency. Finally, we show some new insights regarding bar induction, such as the non-truncated version of bar induction on monotone bars is intuitionistically false (the Bad).**

I. Introduction

Nuprl. The Nuprl interactive theorem prover [25; 5] implements a type theory called *Constructive Type Theory* (CTT), which is a dependent type theory, in the spirit of Martin-Löf's extensional theory [48], based on an untyped functional programming language. Its types include equality types, a hierarchy of universes, W types, quotient types [26], set types, union and (dependent) intersection types [43], image types [50], PER types [6], approximation and computational equivalence types [60], and partial types [66; 29]. CTT "mostly" differs from other similar constructive type theories such as the ones implemented by Agda [17; 1], Coq [13; 28], or Idris [18; 41], in the sense that CTT is an *extensional* theory (i.e., propositional and definitional equality are identified [34]) with types of partial functions [66; 27; 29]. For example, the fixpoint $\mathtt{fix}(\lambda x.x)$ diverges. It is nonetheless a member of types such as the partial type $\overline{\mathbb{Z}}$—the type of integers and diverging terms. In Nuprl, type checking is undecidable but in practice this is mitigated by type inference/checking heuristics implemented as tactics. Following Allen's semantics [3; 4], CTT types are interpreted as Partial Equivalence Relations (PERs) on closed terms, and we have formalized this semantics in Coq [7; 51].

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

978-1-5090-3018-7/17/$31.00

Inductive types. One of our initial motivations for studying Bar Induction (BI), was to derive "standard" induction principles (Howard and Kreisel proved that BI is equivalent to transfinite induction [36]—see Sec. V). Until recently, Nuprl was relying on Mendler's monotone inductive types [49] to build inductive types similar to those of Coq [54]. 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 the version of Nuprl about which Mendler wrote his thesis was terminating [49]. This is not true anymore for several reasons, such as: Nuprl has now types of partial functions [66; 27; 29]. To recover inductive types in Nuprl, we proposed in [16] a solution (discussed in [61, Appx.E] which consists in building indexed families of W types from indexed families of co-W types using a variant of BI. This paper justifies among other things the addition of BI to Nuprl.

Intuitionism. There are two principles that distinguish Brouwer's intuitionistic mathematics [22; 20; 8] from other constructive mathematics, namely *bar induction* and a *continuity principle for numbers* [42; 36; 44; 31; 10; 19; 71; 76; 9; 63; 62; 74; 73; 59]. Also, a central concept in intuitionistic logic is the notion of a *choice sequence* [69], which is a never finished sequence of objects created over time by a *creating subject* [31, 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, or a combination of both. Brouwer developed a notion of intuitionistic continuum by defining real numbers as choice sequences, and proved that all real-valued functions on the unit interval are uniformly continuous [21, Thm.3] 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. Brouwer's (decidable) fan theorem says that every decidable bar on a finitary spread is uniform (this will be made more precise below)—see [71, Ch.7,Sec.7], [31, Sec.3.2], and [61, Appx.G].

Bar induction. We have proved that CTT is consistent with truncated versions of Brouwer's continuity principle [59; 58] (Sec. III-A discusses squashing/truncation). These past few years we have also been experimenting with versions of BI, which is an induction principle on *barred universal spreads*. What does that mean? A spread, as Dummett defines it [31, 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 $\mathcal{B}$ below). A *fan* is a finitely branching spread. A *bar* is a property of spreads that is true about at least one initial segment of each path.

As mentioned by Kleene [42, pp.50-51], BI corresponds to Brouwer's footnote 7 in [21], which roughly speaking says that if a spread is barred then there is a "backward" inductive proof of that. 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 [42, Sec.7.14; 31, Sec.3.4; 62, Rem.3.3; 74, Sec.2]—Kleene showed that it contradicts continuity [42, Sec.7.14,Lem.*27.23]. However, BI is often accepted by intuitionists when bars are restricted to decidable or monotone bars [42; 31; 76]. Also, as proved by Kleene [42, Lem.9.8], functions on numbers, such as $\mathcal{B}$'s members, *are not and cannot* be restricted to general recursive functions for FT and BI to be true (see also [71, p.223; 31, pp.52–53; 36, Sec.4; 42, pp.47–48]). Until recently, CTT's $\mathcal{B}$ type only contained general recursive functions. As we explain here, this is not the case anymore.

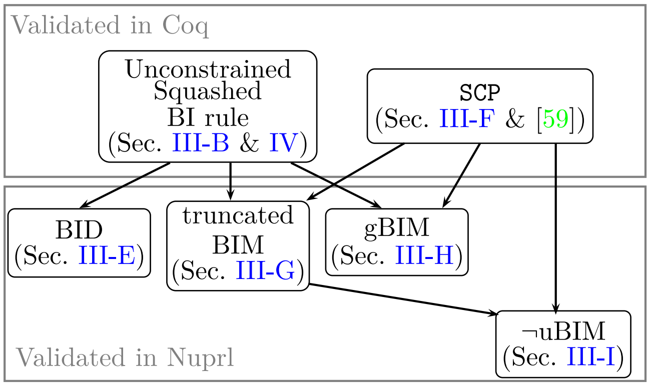
Before stating BI in the next paragraph, we first need to introduce some notation (Sec. II discusses Nuprl's syntax and semantics in more details): We write $\mathcal{B}$ for the Baire space, i.e., the function space $\mathbb{N} \to \mathbb{N}$, which we also write as $\mathbb{N}^{\mathbb{N}}$. We write $\mathcal{B}_k$ for $\mathbb{N}^{\mathbb{N}_k}$, where k is a natural number and $\mathbb{N}_k$ is the type of natural numbers strictly less than k. We use $\mathbf{\Pi}$ and $\mathbf{\Sigma}$ in lieu of the constructive logical quantifiers $\forall$ and $\exists$, respectively. We sometimes write $\mathbf{\Sigma}x_1{:}T_1.\cdots\mathbf{\Sigma}x_n{:}T_n.P$ as $\mathbf{\Sigma}(x_1 : T_1)\cdots(x_n : T_n).P$, and similarly for $\mathbf{\Pi}$ 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 $\mathtt{False}$ be $0 =_{\mathbb{N}} 1$, and $\mathtt{True}$ be $0 =_{\mathbb{N}} 0$. As usual, $\neg T$ is defined as $T \to \mathtt{False}$. $\mathbb{U}_i$ is the universe type at level i. We often omit levels and write either $\mathtt{Type}$ or $\mathbb{P}$ for $\mathbb{U}_i$—as opposed to Coq, there is no distinction between types and propositions in Nuprl.

We now formally state BI. A term P is a *predicate on finite sequences* (of numbers) if it is a member of the type $\mathbf{\Pi}n{:}\mathbb{N}.\mathcal{B}_n \to \mathbb{P}$. A predicate on finite sequences P is a *subset* of another predicate on finite sequences Q if for all $n \in \mathbb{N}$ and $s \in \mathcal{B}_n$, $P(n, s)$ implies $Q(n, s)$—in this context, for readability, we sometimes write $P(a, b)$ for the application $(P\ a\ b)$. A *bar* is a predicate on finite sequences B, such that $\mathbf{\Pi}s{:}\mathcal{B}.\mathbf{\Sigma}n{:}\mathbb{N}.B(n, s)$—we will see below that the $\mathbf{\Sigma}$ type in this formula can sometimes be truncated. A bar B is *decidable* if for all $n \in \mathbb{N}$ and $s \in \mathcal{B}_n$, $B(n, s) \vee \neg B(n, s)$. A bar B is *monotone* if for all $n, m \in \mathbb{N}$ and $s \in \mathcal{B}_n$ if $B(n, s)$ then $B(n{+}1, s{\oplus_n}m)$, where $s \oplus_n m = \lambda x.\mathtt{if}\ x{=}n\ \mathtt{then}\ m\ \mathtt{else}\ s(x)$. A predicate P on finite sequences is *inductive* if for all $n \in \mathbb{N}$ and $s \in \mathcal{B}_n$, if $\mathbf{\Pi}m{:}\mathbb{N}.P(n{+}1, s{\oplus_n}m)$ then $P(n, s)$. The unconstrained BI principle says that if P is an inductive predicate on finite sequences, and B is a bar and a subset of P, then for any term t, $P(0, t)$, i.e., P 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. Kleene proved using continuity that BIM can be reduced to BID, and that BID follows from BIM without any extra assumptions [42, Ch.1,Sec.7.6; 71, Ch.4,Prop.8.13; 31, Thm.3.7&3.8].

Roadmap and Contributions (numbered ① to ⑤). Sec. II discusses Nuprl's syntax and semantics. ① Sec. III-A–III-G introduce the $\downarrow$-squashed (see Sec. III-A) unconstrained BI inference rule that we have proved to be valid w.r.t. Nuprl's PER semantics using CTT's formalization in Coq, and present the versions of BID and BIM that we have derived in Nuprl using bar recursion operators (the Good). ② Sec. III-H presents a new and more general version of BIM. ③ Sec. III-I proves that both this general principle and the standard BIM principle are false in Nuprl when not $\downarrow$-squashed. This means that we can only prove $\downarrow$-squashed formulas with these principles (the Bad), i.e., we can only prove proof-irrelevant predicates. ④ Sec. IV-A provides a model of Nuprl extended with BI, and 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, to prove the validity of this rule we added choice sequences to Nuprl's term language in our model of CTT. These choice sequences are here all Coq functions from numbers to numbers, even those that make use of axioms (that are consistent with CIC—Coq's logic), and are therefore not computable. Our choice sequences are similar to the choice sequences in [11] and are introduced for a similar reason. They are only used in the metatheory and only get exposed to users through a partial axiomatization as illustrated in Sec. IV-A4. Users need only work with finite terms that do not contain choice sequences as illustrated in https://github.com/vrahli/NuprlInCoq/blob/master/rules/sterm.v. ⑤ Sec. IV-B generalizes Sec. IV-A to sequences of name-free (the Ugly) closed terms. Our names, sometimes called *unguessable atoms* [2; 15; 59], are similar to those in nominal logic [55]. Finally, Sec. V discusses additional related work and Sec. VI concludes. In addition, we suggest in [61, Appx.F] a possible externalization of our metatheoretic proof of BI's validity; and [61, Appx.G]. discusses the fan theorem. Fig. 1 summarizes the results presented in this paper.

The results presented here have either been formalized in Coq: https://github.com/vrahli/NuprlInCoq; or in Nuprl: http://www.nuprl.org/LibrarySnapshots/Published/Version2/

Fig. 1 Outline of results



Standard/continuity/index.html. Nuprl lemmas can be accessed by clicking the green hyperlinks or alternatively the reader can search in the continuity library for the lemmas named as the hyperlinks. The text will make it precise whether the results have been proved using Coq or using Nuprl.

II. Background

We first start by presenting some key aspects of Nuprl that will be used in the rest of this paper. Sec. II-A discusses the syntax and operational semantics of a large subset of Nuprl's computation system, and Sec. II-B discusses Nuprl's type system and its PER semantics.

A. Computation System

Fig. 2 presents a subset of Nuprl's syntax and small-step operational semantics [5; 51]. We only show the part that is either mentioned or used in this paper. Nuprl's programming language is an untyped (à la Curry), lazy λ-calculus with pairs, injections, a fixpoint operator, etc. For efficiency, integers are primitive and Nuprl provides operations on integers as well as comparison operators.

A term is either a variable, a value (or canonical term), or a non-canonical term. A non-canonical term t has one or two *principal arguments*—marked using boxes in Fig. 2—which are terms that have to be evaluated to canonical forms before t can be reduced further. For example the application $f(a)$ diverges if f diverges—we often write $f(a)$ for the application $f\ a$. The *canonical form tests* [60] $\mathbf{ifint}(t, a, b)$ and $\mathbf{iflam}(t, a, b)$ are used and explained in Sec. IV-A4.

Fig. 2 also shows part of Nuprl's small-step operational semantics. We omit the rules that reduce principal arguments such as: if $t_1 \mapsto t_2$ then $t_1\ u \mapsto t_2\ u$. Also, the operational semantics of $\boldsymbol{\nu}$ was introduced in [59] and is discussed below in Sec. IV-B1.

We now define abstractions that will be used below:

$$\bot = \mathtt{fix}(\lambda x.x) \qquad \pi_1(a) = \mathtt{let}\ x, y = a\ \mathtt{in}\ x$$
$$\mathtt{tt} = \mathtt{inl}(\star) \qquad \pi_2(a) = \mathtt{let}\ x, y = a\ \mathtt{in}\ y$$
$$\mathtt{ff} = \mathtt{inr}(\star)$$
$$a \le_{\mathtt{z}} b = \mathtt{if}\ a{<}b\ \mathtt{then}\ \mathtt{tt}\ \mathtt{else}\ \mathtt{if}\ a{=}b\ \mathtt{then}\ \mathtt{tt}\ \mathtt{else}\ \mathtt{ff}$$
$$\mathtt{isl}(a) = \mathtt{case}\ a\ \mathtt{of}\ \mathtt{inl}(\_) \Rightarrow \mathtt{tt}\ |\ \mathtt{inr}(\_) \Rightarrow \mathtt{ff}$$
$$\mathtt{if}\ a\ \mathtt{then}\ b\ \mathtt{else}\ c = \mathtt{case}\ a\ \mathtt{of}\ \mathtt{inl}(\_) \Rightarrow b\ |\ \mathtt{inr}(\_) \Rightarrow c$$

Also, we write: $a =_T b$ for the type $a = b \in T$; we write b for ($\mathtt{if}\ b\ \mathtt{then}\ \mathtt{True}\ \mathtt{else}\ \mathtt{False}$), i.e., we use implicit coercions from Booleans to propositions; and we write $\lambda x_1, \dots, x_n.t$ for $\lambda x_1. \dots \lambda x_n.t$.

B. Type System

Following Allen's PER semantics, Nuprl's types are interpreted as partial equivalence relations (PERs) on closed terms [3; 4; 29]. 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. For example, $T_1{\equiv}T_2$ is true if T_1 computes to $\mathbf{\Pi}x_1{:}A_1.B_1$; T_2 computes to $\mathbf{\Pi}x_2{:}A_2.B_2$; $A_1{\equiv}A_2$; and for all closed terms t_1 and t_2 such that $t_1{\equiv}t_2{\in}A_1$, $B_1[x_1\backslash t_1]{\equiv}B_2[x_2\backslash t_2]$. We say that a term t *inhabits* or *realizes* a type T if t is equal to itself in the PER interpretation of T, i.e., $t{\equiv}t{\in}T$. It follows from the PER interpretation of types that the theoretical proposition $a = b \in T$ is true iff $a{\equiv}b{\in}T$ holds in the metatheory [7; 51]. An equality type of the form $a = b \in T$, 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 computational content as opposed to HoTT [72].

As it turns out CTT is not only closed under computation but more generally under Howe's computational equivalence $\sim$, which he proved to be a congruence [37]. In any context C, when $t \sim t'$ we can rewrite t into t' without concern for typing. This relation is especially useful to prove equalities between programs (bisimulations) without concern for typing as illustrated in [60]. For example, using the least upper bound theorem [29, Thm.5.9], we can prove that all diverging expressions such as $\mathtt{fix}(\lambda x.x)$ and $\mathtt{fix}(\lambda x.x(x))$ are computationally equivalent; or that all streams of zeros such as $\mathtt{fix}(\lambda x.\langle 0, x\rangle)$ and $\mathtt{fix}(\lambda x.\langle 0, \langle 0, x\rangle\rangle)$ are computationally equivalent.

The top part of Fig. 2 lists some of Nuprl's types. Among these, $\mathtt{Base}$ is the type of all closed terms of the computation system with $\sim$ as its equality. The type $t_1 \simeq t_2$ is the theoretical counterpart of Howe's metatheoretical relation $t_1 \sim t_2$, and similarly for $\preceq$ and $\preccurlyeq$. Names [2; 59] come with two operations: a fresh operator $\boldsymbol{\nu}$ to generate fresh names, and a test for equality (not shown here). We used names to validate a continuity inference rule [59].

As mentioned above, we have formalized CTT in Coq [7; 51], including: (1) an implementation of Nuprl's computation system; (2) an implementation of Howe's computational equivalence relation, and a proof that it is a congruence; (3) a definition of Allen's PER semantics of CTT; (4) definitions of Nuprl's derivation rules, and proofs that these rules are valid w.r.t. Allen's semantics; (5) and a proof of Nuprl's consistency [7; 51; 58, Appx.A]. We are using CTT's formalization in Coq to prove the validity of all the inference rules of Nuprl, and have already verified a large number of them. See https://github.com/vrahli/NuprlI

Fig. 2 Syntax (top) and operational semantics (bottom) of a subset of Nuprl

$v \in \texttt{Value} ::= vt$ (type) $\mid \texttt{inl}(t)$ (left injection) $\mid \star$ (axiom) $\mid \langle t_1, t_2\rangle$ (pair)
$\mid \lambda x.t$ (lambda) $\mid \texttt{inr}(t)$ (right injection) $\mid i$ (integer) $\mid \boldsymbol{a}$ (name value)

$vt \in \texttt{Type} ::= \mathbb{Z}$ (integer type) $\mid \boldsymbol{\Pi} x{:}t_1.t_2$ (product) $\mid t_1 = t_2 \in t$ (equality)
$\mid \texttt{Base}$ (base) $\mid \boldsymbol{\Sigma} x{:}t_1.t_2$ (sum) $\mid t_1 + t_2$ (disjoint union)
$\mid \texttt{Name}$ (name type) $\mid \cup x{:}t_1.t_2$ (union) $\mid t_1 \preceq t_2$ (simulation)
$\mid \mathbb{U}_i$ (universe) $\mid \texttt{W}(x{:}t_1.t_2)$ (W) $\mid t_1 \simeq t_2$ (bisimulation)
$\mid t_1 // t_2$ (quotient) $\mid \{x : t_1 \mid t_2\}$ (set) $\mid \cap x{:}t_1.t_2$ (intersection)

$t \in \texttt{Term} ::= x$ (variable) $\mid \texttt{let}\ x := \boxed{t_1}\ \texttt{in}\ t_2$ (call-by-value) $\mid \texttt{ifint}(\boxed{t_1}, t_2, t_3)$ (integer test)
$\mid v$ (value) $\mid \texttt{let}\ x, y = \boxed{t_1}\ \texttt{in}\ t_2$ (spread) $\mid \texttt{iflam}(\boxed{t_1}, t_2, t_3)$ (lambda test)
$\mid \boxed{t_1}\ t_2$ (application) $\mid \texttt{if}\ \boxed{t_1} {<} \boxed{t_2}\ \texttt{then}\ t_3\ \texttt{else}\ t_4$ (less than)
$\mid \boldsymbol{\nu} x.\boxed{t}$ (fresh) $\mid \texttt{fix}(\boxed{t})$ (fixpoint)
$\mid \texttt{if}\ \boxed{t_1} {=} \boxed{t_2}\ \texttt{then}\ t_3\ \texttt{else}\ t_4$ (integer equality) $\mid \texttt{case}\ \boxed{t_1}\ \texttt{of}\ \texttt{inl}(x) \Rightarrow t_2 \mid \texttt{inr}(y) \Rightarrow t_3$ (decide)

$(\lambda x.F)\ a$	$\mapsto$	$F[x\backslash a]$	$\texttt{if}\ i_1{=}i_2\ \texttt{then}\ t_1\ \texttt{else}\ t_2$	$\mapsto$	t_1, if $i_1 = i_2$
$\texttt{fix}(v)$	$\mapsto$	$v\ \texttt{fix}(v)$	$\texttt{if}\ i_1{=}i_2\ \texttt{then}\ t_1\ \texttt{else}\ t_2$	$\mapsto$	t_2, if $i_1 \neq i_2$
$\texttt{let}\ x := v\ \texttt{in}\ t$	$\mapsto$	$t[x\backslash v]$	$\texttt{if}\ i_1{<}i_2\ \texttt{then}\ t_1\ \texttt{else}\ t_2$	$\mapsto$	t_1, if $i_1 < i_2$
$\texttt{let}\ x, y = \langle t_1, t_2\rangle\ \texttt{in}\ F$	$\mapsto$	$F[x\backslash t_1; y\backslash t_2]$	$\texttt{if}\ i_1{<}i_2\ \texttt{then}\ t_1\ \texttt{else}\ t_2$	$\mapsto$	t_2, if $i_1 \not< i_2$
$\texttt{ifint}(i, t_1, t_2)$	$\mapsto$	t_1	$\texttt{iflam}(\lambda x.t, t_1, t_2)$	$\mapsto$	t_1
$\texttt{ifint}(v, t_1, t_2)$	$\mapsto$	t_2, if v is not an integer	$\texttt{iflam}(v, t_1, t_2)$	$\mapsto$	t_2, if v is not a λ-term
$\texttt{case}\ \texttt{inl}(t)\ \texttt{of}\ \texttt{inl}(x) \Rightarrow F \mid \texttt{inr}(y) \Rightarrow G$	$\mapsto$	$F[x\backslash t]$	$\texttt{case}\ \texttt{inr}(t)\ \texttt{of}\ \texttt{inl}(x) \Rightarrow F \mid \texttt{inr}(y) \Rightarrow G$	$\mapsto$	$G[y\backslash t]$

nCoq/blob/master/RULES for a list of Nuprl's inference rules along with pointers to the proofs of their validity.

III. Squashing and Bootstrapping BI

This section presents an unconstrained *squashed* BI principle, which we prove to be valid w.r.t. Nuprl's PER semantics in Sec. IV. It also explains how we derived in Nuprl versions of BID and BIM from this squashed BI principle using bar recursion operators, and proves the negation of a non-$\downharpoonleft$-squashed version of BIM.

A. Squashing

In Nuprl, there are various ways of *squashing* or *truncating* a type. The one we use the most throws away the evidence that a type is inhabited and squashes it down to a single inhabitant using set types: $\downarrow T = \{\texttt{True} \mid T\}$ (as defined in [25, p.60]). Because a member of $\{x : T \mid U\}$ is a member t of T (such that $U[x\backslash t]$ holds)—and not a pair of a t in T and a u in $U[x\backslash t]$—the only member of $\downarrow T$ is then the constant $\star$, which is $\texttt{True}$'s single inhabitant. The constant $\star$ inhabits $\downarrow T$ if T is true/inhabited, but we do not keep the proof that it is true. See [58, Appx.F] for more information on squashing. Using the HoTT terminology, we also sometimes truncate types at the *propositional level* [72, Sec.3.7]. In Nuprl, that corresponds to *squashing* a type down to a single equivalence class, i.e. all inhabitants are equal, using quotient types [26]: $\downharpoonleft T = T//\texttt{True}$. Because the members of a quotient type $T//E$ are the members of T, the members of $\downharpoonleft T$ are then the members of T. Also, $\downharpoonleft T$ is a proof-irrelevant type, i.e., its members are all equal to each other because if $x, y \in T$ then $(x =_{\downharpoonleft T} y \iff \texttt{True})$. Note that $\downharpoonleft T \rightarrow \downarrow T$ is true because it is inhabited by $\lambda x.\star$, but we cannot prove the converse because to prove $\downharpoonleft T$ we have to exhibit an inhabitant of T, which $\downarrow T$ does not give us because only $\star$ inhabits $\downarrow T$.

B. Squashed Unconstrained BI Rule

As mentioned above, the unconstrained non-squashed BI principle is not consistent with constructive mathematics. However, it is consistent when proving $\downarrow$-squashed propositions as we prove in Sec. IV. (We do not imply here that Brouwer would have approved such a rule.) Using CTT's formalization in Coq, we prove in this paper the validity w.r.t. Nuprl's PER semantics of inference rules of the following form, which we call `[BarInduction]`:

Definition 1 (*`[BarInduction]` rule*)

$$\begin{array}{ll}
(\texttt{wfd}) & H, n : \mathbb{N}, s : T^{\mathbb{N}_n} \vdash B(n, s) \in \texttt{Type} \\
(\texttt{bar}) & H, s : T^{\mathbb{N}} \vdash \downarrow \boldsymbol{\Sigma} n{:}\mathbb{N}.B(n, s) \\
(\texttt{base}) & H, n : \mathbb{N}, s : T^{\mathbb{N}_n}, b : B(n, s) \vdash P(n, s) \\
(\texttt{ind}) & H, n : \mathbb{N}, s : T^{\mathbb{N}_n}, i : (\boldsymbol{\Pi} m{:}T.P(n + 1, s \oplus_n m)) \\
 & \quad \vdash P(n, s) \\
\hline
 & H \vdash \downarrow P(0, \perp\!\!\!\perp)
\end{array}$$

where T is $\mathbb{N}$ in Sec. IV-A, and the type of name-free closed terms in Sec. IV-B; and $\perp\!\!\!\perp$ is an empty sequence, defined as $\lambda x.\texttt{let}\ \_ := x\ \texttt{in}\ \perp$ for technical reasons discussed in [61, Appx.K]

The conclusion of this rule is $\downarrow$-squashed and therefore does not have any computational content, or rather its computational content is trivially the constant $\star$. This means that we can use whatever means we want in our Coq metatheoretical proof of its validity w.r.t. Nuprl's PER semantics in Sec. IV, even classical ones, because this proof will not be exposed in any way in the theory. Using this $\downarrow$-squashed principle, we show below how to derive in Nuprl, BI principles that have computational content, namely versions of BID and BIM.

The conclusion of the `bar` hypothesis is $\downarrow$-squashed because the bar is sometimes only used for termination, as in BID, and does not contribute to the extract, i.e., to the computational content of the induction principle.

C. BI Hypotheses

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:

$$
\begin{array}{ll}
\texttt{base} & : \mathbf{\Pi} n{:}\mathbb{N}.\mathbf{\Pi} s{:}T^{\mathbb{N}_n}.B(n,s) \to P(n,s) \\
\texttt{bar}_{\downarrow} & : \mathbf{\Pi} s{:}T^{\mathbb{N}}.{\downarrow}\mathbf{\Sigma} n{:}\mathbb{N}.B(n,s) \\
\texttt{bar}_{\lfloor} & : \mathbf{\Pi} s{:}T^{\mathbb{N}}.{\lfloor}\mathbf{\Sigma} n{:}\mathbb{N}.\ B(n,s) \\
\texttt{ind} & : \mathbf{\Pi} n{:}\mathbb{N}.\mathbf{\Pi} s{:}T^{\mathbb{N}_n}.(\mathbf{\Pi} m{:}T.P(n+1, s \oplus_n m)) \to P(n,s) \\
\texttt{dec} & : \mathbf{\Pi} n{:}\mathbb{N}.\mathbf{\Pi} s{:}T^{\mathbb{N}_n}.B(n,s)\ \vee\ \neg B(n,s) \\
\texttt{mon} & : \mathbf{\Pi} n{:}\mathbb{N}.\mathbf{\Pi} s{:}T^{\mathbb{N}_n}.\mathbf{\Pi} t{:}T.B(n,s) \to B(n+1, s \oplus_n t) \\
\texttt{mon}^{*} & : \mathbf{\Pi} n{:}\mathbb{N}.\mathbf{\Pi} m{:}\mathbb{N}_n.\mathbf{\Pi} s{:}T^{\mathbb{N}_n}.B(m,s) \to B(n,s)
\end{array}
$$

Note that the $\mathbf{\Sigma}$ type in $\texttt{bar}_{\lfloor}$'s type is $\lfloor$-squashed and not $\downarrow$-squashed as in $\texttt{bar}_{\downarrow}$ and in `[BarInduction]` because in Sec. III-G we need the bar hypothesis to have some computational content to build a realizer for BIM. We can trivially prove that $\texttt{bar}_{\lfloor}$ implies $\texttt{bar}_{\downarrow}$.

The $\texttt{mon}^{*}$ hypothesis is sometimes more convenient to use than the equivalent, more standard, `mon` hypothesis. It says that if B is true about the initial segment of length m of some sequence s of length at least n, then it is also true about its initial segment of length $n > m$.

D. Spector's Bar Recursion Operator

Spector first introduced a parametrized 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 [67]. Spector mentioned some relation between `SBR` and BID, and Howard showed that his W operator [35, p.111], which can be reduced to `SBR`, realizes BIM (see Sec. III-G). `SBR` can be defined as the following parametrized recursive operator (a minor difference: Spector's operator uses $<_{\mathtt{z}}$ instead of $\leq_{\mathtt{z}}$)—see Nuprl definition spector-bar-rec:

Definition 2 (*Spector's bar recursion operator*)

$$
\begin{array}{l}
\texttt{SBR}(Y,G,H,n,s) = \texttt{if}\ Y\ n\ s \leq_{\mathtt{z}} n\ \texttt{then}\ G\ n\ s \\
\qquad\qquad \texttt{else}\ H\ n\ s\ (\lambda t.\texttt{SBR}(Y,G,H,n+1,s \oplus_n t))
\end{array}
$$

Nuprl being untyped, we do not have to prove that `SBR` is in any type, and we have not done so. However, we show that two of its instances inhabit BI principles in Sec. III-E and III-G.

Spector used a restricted form of `SBR` to interpret the double-negation shift, which he used in his consistency proof [67, Sec.10]. Oliva and Powell [53] later proved that this restricted form of `SBR` is in fact as general as `SBR`. Informally, the way bar recursion works is that it goes up sequences by extending finite sequences using the $\oplus$ operator, until Y tells us we have reached the bar, i.e. the finite sequence given as argument is barred, at which point we apply the base operator G. Once we have reached the bar for all the direct extensions of a finite sequence we apply the induction operator H. As explained for example in [67, Sec.6.4,p.9; 70, Sec.1.9.26,p.83], the continuity of Y implies that the recursion terminates because it implies that for long enough sequences Y returns a number smaller than the length of the sequence it is applied to—see [61, Appx.D]. Also, note that this implies that checking whether we have reached the bar has to be decidable. As mentioned in [67, p.9,Footnote 6], and as further explained in Sec. III-G, this can be ensured by the fact that we can compute the modulus of continuity of the bar.

E. Bar Induction on Decidable Bars

Using an instance of `SBR` we now prove a BID principle, which is both more general than the one presented in Sec. III-B 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 (see Nuprl lemma decidable-bar-rec_wf) by proving that it is realized by the following *decidable bar recursion operator*, parametrized by a $n \in \mathbb{N}$ and a $s \in T^{\mathbb{N}_n}$—see Nuprl definition decidable-bar-rec:

Definition 3 (*Decidable bar recursion operator*)

$$
\begin{array}{l}
\texttt{DBR}(dec, base, ind, n, s) = \\
\ \texttt{case}\ dec\ n\ s\ \texttt{of} \\
\quad \texttt{inl}(r) \Rightarrow base\ n\ s\ r \\
\ \ \texttt{|}\ \texttt{inr}(\_) \Rightarrow ind\ n\ s\ (\lambda t.\texttt{DBR}(dec, base, ind, n+1, s \oplus_n t))
\end{array}
$$

More precisely, using the `[BarInduction]` inference rule presented above in Def. 1, we have proved the following BID principle:

Lemma 1 (*Bar Induction on Decidable bars*)

The hypotheses $\texttt{bar}_{\downarrow}$, `dec`, `base`, and `ind` defined in Sec. III-C imply that the term $\texttt{DBR}(\texttt{dec}, \texttt{base}, \texttt{ind}, 0, \perp\!\!\!\perp)$ inhabits the proposition $P(0, \perp\!\!\!\perp)$.

As mentioned in Sec. III-D, the way this decidable bar recursion operator works (and essentially the way our proof in Nuprl goes—see decidable-bar-rec_wf) is that starting from the empty sequence, we test whether we have reached the bar using `dec`, which inhabits the proposition that says that the bar B is decidable. Given a finite sequence provided by a number n and a sequence s, if $(\texttt{dec}\ n\ s)$ returns $\texttt{inl}(r)$, i.e. we have reached the bar, then r is a proof that $B(n, s)$ is true. In that case, we use our base hypothesis `base`. Otherwise, $(\texttt{dec}\ n\ s)$ returns $\texttt{inr}(r)$ which means that we are not at the bar yet, and in that case we recursively call `DBR` on all possible extensions of the sequence and use our induction hypothesis `ind`.

As mentioned above, `DBR` is an instance of `SBR`—see Nuprl lemma decidable-bar-rec-equal-spector:

Lemma 2 (`DBR` *as* `SBR`)

$$
\begin{array}{l}
\texttt{DBR}(dec, base, ind, n, s) = \\
\texttt{SBR}(\ \lambda n, s.\texttt{if}\ dec\ n\ s\ \texttt{then}\ 0\ \texttt{else}\ n+1 \\
\qquad ,\lambda n, s.\texttt{case}\ dec\ n\ s\ \texttt{of}\ \texttt{inl}(r) \Rightarrow base\ n\ s\ r \\
\qquad\qquad\qquad\qquad\qquad\qquad \texttt{|}\ \texttt{inr}(\_) \Rightarrow \bot \\
\qquad ,ind, n, s)
\end{array}
$$

The term $\perp$ could be any term because the base operator is only applied to n and s when $(dec\ n\ s)$ is an `inl`.

Remark 1. *In Spector's bar recursion operator* `SBR`*, the base case* $(G\ n\ s)$ *does not use the usual base hypothesis of BI that the bar implies the predicate we are trying to prove. More precisely* G *only takes a finite sequence as argument, and* Y*, which checks whether we have reached the bar, does not build anything for* G *to use. It is enough to know that* Y *returns a small enough number. We have not done so, but this suggests that the bar proposition* $B(n,s)$ *in BI's base hypothesis could be squashed as follows:*

$$\mathbf{\Pi} n{:}\mathbb{N}.\mathbf{\Pi} s{:}T^{\mathbb{N}_n}.\downharpoonleft B(n,s) \to P(n,s)$$

It turns out that for both BID and BIM we can always rebuild a proof of $B(n,s)$ *in order to use the base hypothesis.*

F. Continuity

We use a variant of Brouwer's continuity principle below in Sec. III-G to define (a variant of) Howard's W operator. This variant is sometimes called the *strong continuity principle for numbers* [63], which we have proved to be valid w.r.t. Nuprl's PER semantics (see Coq file https://github.com/vrahli/NuprlInCoq/blob/master/continuity/continuity_roadmap.v). The following *barred* variant, called `BSCP`, can be derived from the one presented in [59] as we proved in Nuprl lemma strong-continuity-rel-unique-pair:

Definition 4 (*Barred Strong Continuity Principle*)

$$\begin{array}{l}\mathbf{\Pi} P{:}(\mathcal{B} \to \mathbb{N} \to \mathbb{P}).\\ \quad (\mathbf{\Pi} f{:}\mathcal{B}.\downharpoonleft\mathbf{\Sigma} n{:}\mathbb{N}.\ P(f,n))\\ \quad \to\ \downharpoonleft\mathbf{\Sigma} M{:}(\mathbf{\Pi} n{:}\mathbb{N}.\mathbf{\Pi} s{:}\mathcal{B}_n.(\mathtt{barred}(P,n,s){+}\mathtt{True})).\\ \qquad\quad \mathbf{\Pi} f{:}\mathcal{B}.\mathbf{\Sigma} n{:}\mathbb{N}.\mathbf{\Sigma} p{:}\mathtt{barred}(P,n,f).\\ \qquad\qquad M(n,f) = \mathtt{inl}(p) \in \mathtt{barred}(P,n,f){+}\mathtt{True}\\ \qquad\quad \wedge\ \mathbf{\Pi} m{:}\mathbb{N}.\mathtt{isl}(M(m,f)) \to m =_{\mathbb{N}} n\end{array}$$

where $\mathtt{barred}(P,n,s) = \mathbf{\Sigma} k{:}\mathbb{N}_n.P(s\uparrow^0_n,k)$ is the type of pairs of a k in $\mathbb{N}_n$ and a p in $P(s\uparrow^0_n,k)$, i.e., in the case where P is a predicate on finite sequences as it is the case for our bar predicate B on which we will use `BSCP` below, P is true about the finite sequence s truncated at k; and where $s\uparrow^m_n = \lambda x.\mathtt{if}\ x{<}n\ \mathtt{then}\ s(x)\ \mathtt{else}\ m$ extends a finite sequence s of length n to an infinite sequence by returning the default value m starting from n.

`BSCP` makes it more convenient to define `HBR` below than the standard definition of the strong continuity principle, where $\mathtt{barred}(P,n,s)$ is simply $\mathbb{N}_n$. These strong continuity principles say that there is a uniform way, called M in the above formula (such a function is often called a neighborhood function [71, p.212]), to decide whether n is the modulus of continuity of P at f, and if so returns a number n such that $P(f,n)$ [42, pp.70–71].

As proved in [47, p.154; 68, Thm.IIA; 32], the non-truncated version of a "weaker" version of `BSCP` called `WCP`, and therefore of `BSCP` too, is false in Martin-Löf-like type theories. We have proved that this result is true about Nuprl too. See [61, Appx.D] for more information.

G. Bar Induction on Monotone Bars

A few years after Spector [67] introduced his bar recursion operator, Howard [35] showed that some instance of it, which he called W, realizes BIM, and of which we present a variant here called `HBR`. Let the parameter T from Sec. III-B 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. III-F is only for sequences of numbers. Howard does not explicitly mention continuity. However, Spector mentions continuity in [67, p.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. More precisely, $(\mathtt{BSCP}\ (\lambda s,n.B(n,s))\ \mathtt{bar}_{\downharpoonleft})$ gives us a M that, given a finite sequence, tells us whether the sequence is long enough to know whether we have reached the bar and also where we have reached the bar. Because `BSCP` is $\downharpoonleft$-squashed, assuming that the proposition we are proving by monotone bar induction is $\downharpoonleft$-squashed too, then $(\mathtt{BSCP}\ (\lambda s,n.B(n,s))\ \mathtt{bar}_{\downharpoonleft})$, gives us a:

$$M \in \mathbf{\Pi} n{:}\mathbb{N}.\mathbf{\Pi} s{:}\mathcal{B}_n.(\mathtt{barred}(B,n,s){+}\mathtt{True}) \tag{1}$$

such that:

$$\begin{array}{l}F \in\ \mathbf{\Pi} f{:}\mathcal{B}.\mathbf{\Sigma} n{:}\mathbb{N}.\mathbf{\Sigma} p{:}\mathtt{barred}(B,n,f).\\ \qquad\qquad M(n,f) = \mathtt{inl}(p) \in \mathtt{barred}(B,n,f){+}\mathtt{True}\\ \qquad\quad \wedge\ \mathbf{\Pi} m{:}\mathbb{N}.\mathtt{isl}(M(m,f)) \to m =_{\mathbb{N}} n\end{array} \tag{2}$$

We now define our monotone bar recursion operator `HBR` as follows—see Nuprl definition howard-bar-rec:

Definition 5 (*Monotone bar recursion operator*)

$$\begin{array}{l}\mathtt{HBR}(M, mon, base, ind, n, s) =\\ \mathtt{case}\ M(n,s)\ \mathtt{of}\\ \quad \mathtt{inl}(\langle k,p\rangle) \Rightarrow base\ n\ s\ (mon\ n\ k\ s\ p)\\ \mid \mathtt{inr}(\_) \Rightarrow ind\ n\ s\ (\lambda t.\mathtt{HBR}(M, mon, base, ind, n+1, s \oplus_n t))\end{array}$$

We have proved the following BIM result in Nuprl using the above bar recursion operator —see Nuprl lemma howard-bar-rec_wf:

Lemma 3 (*Bar Induction on Monotone bars*)

The hypotheses $\mathtt{bar}_{\downharpoonleft}$, $\mathtt{mon}^*$, `base`, and `ind` defined in Sec. III-C imply that $\mathtt{HBR}(M, \mathtt{mon}^*, \mathtt{base}, \mathtt{ind}, 0, \perp\!\!\!\perp)$ inhabits $\downharpoonleft P(0,\perp\!\!\!\perp)$.

Note that the proposition we are proving here using bar induction is $\downharpoonleft$-squashed. This is due to the fact that we are using `BSCP` which is $\downharpoonleft$-squashed. Therefore, we can only prove that `HBR` inhabits a $\downharpoonleft$-squashed BIM principle. Does that mean that, using BIM, one can only prove $\downharpoonleft$-squashed propositions? We partly answer this question below in Sec. III-I.

Proof. Let us sketch BIM's proof here. We want to prove that $\downharpoonleft P(0,\perp\!\!\!\perp)$ is true. The first step is to compute the modulus of continuity of $\mathtt{bar}_{\downharpoonleft}$ to get a neighborhood function M as shown above in Equation 1. Once we have unsquashed the existence of this neighborhood function, we can also unsquash our conclusion, i.e., we are now proving $P(0, \perp\!\!\!\perp)$, which we prove by showing that it is inhabited by $\mathtt{HBR}(0, \perp\!\!\!\perp)$, where we write $\mathtt{HBR}(n, s)$ for $\mathtt{HBR}(M, \mathtt{mon}^*, \mathtt{base}, \mathtt{ind}, n, s)$. We are now proving:

$$\mathtt{HBR}(0, \perp\!\!\!\perp) \in P(0, \perp\!\!\!\perp)$$

We now use the [BarInduction] inference rule presented above in Sec. III-B. When instantiating this rule, we have to choose a bar predicate B, which does not necessarily have to be the same as the one in BIM's statement. Here we instantiate [BarInduction] using $B = \lambda n, s.\mathtt{isl}(M(n, s))$, which is a well-formed predicate on finite sequences, and it remains to prove [BarInduction]'s bar hypothesis:

$$\mathbf{\Pi} s{:}\mathcal{B}.{\downarrow}\mathbf{\Sigma} n{:}\mathbb{N}.\mathtt{isl}(M(n, s)) \tag{3}$$

[BarInduction]'s base hypothesis:

$$\mathbf{\Pi} n{:}\mathbb{N}.\mathbf{\Pi} s{:}\mathcal{B}_n.\mathbf{\Pi} b{:}\mathtt{isl}(M(n, s)).\mathtt{HBR}(n, s) \in P(n, s) \tag{4}$$

and [BarInduction]'s induction hypothesis:

$$\begin{array}{l}\mathbf{\Pi} n{:}\mathbb{N}.\mathbf{\Pi} s{:}\mathcal{B}_n. \\ \quad \mathbf{\Pi} i{:}(\mathbf{\Pi} m{:}T.\mathtt{HBR}(n+1, s \oplus_n m) \in P(n+1, s \oplus_n m)). \\ \qquad \mathtt{HBR}(n, s) \in P(n, s)\end{array} \tag{5}$$

We prove 3 using 2: we apply F to s and get a $n \in \mathbb{N}$, a $p \in \mathtt{barred}(B, n, s)$, and a proof that $M(n, s)$ is a left injection, and we conclude by instantiating the conclusion of 3 using n. We now prove 4. Because $M(n, s)$ is a left injection, say $\mathtt{inl}(\langle k, p\rangle)$, such that $\langle k, p\rangle \in \mathtt{barred}(B, n, s)$, we get that $\mathtt{HBR}(n, s)$ computes to $(\mathtt{base}\ n\ s\ (\mathtt{mon}^*\ n\ k\ s\ p))$, and we now have to prove that $(\mathtt{base}\ n\ s\ (\mathtt{mon}^*\ n\ k\ s\ p)) \in P(n, s)$, which is trivial by typing of base and $\mathtt{mon}^*$. Finally, we prove 5. By definition of HBR, if $M(n, s)$ is a left injection, we conclude using the same proof as for 4. If $M(n, s)$ is a right injection, we have to prove that $(\mathtt{ind}\ n\ s\ (\lambda t.\mathtt{HBR}(n+1, s \oplus_n t))) \in P(n, s)$, which is trivial by typing of ind. □

As mentioned above, HBR is an instance of SBR—see Nuprl lemma howard-bar-rec-equal-spector:

Lemma 4 (HBR *as* SBR)

$$\begin{array}{l}\mathtt{HBR}(M, \mathit{mon}, \mathit{base}, \mathit{ind}, n, s) = \\ \mathtt{SBR}(\ \lambda n, s.\mathtt{if}\ M(n, s)\ \mathtt{then}\ 0\ \mathtt{else}\ n+1 \\ \quad , \lambda n, s.\ \mathtt{case}\ M(n, s)\ \mathtt{of} \\ \qquad\quad \mathtt{inl}(\langle k, p\rangle) \Rightarrow \mathit{base}\ n\ s\ (\mathit{ind}\ n\ k\ s\ p) \\ \qquad \mid \mathtt{inr}(\_) \Rightarrow \bot \\ \quad , \mathit{ind}, n, s)\end{array}$$

As in DBR's definition, here the term $\bot$ could be any term because this base operator is only applied to n and s when $M(n, s)$ is a left injection.

As mentioned above, continuity is used here to decide whether we have reached the bar or not. Thanks to continuity we can reduce monotone bar induction to decidable bar induction as proved for example by Kleene [42, p.78], and we can prove that HBR is also an instance of DBR—see Nuprl lemma howard-bar-rec-equal-decidable:

Lemma 5 (HBR *as* DBR)

$$\begin{array}{l}\mathtt{HBR}(M, \mathit{mon}, \mathit{base}, \mathit{ind}, n, s) = \\ \mathtt{DBR}(\ M, \lambda n, s, r.\mathtt{let}\ k, p = r\ \mathtt{in}\ \mathit{base}\ n\ s\ (\mathit{mon}\ n\ k\ s\ p) \\ \quad , \mathit{ind}, n, s)\end{array}$$

H. Generalizing BIM

Before proving that the non-$\downharpoonleft$-squashed version of BIM is false in Sec. III-I, we present here a slightly more general BIM principle than the standard one, which is also only for $\downharpoonleft$-squashed propositions. This principle, which we call gBIM, is inspired by the way Howard's W operator works, and especially by the fact that monotonicity is only used in HBR in the base case—see Nuprl lemma gen-bar-rec:

Definition 6 (*gBIM*)

$$\begin{array}{l}\mathbf{\Pi} P{:}(\mathbf{\Pi} n{:}\mathbb{N}.\mathcal{B}_n \to \mathbb{P}). \\ \quad (\mathbf{\Pi} s{:}\mathcal{B}.{\downharpoonleft}\mathbf{\Sigma} n{:}\mathbb{N}.\ \mathbf{\Pi} m{:}\{n \ldots\}.P(m, s)) \\ \quad \to\ (\mathbf{\Pi} n{:}\mathbb{N}.\mathbf{\Pi} s{:}\mathbb{N}^{\mathcal{B}_n}.(\mathbf{\Pi} m{:}\mathbb{N}.P(n+1, s \oplus_n m)) \to P(n, s)) \\ \quad \to\ {\downharpoonleft} P(0, \perp\!\!\!\perp)\end{array}$$

where $\{n \ldots\}$ is the type $\{k : \mathbb{N} \mid n \leq_{\mathtt{z}} k\}$.

Proof. We prove that this BIM principle is true, using again our unconstrained $\downarrow$-squashed BI principle presented in Def. 1, by proving that assuming that *bar* has type $\mathbf{\Pi} s{:}\mathcal{B}.{\downharpoonleft}\mathbf{\Sigma} n{:}\mathbb{N}.\ \mathbf{\Pi} m{:}\{n \ldots\}.P(m, s)$ and *ind* has type $\mathbf{\Pi} n{:}\mathbb{N}.\mathbf{\Pi} s{:}\mathbb{N}^{\mathcal{B}_n}.(\mathbf{\Pi} m{:}\mathbb{N}.P(n+1, s \oplus_n m)) \to P(n, s)$ then the following instance of Spector's bar recursion operator has type ${\downharpoonleft} P(0, \perp\!\!\!\perp)$:

$$\begin{array}{l}\mathtt{SBR}(\ \lambda n, s.\mathtt{if}\ M(n, s)\ \mathtt{then}\ 0\ \mathtt{else}\ n+1 \\ \quad , \lambda n, s.\mathtt{case}\ M(n, s)\ \ \mathtt{of}\ \mathtt{inl}(\langle k, F\rangle) \Rightarrow F(n) \\ \qquad\qquad\qquad\qquad\qquad \mid \mathtt{inr}(\_) \Rightarrow \bot \\ \quad , \mathit{ind}, n, s)\end{array}$$

where M is the neighborhood function of our *bar* hypothesis, i.e.: $M \in \mathbf{\Pi} n{:}\mathbb{N}.\mathbf{\Pi} s{:}\mathcal{B}_n.(\mathtt{barred}(Q, k, s) + \mathtt{True})$, where $Q = \lambda n, s.\mathbf{\Pi} m{:}\{n \ldots\}.P(m, s)$, and such that:

$$\begin{array}{l}F \in \mathbf{\Pi} f{:}\mathcal{B}.\mathbf{\Sigma} n{:}\mathbb{N}.\mathbf{\Sigma} p{:}\mathtt{barred}(Q, n, f). \\ \qquad M(n, f) = \mathtt{inl}(p) \in \mathtt{barred}(Q, n, f) + \mathtt{True} \\ \qquad \land\ \mathbf{\Pi} m{:}\mathbb{N}.\mathtt{isl}(M(m, f)) \to m =_{\mathbb{N}} n\end{array}$$

The rest proof is similar to the one presented in Sec. III-G. □

Let us mention two differences with a more "standard" version of BIM. (1) 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. (2) Also, here P has to be true at the bar and above the bar[1], 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 gBIM implies the following

[1] The predicate P needs only be true between the bar and its modulus of continuity. Defining such a version of BIM is left for future work.

more "standard" BIM principle, which we simply call BIM here—see Nuprl lemma gen-bar-ind-implies-monotone:

$$
\begin{array}{l}
\mathbf{\Pi} B, P{:}(\mathbf{\Pi} n{:}\mathbb{N}.\mathcal{B}_n \to \mathbb{P}).\\
\quad (\mathbf{\Pi} s{:}\mathcal{B}.\downharpoonleft \mathbf{\Sigma} n{:}\mathbb{N}.\ B(n,s))\\
\quad \to\ (\mathbf{\Pi} n{:}\mathbb{N}.\mathbf{\Pi} s{:}\mathcal{B}_n.(\mathbf{\Pi} m{:}\mathbb{N}.P(n+1, s \oplus_n m)) \to P(n,s))\\
\quad \to\ (\mathbf{\Pi} n,m{:}\mathbb{N}.\mathbf{\Pi} s{:}\mathcal{B}_n.B(n,s) \to B(n+1, s \oplus_n m))\\
\quad \to\ (\mathbf{\Pi} n{:}\mathbb{N}.\mathbf{\Pi} s{:}\mathcal{B}_n.B(n,s) \to P(n,s))\\
\quad \to\ \downharpoonleft P(0, \perp\!\!\!\perp)
\end{array}
$$

which is the principle we have proved above in Sec. III-G by proving that it is inhabited by a variant of Howard's bar recursion operator—except that it uses a one-step monotonicity hypothesis instead of a multi-step monotonicity hypothesis (see `mon` and `mon`* in Sec. III-C).

I. Negation of Non-$\downharpoonleft$-Squashed BIM

We now prove that the $\downharpoonleft$ operator in the above versions of BIM is necessary, i.e., that the following non-$\downharpoonleft$-squashed version of BIM, which we call uBIM, is false—see Nuprl lemma unsquashed-monotone-bar-induction3-false (we have also proved this result in Coq: https://github.com/vrahli/NuprlInCoq/blob/master/continuity/unsquashed_continuity.v):

Definition 7 (*uBIM*)

$$
\begin{array}{l}
\mathbf{\Pi} B, P{:}(\mathbf{\Pi} n{:}\mathbb{N}.\mathcal{B}_n \to \mathbb{P}).\\
\quad (\mathbf{\Pi} s{:}\mathcal{B}.\downharpoonleft \mathbf{\Sigma} n{:}\mathbb{N}.\ B(n,s))\\
\quad \to\ (\mathbf{\Pi} n{:}\mathbb{N}.\mathbf{\Pi} s{:}\mathcal{B}_n.(\mathbf{\Pi} m{:}\mathbb{N}.P(n+1, s \oplus_n m)) \to P(n,s))\\
\quad \to\ (\mathbf{\Pi} n,m{:}\mathbb{N}.\mathbf{\Pi} s{:}\mathcal{B}_n.B(n,s) \to B(n+1, s \oplus_n m))\\
\quad \to\ (\mathbf{\Pi} n{:}\mathbb{N}.\mathbf{\Pi} s{:}\mathcal{B}_n.B(n,s) \to P(n,s))\\
\quad \to\ P(0, \perp\!\!\!\perp)
\end{array}
$$

As discussed below, we still require that the bar be $\downharpoonleft$-squashed. This negative result follows from the fact that uBIM implies a non-squashed version of `WCP` (see Nuprl lemma unsquashed-BIM-implies-unsquashed-weak-continuity), which, as mentioned in Sec. III-F, is false in Nuprl, i.e.:

$$\neg \mathbf{\Pi} F{:}\mathbb{N}^{\mathcal{B}}.\mathbf{\Pi} f{:}\mathcal{B}.\mathbf{\Sigma} n{:}\mathbb{N}.\mathbf{\Pi} g{:}\mathcal{B}.f =_{\mathcal{B}_n} g \to F(f) =_{\mathbb{N}} F(g)$$

is true in Nuprl.

Lemma 6 ($\neg$*uBIM*)

Because the non-squashed version of gBIM implies uBIM, we get that both versions are false.

Proof. The proof that uBIM implies a non-squashed version of `WCP` goes as follows. We assume that $F \in \mathbb{N}^{\mathcal{B}}$ and $f \in \mathcal{B}$, and we have to prove: $\mathbf{\Sigma} n{:}\mathbb{N}.\mathbf{\Pi} g{:}\mathcal{B}.f =_{\mathcal{B}_n} g \to F(f) =_{\mathbb{N}} F(g)$. To prove this, we instantiate uBIM with:

$$
\begin{array}{l}
B = \lambda n, s.\mathbf{\Pi} g{:}\mathcal{B}.(s \boxplus_n f) =_{\mathcal{B}_n} g \to F(s \boxplus_n f) =_{\mathbb{N}} F(g)\\
P = \lambda n, s.\ \mathbf{\Sigma} m{:}\{n\ldots\}.\mathbf{\Pi} g{:}\mathcal{B}.\\
\qquad\qquad (s \boxplus_n f) =_{\mathcal{B}_m} g \to F(s \boxplus_n f) =_{\mathbb{N}} F(g)
\end{array}
$$

where $s \boxplus_n f = \lambda x.\mathtt{if}\ x{<}n\ \mathtt{then}\ s(x)\ \mathtt{else}\ f(x)$. The proposition $P(0, \perp\!\!\!\perp)$ is `WCP`, and we can then easily prove the hypotheses of uBIM:

Bar. The bar hypothesis follows from the $\downharpoonleft$-squashed `WCP` principle, which is true in Nuprl. `WCP` being $\downharpoonleft$-squashed, we also require uBIM's bar hypothesis to be $\downharpoonleft$-squashed.

Base. The base hypothesis is trivial: it suffices to instantiate $P(n,s)$ with n.

Induction. To prove the induction hypothesis we instantiate $\mathbf{\Pi} m{:}\mathbb{N}.P(n+1, s \oplus_n m)$ with $f(n)$. We get to assume $P(n+1, s \oplus_n f(n))$, i.e., that there exists a $m \geq n+1$ such that for all g such that $((s \oplus_n f(n)) \boxplus_{n+1} f) =_{\mathcal{B}_m} g$ then $F((s \oplus_n f(n)) \boxplus_{n+1} f) =_{\mathbb{N}} F(g)$, and have to prove $P(n,s)$. We instantiate our conclusion using m and conclude because $((s \oplus_n f(n)) \boxplus_{n+1} f) =_{\mathcal{B}} (s \boxplus_n f)$.

Monotonicity. To prove the monotonicity hypothesis, we have to prove that $B(n,s)$ implies $B(n+1, s \oplus_n m)$, i.e., assuming $B(n,s)$ and $((s \oplus_n m) \boxplus_{n+1} f) =_{\mathcal{B}_{n+1}} g$, we have to prove that $F((s \oplus_n m) \boxplus_{n+1} f) =_{\mathbb{N}} F(g)$. From $((s \oplus_n m) \boxplus_{n+1} f) =_{\mathcal{B}_{n+1}} g$, we deduce that $(s \boxplus_n f) =_{\mathcal{B}_n} g$, and therefore from $B(n,s)$, we deduce that $F(s \boxplus_n f) =_{\mathbb{N}} F(g)$. Finally, to prove $F((s \oplus_n m) \boxplus_{n+1} f) =_{\mathbb{N}} F(g)$ it is now enough to prove $F(s \boxplus_n f) =_{\mathbb{N}} F((s \oplus_n m) \boxplus_{n+1} f)$, which we get by instantiating $B(n,s)$ with $(s \oplus_n m) \boxplus_{n+1} f$. ☐

One question remains open: can we prove the validity of a non-squashed version of gBIM or of the "standard" BIM principle, where both the bar hypothesis and the conclusion are not squashed? This is left for future work.

IV. Validating BI Inference Rules

Sec. III presented an unconstrained $\downarrow$-squashed BI principle, from which we have derived BID and BIM principles. We now prove the validity of instances of this BI principle w.r.t. Nuprl's PER semantics. Sec. IV-A proves that our `[BarInduction]` inference rule is valid w.r.t. Nuprl's PER semantics when $T = \mathbb{N}$ (see Coq file https://github.com/vrahli/NuprlInCoq/blob/master/bar_induction/bar_induction3.v); while Sec. IV-B proves its validity for sequences of name-free closed terms (see Coq file https://github.com/vrahli/NuprlInCoq/blob/master/bar_induction/bar_induction_cterm4.v).

A. BI for Sequences of Natural Numbers

1) Following the Standard Classical Proof:

Lemma 7 (*Validity of* `[BarInduction]`)

`[BarInduction]` is true in CTT's impredicative Coq metatheory, i.e. in Prop.

Proof. We have proved this following Dummett's standard classical proof [31, p.55], which uses the law of excluded middle and the axiom of choice: see Coq file https://github.com/vrahli/NuprlInCoq/blob/master/bar_induction/bar_induction3.v. His proof goes as follows[2]: first we assume the negation of the conclusion using the law of excluded middle, i.e., the Coq axiom classic (available at https://coq.inria.fr/library/Coq.Logic.Classical_Prop.html). We now get to assume $\neg\downarrow P(0, \perp\!\!\!\perp)$ and therefore $\neg P(0, \perp\!\!\!\perp)$ too. Then, we contrapose our induction hypothesis (`ind`), and using the

[2]For readability, we omit some technicalities here regarding the well-formedness of terms, which are discussed in [61, Appx.K], in particular that finite sequences have to be *normalized.*

axiom of choice FunctionalChoice_on (available at https://coq.inria.fr/library/Coq.Logic.ChoiceFacts.html) we obtain a function F that, for all $n \in \mathbb{N}$, $s \in \mathcal{B}_n$, and proof of $\neg P(n,s)$, returns a natural number m such that $\neg P(n+1, s \oplus_n m)$. Because $\neg P(0, \perp\!\!\!\perp)$, F gives us a sequence $\alpha \in \mathcal{B}$ such that for all $n \in \mathbb{N}$, $\neg P(n,\alpha)$. We now instantiate our bar hypothesis (`bar`) with α to get a number k such that $B(k,\alpha)$. Finally, using our base hypothesis (`base`), we get a proof of $P(k,\alpha)$, which contradicts that for all $n \in \mathbb{N}$, $\neg P(n,\alpha)$. $\square$

2) Adding Coq Sequences to Nuprl: How did we construct the sequence α? F gives us a Coq function from numbers to numbers, but our proof needs a Nuprl term in the Nuprl type $\mathcal{B}$. 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 classic and FunctionalChoice_on, and which are therefore not computable. This coincides with the fact that functions on numbers should not be restricted to general recursive functions for BI to be true [42, Lem.9.8]. We call *choice sequences* these Coq functions from numbers to numbers occurring in Nuprl terms.

Our choice sequences are similar to the infinite sequences in [11] denoted $\lambda\!\!\lambda x.M_x$, where M_1, M_2, ..., is an infinite sequence of terms, which are used in a similar fashion as above to prove that some bar recursion operator realizes the negative translation of the axiom of choice. Similarly, as mentioned in [57], using our choice sequences, we have proved the validity of versions of the axiom of choice. In [11] the authors write: "The infinite terms are not for computational purposes, they only play a role in the termination proof". The same is true for us. The only place where we use choice sequences is in the metatheoretical Coq proof of `[BarInduction]`'s validity, which is not exposed in the theory because the conclusion of this rule is $\downarrow$-squashed and its computational content is the constant $\star$. Therefore, choice sequences do not have to be—and are not—part of the syntax of Nuprl definitions and proofs, i.e., the syntax visible to users. The syntax of terms occurring in definitions and proofs is the proper subset of Nuprl terms that do not contain choice sequences as illustrated in https://github.com/vrahli/NuprlInCoq/blob/master/rules/sterm.v. We talk about the theoretical Nuprl syntax to refer to the user syntax that does not allow choice sequences to occur in terms, as opposed to the syntax of terms implemented in our Coq metatheory that allows choice sequences to occur in terms.

Our choice sequences are also similar to Howe's set-theoretical functions in [39; 40; 38] (also called "oracles"), which he used to provide a set-theoretical semantics of both Nuprl (extended with set-theoretical terms) and HOL, allowing the shallow embedding of HOL in Nuprl.

Definition 8 (*Nuprl's syntax with choice sequences*)

Therefore, we extend Nuprl's (metatheoretical) term syntax presented in Sec. II with choice sequences, as well as an *eager* application operator:

$$\begin{array}{ll} v ::= \cdots \mid \mathtt{seq}(\mathtt{f}) & \text{(choice sequence)} \\ t ::= \cdots \mid \boxed{t_1}@\boxed{t_2} & \text{(eager application)} \end{array}$$

where `f` is a Coq function from numbers to numbers.

For example, $\mathtt{seq}(\mathtt{fun}\ n \Rightarrow n+1)$ is a choice sequence. We use eager applications to reduce lazy applications of choice sequences. Given a lazy application $s(t)$ of a choice sequence s to a term t, we first compute t to a value. If t computes to a natural number n, then $s(t)$ reduces to the application of the choice sequence s to n; otherwise the computation either gets stuck or diverges. For example, $\mathtt{seq}(\mathtt{fun}\ n \Rightarrow n{+}1)(1)$ reduces to 2; $\mathtt{seq}(\mathtt{fun}\ n \Rightarrow n{+}1)(\bot)$ diverges; and $\mathtt{seq}(\mathtt{fun}\ n \Rightarrow n+1)(\star)$ gets stuck.

Definition 9 (*Computing with choice sequences*)

To achieve this, we add the following reduction steps to compute with choice sequences:

$$\mathtt{seq}(\mathtt{f})\ t \mapsto \mathtt{seq}(\mathtt{f})@t$$

i.e., the *lazy* application of a sequence s to a term t computes in one step to the *eager* application of s to t. Eager applications compute as follows:

$$\begin{array}{l} t_1@t \mapsto t_2@t \quad \text{if} \quad t_1 \mapsto t_2 \\ v@t_1 \mapsto v@t_2 \quad \text{if} \quad t_1 \mapsto t_2 \\ (\lambda x.b)@v \mapsto b[x\backslash v] \\ \mathtt{seq}(\mathtt{f})@i \mapsto \mathtt{f}(i) \quad \text{if} \quad 0 \le i \end{array}$$

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 $\mathtt{f}(i)$ 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 $\mathtt{f}(n)$.

3) A Note on Decidability: Adding such choice sequences to Nuprl's (metatheoretical) terms does have interesting consequences such as: many properties become undecidable. For example, syntactic equality or α-equality are now undecidable in general. However, it turns out that even though these properties had been proved and used in the formalization of CTT in Coq, they are not necessary and we managed to do without them. Note that this is only true about Nuprl's metatheoretical syntax. Because Nuprl terms occurring in definitions and proofs do not contain choice sequences, syntactic equality and α-equality are decidable for the user syntax.

4) Consistency: Adding choice sequences to Nuprl's terms also affected Nuprl's consistency: we had to modify the following inference rule, called `[ApplyCases]`:

$$\frac{H \vdash \mathtt{halts}(f(a)) \quad H \vdash f \in \mathtt{Base}}{H \vdash f \simeq \lambda x.f(x)}$$

where the type $\mathtt{halts}(t) = \star \preceq (\mathtt{let}\ x := t\ \mathtt{in}\ \star)$ uses Howe's approximation relation to assert that t computes to a value. This rule says that f is computationally equivalent to its η-expansion $\lambda x.f(x)$ (i.e. f is a function) if $f(a)$

computes to a value, for some term a. Before adding choice sequences to Nuprl's terms, the only way $f(a)$ could compute to a value was if f would compute to a λ-term. This is not true anymore after adding choice sequences to Nuprl's terms. We chose to restate [ApplyCases] as follows:

$$\frac{H \vdash \texttt{halts}(f(a)) \quad H \vdash f \in \texttt{Base}}{H \vdash f \simeq \lambda x.f(x) \ \vee \ \texttt{isChoiceSeq}(x, z, f) \ \lfloor \texttt{iflam}(f, \texttt{tt}, \texttt{ff}) \rfloor}$$

where

$$\begin{aligned}&\texttt{isChoiceSeq}(x, z, f)\\ &= \cap x{:}\texttt{Base}. \cap z{:}\texttt{halts}(x).\texttt{ifint}(x, \texttt{True}, f(x) \preceq \perp)\end{aligned}$$

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

This new [ApplyCases] rule provides a partial axiomatization of choice sequences. Note that because choice sequences are not allowed in Nuprl's theoretical syntax, there is no way in the theory that $f \simeq \lambda x.f(x)$ would not be true for some term f such that $f(a)$ computes to a value, while $\texttt{isChoiceSeq}(x, z, f)$ would be. However, we cannot validate the old [ApplyCases] inference rule that rules out choice sequences, because they do occur in the metatheory.

B. BI For Sequences of Terms

Intuitively a similar proof as the one presented at the beginning of Sec. IV-A could be used at least when T is Base (defined in Sec.II-B). Following the same scheme as in Sec. IV-A, 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 Nuprl's "fresh" $\boldsymbol{\nu}$ operator. We explain this issue here in more details.

1) Banning Names From Choice Sequences: Let us assume that we change our choice sequence operator `seq(f)` so that `f` can now be a Coq function from numbers to closed Nuprl terms. The Coq function (`fun` $n \Rightarrow \boldsymbol{a}$), where $\boldsymbol{a}$ 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 $\boldsymbol{a}$, we have no way of knowing that it mentions $\boldsymbol{a}$. Now, the way Nuprl's $\boldsymbol{\nu}$ operator works, as explained in [59], is that to compute $\boldsymbol{\nu}x.t$, if $t \mapsto u$, we first pick a fresh name $\boldsymbol{b}$ w.r.t. t. The name $\boldsymbol{b}$ being fresh w.r.t. t here means that if $\boldsymbol{b}$ occurs in t then it can only occur in a choice sequence. Then, we compute $t[x\backslash\boldsymbol{b}]$ to w in one computation step, and finally we return $\boldsymbol{\nu}x.(w[\boldsymbol{b}\backslash x])$, where $t[\boldsymbol{a}\backslash u]$ is a capture avoiding substitution function on names similar to the usual substitution operation on variables. Therefore, if t contains `seq`(`fun` $n \Rightarrow \boldsymbol{a}$), we have to make sure that we do not pick $\boldsymbol{a}$. Otherwise, when computing $\boldsymbol{\nu}x.$(`seq`(`fun` $n \Rightarrow \boldsymbol{a}$) 0), we could pick $\boldsymbol{a}$ as our fresh name, reduce (`seq`(`fun` $n \Rightarrow \boldsymbol{a}$) 0)$[x\backslash\boldsymbol{a}]$, which is equal to (`seq`(`fun` $n \Rightarrow \boldsymbol{a}$) 0), to $\boldsymbol{a}$, perform the substitution $\boldsymbol{a}[\boldsymbol{a}\backslash x] = x$, and finally return $\boldsymbol{\nu}x.x$, which would not be correct because the two $\boldsymbol{a}$s are supposed to be different.

We avoid this here by precluding names from occurring in sequences, and change our choice sequence operator `seq(f)` so that `f` is now a Coq function from numbers to name-free closed Nuprl terms. This means that the Coq type of Nuprl terms is now an ordinal with a limit constructor for such sequences (see https://github.com/vrahli/NuprlInCoq/blob/master/terms/terms.v for more details regarding Nuprl's metatheoretical term syntax).

Because choice sequences do not contain free variables or names, most operations on terms do not change because the two substitution operations on names and free variables stay unchanged. Using these choice sequences, we have proved in Coq the validity w.r.t. Nuprl's PER semantics of [BarInduction] when the parameter T is the following type, closed under $\sim$, of name-free closed terms: $\{t : \texttt{Base} \mid (t : \texttt{Base})\#\}$, where the type $(a : A)\#$ asserts that the term a is in the type A and does not contain names (see Coq file https://github.com/vrahli/NuprlInCoq/blob/master/bar_induction/bar_induction_cterm4.v).

2) Could Names Occur in 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 $\boldsymbol{\nu}x.t$, we first pick a fresh name $\boldsymbol{a}$ w.r.t. t by not looking inside choice sequences, then we reduce $t[x\backslash\boldsymbol{a}]$ to u in one computation step, and we compute a new fresh name $\boldsymbol{b}$ w.r.t. both t and u. This is to ensure that if the computation step applies a sequence to a term and "reveals" new names, then $\boldsymbol{b}$ is not one of these names. Finally, we compute $\boldsymbol{\nu}x.t$ using $\boldsymbol{b}$ as our fresh name. Let us consider the example we gave in Sec. IV-B1: $\boldsymbol{\nu}x.$(`seq`(`fun` $n \Rightarrow \boldsymbol{a}$) 0). Following the procedure we just described, we first pick a name that is fresh w.r.t. (`seq`(`fun` $n \Rightarrow \boldsymbol{a}$) 0) by not looking inside the choice sequence. Here it does not matter which one we pick. Let us pick $\boldsymbol{c}$. We reduce the term (`seq`(`fun` $n \Rightarrow \boldsymbol{a}$) 0)$[x\backslash\boldsymbol{c}]$ to $\boldsymbol{a}$ in one computation step. Now we pick a name $\boldsymbol{b}$, which is fresh w.r.t. both (`seq`(`fun` $n \Rightarrow \boldsymbol{a}$) 0) and $\boldsymbol{a}$, and we reduce (`seq`(`fun` $n \Rightarrow \boldsymbol{a}$) 0)$[x\backslash\boldsymbol{b}]$ to $\boldsymbol{a}$ in one computation step. Finally, we return the term $\boldsymbol{\nu}x.(\boldsymbol{a}[\boldsymbol{b}\backslash x])$, which is equal to $\boldsymbol{\nu}x.\boldsymbol{a}$.

Remark 2. *We also want to preserve the property that*

Howe's computational approximation and equivalence relations are congruences [37]. For Nuprl's $\boldsymbol{\nu}$ operator, this means that to prove that $\boldsymbol{\nu}x.t \sim \boldsymbol{\nu}x.u$, it should be enough to prove that $t[x\backslash\boldsymbol{a}] \sim u[x\backslash\boldsymbol{a}]$ for some $\boldsymbol{a}$ fresh w.r.t. t and u. Unfortunately, if names were allowed in choice sequences, we would not be able to compute such a name. See Appx. **??** *for more details.*

V. Related Work

As mentioned in the introduction, Howard and Kreisel studied Brouwer's bar induction and continuity principles in [36] and showed the equivalence between the axiom of transfinite induction (TI)—sometimes called the bar rule [64]—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. They consider the two following notions of well-foundedness: a strong form

$$\mathrm{WF}_1(\rho) = \forall f \exists n \neg (f(n) \rho f(n+1))$$

and a weak form

$$\mathrm{WF}_2(\rho) = \forall f \exists n \neg \forall m \leq n (f(m) \rho f(m+1))$$

Their transfinite induction principle says:

$$\forall x (\forall y (x \rho y \to Q(y)) \to Q(x)) \to \forall x Q(x)$$

In Coq, TI is simply a lemma called `well_founded_ind` for Prop; and `well_founded_induction_type` for Type: see the Coq library https://coq.inria.fr/library/Coq.Init.Wf.html. Well-foundedness is inductively defined in Coq using the *accessibility* predicate Acc. It can be shown that if a decidable relation is well-founded using Coq's definition then it is well-founded using WF_1.

The bar recursion operators mentioned in Sec. III and some of their variants have been extensively studied [65; 11; 14; 52; 12; 56; 33]. 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 [12]) lead to new BI principles.

Troelstra lists some uses of BI in [70, p.114], e.g. to prove strong normalization of systems such as N-HA$^\omega$. Veldman and Bezem proved an intuitionistically valid reformulation of Ramsey's theorem using BIM [77; 75]. We have proved this result in Nuprl: see lemma intuitionistic-Ramsey. In [79], the authors proved similar results using directly Coq's inductive types rather than BI.

Choice sequences have also been widely studied over the years [45; 42; 46; 44; 69; 31; 71; 78]. One interesting result regarding choice sequences is the so-called "elimination of choice sequences" theorem [46, Sec.2; 44, Ch.7; 69, Ch.3; 31, pp.221–222; 30] that eliminates quantifications over choice sequences. This theorem relies on a mapping from the formulae of the CS formal system [44] to formulae of the IDB_1 formal system [44] that do not contain choice sequence variables. It is left to future work to study whether a similar result could be used to prove that BI is consistent with Nuprl without using choice sequences.

Finally, it is worth noting that our method of building a model of Nuprl extended with BI principles bears some resemblance with forcing [23; 24] where our forcing conditions are our choice sequences.

VI. Conclusion

We have recently proved, using CTT's formalization in Coq, that $\downharpoonleft$-squashed versions of Brouwer's continuity principle for numbers are consistent with Nuprl [59]. We have now also proved the validity of a $\downarrow$-squashed BI inference rule for sequences of name-free closed terms. From this $\downarrow$-squashed BI rule, we have derived a non-squashed version of BID for sequences of name-free closed terms, as well as a $\downharpoonleft$-version of BIM for sequences of numbers (because Nuprl's version of continuity is only for sequences of numbers). We have also shown that BIM is not true in general for non-$\downharpoonleft$-squashed propositions. Several questions remain open such as: (1) Can we generalize the $\downharpoonleft$-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 David Guaspari and Evan Moran for their helpful criticism.

References

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

[2] Stuart Allen. *An Abstract Semantics for Atoms in Nuprl*. Tech. rep. Cornell University, 2006.

[3] Stuart F. Allen. "A Non-Type-Theoretic Definition of Martin-Löf's Types". In: *LICS*. IEEE Computer Society, 1987, pp. 215–221.

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

[5] 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.

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

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

[8] Mark van Atten. *On Brouwer*. Wadsworth Philosophers. Cengage Learning, 2004.

[9] Mark van Atten and Dirk van Dalen. "Arguments for the continuity principle". In: *Bulletin of Symbolic Logic* 8.3 (2002), pp. 329–347.

[10] Michael J. Beeson. *Foundations of Constructive Mathematics*. Springer, 1985.

[11] 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.

[12] Ulrich Berger and Paulo Oliva. "Modified bar recursion". In: *Mathematical Structures in Computer Science* 16.2 (2006), pp. 163–183.

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

[14] Marc Bezem. "Equivalence of Bar Recursors in the Theory of Functionals of Finite Type". In: *Archive for Mathematical Logic* 27.2 (1988), pp. 149–160.

[15] Mark Bickford. “Unguessable Atoms: A Logical Foundation for Security”. In: *Verified Software: Theories, Tools, Experiments, Second Int'l Conf.* Vol. 5295. LNCS. Springer, 2008, pp. 30–53.

[16] 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.

[17] 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.

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

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

[20] L.E.J. Brouwer. *Brouwer's Cambridge Lectures on Intuitionism*. Edited by D. Van Dalen. Cambridge University Press, 1981, pp. 214–215.

[21] L.E.J. Brouwer. “From frege to Gödel: A Source Book in Mathematical Logic, 1879–1931”. In: Harvard University Press, 1927. Chap. On the Domains of Definition of Functions.

[22] L.E.J. Brouwer. “Historical background, principles and methods of intuitionism”. In: *South African journal of science* 49.3,4 (1952).

[23] Paul J. Cohen. “The independence of the continuum hypothesis”. In: *the National Academy of Sciences of the United States of America* 50.6 (Dec. 1963), pp. 1143–1148.

[24] Paul J. Cohen. “The independence of the continuum hypothesis II”. In: *the National Academy of Sciences of the United States of America* 51.1 (Jan. 1964), pp. 105–110.

[25] 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.

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

[27] Robert L. Constable and Scott F. Smith. “Computational Foundations of Basic Recursive Function Theory”. In: *Theoretical Computer Science* 121.1&2 (1993), pp. 89–112.

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

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

[30] Gerrit van Der Hoeven and Ieke Moerdijk. “Sheaf models for choice sequences”. In: *Ann. Pure Appl. Logic* 27.1 (1984), pp. 63–107.

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

[32] 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.

[33] Martín Escardó and Paulo Oliva. “Bar Recursion and Products of Selection Functions”. In: *J. Symb. Log.* 80.1 (2015), pp. 1–28.

[34] Martin Hofmann. “Extensional concepts in intensional type theory”. PhD thesis. University of Edinburgh, 1995.

[35] William A. Howard. “Functional interpretation of bar induction by bar recursion”. eng. In: *Compositio Mathematica* 20 (1968), pp. 107–124.

[36] 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.

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

[38] 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.

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

[40] 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.

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

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

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

[44] G. Kreisel and A.S. Troelstra. “Formal systems for some branches of intuitionistic analysis”. In: *Annals of Mathematical Logic* 1.3 (1970), pp. 229–387.

[45] Georg Kreisel. “A Remark on Free Choice Sequences and the Topological Completeness Proofs”. In: *J. Symb. Log.* 23.4 (1958), pp. 369–388.

[46] Georg Kreisel. “Lawless sequences of natural numbers”. eng. In: *Compositio Mathematica* 20 (1968), pp. 222–248.

[47] Georg Kreisel. “On weak completeness of intuitionistic predicate logic”. In: *J. Symb. Log.* 27.2 (1962), pp. 139–158.

[48] Martin-Löf. “Constructive Mathematics and Computer Programming”. In: *6th International Congress for Logic, Methodology and Philosophy of Science*. Noth-Holland, Amsterdam, 1982, pp. 153–175.

[49] Paul F. Mendler. “Inductive Definition in Type Theory”. PhD thesis. Ithaca, NY: Cornell University, 1988.

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

[51] *Nuprl in Coq*. https://github.com/vrahli/NuprlInCoq.

[52] Paulo Oliva. “Understanding and Using Spector's Bar Recursive Interpretation of Classical Analysis”. In: *CiE 2006*. Vol. 3988. LNCS. Springer, 2006, pp. 423–434.

[53] Paulo Oliva and Thomas Powell. “On Spector's bar recursion”. In: *Math. Log. Q.* 58.4-5 (2012), pp. 356–265.

[54] Christine Paulin-Mohring. “Inductive Definitions in the system Coq - Rules and Properties”. In: *TLCA'93*. Vol. 664. LNCS. Springer, 1993, pp. 328–345.

[55] Andrew M. Pitts. “Nominal Logic: A First Order Theory of Names and Binding”. In: *TACS 2001*. Vol. 2215. LNCS. Springer, 2001, pp. 219–242.

[56] Thomas Powell. “On Bar Recursive Interpretations of Analysis”. PhD thesis. Queen Mary University of London, Aug. 2013.

[57] Vincent Rahli. “Exercising Nuprl's Open-Endedness”. In: *ICMS 2016*. Vol. 9725. LNCS. Springer, 2016, pp. 18–27.

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

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

[60] 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.

[61] Vincent Rahli, Mark Bickford, and Robert L. Constable. “Bar Induction: The Good, the Bad, and the Ugly”. Extended version avaible at http://www.nuprl.org/html/Nuprl2Coq/bar-induction-lics-long.pdf. Apr. 2017.

[62] Michael Rathjen. “A note on Bar Induction in Constructive Set Theory”. In: *Math. Log. Q.* 52.3 (2006), pp. 253–258.

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

[64] Michael Rathjen. “The Role of Parameters in Bar Rule and Bar Induction”. In: *J. Symb. Log.* 56.2 (1991), pp. 715–730.

[65] Helmut Schwichtenberg. “On Bar Recursion of Types 0 and 1”. In: *J. Symb. Log.* 44.3 (1979), pp. 325–329.

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

[67] Clifford Spector. “Provably recursive functionals of analysis: a consistency proof of analysis by an extension of principles in current intuitionistic mathematics”. In: *Recursive Function Theory: Proc. Symposia in Pure Mathematics*. Vol. 5. American Mathematical Society, 1962, pp. 1–27.

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

[69] A.S. Troelstra. *Choice Sequences: A Chapter of Intuitionistic Mathematics*. Clarendon Press, 1977.

[70] A.S. Troelstra. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*. New York, Springer, 1973.

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

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

[73] Wim Veldman. "Brouwer's Fan Theorem as an axiom and as a contrast to Kleene's alternative". In: *Arch. Math. Log.* 53.5-6 (2014), pp. 621–693.

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

[75] Wim Veldman. "Some Applications of Brouwer's thesis on Bars". In: *One Hundred Years of Intuitionism*. Birkhäuser, 2008, pp. 326–340.

[76] 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.

[77] Wim Veldman and Mark Bezem. "Ramsey's theorem and the pigeonhole principle in intuitionistic mathematics". In: *J. of the London Mathematical Society* s2-47 (2 1993), pp. 193–211.

[78] Richard Vesley. "Realizing Brouwer's Sequences". In: *Ann. Pure Appl. Logic* 81.1-3 (1996), pp. 25–74.

[79] Dimitrios Vytiniotis, Thierry Coquand, and David Wahlstedt. "Stop When You Are Almost-Full - Adventures in Constructive Termination". In: *ITP 2012*. Vol. 7406. LNCS. Springer, 2012, pp. 250–265.