Bar Induction is Compatible with Constructive Type Theory

VINCENT RAHLI, SnT, University of Luxembourg, Luxembourg
MARK BICKFORD, Cornell University, USA
LIRON COHEN, Cornell University, USA
ROBERT L. CONSTABLE, Cornell University, USA

Powerful yet effective induction principles play an important role in computing, being a paramount component of programming languages, automated reasoning and program verification systems. The *Bar Induction* principle is a fundamental concept of intuitionism, which is equivalent to the standard principle of transfinite induction. In this work we investigate the compatibility of several variants of Bar Induction with *Constructive Type Theory* (CTT), a dependent type theory in the spirit of Martin-Löf's extensional theory. We first show that CTT is compatible with a Bar Induction principle for sequences of numbers. Then, we establish the compatibility of CTT with a more general Bar Induction principle for sequences of name-free closed terms. The formalization of the latter principle within the theory involved enriching CTT's term syntax with a limit constructor and showing that consistency is preserved. Furthermore, we provide novel insights regarding Bar Induction, such as the non-truncated version of Bar Induction on monotone bars being intuitionistically false. These enhancements are carried out formally using the Nuprl proof assistant which implements CTT, and the formalization of CTT within the Coq proof assistant presented in previous works.

CCS Concepts: • **Theory of computation → Constructive mathematics**; **Type theory**.

Additional Key Words and Phrases: Bar Induction, Computational Type Theory, Intuitionistic Logic, W types, Semantics, Choice Sequences, Nuprl, Coq

ACM Reference Format:
Vincent Rahli, Mark Bickford, Liron Cohen, and Robert L. Constable. 2019. Bar Induction is Compatible with Constructive Type Theory. *J. ACM* 1, 1, Article 1 (January 2019), 35 pages. https://doi.org/10.1145/3305261

1 INTRODUCTION

Constructive Type Theory (CTT) is a powerful dependent type theory in the spirit of Martin-Löf's extensional theory. Variants of CTT are implemented by widely used proof assistants such as Agda [24; 3], Coq [18; 39], Idris [25; 56], Lean [73], and Nuprl [35; 7]. The capabilities of such proof assistants have been exploited to advance the state of the art in many computer science applications such as compilers [69], microkernels [61], or operating systems [47]. CTT and its variants have their roots in Brouwer's intuitionistic mathematics [30; 28; 11], which goes beyond CTT by exceeding Church-Turing computability. Moreover, intuitionistic mathematics comes with a powerful and general induction principle called *Bar Induction* (BI) [60; 50; 67; 43; 13; 27; 102; 107; 12; 92; 91; 105; 104; 87]. The goal of this work is two-fold:

Authors' addresses: Vincent Rahli, SnT, University of Luxembourg, 6 Avenue de la Fonte, Esch-sur-Alzette, 4364, Luxembourg, vincent.rahli@gmail.com; Mark Bickford, Cornell University, 107 Hoy Rd, Ithaca, NY, 14853, USA, markb@cs.cornell.edu; Liron Cohen, Cornell University, 107 Hoy Rd, Ithaca, NY, 14853, USA, lironcohen@cornell.edu; Robert L. Constable, Cornell University, 107 Hoy Rd, Ithaca, NY, 14853, USA, rc@cs.cornell.edu.

0004-5411/2019/1-ART1 $15.00
https://doi.org/10.1145/3305261

- While all the aforementioned proof assistants come equipped with strong inductive principles, ensuring their correctness is, in general, very complex. In search for a powerful, computational inductive principle, in this work we formally show that CTT can be consistently enhanced with variants of BI.
- BI is a salient principle of intuitionistic mathematics that goes beyond constructive mathematics, whose additional computational capabilities have yet to be studied in the context of CTT. This work takes a step towards adding such capabilities to CTT, resulting in a new kind of type theory, whose notion of computation extends Church-Turing computability.

Bar Induction was initially introduced by Brouwer as an induction principle to reason about *choice sequences* [99]. They are never finished sequences of objects created over time by a *creating subject* [43, Sec.6.3], which lie at the heart of intuitionistic logic. 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 dice), or a combination of both. Brouwer developed a notion of intuitionistic continuum by defining real numbers as choice sequences of nested rational intervals, and proved that all real-valued functions on the unit interval are uniformly continuous [29, 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 the 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 Sec. 6.1, [102, Ch.7,Sec.7], and [43, Sec.3.2].

Bar Induction is an induction principle on *barred universal spreads*. What does that mean? A *spread*, as Dummett defines it [43, 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. Below is a formal definition of Bar Induction together with some visualization in Fig 1.

As mentioned by Kleene [60, pp.50-51], BI corresponds to Brouwer's footnote 7 in [29], 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 [60, Sec.7.14; 43, Sec.3.4; 91, Rem.3.3; 105, Sec.2]—Kleene showed that it contradicts continuity [60, Sec.7.14,Lem.*27.23]. However, BI is often accepted by intuitionists when bars are restricted to decidable or monotone bars [60; 43; 107]. Also, as proved by Kleene [60, 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 [102, p.223; 43, pp.52–53; 50, Sec.4; 60, pp.47–48]). Until recently, CTT's $\mathcal{B}$ type only contained general total recursive functions: λ-expressions that produce numbers from numbers. This is not the case anymore because in this work we enhance the $\mathcal{B}$ type with non-recursive objects (infinite choice sequences), as we explain below.

We now formally state BI. We use the following notations (see Fig. 4 for a list of Nuprl's primitive types): $\mathcal{B}$ for the Baire space, i.e., the function space $\mathbb{N} \rightarrow \mathbb{N}$, which we also write as $\mathbb{N}^{\mathbb{N}}$; $\mathcal{B}_k$ 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; $\mathbf{\Pi}$ and Σ in lieu of the constructive logical quantifiers $\forall$ and $\exists$, respectively; $P \ \vee \ Q$ for the disjoint union P+Q; $\mathbb{U}_i$ for the type of propositions at level i. We often omit universe levels and write either Type or $\mathbb{P}$ for $\mathbb{U}_i$—as opposed to Coq for example, there is no distinction between types and propositions in Nuprl. A term P is a *predicate on finite sequences* (of numbers) if it is a member of the type $\Pi n{:}\mathbb{N}.\mathcal{B}_n \longrightarrow \mathbb{P}$. 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 $Q(n, s)$.[1] A predicate P on finite sequences is *inductive* if for all $n \in \mathbb{N}$ and $s \in \mathcal{B}_n$, if $\Pi m{:}\mathbb{N}.P(n + 1, s \oplus_n m)$ then $P(n, s)$, where $s \oplus_n m$ updates the sequence s by setting the nth value to be m, i.e., $s \oplus_n m = \lambda x.\texttt{if } x{=}n \texttt{ then } m \texttt{ else } s(x)$. A *bar* is a predicate on finite sequences B, such that $\Pi s{:}\mathcal{B}.\Sigma n{:}\mathbb{N}.B(n, s)$—we will see below that the Σ type in this formula can sometimes be truncated.

> The *Bar Induction principle* states 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.

We refer to the BI principle above as the unconstrained BI principle. A bar B is *decidable* if for all $n \in \mathbb{N}$ and $s \in \mathcal{B}_n$, $B(n, s) \lor \neg B(n, s)$. A bar B is *monotone* if for all $n, m \in \mathbb{N}$ and $s \in \mathcal{B}_n$ if $B(n, s)$ then $B(n + 1, s \oplus_n m)$, Bar Induction on Decidable bars (BID) also assumes that B is decidable, and Bar Induction on Monotone bars (BIM) assumes that B is monotone. Fig. 1 illustrates the BIM principle.

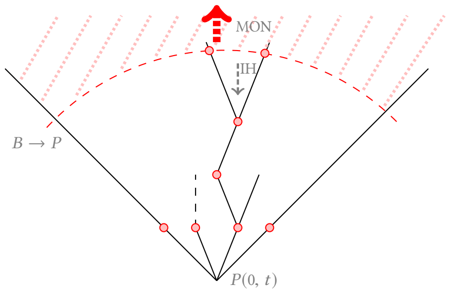

Fig. 1. Bar Induction

While, as noted, the unconstrained BI principle is not valid in constructive mathematics, we here prove that a $\downarrow$-squashed variant of it (i.e. one in which some evidence is omitted—see Sec. 2.4) is valid w.r.t. Nuprl's PER semantics. This is done using CTT's formalization in Coq. In addition, we also discuss possible ways of externalizing this proof (in the sense that the proof is done within CTT itself, and not within its metatheory). Furthermore, using this $\downarrow$-squashed BI principle we derive both a non-squashed BID principle and a $\downarrow$-squashed BIM principle using bar recursion operators.[2] We also present a novel, more general BIM principle and show that both it and the standard BIM principle are false in CTT in general, except when proving proof irrelevant propositions (i.e., to prove $\downarrow$-squashed propositions—see Sec. 2.4).

We provide a model of CTT extended with BI, and prove 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 a notion of 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 [14] 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. 4.1.2.[3] We generalize the validity proof to also support sequences of name-free closed terms. Our names, sometimes called *unguessable atoms* [4; 20; 87], are similar to those in nominal logic [83].

Inductive types and principles are an essential ingredient in any programming language. There are several approaches to building them using CTT. For instance, until recently, Nuprl was relying

[1]In this context, for readability, we sometimes write $P(a, b)$ for the application $(P\ a\ b)$.
[2]Kleene proved using continuity that BIM can be reduced to BID, and that BID follows from BIM without any extra assumptions [60, Ch.1,Sec.7.6; 102, Ch.4,Prop.8.13; 43, Thm.3.7&3.8].
[3]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.

Name	Formula	Where	Comments
$\mathtt{BI}_{\downarrow}$	$\mathtt{WF}(B) \rightarrow \mathtt{BAR}_{\downharpoonleft}(B) \rightarrow \mathtt{BASE}(B, P) \rightarrow \mathtt{IND}(P) \rightarrow \downarrow P(0, \perp\!\!\!\perp)$	Coq	uses classical logic
BID	$\mathtt{WF}(B) \rightarrow \mathtt{BAR}_{\downharpoonleft}(B) \rightarrow \mathtt{DEC}(B) \rightarrow \mathtt{BASE}(B, P) \rightarrow \mathtt{IND}(P) \rightarrow P(0, \perp\!\!\!\perp)$	Nuprl	uses $\mathtt{BI}_{\downarrow}$
$\mathtt{BIM}_{\downharpoonleft}$	$\mathtt{WF}(B) \rightarrow \mathtt{BAR}_{\downharpoonleft}(B) \rightarrow \mathtt{MON}(B) \rightarrow \mathtt{BASE}(B, P) \rightarrow \mathtt{IND}(P) \rightarrow \downharpoonleft P(0, \perp\!\!\!\perp)$	Nuprl	uses $\mathtt{BI}_{\downarrow}$
$\neg\mathtt{uBIM}$	$\neg\Pi B, P{:}(\Pi n{:}\mathbb{N}.\mathbb{P}^{\mathcal{B}_n}).\mathtt{BAR}_{\downharpoonleft}(B) \rightarrow \mathtt{MON}(B) \rightarrow \mathtt{BASE}(B, P) \rightarrow \mathtt{IND}(P) \rightarrow P(0, \perp\!\!\!\perp)$	Nuprl	contradicts continuity
$\mathtt{gBIM}_{\downharpoonleft}$	$\Pi P{:}(\Pi n{:}\mathbb{N}.\mathcal{B}_n \rightarrow \mathbb{P}).\mathtt{MONBAR}(P) \rightarrow \mathtt{IND}(P) \rightarrow \downharpoonleft P(0, \perp\!\!\!\perp)$	Nuprl	uses $\mathtt{BI}_{\downarrow}$
$\neg\mathtt{WCP}_{1,0}$	$\neg\Pi F{:}\mathbb{N}^{\mathcal{B}}.\Pi f{:}\mathcal{B}.\Sigma n{:}\mathbb{N}.\Pi g{:}\mathcal{B}.f =_{\mathcal{B}_n} g \rightarrow F(f) =_{\mathbb{N}} F(g)$	Nuprl	
$\mathtt{WCP}_{1,0\downharpoonleft}$	$\Pi F{:}\mathbb{N}^{\mathcal{B}}.\Pi f{:}\mathcal{B}.\downharpoonleft\Sigma n{:}\mathbb{N}.\ \Pi g{:}\mathcal{B}.f =_{\mathcal{B}_n} g \rightarrow F(f) =_{\mathbb{N}} F(g)$	Coq	uses named exceptions
$\mathtt{WCP}_{1,0\downarrow}$	$\Pi F{:}\mathbb{N}^{\mathcal{B}}.\Pi f{:}\mathcal{B}.\downarrow\Sigma n{:}\mathbb{N}.\Pi g{:}\mathcal{B}.f =_{\mathcal{B}_n} g \rightarrow F(f) =_{\mathbb{N}} F(g)$	Coq	uses diverging terms
$\neg\mathtt{WCP}_{1,1}$	$\neg\Pi P{:}\mathcal{B} \rightarrow \mathbb{P}^{\mathcal{B}}.(\Pi a{:}\mathcal{B}.\Sigma b{:}\mathcal{B}.P(a, b)) \rightarrow \Sigma c{:}\mathbb{N}^{\mathcal{B}}.\mathtt{CONT}(c)\ \wedge\ \Pi a{:}\mathcal{B}.\mathtt{shift}(c, a)$	Nuprl	
$\mathtt{WCP}_{1,1\downharpoonleft}$	$?\neg\Pi P{:}\mathcal{B} \rightarrow \mathbb{P}^{\mathcal{B}}.(\Pi a{:}\mathcal{B}.\Sigma b{:}\mathcal{B}.P(a, b)) \rightarrow \downharpoonleft\Sigma c{:}\mathbb{N}^{\mathcal{B}}.\ \mathtt{CONT}(c)_{\downharpoonleft}\ \wedge\ \Pi a{:}\mathcal{B}.\mathtt{shift}(c, a)$	?	
$\mathtt{WCP}_{1,1\downarrow}$	$?\neg\Pi P{:}\mathcal{B} \rightarrow \mathbb{P}^{\mathcal{B}}.(\Pi a{:}\mathcal{B}.\Sigma b{:}\mathcal{B}.P(a, b)) \rightarrow \downarrow\Sigma c{:}\mathbb{N}^{\mathcal{B}}.\mathtt{CONT}(c)_{\downarrow}\ \wedge\ \Pi a{:}\mathcal{B}.\mathtt{shift}(c, a)$	?	
$\mathtt{AC}_{0,0}$	$\Pi P{:}\mathbb{N} \rightarrow \mathbb{P}^{\mathbb{N}}.(\Pi n{:}\mathbb{N}.\Sigma m{:}\mathbb{N}.P(n, m)) \rightarrow \Sigma f{:}\mathcal{B}.\Pi n{:}\mathcal{B}.P(n, f(n))$	Nuprl	
$\mathtt{AC}_{0,0\downharpoonleft}$	$\Pi P{:}\mathbb{N} \rightarrow \mathbb{P}^{\mathbb{N}}.(\Pi n{:}\mathbb{N}.\downharpoonleft\Sigma m{:}\mathbb{N}.\ P(n, m)) \rightarrow \downharpoonleft\Sigma f{:}\mathcal{B}.\ \Pi n{:}\mathcal{B}.P(n, f(n))$	Nuprl	
$\mathtt{AC}_{0,0\downarrow}$	$\Pi P{:}\mathbb{N} \rightarrow \mathbb{P}^{\mathbb{N}}.(\Pi n{:}\mathbb{N}.\downarrow\Sigma m{:}\mathbb{N}.P(n, m)) \rightarrow \downarrow\Sigma f{:}\mathcal{B}.\Pi n{:}\mathcal{B}.P(n, f(n))$	Coq	uses classical logic
$\mathtt{AC}_{1,0}$	$\Pi P{:}\mathcal{B} \rightarrow \mathbb{P}^{\mathbb{N}}.(\Pi f{:}\mathcal{B}.\Sigma n{:}\mathbb{N}.P(f, n)) \rightarrow \Sigma F{:}\mathbb{N}^{\mathcal{B}}.\Pi f{:}\mathcal{B}.P(f, F(f))$	Nuprl	
$\mathtt{AC}_{1,0\downharpoonleft}$	$\Pi P{:}\mathcal{B} \rightarrow \mathbb{P}^{\mathbb{N}}.(\Pi f{:}\mathcal{B}.\downharpoonleft\Sigma n{:}\mathbb{N}.\ P(f, n)) \rightarrow \downharpoonleft\Sigma F{:}\mathbb{N}^{\mathcal{B}}.\ \Pi f{:}\mathcal{B}.P(f, F(f))$	Nuprl	
$\mathtt{AC}_{1,0\downarrow}$	$?\neg\Pi P{:}\mathcal{B} \rightarrow \mathbb{P}^{\mathbb{N}}.(\Pi f{:}\mathcal{B}.\downarrow\Sigma n{:}\mathbb{N}.P(f, n)) \rightarrow \downarrow\Sigma F{:}\mathbb{N}^{\mathcal{B}}.\Pi f{:}\mathcal{B}.P(f, F(f))$	?	
$\mathtt{AC}_{2,0}$	$\Pi P{:}\mathbb{N}^{\mathcal{B}} \rightarrow \mathbb{P}^{\mathbb{N}}.(\Pi f{:}\mathbb{N}^{\mathcal{B}}.\Sigma n{:}T.P(f, n)) \rightarrow \Sigma F{:}T^{(\mathbb{N}^{\mathcal{B}})}.\Pi f{:}\mathbb{N}^{\mathcal{B}}.P(f, F(f))$	Nuprl	
$\neg\mathtt{AC}_{2,0\downharpoonleft}$	$\neg(\Pi P{:}\mathbb{N}^{\mathcal{B}} \rightarrow \mathbb{P}^{T}.(\Pi f{:}\mathbb{N}^{\mathcal{B}}.\downharpoonleft\Sigma n{:}T.\ P(f, n)) \rightarrow \downharpoonleft\Sigma F{:}T^{(\mathbb{N}^{\mathcal{B}})}.\ \Pi f{:}\mathbb{N}^{\mathcal{B}}.P(f, F(f)))$	Nuprl	contradicts continuity
$\neg\mathtt{AC}_{2,0\downarrow}$	$\neg(\Pi P{:}\mathbb{N}^{\mathcal{B}} \rightarrow \mathbb{P}^{T}.(\Pi f{:}\mathbb{N}^{\mathcal{B}}.\downharpoonleft\Sigma n{:}T.\ P(f, n)) \rightarrow \downarrow\Sigma F{:}T^{(\mathbb{N}^{\mathcal{B}})}.\Pi f{:}\mathbb{N}^{\mathcal{B}}.P(f, F(f)))$	Nuprl	contradicts continuity
$\neg\mathtt{LEM}$	$\neg\Pi P{:}\mathbb{P}.P\ \vee\ \neg P$	Nuprl	
$\neg\mathtt{LEM}_{\downharpoonleft}$	$\neg\Pi P{:}\mathbb{P}.\downharpoonleft(P\ \vee\ \neg P)$	Nuprl	
$\mathtt{LEM}_{\downarrow}$	$\Pi P{:}\mathbb{P}.\downarrow(P\ \vee\ \neg P)$	Coq	uses classical logic
MP	$\Pi P{:}\mathbb{P}^{\mathbb{N}}.(\Pi n{:}\mathbb{N}.P(n)\ \vee\ \neg P(n)) \rightarrow (\neg\Pi n{:}\mathbb{N}.\neg P(n)) \rightarrow \Sigma n{:}\mathbb{N}.P(n)$	Nuprl	uses $\mathtt{LEM}_{\downarrow}$
$\neg\mathtt{KS}$	$\neg\Pi A{:}\mathbb{P}.\Sigma a{:}\mathcal{B}.((\Sigma x{:}\mathbb{N}.a(x) =_{\mathbb{N}} 1)\ \iff\ A)$	Nuprl	uses MP
$\neg\mathtt{KS}_{\downharpoonleft}$	$\neg\Pi A{:}\mathbb{P}.\downharpoonleft\Sigma a{:}\mathcal{B}.((\Sigma x{:}\mathbb{N}.a(x) =_{\mathbb{N}} 1)\ \iff\ A)$	Nuprl	uses MP
$\mathtt{KS}_{\downarrow}$	$\Pi A{:}\mathbb{P}.\downarrow\Sigma a{:}\mathcal{B}.((\Sigma x{:}\mathbb{N}.a(x) =_{\mathbb{N}} 1)\ \iff\ A)$	Coq	uses classical logic

Fig. 2. Valid intuitionistic axioms

on Mendler's monotone inductive types [72] to build inductive types similar to those of Coq [82]—Mendler's constructions go beyond those of Coq because they allow one to build monotone inductive types from monotonic functions on types. Nonetheless, these sorts of constructions usually rely on the assumption that the computation system is terminating. This was key, for example, in Mendler's proof of the validity of the inference rules for (co-)inductive types. The current version of Nuprl has an extremely rich collection of types which includes, among other things, types of partial functions [96; 36; 40]. Therefore, its computation system is no longer terminating. To recover inductive types, instead of adapting Mendler's proof to deal with a non-terminating computation system, we exploit BI to derive "standard" induction principles (Howard and Kreisel proved that BI is equivalent to transfinite induction [50]—see Sec. 7). Thus, we use a variant of BI to construct indexed families of W types [71; 78] from indexed families of co-W types (see Sec. 5).

As previously mentioned, other than Bar Induction, there are several principles that go beyond constructive mathematics. Those include the classical Law of Excluded Middle (LEM) and Axiom of Choice (AC), the (Russian) constructive Markov's Principle (MP), and the intuitionistic Continuity Principle (WCP) and Kripke's Schema (KS). Fig. 2 summarizes the status of variants of those principles w.r.t. CTT. Those axioms have either been validated using our Coq model of CTT, or have been proved to be true directly in Nuprl, or we use "?" to indicate those that have not yet been proved or disproved. WCP, AC, and LEM are discussed in [88]. KS and MP are discussed in Sec. 6.2. Both CONT and shift are defined in Sec. 6.2. We write $\mathtt{CONT}(c)_{\downarrow}$ for the version of $\mathtt{CONT}(c)$ where $\underline{\Sigma}$ is $\downarrow\Sigma$; we write $\mathtt{CONT}(c)_{\downharpoonleft}$ for the version of $\mathtt{CONT}(c)$ where $\underline{\Sigma}$ is $\downharpoonleft\Sigma$; and we simply write $\mathtt{CONT}(c)$ for the version of $\mathtt{CONT}(c)$ where $\underline{\Sigma}$ is Σ. The empty sequence $\perp\!\!\!\perp$ is defined in Sec. 3.1. The

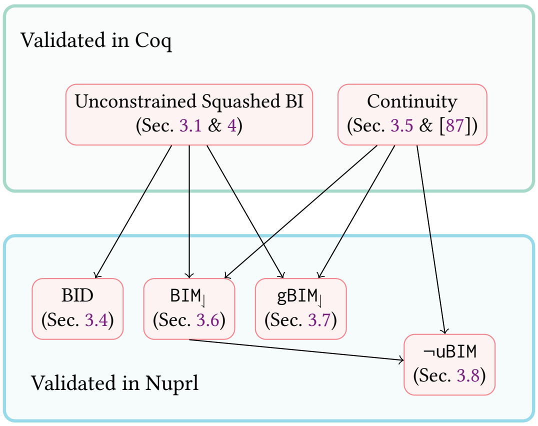

Fig. 3. Outline of main results

$\{n\dots\}$ type, where n is a natural number, is the type of numbers greater than or equal to n. Also, let (these predicates correspond to the hypotheses presented below in Sec.3.2)

$$\begin{array}{ll}
\mathtt{WF}(B) & = \Pi n{:}\mathbb{N}.\Pi s{:}\mathcal{B}_n.B(n,s) \in \mathsf{Type} \\
\mathtt{BAR}_{\downarrow}(B) & = \Pi s{:}\mathcal{B}.{\downarrow}\Sigma n{:}\mathbb{N}.B(n,s) \\
\mathtt{BAR}_{\lfloor}(B) & = \Pi s{:}\mathcal{B}.{\lfloor}\Sigma n{:}\mathbb{N}.B(n,s) \\
\mathtt{DEC}(B) & = \Pi n{:}\mathbb{N}.\Pi s{:}\mathcal{B}_n.B(n,s) \ \vee\ \neg B(n,s) \\
\mathtt{MON}(B) & = \Pi n{:}\mathbb{N}.\Pi s{:}\mathcal{B}_n.\Pi m{:}\mathbb{N}.B(n,s) \longrightarrow B(n+1, s \oplus_n m) \\
\mathtt{MONBAR}(P) & = \Pi s{:}\mathcal{B}.{\lfloor}\Sigma n{:}\mathbb{N}.\ \Pi m{:}\{n\dots\}.P(m,s) \\
\mathtt{BASE}(B,P) & = \Pi n{:}\mathbb{N}.\Pi s{:}\mathcal{B}_n.B(n,s) \longrightarrow P(n,s) \\
\mathtt{IND}(P) & = \Pi n{:}\mathbb{N}.\Pi s{:}\mathcal{B}_n.(\Pi m{:}\mathbb{N}.P(n+1, s \oplus_n m)) \longrightarrow P(n,s)
\end{array}$$

Note that the bar properties $\mathtt{BAR}_{\downarrow}(B)$ and $\mathtt{BAR}_{\lfloor}(B)$ guarantee that the bar predicate B is well-formed on infinite sequences. However, in addition, we typically require that the bar property in bar induction principles is well-formed on finite sequences of type $\mathcal{B}_n$ for any $n \in \mathbb{N}$ using $\mathtt{WF}(B)$.

Roadmap. First, Sec. 2 presents part of Nuprl's syntax and semantics that are used in the rest of this paper. Sec. 3 discusses the variants of BI explored. Sec. 3.1 introduces the ↓-squashed unconstrained BI inference rule that we have proved to be valid w.r.t. Nuprl's PER semantics using CTT's formalization in Coq. Sec. 3.3–3.6 introduce the versions of BID and BIM that we have derived using bar recursion operators. In Sec. 3.7 we present a new and more general version of BIM, and in Sec. 3.8 we prove that both this general principle and the standard BIM principle are false in CTT when not ⌊-squashed. Sec. 4 establishes the consistency of the extended framework. Sec. 4.1 provides a model of CTT extended with BI, and proves the validity of a BI inference rule for sequences of numbers. Then, Sec. 4.2 generalizes Sec. 4.1 to sequences of name-free closed terms. Sec. 4.3 suggests a possible externalization of our metatheoretic proof of BI's validity. In Sec. 5 we demonstrate how BI can be used to construct W types. Sec. 6 investigates the connections between BI and other key intuitionistic principles, namely the Fan Theorem and Kripke's Schema. Finally, Sec. 7 discusses related work and Sec. 8 concludes.[4]

[4]Some of the results presented here were presented in our LICS 2017 paper [90]. We here present additional details regarding Nuprl in Sec. 2, continuity in Sec. 3.5 and Bar Induction throughout the paper. Most notably Sec. 4.3 suggests a possible externalization of our metatheoretic proof of BI's validity; Sec. 5 demonstrate how BI can be used to derive induction principles for W types; and Sec. 6 discusses related principles, namely the Fan Theorem and Kripke's Schema.

Fig. 3 summarizes the main results presented in this paper. All of them have either been formalized in Coq: https://github.com/vrahli/NuprlInCoq; or in Nuprl: http://www.nuprl.org/LibrarySnapshots/Published/Version2/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.

2 BACKGROUND

The Nuprl interactive theorem prover [35; 7] implements a type theory called *Constructive Type Theory* (CTT), which is a dependent type theory, in the spirit of Martin-Löf's extensional theory [70], based on an untyped functional programming language. Its types include equality types, a hierarchy of universes, W types, quotient types [34], set types, union and (dependent) intersection types [62], image types [77], PER types [9], approximation and computational equivalence types [89], and partial types [96; 40]. CTT "mostly" differs from other similar constructive type theories such as the ones implemented by Agda [24; 3], Coq [18; 39], or Idris [25; 56], in the sense that CTT is an *extensional* theory (i.e., propositional and definitional equality are identified [48]) with types of partial functions [96; 36; 40]. For example, the fixpoint $\texttt{fix}(\lambda x.x)$ computes to itself in two computation steps and therefore 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 [5; 6], CTT types are interpreted as Partial Equivalence Relations (PERs) on closed terms, and we have formalized this semantics in Coq [10; 79].

We next present some key aspects of Nuprl that will be used in the rest of this paper. Sec. 2.1 discusses the syntax and operational semantics of a large subset of Nuprl's computation system, and Sec. 2.2 discusses Nuprl's type system and its PER semantics.

2.1 Computation System

Fig. 4 presents a subset of Nuprl's syntax and small-step operational semantics [7; 79]. 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. 4—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* [89] $\textbf{ifint}(t, a, b)$ and $\textbf{iflam}(t, a, b)$ are used and explained in Sec. 4.1.2.

Fig. 4 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 [87] and is discussed below in Sec. 4.2.1.

We now define abstractions that will be used below:

$$\begin{array}{ll} \bot = \texttt{fix}(\lambda x.x) & \\ \texttt{tt} = \texttt{inl}(\star) & \pi_1(a) = \texttt{let}\ x,y = a\ \texttt{in}\ x \\ \texttt{ff} = \texttt{inr}(\star) & \pi_2(a) = \texttt{let}\ x,y = a\ \texttt{in}\ y \end{array}$$

$$a \leq_z b = \texttt{if}\ a{<}b\ \texttt{then}\ \texttt{tt}\ \texttt{else}\ \texttt{if}\ a{=}b\ \texttt{then}\ \texttt{tt}\ \texttt{else}\ \texttt{ff}$$

$$\texttt{isl}(a) = \texttt{case}\ a\ \texttt{of}\ \texttt{inl}(\_) \Rightarrow \texttt{tt}\ \textbf{|}\ \texttt{inr}(\_) \Rightarrow \texttt{ff}$$

$$\texttt{if}\ a\ \texttt{then}\ b\ \texttt{else}\ c = \texttt{case}\ a\ \texttt{of}\ \texttt{inl}(\_) \Rightarrow b\ \textbf{|}\ \texttt{inr}(\_) \Rightarrow c$$

Also, we write $\lambda x_1, \ldots, x_n.t$ for $\lambda x_1. \ldots \lambda x_n.t$.

$$\begin{array}{llllllll}
v \in \texttt{Value} ::= & vt & \text{(type)} & \mid \texttt{inl}(t) & \text{(left injection)} & \mid \star & \text{(axiom)} \\
& \mid \langle t_1, t_2 \rangle & \text{(pair)} & \mid \lambda x.t & \text{(lambda)} & \mid \texttt{inr}(t) & \text{(right injection)} \\
& \mid i & \text{(integer)} & \mid \boldsymbol{a} & \text{(name value)} & & \\
vt \in \texttt{Type} ::= & \mathbb{Z} & \text{(integer type)} & \mid \mathbf{\Pi} x{:}t_1.t_2 & \text{(product)} & \mid t_1 = t_2 \in t & \text{(equality)} \\
& \mid \texttt{Base} & \text{(base)} & \mid \Sigma x{:}t_1.t_2 & \text{(sum)} & \mid t_1 {+} t_2 & \text{(disjoint union)} \\
& \mid \texttt{Name} & \text{(name type)} & \mid \cup x{:}t_1.t_2 & \text{(union)} & \mid t_1 \preccurlyeq t_2 & \text{(simulation)} \\
& \mid \mathbb{U}_i & \text{(universe)} & \mid \cap x{:}t_1.t_2 & \text{(intersection)} & \mid t_1 \simeq t_2 & \text{(bisimulation)} \\
& \mid t_1 // t_2 & \text{(quotient)} & \mid \{x : t_1 \mid t_2\} & \text{(set)} & & \\
t \in \texttt{Term} ::= & x & \text{(variable)} & \mid \texttt{let}\ x := \boxed{t_1}\ \texttt{in}\ t_2 & \text{(call-by-value)} & & \\
& \mid v & \text{(value)} & \mid \texttt{let}\ x, y = \boxed{t_1}\ \texttt{in}\ t_2 & \text{(spread)} & & \\
& \mid \boxed{t_1}\ t_2 & \text{(application)} & \mid \texttt{if}\ \boxed{t_1}{<}\boxed{t_2}\ \texttt{then}\ t_3\ \texttt{else}\ t_4 & \text{(less than)} & & \\
& \mid \nu x.\boxed{t} & \text{(fresh)} & \mid \texttt{fix}(\boxed{t}) & \text{(fixpoint)} & & \\
& \mid \textbf{iflam}(\boxed{t_1}, t_2, t_3) & \text{(lambda test)} & \mid \textbf{ifint}(\boxed{t_1}, t_2, t_3) & \text{(integer test)} & & \\
& \mid \texttt{if}\ \boxed{t_1}{=}\boxed{t_2}\ \texttt{then}\ t_3\ \texttt{else}\ t_4 & \text{(integer equality)} & & & & \\
& \mid \texttt{case}\ \boxed{t_1}\ \texttt{of}\ \texttt{inl}(x) \Rightarrow t_2 \mid \texttt{inr}(y) \Rightarrow t_3 & \text{(decide)} & & & &
\end{array}$$

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

Fig. 4. Syntax (top) and operational semantics (bottom) of a subset of Nuprl

2.2 Type System

Following Allen's PER semantics, Nuprl's types are interpreted as partial equivalence relations (PERs) on closed terms [5; 6; 40]. Allen's PER semantics can be seen as an inductive-recursive [44] 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.[5] For example, $T_1 {\equiv} T_2$ is true whenever 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 $t_1 {\equiv} t_2 {\in} A_1$, $B_1[x_1 \backslash t_1] {\equiv} B_2[x_2 \backslash t_2]$. The definitions of $a {\equiv} b {\in} T$ and $T_1 {\equiv} T_2$ include similar rules for all the other type constructors of CTT. The top part of Fig. 4 lists some of Nuprl's types (we only list here the ones that either mentioned or used in this paper—see `close`'s definition in https://github.com/vrahli/NuprlInCoq/blob/master/per/per.v for a complete list). 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 [10; 79]. 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 (expressions computing to) the constant $\star$, i.e., equality types do not have computational content. In the rest of this paper, we often write $a =_T b$ for the type $a = b \in T$, and $a \in T$ for the type $a = a \in T$.

[5]Following Allen's method, our Coq formalization uses a purely inductive definition instead of an inductive-recursive definition. Methods to translate a mutually inductive-recursive definition to a single inductive definition have been formally studied, for example by Capretta [31].

We further define `True` to be $0 =_{\mathbb{Z}} 0$ and `False` to be $0 =_{\mathbb{Z}} 1$. Note that the only member of `True` is $\star$, which is the only inhabitant of true equality types, as discussed above; and `False` is an uninhabited empty type. We sometimes write b for (`if` b `then True else False`), i.e., we use an implicit coercion from Booleans to propositions.

Let $P \longrightarrow Q$ denote the non-dependent product $\Pi n{:}P.Q$, and let $P \land Q$ denote the non-dependent sum $\Sigma n{:}P.Q$ (where non-dependent means that Q does not depend on n). In addition, Nuprl also features types beyond those in "standard" Martin-Löf-like type theories such as union and intersection types that were introduced as part of CTT in [62]. Among other things, b is a member of $\cup x{:}A.B$, if there exists a $a \in A$ such that $b \in B[x\backslash a]$; and b is a member of $\cap x{:}A.B$, if for all $a \in A$, then $b \in B[x\backslash a]$. Moreover, `Name` is the type of names [4], which we used in [87; 88] to validate a continuity inference rule. Our proof uses named exceptions to probe terms to compute the modulus of continuity of a function $F \in \mathcal{B} \longrightarrow \mathbb{N}$ at a point $f \in \mathcal{B}$. Names come with two operations [87; 88]: a fresh operator $\boldsymbol{\nu}$ to generate fresh names, and a test for equality (not shown here).

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 [51]. 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 [89]. For example, using the least upper bound theorem [40, 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. CTT provides the following types to reason about Howe's computational equivalent relation within Nuprl: `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 $\preccurlyeq$.

As mentioned above, we have formalized CTT in Coq [10; 79], 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 [10; 79; 88, Sec.2.4]. 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/NuprlInCoq/blob/master/RULES for a list of Nuprl's inference rules along with pointers to the proofs of their validity.

2.3 Diverging Terms

As mentioned in the introduction, Nuprl can assign types to diverging terms [96; 36; 40]. For example, the fixpoint $\mathtt{fix}(\lambda x.x)$ is a member of, among others, the partial type $\overline{\mathbb{Z}}$, which is the type of integers and diverging terms. The type $\overline{\mathbb{Z}}$ can be seen as the integer type of ML-like programming languages such as OCaml. Partial types are not the only ones that can be assigned to diverging terms. Nuprl's current function/pi and intersection types also 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 `False` $\longrightarrow$ `False` (or as $\cap$`False.False` using intersection types). By definition of function types all terms inhabit `Top`, even diverging terms such as $\mathtt{fix}(\lambda x.x)$—this was not the case in [35], where only λ-terms were allowed to inhabit function types.

2.4 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 [35, 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 [88, Sec.2.6] for more information on squashing. Using the HoTT terminology, we also sometimes truncate types at the *propositional level* [103, 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 [34]: $\rfloor T = T//\texttt{True}$. Because the members of a quotient type $T//E$ are the members of T, the members of $\rfloor T$ are then the members of T. Also, $\rfloor T$ is a proof-irrelevant type, i.e., its members are all equal to each other because if $x, y \in T$ then $(x =_{\rfloor T} y \iff \texttt{True})$. Note that $\rfloor T \rightarrow \downarrow T$ is true because it is inhabited by $\lambda x.\star$, but we cannot prove the converse because to prove $\rfloor T$ we have to exhibit an inhabitant of T, which $\downarrow T$ does not give us because only $\star$ inhabits $\downarrow T$.

2.5 Sequents and Rules

Sequents are of the form $h_1, \ldots, h_n \vdash T \lfloor \mathbf{ext}\ t \rfloor$. The term t is a member of the type T, which in this context is called the *extract* or *evidence* of T. Extracts are programs that are computed by the system once a proof is complete. We will sometimes omit proof extracts when they are irrelevant to the discussion. An hypothesis h is of the form $x : A$, where the variable x is referred to as the name of the hypothesis and A its type. Such a sequent states, among other things, that T is a type and t is a member of T. A rule is a pair of a conclusion sequent S and a list of premise sequents, $S_1, \cdots, S_n$, which we write as:

$$\frac{S_1 \quad \cdots \quad S_n}{S}$$

Several equivalent definitions for the validity of sequents appear in the Nuprl literature [35; 40; 62; 10; 88]. Since our results are invariant to the specific semantics, we do not repeat them here. The sequent semantics standardly induces the notion of validity of a rule, i.e., the validity of the premises entails the validity of the conclusion.

3 VARIANTS OF BAR INDUCTION

This section presents an unconstrained *squashed* BI principle, which we prove to be valid w.r.t. the PER semantics in Sec. 4. It also explains how versions of BID and BIM are derived from this squashed BI principle using bar recursion operators, and proves the negation of a non-$\rfloor$-squashed version of BIM.

3.1 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. 4. (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*)

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

where $\perp\!\!\!\perp$ is an empty sequence, defined as $\lambda x.\texttt{let}\ \_ := x\ \texttt{in}\ \perp$.

In Sec. 4.1 we take T to be $\mathbb{N}$, and in Sec. 4.2 the type of name-free closed terms.

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 $\star$ (we often omit the trivial extract $\star$ from sequents). 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. 4, 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 do have computational content, namely versions of BID and BIM. The conclusion of the `bar` hypothesis is $\downarrow$-squashed because some applications of this rule, such as BID, only use the bar for termination, in which case the `bar` hypothesis does not contribute to the extract, i.e., to the computational content of the induction principle.

We next discuss the $\perp\!\!\!\perp$ operator, which is used in the conclusion of `[BarInduction]` for technical reasons. Intuitively, in $(P\ 0\ \perp\!\!\!\perp)$, $\perp\!\!\!\perp$ could be replaced by any sequence because a sequence of length n is also a sequence of length 0. However, this is only true if P is a well-formed predicate on finite sequences, i.e., of type $\Pi n{:}\mathbb{N}.T^{\mathbb{N}_n} \rightarrow \texttt{Type}$. Here we want to avoid requiring one to have to prove that P is well-formed because we sometimes want to use this rule to prove that P is indeed well-formed. (However, we require the bar B to be well-formed as stated by the subgoal called `wfd`.) For example, we derive in Sec. 3 principles for non-$\downarrow$-squashed propositions by proving that some bar recursion operator br inhabits some proposition Q, i.e. $br \in Q$, using our $\downarrow$-squashed BI principle. The proposition $br \in Q$ is a $\downarrow$-squashed proposition, i.e. $br \in Q \iff \downarrow(br \in Q)$, because Nuprl's equality types can only be inhabited by the constant $\star$. To prove that $br \in Q$ is a true/inhabited type, we have to prove that it is well-formed, i.e., that $br \in Q$ is indeed a type, which we might not be able to prove because we might again need to use Bar Induction for that.

P being a well-formed predicate on finite sequences would allow us, in proving the validity of `[BarInduction]`, to sometimes replace a sequence s_1 of type $T^{\mathbb{N}_n}$ by another sequence s_2 of type $T^{\mathbb{N}_n}$ in an expression of the form $(P\ n\ s_1)$, given that $s_1 = s_2 \in T^{\mathbb{N}_n}$. This is what $\perp\!\!\!\perp$ allows us to do. We can then prove that this sequence is computationally equivalent to the sequence $\texttt{norm}(c, 0)$ for any term c, i.e. $\perp\!\!\!\perp \sim \texttt{norm}(c, 0)$ (see Sec. 2.2), where `norm` is defined as follows:

$$\texttt{norm}(s, n) = \lambda x.\texttt{if}\ x{<}0\ \texttt{then}\ \perp\ \texttt{else}\ \texttt{if}\ x{<}n\ \texttt{then}\ s(x)\ \texttt{else}\ \perp$$

This *normalization* operator returns $s(x)$ for $x \in \{0, \ldots, n-1\}$, and otherwise returns $\perp$. Therefore, when sequences are *normalized* using `norm`, if $s_1 = s_2 \in T^{\mathbb{N}_n}$, the two sequences $\texttt{norm}(s_1, n)$ and $\texttt{norm}(s_2, n)$ are then computationally equivalent, i.e. $\texttt{norm}(s_1, n) \sim \texttt{norm}(s_2, n)$ (see Sec. 2.2), which allows us to substitute $\texttt{norm}(s_1, n)$ for $\texttt{norm}(s_2, n)$ in $(P\ n\ \texttt{norm}(s_1, n))$ without having to prove, e.g., that P is a well-formed predicate on finite sequences.

Unfortunately, we cannot simply define $\perp\!\!\!\perp$ as $\lambda x.\perp$ because of exceptions, which we have added to Nuprl in [87]. If $\perp\!\!\!\perp$ was defined $\lambda x.\perp$, we would not be able to prove $\perp\!\!\!\perp \sim \texttt{norm}(c, 0)$, because Nuprl's computation system is lazy: $\lambda x.\perp$ returns $\perp$ when applied to an exception while $\texttt{norm}(c, 0)$ returns the exception.

3.2 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 as discussed in Sec. 1. We provide a list of such terms along with their types:

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

Note that the Σ type in $\texttt{bar}_{\downharpoonright}$'s type is $\downharpoonright$-squashed and not $\downarrow$-squashed as in $\texttt{bar}_{\downarrow}$ and in `[BarInduction]` because in Sec. 3.6 we need the bar hypothesis to have some computational content to build a realizer for BIM. We can trivially prove that $\texttt{bar}_{\downharpoonright}$ 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 $n > m$.

3.3 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 [97]. Spector mentioned some relation between SBR and BID, and Howard showed that his W operator [49, p.111], which can be reduced to SBR, realizes BIM (see Sec. 3.6). SBR can be defined as the following parametrized recursive operator (a minor difference: Spector's operator uses $<_z$ instead of $\leq_z$)—see Nuprl definition spector-bar-rec:

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

$$\mathsf{SBR}(Y,G,H,n,s) = \texttt{if}\ Y\ n\ s \leq_z n\ \texttt{then}\ G\ n\ s\ \texttt{else}\ H\ n\ s\ (\lambda t.\mathsf{SBR}(Y,G,H,n+1,s \oplus_n t))$$

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. 3.4 and 3.6.

Spector used a restricted form of SBR to interpret the double-negation shift, which he used in his consistency proof [97, Sec.10]. Oliva and Powell [81] 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 [97, Sec.6.4,p.9; 100, 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 Sec. 3.5. Also, note that this implies that checking whether we have reached the bar has to be decidable. As mentioned in [97, p.9,Footnote 6], and as further explained in Sec. 3.6, this can be ensured by the fact that we can compute the modulus of continuity of the bar.

3.4 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. 3.1 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* – DBR)

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

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

Definition 4 (BID)

$$\begin{array}{rl} \mathtt{BID} = & \Pi B, P{:}(\Pi n{:}\mathbb{N}.\mathcal{B}_n \rightarrow \mathbb{P}). \\ & (\Pi s{:}\mathcal{B}.\downarrow\Sigma n{:}\mathbb{N}.B(n, s)) \\ & \rightarrow (\Pi n{:}\mathbb{N}.\Pi s{:}\mathcal{B}_n.(\Pi m{:}\mathbb{N}.P(n+1, s \oplus_n m)) \rightarrow P(n, s)) \\ & \rightarrow (\Pi n{:}\mathbb{N}.\Pi s{:}\mathcal{B}_n.B(n, s) \lor \neg B(n, s)) \\ & \rightarrow (\Pi n{:}\mathbb{N}.\Pi s{:}\mathcal{B}_n.B(n, s) \rightarrow P(n, s)) \\ & \rightarrow P(0, \perp\!\!\perp) \end{array}$$

Theorem 1 (*Bar Induction on Decidable bars – BID*) BID is true in CTT assuming `[BarInduction]` (i.e., the hypotheses $\mathtt{bar}_\downarrow$, dec, base, and ind defined in Sec. 3.2 imply that $\mathtt{DBR}(\mathtt{dec}, \mathtt{base}, \mathtt{ind}, 0, \perp\!\!\perp)$ inhabits the proposition $P(0, \perp\!\!\perp)$).

Proof outline. As mentioned in Sec. 3.3, 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 ($\mathtt{dec}\ n\ s$) returns $\mathtt{inl}(r)$, 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, ($\mathtt{dec}\ n\ s$) returns $\mathtt{inr}(r)$ 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 1 (DBR *as* SBR)

$$\begin{array}{rl} \mathtt{DBR}(dec, base, ind, n, s) = & \mathtt{SBR}(\lambda n, s.\mathtt{if}\ dec\ n\ s\ \mathtt{then}\ 0\ \mathtt{else}\ n+1 \\ & , \lambda n, s.\mathtt{case}\ dec\ n\ s\ \mathtt{of}\ \mathtt{inl}(r) \Rightarrow base\ n\ s\ r \\ & \qquad\qquad\qquad\qquad \mid \mathtt{inr}(\_) \Rightarrow \perp \\ & , ind, n, s) \end{array}$$

Note that 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:*

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

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.*

3.5 Continuity Principle

As mentioned above, continuity is used to ensure the termination of bar recursors. We use here variants of what is sometimes called the *Strong Continuity Principle for numbers* [92], which we have proved to be valid w.r.t. Nuprl's PER semantics (see [87; 88] as well as the Coq file https://github.com/vrahli/NuprlInCoq/blob/master/continuity/continuity_roadmap.v). More precisely, in Sec. 3.6 we use a variant of Brouwer's Strong Continuity Principle to define (a variant of) Howard's W bar recursion operator.

3.5.1 Strong Continuity Principle

We first define the following variant of Brouwer's Strong Continuity Principle, which is derivable from the one in [87]—see lemma strong-continuity-rel-unique:

Definition 5 (*Strong Continuity Principle* – SCP)

$$\begin{aligned}
\mathsf{SCP} = \ & \Pi P{:}(\mathcal{B} \to \mathbb{N} \to \mathbb{P}).\\
& \quad (\Pi f{:}\mathcal{B}.\downharpoonleft\Sigma n{:}\mathbb{N}.\ P(f,n))\\
& \quad \to \ \downharpoonleft\Sigma M{:}(\Pi n{:}\mathbb{N}.\mathcal{B}_n \to (\mathbb{N}_n+\mathtt{True})).\\
& \qquad\quad \Pi f{:}\mathcal{B}.\Sigma n{:}\mathbb{N}.\Sigma k{:}\mathbb{N}_n.\\
& \qquad\qquad P(f,k)\\
& \qquad\qquad \land M(n,f) = \mathtt{inl}(k) \in \mathbb{N}_n+\mathtt{True}\\
& \qquad\qquad \land \Pi m{:}\mathbb{N}.\mathtt{isl}(M(m,f)) \to m =_{\mathbb{N}} n
\end{aligned}$$

This Strong Continuity Principle essentially says that there is a uniform way, called M in the above formula (such a function is often called a neighborhood function [102, p.212]), to decide whether n is the modulus of continuity of P at f, and if so returns a number n such that $P(f,n)$ [60, pp.70–71].

This version of SCP differs from the one in [87] as follows: (1) here we present its relational version instead of its functional version, i.e., we assume the existence of a predicate that relates numbers and infinite sequences using a $\downharpoonleft$-squashed Σ type, while [87] assumes the existence of a function; and (2) here M is of type $(\Pi n{:}\mathbb{N}.\mathcal{B}_n \to (\mathbb{N}_n+\mathtt{True}))$ as opposed to $(\Pi n{:}\mathbb{N}.\mathcal{B}_n \to (\mathbb{N}+\mathtt{True}))$ in [87], i.e., we are guaranteed that the modulus of continuity n of P at f will be larger than the value k that M returns (i.e., such that $M(n,f) = \mathtt{inl}(k)$) such that $P(f,k)$ is true—or taking P as a function as in [87], that $P(f) < n$. In Sec. 3.6 we use the modulus of continuity of BI's bar hypothesis to define the monotone bar recursion operator HBR so that we know that we only need to check initial segments of infinite sequences to decide whether we have reached the bar. Therefore, (2) is useful because we then know that if we have reached the modulus of continuity of the bar then we are past the bar. As it turns out, Sec.3.5.2 presents an even more convenient version of the Strong Continuity Principle, which we have used to define HBR.

As mentioned by Bridges and Richman [27, p.119], SCP is equivalent to a "principle of continuous choice", which they divide into a continuous part, namely what is often called the Weak Continuity Principle (WCP), and a choice part, namely the axiom of choice often referred to as AC $_{1,0}$, which is true in Nuprl (see Nuprl lemma axiom-choice-1X-quot):

$$\begin{aligned}\mathtt{WCP} &= \Pi F{:}\mathbb{N}^{\mathcal{B}}.\Pi f{:}\mathcal{B}.\downharpoonright\Sigma n{:}\mathbb{N}.\ \Pi g{:}\mathcal{B}.f =_{\mathcal{B}_n} g \to F(f) =_{\mathbb{N}} F(g)\\ \mathtt{AC}_{1,0} &= \Pi f{:}\mathcal{B}.\downharpoonright\Sigma n{:}\mathbb{N}.\ P(f,n) \to \downharpoonright\Sigma F{:}\mathbb{N}^{\mathcal{B}}.\ \Pi f{:}\mathcal{B}.P(f,F(f))\end{aligned}$$

where P is a predicate of type $\mathcal{B} \to \mathbb{N} \to \mathbb{P}$.

As first shown by Kreisel in [66, p.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 [98, Thm.IIA], the inconsistency of N-HA$^{\omega}$ (a "neutral" version of HA$^{\omega}$ that "permits extensional as well as intensional interpretations of equality at higher types" [101]) extended with (1) Brouwer's continuity principle, (2) a function extensionality axiom, and (3) a version of the axiom of choice AC $_{2,0}$. We have proved this inconsistency in Nuprl when the existential quantifier is interpreted as Σ: unsquashed-continuity-false-troelstra; and we have proved that both the $\downharpoonright$-squashed version of AC $_{2,0}$ and its $\downarrow$-squashed version:

$$\begin{aligned}\mathtt{AC}_{2,0_{\downharpoonright}} &= \Pi f{:}\mathbb{N}^{\mathcal{B}}.\downharpoonright\Sigma n{:}T.\ P(f,n) \to \downharpoonright\Sigma F{:}T^{(\mathbb{N}^{\mathcal{B}})}.\ \Pi f{:}\mathbb{N}^{\mathcal{B}}.P(f,F(f))\\ \mathtt{AC}_{2,0_{\downarrow}} &= \Pi f{:}\mathbb{N}^{\mathcal{B}}.\downharpoonright\Sigma n{:}T.\ P(f,n) \to \downarrow\Sigma F{:}T^{(\mathbb{N}^{\mathcal{B}})}.\Pi f{:}\mathbb{N}^{\mathcal{B}}.P(f,F(f))\end{aligned}$$

where T is a non-empty type (such as $\mathbb{N}$) and P is a predicate of type $\mathbb{N}^{\mathcal{B}} \to T \to \mathbb{P}$, are false in Nuprl because they contradict continuity: see Nuprl lemmas notAC20 and notAC20-ssq. Escardó and Xu [45] proved in Agda, without using function extensionality but allowing reductions under λs, that the non-truncated version of WCP is false in a Martin-Löf-like type theory such as Nuprl.

3.5.2 Barred Strong Continuity Principle

Instead of SCP, we will use the following *barred* variant, called BSCP, which is derivable from SCP–see Nuprl lemma strong-continuity-rel-unique-pair:

Definition 6 (*Barred Strong Continuity Principle* – BSCP)

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

where $\mathtt{barred}(P,n,s) = \Sigma k{:}\mathbb{N}_n.P(s\uparrow^0_n,k)$ 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 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 the HBR monotone bar recursion operator below in Def. 7 than when using the standard definition of the Strong Continuity Principle as in Def. 5, where $\mathtt{barred}(P,n,s)$ is simply $\mathbb{N}_n$. As it turns out, in Sec. 3.6 we use BSCP instead of SCP because the information provided by M in SCP is not convenient to use BI's base hypothesis. As in DBR, we also need a proof that we have reached the bar, i.e., a proof of $B(n,s)$ for some finite sequence given by n and s. This information is provided by the condition on M in SCP. In order to simplify the definition of HBR, we used the BSCP variant of SCP instead, where M returns all the information we need to define HBR.

3.6 $\downharpoonleft$-Squashed Bar Induction on Monotone Bars

A few years after Spector [97] introduced his bar recursion operator, Howard [49] 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. 3.1 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 [97, 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 an 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, $(\mathtt{BSCP}\ (\lambda s, n.B(n, s))\ \mathtt{bar}_{\downharpoonleft})$, provides the following M:

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

such that:

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

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

Definition 7 (*Monotone bar recursion operator* – HBR)

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

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

Definition 8 ($\mathtt{BIM}_{\downharpoonleft}$)

$$\begin{aligned} \mathtt{BIM}_{\downharpoonleft} = \ & \Pi B, P{:}(\Pi n{:}\mathbb{N}.\mathcal{B}_n \to \mathbb{P}). \\ & (\Pi s{:}\mathcal{B}.\downharpoonleft\Sigma n{:}\mathbb{N}.\ B(n, s)) \\ & \to (\Pi n{:}\mathbb{N}.\Pi s{:}\mathcal{B}_n.(\Pi m{:}\mathbb{N}.P(n+1, s \oplus_n m)) \to P(n, s)) \\ & \to (\Pi n{:}\mathbb{N}.\Pi m{:}\mathbb{N}_n.\Pi s{:}\mathcal{B}_n.B(m, s) \to B(n, s)) \\ & \to (\Pi n{:}\mathbb{N}.\Pi s{:}\mathcal{B}_n.B(n, s) \to P(n, s)) \\ & \to \downharpoonleft P(0, \perp\!\!\!\perp) \end{aligned}$$

Theorem 2 ($\downharpoonleft$*-squashed Bar Induction on Monotone bars* – $\mathtt{BIM}_{\downharpoonleft}$)

$\mathtt{BIM}_{\downharpoonleft}$ is true in CTT assuming [BarInduction] (i.e., the hypotheses $\mathtt{bar}_{\downharpoonleft}$, $\mathtt{mon}^*$, base, and ind defined in Sec. 3.2 imply that $\downharpoonleft P(0, \perp\!\!\!\perp)$ is inhabited by $\mathtt{HBR}(M, \mathtt{mon}^*, \mathtt{base}, \mathtt{ind}, 0, \perp\!\!\!\perp)$).

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 partially answer this question below in Sec. 3.8.

Proof outline. 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. 3.1. 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:

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

`[BarInduction]`'s base hypothesis:

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

and `[BarInduction]`'s induction hypothesis:

$$\Pi n{:}\mathbb{N}.\Pi s{:}\mathcal{B}_n.\Pi i{:}(\Pi m{:}\mathbb{N}.\mathtt{HBR}(n + 1, s \oplus_n m) \in P(n + 1, s \oplus_n m)).\mathtt{HBR}(n, s) \in P(n, s) \tag{5}$$

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 2 (HBR *as* SBR)

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

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 [60, p.78], and we can prove that HBR is also an instance of DBR—see Nuprl lemma howard-bar-rec-equal-decidable:

Lemma 3 (HBR *as* DBR)

$\mathtt{HBR}(M, mon^*, base, ind, n, s) = \mathtt{DBR}(M, \lambda n, s, r.\mathtt{let}\ k, p = r\ \mathtt{in}\ base\ n\ s\ (mon^*\ n\ k\ s\ p), ind, n, s)$

3.7 Generalizing BIM

Before proving that the non-$\downharpoonleft$-squashed version of BIM is false in Sec. 3.8, 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 $\mathtt{gBIM}_{\downharpoonleft}$, 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 9 (*Generalized BIM* – $\mathtt{gBIM}_{\downharpoonleft}$)

$$\begin{aligned}\mathtt{gBIM}_{\downharpoonleft} = \Pi P{:}(\Pi n{:}\mathbb{N}.\mathcal{B}_n \to \mathbb{P}).&\\ &(\Pi s{:}\mathcal{B}.\downharpoonleft\Sigma n{:}\mathbb{N}.\ \Pi m{:}\{n\ldots\}.P(m,s))\\ &\to\ (\Pi n{:}\mathbb{N}.\Pi s{:}\mathcal{B}_n.(\Pi m{:}\mathbb{N}.P(n+1, s \oplus_n m)) \to P(n,s))\\ &\to\ \downharpoonleft P(0, \perp\!\!\!\perp)\end{aligned}$$

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

Theorem 3

$\mathtt{gBIM}_{\downharpoonleft}$ is true in CTT assuming `[BarInduction]`.

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

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

where M is the neighborhood function of our *bar* hypothesis, i.e.:

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

where $Q = \lambda n, s.\Pi m{:}\{n\ldots\}.P(m,s)$, and such that:

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

The rest of the proof is similar to the one presented in Sec. 3.6. □

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[6], 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. It is straightforward to prove that $\mathtt{gBIM}_{\downharpoonleft}$ implies $\mathtt{BIM}_{\downharpoonleft}$ (see Def. 8)—see Nuprl lemma gen-bar-ind-implies-monotone:

[6]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.

Lemma 4 ($\texttt{gBIM}_\downharpoonright$ *implies* $\texttt{BIM}_\downharpoonright$)

$\texttt{gBIM}_\downharpoonright$ implies $\texttt{BIM}_\downharpoonright$ in CTT.

3.8 Negation of Non-$\downharpoonright$-Squashed BIM

We now prove that the $\downharpoonright$ operator in the above versions of BIM (see for example Def. 8) is necessary, i.e., that the following non-$\downharpoonright$-squashed version of BIM, which we call $\texttt{uBIM}$, 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 10 ($\texttt{uBIM}$)

$$\begin{aligned}\texttt{uBIM} = \ & \Pi B, P{:}(\Pi n{:}\mathbb{N}.\mathcal{B}_n \to \mathbb{P}).\\ & (\Pi s{:}\mathcal{B}.\downharpoonright\Sigma n{:}\mathbb{N}.\ B(n,s))\\ & \to\ (\Pi n{:}\mathbb{N}.\Pi s{:}\mathcal{B}_n.(\Pi m{:}\mathbb{N}.P(n+1, s \oplus_n m)) \to P(n,s))\\ & \to\ (\Pi n, m{:}\mathbb{N}.\Pi s{:}\mathcal{B}_n.B(n,s) \to B(n+1, s\oplus_n m))\\ & \to\ (\Pi n{:}\mathbb{N}.\Pi s{:}\mathcal{B}_n.B(n,s) \to P(n,s))\\ & \to\ P(0, \perp\!\!\!\perp)\end{aligned}$$

As discussed below, we still require that the bar be $\downharpoonright$-squashed.

Lemma 5

$\texttt{uBIM}$ implies a non-squashed version of $\texttt{WCP}$ in CTT.

Proof outline. Suppose $F \in \mathbb{N}^{\mathcal{B}}$ and $f \in \mathcal{B}$. We have to show: $\Sigma n{:}\mathbb{N}.\Pi g{:}\mathcal{B}.f =_{\mathcal{B}_n} g \to F(f) =_{\mathbb{N}} F(g)$. To prove this, we instantiate $\texttt{uBIM}$ with:

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

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

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

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

Induction. To prove the induction hypothesis we instantiate $\Pi m{:}\mathbb{N}.P(n+1, s\oplus_n m)$ 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 $(s \oplus_n m) \boxplus_{n+1} f$.

For the full formal proof see Nuprl lemma unsquashed-BIM-implies-unsquashed-weak-continuity. □

Lemma 6 ($\neg\texttt{uBIM}$)

$\texttt{uBIM}$ and the non-squashed version of $\texttt{gBIM}_\downharpoonright$ are both false in CTT.

Proof outline. Recall that the non-squashed Continuity Principle, as mentioned in Sec. 3.5, is false in Nuprl, i.e.:

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

is true in Nuprl. This, together with Lemma 5 and the fact that the non-squashed $\texttt{gBIM}_{\downarrow}$ implies `uBIM` yields the negative results. □

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

4 VALIDATING BI INFERENCE RULES

Sec. 3 presented an unconstrained $\downarrow$-squashed BI principle, from which we have derived BID and BIM principles. In this section we establish the validity of instances of this BI principle w.r.t. Nuprl's PER semantics. Sec. 4.1 proves that our `[BarInduction]` inference rule is valid w.r.t. Nuprl's PER semantics when $T = \mathbb{N}$ (see https://github.com/vrahli/NuprlInCoq/blob/master/bar_induction/bar_induction3.v); while Sec. 4.2 proves its validity for sequences of name-free closed terms (see https://github.com/vrahli/NuprlInCoq/blob/master/bar_induction/bar_induction_cterm4.v).

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 P is true about the empty sequence [43, Sec.3.4; 102, Sec.8.18; 105, 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 branches is finite, and which is built-up from three kinds of inference steps: *monotone* (also called upward [102], backward [104], and ζ-inferences [29; 43]) and *inductive* (also called downward [102], forward [104], and $\digamma$-inferences [29; 43]) steps corresponding to the monotone and inductive predicates introduced above, as well as *immediate* steps [102] (also called opening statements [104] or η-inferences [43]) 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 [49], which realizes BIM (see Sec. 3.6). As explained for example in [43, Sec.3.4], while Brouwer might have believed that the monotone ζ-steps were not necessary in canonical proofs, this was then refuted by Kleene [60, Sec.7.14,Lem.*27.23]. In [102, p.233] it was shown that monotone ζ-steps can only be eliminated when the bar is monotone or decidable. As explained below in more detail, monotone steps are not necessary when proving *squashed* propositions, which do not have any computational content.

4.1 BI for Sequences of Natural Numbers

The theorem below establishes the consistency of Nuprl with the squashed-BI principle.

Theorem 4 (*Validity of* `[BarInduction]`)

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

Proof outline. We have proved this following Dummett's standard classical proof [43, 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[7]: 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

[7]For readability, we omit some technicalities here regarding the well-formedness of terms, which are discussed in Sec. 3.1, in particular that finite sequences have to be *normalized*.

$\neg P(0, \bot\!\!\!\bot)$ too. Then, we contrapose our induction hypothesis (`ind`), and using the 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 $\neg P(0, \bot\!\!\!\bot)$, 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 $n \in \mathbb{N}$, $\neg P(n, \alpha)$. □

4.1.1 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 [60, 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 [14] denoted $\lambda\!\!\!\lambda x.M_x$, 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 realizes the negative translation of the axiom of choice. Similarly, as mentioned in [85], using our choice sequences, we have proved the validity of versions of the axiom of choice. In [14] 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 ↓-squashed and its computational content is the constant ⋆. 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. By the theoretical Nuprl syntax, we 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.[8]

Our choice sequences are also similar to Howe's set-theoretical functions in [53; 54; 52] (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 11 (*Nuprl's syntax with choice sequences*)

Nuprl's (metatheoretical) term syntax presented in Sec. 2 is extended with choice sequences, as well as an *eager* application operator:

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

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

For example, `seq`(`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,

[8]In recent work, we have explored ways to finitely extend the theoretical Nuprl syntax in order to include choice sequences in terms.

seq(fun $n \Rightarrow n+1$)(1) reduces to 2; seq(fun $n \Rightarrow n+1$)($\bot$) diverges; and seq(fun $n \Rightarrow n+1$)($\star$) gets stuck.

Definition 12 (*Computing with choice sequences*)

The following reduction steps are added to support computations with choice sequences:

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

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:

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

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

Remark 2 (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.*

4.1.2 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 a is an expression, which is not constrained in any way (i.e., one does not need to show that it is in some type); and 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:

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

where

$$\mathtt{isChoiceSeq}(x,z,f) = \cap x{:}\mathtt{Base}.\cap z{:}\mathtt{halts}(x).\mathbf{ifint}(x,\mathtt{True},f(x) \preceq \bot)$$

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 $\bot$ 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 **iflam**$(f, \mathtt{tt}, \mathtt{ff})$, which checks whether f computes to a λ-term: if it does then the conclusion is realized by `tt`, i.e. `inl`$(\star)$, because $\star$ realizes the left-hand-side of the $\vee$; otherwise, the conclusion is realized by `ff`, i.e. `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 `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.

4.2 BI For Sequences of Terms

Intuitively a similar proof as the one presented at the beginning of Sec. 4.1 could be used at least when T is `Base` (defined in Sec. 2.2). Deriving a Bar Induction rule for a larger class of sequences than sequences of numbers as in Sec. 4.1 would allow deriving induction principles for a larger class of types using the technique presented in Sec. 5. Following the same scheme as in Sec. 4.1, 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 Nuprl's "fresh" $\boldsymbol{\nu}$ operator. We explain this issue below in more details.

4.2.1 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 [87], 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: $\{t : \mathtt{Base} \mid (t : \mathtt{Base})\#\}$, 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_ctemr4.v).

4.2.2 Could Names Occur in Sequences?

We next suggest a possible solution that consists in introspecting computations, whose further study is left for future work. 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. 4.2.1: $\boldsymbol{\nu} x.(\texttt{seq}(\texttt{fun}\ n \Rightarrow \boldsymbol{a})\ 0)$. Following the procedure we just described, we first pick a name that is fresh w.r.t. $(\texttt{seq}(\texttt{fun}\ n \Rightarrow \boldsymbol{a})\ 0)$ 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 $(\texttt{seq}(\texttt{fun}\ n \Rightarrow \boldsymbol{a})\ 0)[x\backslash \boldsymbol{c}]$ to $\boldsymbol{a}$ in one computation step. Now we pick a name $\boldsymbol{b}$, which is fresh w.r.t. both $(\texttt{seq}(\texttt{fun}\ n \Rightarrow \boldsymbol{a})\ 0)$ and $\boldsymbol{a}$, and we reduce $(\texttt{seq}(\texttt{fun}\ n \Rightarrow \boldsymbol{a})\ 0)[x\backslash \boldsymbol{b}]$ 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 $\boldsymbol{\nu} x.\boldsymbol{a}$.

Names pose an additional challenge in preserving the key property that Howe's equivalence relation $\sim$, mentioned in Sec. 2.2, is a congruence as proved in [51]. To prove this, Howe first shows that his approximation relation $\preccurlyeq$ is a congruence [51]—Howe's computational equivalence relation is defined on closed terms as follows: $t \sim u$ if $t \preccurlyeq u \land u \preccurlyeq t$. Unfortunately, this is not easy to prove directly. Howe's "trick" was to define another inductive relation $\preccurlyeq^*$, which is a congruence and contains $\preccurlyeq$ by definition.

In order to deal with the $\boldsymbol{\nu}$ operator in [87], we had to slightly modify $\preccurlyeq^*$ in the following way: in order to prove that $\boldsymbol{\nu} x.t \preccurlyeq^* u$, we have to prove that there exists a t' such that $t[x\backslash \boldsymbol{a}] \preccurlyeq^* t'[x\backslash \boldsymbol{a}]$ and $\boldsymbol{\nu} x.t' \preccurlyeq u$, where the name $\boldsymbol{a}$ has to be fresh w.r.t. t and t'. Unfortunately, when names are allowed in choice sequences we cannot anymore compute such a name because it is not decidable anymore whether a name occurs in a term. Thus, the question of whether the $\preccurlyeq^*$ relation can be adapted so to deal with names in choice sequences remains open.

4.3 Externalizing BI's Validity Proof

The proof of BI's validity presented in Sec. 4.1 seemingly relies on classical axioms. It turns out that these principles are consistent with Nuprl's PER semantics [6; 5; 40], and can therefore be arguably considered as being constructive (but not intuitionistic according to Troelstra and van Dalen [102, pp.4–5]). In this section we identify the "minimal" such axioms that are truly indispensable and show that they are compatible with Nuprl at the theory level. See squashed-bar-ind-as-a-lemma for a formalization in Nuprl of the proof presented in this section.

Let us prove $\downarrow P(0, \perp\!\!\!\perp)$ in Nuprl. First we use the law of excluded middle in order to get to assume $\neg\downarrow P(0, \perp\!\!\!\perp)$. Unfortunately the law of excluded middle is false in Nuprl when non squashed because, for example, it contradicts continuity (see [88, Sec.6.3]). However, because the proposition we are proving is $\downarrow$-squashed, we only need the following $\downarrow$-squashed version of the law of excluded middle, which is consistent with Nuprl, as explained in [10; 40; 63]:

$$\begin{array}{c} H \vdash \downarrow(P + \neg P) \\ \text{BY [LEM]} \\ H \vdash P \in \mathbb{U}_i \end{array}$$

One can prove that this inference rule is consistent with Nuprl using the law of excluded middle in the metatheoretical proof of its validity w.r.t. Nuprl's PER semantics as shown in https://github.com/vrahli/NuprlInCoq/blob/master/rules/rules_classical.v. This seemingly classical principle is therefore computationally justified in the sense that the conclusion of the rule is inhabited by $\star$. As proved in [63, Thm.4.2], it implies Markov's principle, which is a principle of constructive recursive mathematics (CRM), also called Russian constructive mathematics [27, Ch.3]. We instantiate this $\downarrow$-squashed law of excluded middle principle with $\downarrow P(0, \perp\!\!\!\perp)$, and get to assume $\downarrow(\downarrow P(0, \perp\!\!\!\perp) + \neg\downarrow P(0, \perp\!\!\!\perp))$, which we can unsquash because we are proving a squashed proposition. If $\downarrow P(0, \perp\!\!\!\perp)$ is true then we can conclude directly. Let us now assume that $\neg\downarrow P(0, \perp\!\!\!\perp)$ is true, and let us prove `False`. From BI's base and bar hypotheses we deduce:

$$\Pi s{:}\mathcal{B}.\downarrow\Sigma n{:}\mathbb{N}.\downarrow P(n, s)$$

Next, we use again the $\downarrow$-squashed law of excluded middle to turn the induction hypothesis:

$$\Pi n{:}\mathbb{N}.\Pi s{:}\mathcal{B}_n.(\Pi m{:}\mathbb{N}.P(n+1, s \oplus_n m)) \longrightarrow P(n, s)$$

into:

$$\Pi n{:}\mathbb{N}.\Pi s{:}\mathcal{B}_n.\neg\downarrow P(n, s) \longrightarrow \downarrow\Sigma m{:}\mathbb{N}.\neg\downarrow P(n+1, s \oplus_n m)$$

In the metatheoretical Coq proof presented in Sec. 4.1, we used the axiom of choice to extract a choice sequence $\alpha \in \mathcal{B}$ from this formula such that for all $n \in \mathbb{N}$, $\neg P(n, \alpha)$. Instead here we use the following principle to recursively define choice sequences:

$$\begin{array}{l} H \vdash \downarrow\Sigma f{:}\mathcal{B}.\Pi n{:}\mathbb{N}.\downarrow P(n, f) \\ \quad \text{BY } \texttt{[ChoiceSequenceRec]} \\ \quad H \vdash P(0, s) \\ \quad H, n : \mathbb{N}, f : \mathcal{B}_n, z : P(n, f) \vdash \downarrow\Sigma m{:}\mathbb{N}.P(n+1, f \oplus_n m) \\ \quad H, n : \mathbb{N}, f : \mathcal{B}_n \vdash P(n, f) \in \texttt{Type} \end{array}$$

We have proved that this inference rule is valid w.r.t. Nuprl's PER semantics using Coq's axiom of choice: see Coq file https://github.com/vrahli/NuprlInCoq/blob/master/axiom_of_choice/choice_sequence_ind.v. Using this principle we get to assume $\downarrow\Sigma f{:}\mathcal{B}.\Pi n{:}\mathbb{N}.\downarrow\neg\downarrow P(n, f)$, which is inconsistent with our hypothesis $\Pi s{:}\mathcal{B}.\downarrow\Sigma n{:}\mathbb{N}.\downarrow P(n, s)$.

This allows us to prove the $\downarrow$-squashed and unconstrained BI principle presented in Sec.3.1 directly in Nuprl, with one drawback, which we discuss below. In the spirit of reverse mathematics [95; 58], we have decomposed our $\downarrow$-squashed BI rule into a $\downarrow$-squashed excluded middle rule and the `[ChoiceSequenceRec]` choice principle.

Unfortunately, to use `[LEM]` we need to be able to prove that P is a type as stated in `[LEM]`'s single subgoal. Similarly, to use the `[ChoiceSequenceRec]` inference rule we need to be able to prove that P is a well-formed predicate on finite sequences: see `[ChoiceSequenceRec]`'s third subgoal, which as we see below is necessary for the rule to be valid. This means that this direct proof in Nuprl of the $\downarrow$-squashed and unconstrained BI principle is only for well-formed predicates on finite sequences, while the BI rule presented in Sec. 3.1 does not require one to prove that P is a well-formed predicate on finite sequences. The next technical paragraph explains why this is an issue.

If we require one to prove the predicate's well-formedness in order to use the $\downarrow$-squashed and unconstrained BI principle, then it is not clear whether we can prove BID or BIM from this version of BI, which might render our direct Nuprl proof of BI less useful than the rule presented in Sec. 3.1—squashed-bar-ind-as-a-lemma can still be used to prove $\downarrow$-squashed propositions. The reason is that, in the case of BID for example (see Nuprl lemma decidable-bar-rec_wf), we use the $\downarrow$-squashed and unconstrained BI principle, i.e. rule `[BarInduction]`, to prove $\mathtt{DBR}(\mathtt{dec}, \mathtt{base}, \mathtt{ind}, 0, \perp\!\!\!\perp) \in P(0, \perp\!\!\!\perp)$, which is a $\downarrow$-squashed proposition because equality types can only be inhabited by $\star$, but in general we have no way of proving that $\lambda n, s.(\mathtt{DBR}(\mathtt{dec}, \mathtt{base}, \mathtt{ind}, n, s) \in P(n, s))$ is a well-formed predicate on finite sequences because in general it means that $\mathtt{DBR}(\mathtt{dec}, \mathtt{base}, \mathtt{ind}, n, s) \in P(n, s)$ has to be true, which is basically what we are trying to prove. It is therefore essential for our proof of BID that `[BarInduction]` does not require one to prove that P is a well-formed predicate on finite sequences.

As mentioned above, `[ChoiceSequenceRec]` would not be valid without the third subgoal that says that P has to be a well-formed predicate on finite sequences. The reason is that the base and induction hypotheses of this rule only ensure that P is well-formed on the sequence α they define (and the sequences that differ from α only at one place), while, according to the semantics of Σ types, the conclusion of this rule requires us to prove that P is well-formed on all possible sequences in $\mathcal{B}$. Let us provide an example. Let P be:

```
λn, s. if n=0 then True
          else if n=1 then s(0) ≃ 0
                 else if s(0)=0 then True
                        else ★
```

`[ChoiceSequenceRec]`'s base subgoal is true because $P(0, s)$ computes to True. Its induction subgoal is also true because if $P(n, f)$ is true then: (1) either $n = 0$ and then we can prove that $P(1, f \oplus_0 0)$ is true and $P(1, f \oplus_0 m)$ is well-formed for all $m \in \mathbb{N}$; (2) or $n = 1$ and then we get that $f(0) \simeq 0$, and we can prove that $P(2, f \oplus_1 0)$ and $P(2, f \oplus_1 m)$ is well-formed for all $m \in \mathbb{N}$; (3) or $n > 1$ and then we again get that $f(0) \simeq 0$ because otherwise $P(n, f)$ is not well-formed, and we can prove that $P(n + 1, f \oplus_n 0)$ and $P(n + 1, f \oplus_n m)$ is well-formed for all $m \in \mathbb{N}$. However, $P(n, f)$ is not well-formed for all $n \in \mathbb{N}$ and $f \in \mathcal{B}$ because $P(2, \lambda x.1)$ computes to $\star$, which is not a type. See Coq file https://github.com/vrahli/NuprlInCoq/blob/master/axiom_of_choice/choice_sequence_ind2.v for a formal proof.

5 DERIVING W TYPES FROM BI

This section describes how one can derive inductive types as W types [71; 78], and especially their induction principles using BI. A similar construction was described in [21], where the authors built indexed W types. For simplicity, we only focus here on non-indexed W types.

The construction goes as follows:

(1) We first define co-W type, also sometimes called M types, in Sec. 5.1.
(2) We then define W types as finite co-W types in Sec. 5.2.
(3) We prove an induction principle for W types using Bar Induction in Sec. 5.3.

See for example [1, Sec.5.2] for a discussion of W and M types. Related to the construction presented here and in [21], Altenkirch et al. showed how to build M types from W types [2; 8]. Instead, here we build W types from M types. The results presented here have been formalized in Nuprl: http://www.nuprl.org/LibrarySnapshots/Published/Version2/Standard/co-recursion.

5.1 M Types

One way of building coinductive types in Nuprl is using intersection types as follows:

$$\mathtt{corec}(F) = \cap n{:}\mathbb{N}.F^n(\mathtt{Top})$$

where $F^0(T) = T$ and $F^{n+1}(T) = F(F^n(T))$ (see Sec. 2.3 for details on Top). As explained in [21], $\mathtt{corec}(F)$ is the greatest fixed point of F if F is monotone and an ω limit preserving function. A function F on types is monotone if $T_1 \sqsubseteq T_2$ implies $F(T_1) \sqsubseteq F(T_2)$ for any two types T_1 and T_2. The type $T_1 \sqsubseteq T_2$ expresses that T_1 is a subtype of T_2, and is defined as $\lambda x.x \in T_1 \to T_2$. A function F on types is an ω limit preserving function if $\cap n{:}\mathbb{N}.F(X(n)) \sqsubseteq F(\cap n{:}\mathbb{N}.X(n))$ for any $X \in \mathtt{Type}^{\mathbb{N}}$.

Using corec, we define co-W types as follows (see Nuprl definition coW):

$$\mathtt{coW}(A, B) = \mathtt{corec}(\lambda W.\Sigma a{:}A.(B(a) \to W))$$

where $A \in \mathtt{Type}$ and $B \in \mathtt{Type}^A$. An element of a co-W type $\mathtt{coW}(A, B)$ is therefore a pair $\langle a, f\rangle$ of a a in A, and a function f in $B(a) \to \mathtt{coW}(A, B)$.

For example, we define co-numbers (i.e., steams of numbers) as follows:

$$\texttt{co}\mathbb{N} = \texttt{coW}(\mathbb{B}, \lambda a.\texttt{if } a \texttt{ then False else True})$$

where $\mathbb{B}$ is the Boolean type defined as True+True. Zero can then be represented by the pair $\langle \texttt{tt}, \lambda x.\bot\rangle$ where $\lambda x.\bot$ is a function of type $\texttt{False} \longrightarrow \texttt{co}\mathbb{N}$; and the successor of n can be represented by the pair $\langle \texttt{ff}, \lambda x.n\rangle$ where $\lambda x.n$ is a function of type $\texttt{True} \longrightarrow \texttt{co}\mathbb{N}$.

5.2 W Types

A W type will be defined as the finite elements of a co-W type. Because an element of a co-W type is a (possibly infinite) tree, the finite ones are those that have finite branches. For that we define the concept of path in an element of a co-W type as follows (see Nuprl definition coPath):

$$\texttt{Path}(A, B) = \mathbb{N} \longrightarrow (\Sigma a{:}A.B(a)) + \texttt{True}$$

where $A \in \texttt{Type}$ and $B \in \texttt{Type}^A$. Paths can be infinite or finite. We use $\texttt{inr}(\star)$ to indicate the end of a path. Given an element of a co-W type $\langle a, f\rangle$, a path indicates what b we want to apply the f to. We then say that a path p is correct up to depth n w.r.t. an element w of a co-W type if $\texttt{correctPath}(A, n, p, w)$ is true, where the recursive correctPath function is defined as follows (see Nuprl definition correctCoPath):

Definition 13 (correctPath)

```
correctPath($A, n, p, w$) =
case $p(0)$ of
  inl($x$) $\Rightarrow$ let $a, b = x$ in
             let $a', f = w$ in
             $a =_A a' \wedge$ if $n =_{\mathbb{Z}} 0$ then True else correctPath($A, n-1, \upuparrows(p), f(b)$)
| inr($x$) $\Rightarrow$ True
```

where the operator $\upuparrows$ shifts a path by 1 as follows: $\upuparrows(p) = \lambda n.p(n+1)$.

We are now ready to define W types as follows (see Nuprl definition finiteCoW):

Definition 14 (*W types*)

$$\texttt{W}(A, B) = \{w : \texttt{coW}(A, B) \mid \texttt{finiteCoW}(A, w)\}$$

where

$$\texttt{finiteCoW}(A, w) = \Pi p{:}\texttt{Path}(A, B).(\Pi n{:}\mathbb{N}.\texttt{correctPath}(A, n, p, w)) \longrightarrow \downarrow\Sigma n{:}\mathbb{N}.\texttt{isr}(p(n))$$

and

$$\texttt{isr}(t) = \texttt{if } t \texttt{ then ff else tt}$$

The finiteCoW operator states that each path p that is correct w.r.t. w must end at some depth n.

5.3 Induction Principle

Next we prove the following induction principle for W types (see Nuprl lemma wrec_wf):

Theorem 5 (*Induction principle for W types*)

$\Pi w{:}\mathtt{W}(A,B).\mathtt{wrec}(c,w) \in P(w)$ is true in CTT, assuming `[BarInduction]`, where

$$\begin{aligned} A &\in \mathtt{Type} \\ B &\in A \to \mathtt{Type} \\ P &\in \mathtt{W}(A,B) \to \mathtt{Type} \\ c &\in (\Pi a{:}A.\Pi f{:}(B(a) \to \mathtt{W}(A,B)).(\Pi b{:}B(a).P(f(b))) \to P(\langle a,f\rangle)) \end{aligned}$$

and where `wrec` is the following recursive function (see Nuprl lemma wrec):

$$\mathtt{wrec}(c,w) = \mathtt{let}\ a,f = w\ \mathtt{in}\ c\ a\ f\ (\lambda b.\mathtt{wrec}(c,f(b)))$$

In order to prove the above lemma, we use the following variant of the BI rule presented in Sec. 3.1:

$$\frac{\begin{array}{ll} (\mathtt{wfb}) & H, n:\mathbb{N}, s:\mathcal{B}_n \vdash B(n,s) \in \mathtt{Type} \\ (\mathtt{wfr}) & H, n:\mathbb{N}, s:\mathcal{B}_n \vdash R(n,s) \in \mathtt{Type} \\ (\mathtt{init}) & H \vdash R(0,\perp\!\!\!\perp) \\ (\mathtt{bar}) & H, s:\mathcal{B}, z:\cap m{:}\mathbb{N}.{\downarrow}R(m,s) \vdash {\downarrow}\Sigma n{:}\mathbb{N}.B(n,s) \\ (\mathtt{base}) & H, n:\mathbb{N}, s:\mathcal{B}_n, z:{\downarrow}R(n,s), b:B(n,s) \vdash P(n,s) \\ (\mathtt{ind}) & H, n:\mathbb{N}, s:\mathcal{B}_n, z:{\downarrow}R(n,s), i:(\Pi m{:}\mathbb{N}.R(n+1, s \oplus_n m) \to P(n+1, s \oplus_n m)) \vdash P(n,s) \end{array}}{H \vdash {\downarrow}P(0,\perp\!\!\!\perp)}$$

where R is here a spread law. Note that this rule is at least as strong as the one presented in Sec. 3.1 because the spread law could simply be $\lambda n,s.\mathtt{True}$. Using our Coq formalization, we have proved the validity of this rule: https://github.com/vrahli/NuprlInCoq/blob/master/bar_induction/bar_induction5_con.v.

For simplicity we restrict ourselves to spreads of natural numbers, but we conjecture that a similar rule will be true about spreads of terms in $\mathtt{NBase} = \{t:\mathtt{Base} \mid (t:\mathtt{Base})\#\}$ (see Sec. 4.2.1). Therefore, again for simplicity, we only prove here the above induction principle where A and $B(a)$ are essentially subtypes of $\mathbb{N}$, but again the same principle is true for subtypes of `NBase`. For this reason, we treat $(\Sigma a{:}A.B(a))+\mathtt{True}$ as if it was $\mathbb{N}$.

Proof outline. We use the following spread law:

$$\lambda n,p.\mathtt{correctPath}(A,n,\lambda m.\mathtt{if}\ m{<}n\ \mathtt{then}\ p(m)\ \mathtt{else}\ \mathtt{inr}(\star),w)$$

and the following bar predicate:

$$\lambda n,p.\mathtt{if}\ 0{<}n\ \mathtt{then}\ \mathtt{isr}(p(n-1))\ \mathtt{else}\ \mathtt{False}$$

Also, instead of proving: $\mathtt{wrec}(c,w) \in P(w)$ we switch to proving the equivalent proposition ${\downarrow}G(0,\perp\!\!\!\perp)$, where

$$\begin{aligned} G = \lambda n,p.\mathtt{walkPathF}(\,& n, \\ & \lambda m.\mathtt{if}\ m{<}n\ \mathtt{then}\ p(m)\ \mathtt{else}\ \mathtt{inr}(\star), \\ & w, \\ & \lambda w.\mathtt{wrec}(c,w) \in P(w), \\ & \mathtt{True}) \end{aligned}$$

The recursive function walkPathF is defined as follows—assuming that p is correct w.r.t. w—(see Nuprl lemma walkCoPathF):

```
walkPathF(0, p, w, F, d) = F(w)
walkPathF(n + 1, p, w, F, d) = let a, f = w in
                               case p(0) of
                                inl(x) ⇒ let a′, b = x in
                                          walkPathF(n, ⇈(p), f(b), F, d)
                               | inr(x) ⇒ d
```

We then use [LawlikeBarInduction] with the above mentioned spread law and bar predicate. It then remains to verify that the [LawlikeBarInduction]'s hypotheses are true, i.e., essentially that (bar), (base), and (ind) are true about these spread law and bar predicate—the other hypotheses are trivial. This is done straightforwardly. More details can be found at wrec_wf. □

6 RELATION TO OTHER INTUITIONISTIC NOTIONS

In this section we explore the connections to other salient intuitionistic principles, namely the Fan Theorem in Sec. 6.1, which is a widely used consequence of Bar Induction; as well as Kripke's Schema 6.2, which formalizes Brouwer's notion of the creative (or creating) subject.

6.1 The Fan Theorem

As mentioned in Sec. 1, the Fan Theorem says that every decidable (or detachable) bar on a finitary spread is uniform [102, Ch.7,Sec.7; 43, Sec.3.2]. The more general version of FT, sometimes called the "Full Fan Theorem" [26] (FFT), that does not require the bar to be decidable is also intuitionistically valid [102, Ch.7,Prop.7.4] and can be derived from FT and continuity (see below). FT is the classical contrapositive of *Weak König's Lemma* (WKL), which says that every infinite binary tree has an infinite path—see for example [57; 59; 16]. Constructively, FT is equivalent to a "unique" version of WKL, often denoted WKL! [16]. It turns out that FT is equivalent to the the Uniform Continuity principle (UC), when assuming the continuous choice axiom (the Weak Continuity Principle plus some version of the axiom of choice often denoted $\mathsf{AC}_{1,0}$) [27; 15; 86].

As mentioned in Sec. 3.5, the process of finding the modulus of continuity of a function is not extensional in the sense that it can return different results for extensionally equal functions. Therefore, as proved by Kreisel [66, p.154], Troelstra [98, Thm.IIA], and Escardó and Xu [45], Brouwer's Continuity Principle has to be truncated in a Martin-Löf-like type theory such as Nuprl. However, as proved by Escardó and Xu [45], truncation is not always required when moving from the Baire space to the Cantor space (i.e., sequence of numbers to sequence of Booleans): they showed that in a Martin-Löf-like type theory such as Nuprl, the truncated version of the *uniform Continuity Principle* is equivalent to its non-truncated version (in [111], they also developed a continuous model of Gödel's system T and its logic HA$^\omega$, within which the uniform continuity holds). Following their method, we also derived within Nuprl the following non-truncated uniform Continuity Principle for functions on the Cantor space (see Nuprl lemma strong-continuity2-implies-uniform-continuity2-nat):

$$\mathsf{UCP} = \Pi F{:}C \to \mathbb{N}.\Sigma n{:}\mathbb{N}.\Pi f,g{:}C.f =_{C_n} g \to F(f) =_{\mathbb{N}} F(g)$$

where $C = \mathbb{B}^{\mathbb{N}}$, $C_n = \mathbb{B}^{\mathbb{N}_n}$, and where the Σ type that asserts the existence of a uniform modulus of continuity is not squashed. Therefore, by using continuity to prove FFT from FT, it turns out that we can derive the following non-truncated version of FFT where none of the Σs are truncated—see Nuprl lemma general-fan-theorem-troelstra2, whose proof follows the one of [102, Prop.7.4.(i)]:

$$\Pi P{:}(\Pi n{:}\mathbb{N}.C_n \to \mathbb{P}).(\Pi f{:}C.\Sigma n{:}\mathbb{N}.P\ n\ f) \to (\Sigma k{:}\mathbb{N}.\Pi f{:}C.\Sigma n{:}\mathbb{N}_k.P\ n\ f)$$

as well as the following truncated version—see Nuprl lemma general-fan-theorem-troelstra-sq—where both Σs are truncated:

$$\Pi P{:}(\Pi n{:}\mathbb{N}.C_n \to \mathbb{P}).(\Pi f{:}C.\downharpoonleft\Sigma n{:}\mathbb{N}.\ P\ n\ f) \to (\downharpoonleft\Sigma k{:}\mathbb{N}.\ \Pi f{:}C.\Sigma n{:}\mathbb{N}_k.P\ n\ f)$$

which follows from the non-squashed version of FFT and a $\downharpoonleft$-squashed version of $\mathtt{AC}_{1,0}$.

6.2 Kripke's Schema

Kripke's Schema (KS for short—according to Troelstra and Van Dalen [102, p.241], a name coined by Myhill [76]) formalizes Brouwer's notion of the creative subject. It is often stated as follows:

$$\forall A : \mathbb{P}.\ \exists a : \mathcal{B}.\ (\exists x : \mathbb{N}.\ a(x) =_{\mathbb{N}} 1) \iff A$$

As proved for example by Bridges and Richman [27, p.116] or Troelstra and Van Dalen [102, Ch.4,Sec.9.5], KS is inconsistent with Markov's principle (MP). As discussed below, Myhill proved that KS contradicts some continuity axiom. Van Atten and Van Dalen also used KS to prove that there are no discontinuous functions in [12, Sec.3.2]. KS is classically valid and we have proved the validity of the following squashed version of KS in Coq (see https://github.com/vrahli/NuprlInCoq/blob/master/rules/kripkes_schema.v):

$$\frac{H \vdash A \in \mathbb{U}_i}{H \vdash \downarrow\Sigma a{:}\mathcal{B}.\,(\Sigma x{:}\mathbb{N}.(a(x) =_{\mathbb{N}} 1\ \wedge\ \Pi y{:}\mathbb{N}.x \neq_{\mathbb{N}} y \to a(y) =_{\mathbb{N}} 0)) \iff A}$$

Several variants of this schema are discussed in the literature. The one stated above corresponds to the strong form of Kripke's schema, which is sometimes stated as follows (as in [43, p.244; 41, p.238; 102, Ch.4,Sec.9.3; 12, Sec.3.2]):

$$\downharpoonleft\Sigma a{:}\mathcal{B}.\left(\begin{array}{l} \quad\Sigma x{:}\mathbb{N}.a(x) =_{\mathbb{N}} 1 \iff A \\ \wedge\ \Pi n, m{:}\mathbb{N}.n \leq m \to a(n) \leq a(m) \leq 1 \end{array}\right)$$

There is also a weaker form of this axiom which reads as follows (see also [43, p.244; 76, p.295; 74, p.168; 55, p.241; 75, p.152; 41, p.238; 102, Ch.4,Sec.10.6]):

$$\downharpoonleft\Sigma a{:}\mathcal{B}.\left(\begin{array}{l} \quad\Pi x{:}\mathbb{N}.a(x) =_{\mathbb{N}} 1 \to A \\ \wedge\ \neg A \iff \Pi x{:}\mathbb{N}.a(x) =_{\mathbb{N}} 0 \\ \wedge\ \Pi n, m{:}\mathbb{N}.n \leq m \to a(n) \leq a(m) \leq 1 \end{array}\right)$$

As mentioned above, Myhill proved that KS contradicts $\forall\alpha\exists\beta$-continuity, which is sometimes referred to as $\mathrm{CP}_{\exists\beta}$, and which can be stated as follows:

$$\Pi A{:}\mathcal{B} \to \mathcal{B} \to \mathbb{P}.(\Pi a{:}\mathcal{B}.\underline{\Sigma} b{:}\mathcal{B}.\ A(a, b)) \to \underline{\Sigma} c{:}\mathbb{N}^{\mathcal{B}}.\ \mathtt{CONT}(c)\ \wedge\ \Pi a{:}\mathcal{B}.A(a, \mathtt{shift}(c, a))$$

where

$$\mathtt{shift}(c, a) = \lambda n.c(\lambda k.\mathtt{if}\ k{=_{\mathbb{Z}}}0\ \mathtt{then}\ n\ \mathtt{else}\ a(k))$$
$$\mathtt{CONT}(F) = \Pi f{:}\mathcal{B}.\underline{\Sigma} n{:}\mathbb{N}.\ \Pi g{:}\mathcal{B}.f =_{\mathcal{B}_n} g \to F(f) =_{\mathbb{N}} F(g)$$

As we proved in https://github.com/vrahli/NuprlInCoq/blob/master/continuity/unsquashed_continuity.v, the version of $\mathrm{CP}_{\exists\beta}$, where all the occurrences of $\underline{\Sigma}$ are replaced by Σ, is false in CTT because it follows trivially from the fact that the untruncated version of WCP is false. Following Dummett's version [43, p.246] of Myhill's proof, we have proved that the $\downharpoonleft$-truncated version of KS contradicts $\mathrm{CP}_{\exists\beta}$ where $\underline{\Sigma}$ is $\downharpoonleft\Sigma$ (see for example: https://github.com/vrahli/NuprlInCoq/blob/master/continuity/unsquashed_continuity.v). However, note that we have not validated either of (the $\downharpoonleft$-truncated versions of) $\mathrm{CP}_{\exists\beta}$ or of $\mathrm{KS}_{\downharpoonleft}$ (see Fig. 2). On the contrary, following Troelstra and Van Dalen's proof, we have proved that KS is inconsistent with MP [102, Ch.4,Sec.9.5], and, as mentioned in Sec. 4.3, we have proved that MP is true in Nuprl using some truncated form of excluded middle, which we validated using our Coq model.

7 FURTHER RELATED WORK

As mentioned in the introduction, Howard and Kreisel studied Brouwer's Bar Induction and Continuity Principles in [50] and showed the equivalence between the axiom of transfinite induction (TI)—sometimes called the bar rule [93]—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:

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

In Coq, TI is captures simply by the following lemmas: (1) `well_founded_ind` for Prop, and (2) `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. 3 and some of their variants have been extensively studied [94; 14; 19; 80; 17; 84; 46; 23; 22]. 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 [17]) lead to new BI principles.

Troelstra lists some uses of BI in [100, p.114], e.g. to prove strong normalization of systems such as $\mathrm{N\text{-}HA}^{\omega}$. Veldman and Bezem proved an intuitionistically valid reformulation of Ramsey's theorem using BIM [108; 106]. We have proved this result in Nuprl: see lemma intuitionistic-Ramsey. In [110], the authors proved similar results using directly Coq's inductive types rather than BI.

Choice sequences have also been widely studied over the years [64; 60; 65; 67; 99; 43; 102; 109]. One interesting result regarding choice sequences is the so-called "elimination of choice sequences" theorem [65, Sec.2; 67, Ch.7; 99, Ch.3; 43, pp.221–222; 42] that eliminates quantifications over choice sequences. This theorem relies on a mapping from the formulae of the CS formal system [67] to formulae of the IDB_1 formal system [67] 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.

As mentioned above MP was shown to be consistent with Nuprl (using a squashed version of the law of excluded middle), and it was also shown directly in Nuprl that MP is inconsistent with KS (following [27, p.116; 102, Ch.4,Sec.9.5]). In contrast, in [37] the authors established the independence of MP with Martin-Löf's type theory. Their method relies on a forcing extension of type theory, which contains a "generic" infinite sequence of Booleans. Kripke also proved a similar result for Kreisel's FC system of absolutely free choice sequences [68, p.104]. In [38], the authors showed that MP and the axiom of countable choice are not provable in dependent type theory with one univalent universe and propositional truncation.

Finally, it is worth noting that our method of building a model of Nuprl extended with BI principles bears some resemblance with forcing [32; 33] where our forcing conditions are our choice sequences.

8 CONCLUSION

Bar Induction is a salient intuitionistic principle which allows for powerful constructive inductive reasoning, equivalent to that of the standard transfinite induction. In this work we formally established its compatibility with Constructive Type Theory, the type theory implemented by the Nuprl proof assistant. We proved the validity of a ↓-squashed BI inference rule for sequences of name-free closed terms, from which we derived a non-squashed version of BID for sequences of name-free closed terms, as well as a ⇃-version of BIM for sequences of numbers. We have also shown that the general BIM does not hold for non-⇃-squashed propositions.

Several questions remain open such as: (1) Can the ⇃-squashed Continuity Principle for numbers be generalized to sequences of terms? (2) Can the ↓-squashed Bar Induction principle be generalized 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?

In [87] it was shown that versions of the Continuity Principle for numbers are compatible with CTT. This work establishes CTT's compatibility with variants of another key intuitionistic principle, BI, and therefore takes another step towards extending CTT into a new kind of type theory, one enriched with intuitionistic concepts and principles. There remains a great deal to investigate in this direction, with the goal of creating a highly expressive constructive type theory that will advance the state of the art in powerful type systems and improve the performance and expressive power of programming languages.

ACKNOWLEDGEMENTS

We thank David Guaspari and Evan Moran, as well as our anonymous reviewers, for their helpful criticism. Rahli was partially supported by the SnT and the National Research Fund Luxembourg (FNR), through PEARL grant FNR/P14/8149128. Cohen was supported by: Fulbright Post-doctoral Scholar program; Weizmann Institute of Science – National Postdoctoral Award program for Advancing Women in Science; Eric and Wendy Schmidt Postdoctoral Award program for Women in Mathematical and Computing Sciences.

REFERENCES

[1] Michael Gordon Abbott. "Categories of containers". PhD thesis. University of Leicester, England, UK, 2003.
[2] Michael Gordon Abbott, Thorsten, and Neil Ghani. "Containers: Constructing strictly positive types". In: *Theor. Comput. Sci.* 342.1 (2005), pp. 3–27.
[3] *Agda Wiki*. http://wiki.portal.chalmers.se/agda/pmwiki.php.
[4] Stuart Allen. *An Abstract Semantics for Atoms in Nuprl*. Tech. rep. Cornell University, 2006.
[5] Stuart F. Allen. "A Non-Type-Theoretic Definition of Martin-Löf's Types". In: *LICS*. IEEE Computer Society, 1987, pp. 215–221.
[6] Stuart F. Allen. "A Non-Type-Theoretic Semantics for Type-Theoretic Language". PhD thesis. Cornell University, 1987.
[7] 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.
[8] Thorsten Altenkirch, Neil Ghani, Peter Hancock, Conor McBride, and Peter Morris. "Indexed containers". In: *J. Funct. Program.* 25 (2015).
[9] Abhishek Anand, Mark Bickford, Robert L. Constable, and Vincent Rahli. "A Type Theory with Partial Equivalence Relations as Types". Presented at TYPES 2014. 2014.
[10] Abhishek Anand and Vincent Rahli. "Towards a Formally Verified Proof Assistant". In: *ITP 2014*. Vol. 8558. LNCS. Springer, 2014, pp. 27–44.
[11] Mark van Atten. *On Brouwer*. Wadsworth Philosophers. Cengage Learning, 2004.
[12] Mark van Atten and Dirk van Dalen. "Arguments for the continuity principle". In: *Bulletin of Symbolic Logic* 8.3 (2002), pp. 329–347.
[13] Michael J. Beeson. *Foundations of Constructive Mathematics*. Springer, 1985.

[14] 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.

[15] Josef Berger. "The Fan Theorem and Uniform Continuity". In: *CiE 2005*. Vol. 3526. LNCS. Springer, 2005, pp. 18–22.

[16] Josef Berger and Hajime Ishihara. "Brouwer's fan theorem and unique existence in constructive analysis". In: *Math. Log. Q.* 51.4 (2005), pp. 360–364.

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

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

[19] 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.

[20] 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.

[21] 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.

[22] Valentin Blot. "An interpretation of system F through bar recursion". In: *LICS 2017*. IEEE Computer Society, 2017, pp. 1–12.

[23] Valentin Blot. "Hybrid realizability for intuitionistic and classical choice". In: *LICS '16*. ACM, 2016, pp. 575–584.

[24] 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.

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

[26] Douglas Bridges. "A reverse look at Brouwer's fan theorem". In: *One Hundred Years of Intuitionism*. Birkhäuser, 2008, pp. 316–325.

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

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

[29] 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.

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

[31] Venanzio Capretta. "A polymorphic representation of induction-recursion". www.cs.ru.nl/~venanzio/publications/induction_recursion.ps. 2004.

[32] 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.

[33] 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.

[34] 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.

[35] Robert L. Constable, Stuart F. Allen, Mark Bromley, Rance Cleaveland, J. F. Cremer, Robert W. Harper, Douglas J. Howe, Todd B. Knoblock, Nax P. Mendler, Prakash Panangaden, James T. Sasaki, and Scott F. Smith. *Implementing mathematics with the Nuprl proof development system*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1986.

[36] 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.

[37] Thierry Coquand and Bassel Mannaa. "The Independence of Markov's Principle in Type Theory". In: *FSCD 2016*. Vol. 52. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, 17:1–17:18.

[38] Thierry Coquand, Bassel Mannaa, and Fabian Ruch. "Stack semantics of type theory". In: *LICS 2017*. IEEE Computer Society, 2017, pp. 1–11.

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

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

[41] Dirk van Dalen. "The Use of Kripke's Schema as a Reduction Principle". In: *J. Symb. Log.* 42.2 (1977), pp. 238–240.

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

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

[44] Peter Dybjer and Anton Setzer. “Induction-recursion and initial algebras”. In: *Ann. Pure Appl. Logic* 124.1-3 (2003), pp. 1–47.

[45] Martín H. 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.

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

[47] Ronghui Gu, Zhong Shao, Hao Chen, Xiongnan (Newman) Wu, Jieung Kim, Vilhelm Sjöberg, and David Costanzo. “CertiKOS: An Extensible Architecture for Building Certified Concurrent OS Kernels”. In: *OSDI 2016*. USENIX Association, 2016, pp. 653–669.

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

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

[50] 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.

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

[52] 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.

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

[54] 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.

[55] Richard G. Hull. “Counterexamples in Intuitionistic Analysis Using Kripke’s Schema”. In: *Mathematical Logic Quarterly* 15.1618 (1969), pp. 241–246.

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

[57] Hajime Ishihara. “An omniscience principle, the König Lemma and the Hahn-Banach theorem”. In: *Math. Log. Q.* 36.3 (1990), pp. 237–240.

[58] Hajime Ishihara. “Reverse Mathematics in Bishop’s Constructive Mathematics”. In: *Philosophia Scientiæ* CS6 (2006), pp. 43–59.

[59] Hajime Ishihara. “Weak König’s Lemma Implies Brouwer’s Fan Theorem: A Direct Proof”. In: *Notre Dame Journal of Formal Logic* 47.2 (2006), pp. 249–252.

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

[61] Gerwin Klein et al. “seL4: Formal Verification of an OS Kernel”. In: *SOSP 2009*. ACM, 2009, pp. 207–220.

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

[63] Alexei Kopylov and Aleksey Nogin. “Markov’s Principle for Propositional Type Theory”. In: *CSL 2001*. Vol. 2142. LNCS. Springer, 2001, pp. 570–584.

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

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

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

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

[68] Saul A. Kripke. “Semantical Analysis of Intuitionistic Logic I”. In: *Formal Systems and Recursive Functions*. Vol. 40. Studies in Logic and the Foundations of Mathematics. Elsevier, 1965, pp. 92–130.

[69] Xavier Leroy. “Formal certification of a compiler back-end or: programming a compiler with a proof assistant”. In: *POPL’06*. ACM, 2006, pp. 42–54.

[70] 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.

[71] Per Martin-Löf. *Intuitionistic Type Theory*. Studies in Proof Theory, Lecture Notes 1. Napoli: Bibliopolis, 1984.

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

[73] Leonardo Mendonça de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. “The Lean Theorem Prover (System Description)”. In: *CADE-25*. Vol. 9195. LNCS. Springer, 2015, pp. 378–388.

[74] J. Myhill. “Formal Systems of Intuitionistic Analysis I”. In: *Studies in Logic and the Foundations of Mathematics* 52 (1968), pp. 161–178.

[75] John Myhill. “Formal Systems of Intuitionistic Analysis II: The Theory of Species”. In: *Studies in Logic and the Foundations of Mathematics* 60 (1970), pp. 151–162.

[76] John Myhill. “Notes towards an axiomatization of intuitionistic analysis”. In: *Logique et Analyse* 9 (1967), pp. 280–297.

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

[78] Bengt Nordström, Kent Petersson, and Jan M. Smith. *Programming in Martin-Löf’s Type Theory: An Introduction.* New York, NY, USA: Clarendon Press, 1990.

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

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

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

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

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

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

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

[86] 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.

[87] Vincent Rahli and Mark Bickford. “A nominal exploration of intuitionism”. In: *CPP 2016.* Extended version: http://www.nuprl.org/html/Nuprl2Coq/continuity-long.pdf. ACM, 2016, pp. 130–141.

[88] Vincent Rahli and Mark Bickford. “Validating Brouwer’s continuity principle for numbers using named exceptions”. In: *Mathematical Structures in Computer Science* (2017), pp. 1–49.

[89] 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.

[90] Vincent Rahli, Mark Bickford, and Robert L. Constable. “Bar induction: The good, the bad, and the ugly”. In: *LICS 2017.* Extended version avaible at https://vrahli.github.io/articles/bar-induction-lics-long.pdf. IEEE Computer Society, 2017, pp. 1–12.

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

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

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

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

[95] Stephen G. Simpson. *Subsystems of Second Order Arithmetic.* Second Edition. 2006.

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

[97] 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.

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

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

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

[101] A.S. Troelstra. “Non-extensional equality”. In: *Fundamenta Mathematicae* 82.4 (1975), pp. 307–322.

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

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

[104] 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.

[105] Wim Veldman. “Brouwer’s real thesis on bars”. In: *Philosophia Scientiæ* CS6 (2006), pp. 21–42.

[106] Wim Veldman. “Some Applications of Brouwer’s thesis on Bars”. In: *One Hundred Years of Intuitionism.* Birkhäuser, 2008, pp. 326–340.

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

[108] 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.
[109] Richard Vesley. “Realizing Brouwer’s Sequences”. In: *Ann. Pure Appl. Logic* 81.1-3 (1996), pp. 25–74.
[110] 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.
[111] Chuangjie Xu and Martín Hötzel Escardó. “A Constructive Model of Uniform Continuity”. In: *TLCA 2013.* Vol. 7941. LNCS. Springer, 2013, pp. 236–249.