A Constructive Proof of Dependent Choice, Compatible with Classical Logic

Hugo Herbelin
INRIA - PPS - Univ. Paris Diderot
Paris, France
e-mail: Hugo.Herbelin@inria.fr

Abstract—Martin-Löf’s type theory has strong existential elimination (dependent sum type) that allows to prove the full axiom of choice. However the theory is intuitionistic. We give a condition on strong existential elimination that makes it computationally compatible with classical logic. With this restriction, we lose the full axiom of choice but, thanks to a lazily-evaluated coinductive representation of quantification, we are still able to constructively prove the axiom of countable choice, the axiom of dependent choice, and a form of bar induction in ways that make each of them computationally compatible with classical logic.

Keywords—Dependent choice; classical logic; constructive logic; strong existential

I. Introduction

a) Scaling Martin-Löf’s proof of the axiom of choice to classical logic: In Martin-Löf’s intuitionistic type theory [26], the functional form of the axiom of choice has a simple proof:

\[
\begin{align*}
AC_A \coloneqq & \lambda H.(\lambda x. \text{wit}(H\ x), \lambda x. \text{prf}(H\ x)) \\
& : \forall x^A \exists y^B P(x, y) \rightarrow \exists f^A \forall x^A P(x, f(x))
\end{align*}
\]

where \(\text{wit}\) and \(\text{prf}\) are the first and second projections of a strong existential quantifier\(^1\).

The proof is constructive: it is a program which we can compute with in the sense that any closed proof of some \(\Sigma_1^0\)-statement \(\exists \ g(z) = 0\) that uses the axiom of choice will eventually provide with a witness \(t\) such that \(g(t) = 0\).

On the other side, classical logic is “constructive” too [17], [31] and by interpreting Peirce’s law by means of the \text{callcc} and \text{throw} control operators\(^2\), we can also compute witnesses from closed proofs of \(\Sigma_1^0\)-statements.

Combining the two is however delicate. Reminding that \text{callcc}_\alpha p\ has type \(A\) and binds the continuation variable \(\alpha\) of input type \(A\) when \(p\) has type \(A\) while \text{throw}_\alpha p\ has arbitrary type \(B\) for \(p\) of type \(A\) and \(\alpha\) of input type \(A\), we cannot accept the following instance of the standard reduction rule for \text{callcc} in natural deduction:

\[
\begin{align*}
\text{prf}(\text{callcc}_\alpha(t_1, \phi(\text{throw}_\alpha(t_2, p)))) \\
= & \text{callcc}_{\alpha}(\text{prf}(t_1, \phi(\text{throw}_\alpha(\text{prf}(t_2, p))))
\end{align*}
\]

since if the continuation \(\alpha\) had input type \(\exists P(n)\) in the left-hand side then it would have to have both input types \(P(t_1)\) and \(P(t_2)\) in the right-hand side, leading to an unexpected degeneracy of the domain of discourse\(^3\) [19]. This first problem is solved by using higher-level reduction rules such as

\[
\begin{align*}
E[\text{prf}(\text{callcc}_\alpha(t_1, \phi(\text{throw}_\alpha(t_2, p)))] \\
= & \text{callcc}_{\alpha}(E[\text{prf}(t_1, \phi(\text{throw}_\alpha(E[\text{prf}(t_2, p)])]] \\
E[\text{wit}(\text{callcc}_\alpha(t_1, \phi(\text{throw}_\alpha(t_2, p)))] \\
= & \text{callcc}_{\alpha}(E[\text{wit}(t_1, \phi(\text{throw}_\alpha(E[\text{wit}(t_2, p)]))])
\end{align*}
\]

where the reduction is allowed only when \(E\) is an evaluation context whose return type does not depend on its hole. However, this does not help much because if \(E\) contained other occurrences of the expression \(\text{prf}(\text{callcc}_\alpha(t_1, \phi(\text{throw}_\alpha(t_2, p)))]\) derived from the same initial proof (and this is precisely what would happen in Martin-Löf’s proof of \(AC_A\) if the two copies of \(H\ x\) were classical proofs of the form \(\text{callcc}_\alpha(t_1, \phi(\text{throw}_\alpha(t_2, p)))]\), the synchronisation between the two proofs would be lost.

b) Realising the axioms of countable choice and dependent choice in the presence of classical logic: The axiom of countable choice

\[
AC_{\exists^1} : \forall x^A \exists y^A P(x, y) \rightarrow \exists f^A \forall x^A P(x, f(x))
\]

and the slightly stronger axiom of dependent choice

\[
DC : \forall x^A \exists y^A P(x, y) \rightarrow \\
\forall x_0 \exists f^A \forall x (f(0) = x_0 \land \forall n P(f(n), f(S(n))))
\]

are two weak instances of the full axiom of choice and realisability contributed to understand their computational content in the presence of classical logic. Three approaches were followed.

A breakthrough was made in 1961 in the context of Gödel’s functional interpretation (Dialectica) with the definition by Spector [35] of a notion of bar recursion so as to realise the principle of double negation shift from which the functional interpretation of the axiom of dependent choice follows.

Much later, in 1997, a direct realiser, in a sense close to the one of Kleene [22], was proposed in the context of the arithmetic in finite types by Berardi, Bezem and Coquand [6] for the negative translation of the axiom of dependent choice.

In both cases, the key ingredient is a recursive loop parameterised by a finite portion of the function being built, each

\(^1\)Also known as \(\Sigma\)-type, dependent sum, or strong sum.

\(^2\)We use the SML names of these operators that exist also with other names in various other programming languages.

\(^3\)Failure of subject reduction when combining strong existential quantification and computational classical logic was also observed by P. Blain Levy (private communication).
recursive call carrying one more piece of information than the preceding one, the whole process being terminating because, for the simply-typed \(\lambda\)-calculus based language of realisers they consider, closed programs over functions only uses a finite amount of information of their argument. Later on, Berger and Oliva \[8\] reformulated Berardi, Bezem and Coquand’s realiser in terms of some notion of modified bar recursion. Then, in 2004, Berger \[7\] reduced the termination of these realisers to in terms of some notion of modified bar recursion. Then, in Oliva \[8\] reformulated Berardi, Bezem and Coquand’s realiser amount of information of their argument. Later on, Berger and recursive call carrying one more piece of information than the bar recursion. They do not seem either to generalise to choice the ones of Berardi, Bezem and Coquand, and a fortiori from particular, Krivine’s realisers are rather di which, informally, maps those set of realisers.

Alternatively, it can be seen as a form of realisability allowing the variant of countable choice realised by Krivine is

\[ \forall x \exists y f(x, y) \rightarrow \exists y \forall x f(x, y) \forall x \exists y f(x, y) \]

where \(\mathbb{N} \rightarrow \star\) and \(\mathbb{N} \rightarrow \star\) respectively denote the type of predicates and relations over \(\mathbb{N}\). Krivine’s realiser for \(AC^*\) does not use a fixpoint but instead a “quote” function which, informally, maps those \(Y\) such that \(P(x, Y)\) into natural numbers that can then be compared so that the \(Y\) with least “quote” is used to define \(F\) on \(x\). The realiser also crucially uses control operators: if some \(Y\) is found that has lesser quoted value than the \(Y\) currently used to build \(F\), the evaluation context at the time \(F(x)\) was requested is restored and a new computation of \(F\) on \(x\) with new \(Y\) is started. In particular, Krivine’s realisers are rather different in style from the ones of Berardi, Bezem and Coquand, and a fortiori from bar recursion. They do not seem either to generalise to choice functions with arbitrary, non relational, codomain \(A\).

\[ \text{c) Call-by-name, call-by-value and call-by-need: } \]

Church’s \(\lambda\)-calculus \[9, 5\] comes naturally as a “call-by-name” calculus and it is its use in computer programming languages that motivated the theoretical study of its more intricate\(^3\) call-by-value counterpart, thanks successively to Plotkin \[32\], Moggi \[27\], Sabry and Felleisen \[33\], Sabry and Wadler \[34\], etc. Similarly, call-by-need \(\lambda\)-calculus, which is at the heart of programming languages like Haskell \[15\], progressively tends to be studied at the same foundational level its call-by-name and call-by-value variants are, see \[2, 25\], or, in the presence of control, \[29, 4, 3\]. Call-by-value and call-by-need are appropriate for sharing values and will turn to be useful for dealing with theories that might reflect proofs inside terms.

\[ \text{d) Internalising the construction of an approximation of the choice function at the level of proofs: } \]

In order to preserve the synchronisation between different instances of proofs, that are classical and hence liable to duplicate their evaluation context, call-by-value evaluation is indeed appropriate. However, in the proof of the axiom of choice above, the two occurrences of \(H x\) are in the scope of different binders of \(x\) what forbids the possibility to share them.

Let us assume that the domain of quantification \(A\) is the domain of natural numbers. Let us also assume for a while that we could define the choice function and its property by infinite terms. Then we could prove the axiom of countable choice with the following infinite proof:

\[ AC_N \triangleq \lambda H. (\lambda n. \text{if } n = 0 \text{ then } \text{wit}(H 0) \text{ else } \text{if } n = 1 \text{ then } \text{wit}(H 1) \text{ else } \ldots, \lambda n. \text{if } n = 0 \text{ then } \text{prf}(H 0) \text{ else } \text{if } n = 1 \text{ then } \text{prf}(H 1) \text{ else } \ldots) \]

Now, we have an infinite number of calls to \(H\) but each of these calls is parameter-free and hence shareable. Using the \text{let} operator of call-by-value, we can then make sharing explicit:

\[ AC_N \triangleq \lambda H. \text{let } H_0 = H 0 \text{ in } \text{let } H_1 = H 1 \text{ in } \ldots (\lambda n. \text{if } n = 0 \text{ then } \text{wit}H_0 \text{ else } \text{if } n = 1 \text{ then } \text{wit}H_1 \text{ else } \ldots, \lambda n. \text{if } n = 0 \text{ then } \text{prf}H_0 \text{ else } \text{if } n = 1 \text{ then } \text{prf}H_1 \text{ else } \ldots) \]

Now we have to capture the infinity by finitary means and this is possible by turning the infinite sequence of \text{let} into a single stream definition \((H 0, H 1, \ldots)\). This leads to the following proof of the countable axiom of choice:

\[ AC_N \triangleq \lambda H. \text{let } s = \text{cofix}_{\star}(H n, f n) \text{ in } (\lambda n. \text{wit}(\text{nth} n s), \lambda n. \text{prf}(\text{nth} n s)) \]

where \(\text{cofix}_{\star}(H n, f n)\) is a corecursive definition of the stream iterating on \(f\) with parameter \(n\) and started at 0

\(^3\) Though, when looking at \(\lambda\)-calculus from the point of view of sequent calculus instead of from the point of view of natural deduction \[12, 18\], call-by-value \(\lambda\)-calculus gets no more complicated than call-by-name, both having the same - intermediate - level of intrinsic technical complexity.

\(^4\) See e.g. Berger and Oliva \[8\] for a notion of realisability obtained by combination of Kleene’s realisability and Friedman-Dragalin \(\lambda\)-translation and in which \(\perp\) is realisable. That Krivine’s classical realisability contains \(\lambda\)-translation comes from the fact that \(\perp\) is not empty but realised by a fixed set of realisers.
while $n\text{th}_n s$ is a recursive definition of the access to the $n^{th}$ component of the stream $s$.

At the level of formulae, the stream is an inhabitant of a coinductively defined infinite conjunction $\forall n (\exists y P(y, n) \land X(n + 1))$. At the level of computation, since a stream is infinite, we cannot afford evaluating each of its component in advance, so we have to use a lazy call-by-value mechanism.

e) Outline: To make a sound formal system of this analysis, it remains to characterise the restriction required on strong existential elimination so that it becomes compatible with classical logic. In Section II, we study this restriction in the classical arithmetic in finite types, showing in passing how to define coinductive formulae in this context. By lazy evaluating the coinductive proofs, termination can reasonably be claimed, from which conservativity of classical logic over intuitionistic logic for $\Sigma^0_1$ formulae in the presence of strong existential elimination entails. In Section III, we show how to exploit the coinductive connectives to give a proof of the axioms of countable choice, axiom of dependent choice, and bar induction. Open issues will be discussed in Section IV together with a comparison with some other works.

II. $dPA^w$: CLASSICAL ARITHMETIC IN FINITE TYPES WITH STRONG EXISTENTIAL

We now focus on the arithmetic in finite types and extend $dPL$ with quantification over functions of higher-order types and recursion. In this logic, that we call $dPA^w$, the axioms of countable choice and dependent choice can be proved as will be shown in the next section.

Even though coinductive formulae can be defined in $dPA^w$, thanks to the quantification over functions, we will consider a primitive notion of coinductive formulae, considered positive, and that will be precisely convenient for proving the axioms of countable choice and dependent choice.

A. Proofs and Terms

Strong existential elimination forces formulae to be dependent. In $dPA^w$, terms $t, u, \ldots$ can depend on proofs $p, q, \ldots$, and vice versa so that both are defined mutually:

$$\begin{align*}
t, u & \ ::= \ x \mid 0 \mid S(t) \mid \text{rec of } \tau \mid \text{wit } p \\
p, q & \ ::= \ a \mid i_1(p) \mid (p_1, p_2) \mid (t, p) \mid \lambda a.p \mid \lambda x.p \\
& \quad | \text{case } p \text{ of } [a_1, p_1 | a_2, p_2] \\
& \quad | \text{split } p \text{ as } (x, a) \text{ in } q \\
& \quad | \text{dest } p \text{ as } (x, a) \text{ in } q | \text{prf } p \\
& \quad | p \mid \lambda x. p | \text{exfalso } p \\
& \quad | \text{refl } | \text{subst } p q \\
& \quad | \text{ind } t \text{ of } [p] \text{ in } q \text{ in } \lambda x.p \\
& \quad | \text{cofix}^w_{x,p} \\
& \quad | \text{catch}_{x,p} | \text{throw}_{x,p} \\
& \quad | \text{let } a = p \text{ in } q
\end{align*}$$

where $f$ ranges over function symbols, $r$ denotes in $f(r)$ a sequence of terms of length the arity of $f$, the names $x, y, \ldots$ range over a set of term variables, $a, b, \ldots$ over a set of proof variables, $\alpha, \beta, \ldots$ over a set of continuation variables. The constructions $\lambda x.p$, case $p$ of $[a_1, p_1 | a_2, p_2]$, split $p$ as $(a_1, a_2)$ in $q$, dest $p$ as $(x, a)$ in $q$ and ind $t$ of $[p] \text{ in } q$ bind $a, a_1$ and $a_2$. The constructions $\lambda x.p$, dest $p$ as $(x, a)$ in $q$, dest $p$ as $(x, a)$ in $t$, $\lambda x.t$, ind $t$ of $[p] \text{ in } q$ and rec $t$ of $[\tau] \text{ in } \lambda x.t$ bind $x$ and $y$. The construction catch$_{x,p}$ binds $\alpha$. The binders are considered up to the actual name used to represent the binder ($\alpha$-conversion) and the set of free variables $FV(p)$ of a proof $p$ is, as usual, the set of variables of $p$ that are not bound inside $p$ itself.

Most constructions speak by themselves with the peculiarity that terms can be built by case analysis (case) or destruction of proofs (split and dest). The operator rec $t$ of $[\tau] \text{ in } \lambda x.t$ is for recursion in finite types while ind $t$ of $[p] \text{ in } q$ is for induction. The construction cofix$^w_{x,p}$ is for building coinductive formulae.

Let us say also that the operators catch and throw implement classical reasoning. They are similar to the operators of same name in Nakano [28] or Crolard [11]. In terms of Parigot’s $\lambda$-calculus [30], catch$_{x,p}$ is basically equivalent to $\mu x.((x, a)p$ and throw$_{x,p}$ to $x \cdot \delta_x(x, p)$ for $\delta$ not occurring in $p$.

The abbreviations $\pi_1(p) \equiv \text{split } p$ as $(a_1, a_2)$ in $a_1$ and $\pi_2(p) \equiv \text{split } p$ as $(a_1, a_2)$ in $a_2$ might occasionally be useful.

To emphasise that a term variable ranges over functions, we might use symbols derived from the letters $f$ or $g$ instead of $x$ or $y$. We might also use $n$ or $m$ for a variable ranging over natural numbers.

B. Operational Semantics

We equip $dPA^w$ with a call-by-value evaluation semantics and for that, a subclass of proofs will play a particular role in extracting the intuitionistic content of positive formulae. These are the values defined by:

$$V \ ::= \ a | i_1(V) | (V, V) | (t, V) | \lambda a.p | \lambda x.p | () | \text{refl}$$

To define the operational semantics of $dPA^w$, we also need to define the class of elementary call-by-value evaluation contexts. Because of corecursion, we have potentially infinite values and we do not want to fully reduce proofs using a call-by-value semantics. Therefore, we use an incremental reduction semantics which is lazy on the evaluation of corecursive values. Lazy evaluation requires to introduce specific contexts, written $D$, which accumulate pending delayed computation of cofixpoints. Altogether, evaluation contexts are defined as follows:

$$\begin{align*}
F[ ] & \ ::= \ u([ ] | [l, ]) | (V, [ ]) | (t, [ ]) \\
& | \text{case } [ ] \text{ of } [a_1, p_1 | a_2, p_2] \\
& | \text{split } [ ] \text{ as } (a_1, a_2) \text{ in } q \\
& | \text{dest } [ ] \text{ as } (x, a) \text{ in } p | \text{prf } [ ] \\
& | [ ] | [ ] | [ ] | \text{let } a = [ ] \text{ in } q \\
& | \text{subst } [ ] p
\end{align*}$$

$$\begin{align*}
D[ ] & \ ::= \ [ ] | D[F[ ]] | \text{let } a = \text{cofix}^w_{x,p} \text{ in } D[ ]
\end{align*}$$

For $F[ ]$ an elementary call-by-value evaluation context and $p$ a proof, we write $F[p]$ for the proof obtained by plugging $p$ into the hole of $F[ ]$ and similarly for $D[ ]$. 
let $a = i(p)$ in $q$
let $a = (p_1, p_2)$ in $q$
let $a = (t, p)$ in $q$
let $a = \lambda b.p$ in $q$
let $a = \lambda x.p$ in $q$
let $a = ()$ in $q$
let $a = b$ in $q$
case $i(p)$ of $[a_1, p_1] | a_2, p_2$
split $(p_1, p_2)$ as $(a_1, a_2)$ in $q$
dest $(t, p)$ as $(x, a)$ in $q$
prf $(t, p)$
$(\alpha.a) p$
$(\lambda x.p) t$
subst refl $p$
ind $0$ of $[p | (x, a), q]$
ind $S(t)$ of $[p | (x, a), q]$
case cofix$\alpha \gamma$, $p$ of $[a_1, p_1] | a_2, p_2$
split cofix$\alpha \gamma$, $p$ as $(a_1, a_2)$ in $q$
dest cofix$\alpha \gamma$, $p$ as $(x, a)$ in $q$
let $a = cofix\alpha \gamma$, $p$ in exfalso $q$
let $a = cofix\alpha \gamma$, $p$ in throw$w$, $q$
let $a = cofix\alpha \gamma$, $p$ in catch$w$, $q$
let $a = cofix\alpha \gamma$, $p$ in $D$[case $a$ of $[a_1, p_1] | a_2, p_2$]
let $a = cofix\alpha \gamma$, $p$ in $D$[split $a$ as $(a_1, a_2)$ in $q$]
let $a = cofix\alpha \gamma$, $p$ in $D$[dest $a$ as $(x, a')$ in $q$]
$F[let a = cofix\beta \gamma$, $p$ in $q]$  
$F[exfalso p]$  
$F[throw w, p]$  
$F[catch w, p]$  
exfalso exfalso $p$
exfalso throw$\beta$, $p$
exfalso catch$\beta$, $p$
throw$w$, exfalso $p$
$\lambda t.(let a = intro\alpha \gamma$, $p$ in $D$[case $a$ of $[a_1, p_1] | a_2, p_2$])
$\lambda t.(let a = intro\alpha \gamma$, $p$ in $D$[split $a$ as $(a_1, a_2)$ in $q$])
$\lambda t.(let a = intro\alpha \gamma$, $p$ in $D$[dest $a$ as $(x, a')$ in $q$])$
$\lambda t.(let a = intro\alpha \gamma$, $p$ in $F[q]$)
exfalso $p$
throw$w$, $p$
catch$\beta$, $p$
exfalso $[\beta/\alpha]$  
exfalso $p$
exfalso $\beta/\alpha$
throw$\beta$, $p$
catch$\beta$, $p$
$\lambda t.(let a = intro\alpha \gamma$, $p$ in $D$[case $a$ of $[a_1, p_1] | a_2, p_2$])$
$\lambda t.(let a = intro\alpha \gamma$, $p$ in $D$[split $a$ as $(a_1, a_2)$ in $q$])$
$\lambda t.(let a = intro\alpha \gamma$, $p$ in $D$[dest $a$ as $(x, a')$ in $q$])$
$\lambda t.(let a = intro\alpha \gamma$, $p$ in $F[q]$)
$\lambda t.(let a = intro\alpha \gamma$, $p$ in $D$[case $a$ of $[a_1, p_1] | a_2, p_2$])$
$\lambda t.(let a = intro\alpha \gamma$, $p$ in $D$[split $a$ as $(a_1, a_2)$ in $q$])$
$\lambda t.(let a = intro\alpha \gamma$, $p$ in $D$[dest $a$ as $(x, a')$ in $q$])$
$\lambda t.(let a = intro\alpha \gamma$, $p$ in $F[q]$)

Fig. 1. Reduction rules on terms and proofs of $dPA^\alpha$

The reduction rules are shown in Figure 1 where the substitutions $p[V/a]$, $p[u/x]$, $t[V/a]$, $t[u/x]$ and $p[\beta/\alpha]$ are capture-free with respect to the three kinds of variables $(x$, $a$ and $\alpha)$ and where the substitution $p[F/\alpha]$ means replacing subterms of the form $\text{throw}_w, q$ in $p$ by $\text{throw}_w, F[q]$ (including the recursive replacements in $q$).

We write $\ast$ for the reflexive-transitive closure of $\Rightarrow$. We write $\equiv$ for the reflexive-symmetric-transitive closure of $\Rightarrow$.

$\text{strictly speaking, the definition of } p[\beta/\alpha] \text{ and } p[F/\alpha] \text{ requires also to consider their variants } t[\beta/\alpha] \text{ and } t[F/\alpha] \text{ on terms}$

C. Types, Formulae and Inference Rules

Terms are simply typed, with the natural numbers as base type. Finite types are thus defined by:

$$T, U ::= \mathbb{N} \mid T \rightarrow U$$

In $dPA^\alpha$, we consider implication to be possibly dependent in its antecedent and use the notation $[a : A] \rightarrow B$ to express this dependency, underlining the fact that $a$ can occur in some term in $B$. An advantage of allowing this dependency is the ability to express statements such as $[a : \exists x P(x)] \rightarrow}$
Fig. 2. dPA*: Classical arithmetic in finite types with strong existential
Proved in the case of predicate logic that the logic with dependent implication and strong elimination rule of existential quantification and the conjunction over functional symbols and, as will be shown in Section III, dependent choice, it can simulate quantification over predicates talking about larger domains than \( \mathbb{N} \), one would also typically need the axiom of unique choice\(^{12}\) on arbitrary large domains.

\[
\forall x^T \exists n P(x, n) \rightarrow \exists f \forall x^T P(x, f(x))
\]

and there is no reason to think that this holds.

We are now ready to state the operational and logical properties of \( dPA^\omega \).

**Theorem 1 (Subject reduction):** If \( \Gamma \vdash p : A \) and \( p \triangleright q \) then \( \Gamma \vdash q : A \).

\(^7\)We do not get extra logical strength from this design choice. It can be proved in the case of predicate logic that the logic with dependent implication is conservative over ordinary predicate logic and we conjecture that \( dPA^\omega \) with dependent implication is conservative over its version without dependent implication.

\(^8\)Weakening the no rule from the context, or not relevant, we may occasionally drop the type of the variable in the quantifiers.

Since there are terms in all finite types in \( dPA^\omega \), it is convenient to indicate the types of variables in typing contexts. Hence, contexts are defined by:

\[
\Gamma ::= \emptyset | \Gamma, x : T | \Gamma, a : A | \Gamma, a : A^k
\]

where \( a : A \) stands for an assumption of \( A \) and \( a : A^k \) for an assumption of \( A \) with the objective of obtaining a proof by contradiction). On its side, \( x : T \) stands for the declaration of a variable of type \( T \). It is assumed that assumptions have distinct variable names and we write \( \text{Dom}(\Gamma) \) for the set of names \( a \) and \( a \) thus declared in \( \Gamma \).

Inference rules are given in Figure 2 with the typing rules in the bottom. The main difference with ordinary logic is the strong elimination rule of existential quantification and the appropriate support for formulae depending on proofs.

\(^9\)Not to be confused with the notation \( A^k \) sometimes used for powerset.

\(^{10}\)See e.g. Allali \([1]\) for such a presentation of arithmetic.

\(^{11}\)Unfolding of coinductive formulae makes the reduction system non terminating. One might wonder if it would make \( \equiv \) undecidable: no, because unfolding can just be used lazily.

\(^{12}\)I.e. reification of functional relations into functions.
of the original interacting normal subproofs. Such modified interaction between modified proofs obtained by artificially the infinite reduction sequence can be turned into an infinite portion of the infinite branches can be explored. Therefore, dependent choice, there is an infinite sequence of nested rules of another branch are explored simultaneously. If nested seen as an infinite interaction between the immediate normal be found. Let us consider a minimal proof having an infinite beyond some given depth, only introduction rules of positive coming from the expansion of cofixpoints and such that, founded infinitary trees up to the presence of infinite branches proof in infinitary logic. Normal proofs expand into well-able.

cannot happen. In particular, the only rule involving prf prf (prf)

They preserve the correctness of derivations. The di

Claim 1 (Normalisation): If \( \Gamma \vdash p : A \) then \( p \) is normalisable.

Proof: (sketch) We follow the ideas of [10] and interpret proofs in infinitary logic. Normal proofs expand into well-founded infinitary trees up to the presence of infinite branches coming from the expansion of cofixpoints and such that, beyond some given depth, only introduction rules of positive connectives occur. Otherwise said, along all branches, infinitely many nested introduction rules of positive connectives can occur but only finitely many nested elimination rules can be found. Let us consider a minimal proof having an infinite reduction sequence. Such an infinite reduction sequence can be seen as an infinite interaction between the immediate normal subproofs of the given proof. Laziness of cofixpoint unfolding now ensures that any time an infinite branch is explored in its part made only of introduction rules, nested elimination rules of another branch are explored simultaneously. If nested elimination rules of arbitrary depths are explored, then, by dependent choice, there is an infinite sequence of nested elimination rules. This is not the case, hence, only a finite portion of the infinite branches can be explored. Therefore, the infinite reduction sequence can be turned into an infinite interaction between modified proofs obtained by artificially cutting at some large enough occurrences the infinite branches of the original interacting normal subproofs. Such modified proofs are well-founded. But using the result of [10], interaction between well-founded normal proofs in infinitary logic necessarily terminates, a contradiction.

Theorem 2 (Conservativity, first version): If \( A \) is a closed \( \forall \rightarrow \forall \)-wit-free formula then \( \vdash p : A \) in \( dPA^\omega \) implies that there is some \( V \) such that \( \forall V : A \) in \( HA^\omega \).

Proof: Our choice of rules makes that any closed proof of \( A \) eventually produces, by normalisation, either an expression \( D[V] \) or an expression \( \text{catch}_r[D[V]] \) where \( D \) is made only of nested let \( a = \text{cofix}_{b,q} \) in \( [ ] \) (the case exfalso \( D[V] \) cannot happen because there is no value of type \( \bot \), the cases \( D[\text{cofix}_{b,q}] \) and \( \text{catch}_r[D[\text{cofix}_{b,q}] \) cannot happen because \( A \) is \( \forall \)-free, all other possible configurations are reducible since \( p \) is closed). Now, because \( A \) is \( \forall \rightarrow \forall \)-wit-free, \( V \) does not contain any subexpression of the form \( \lambda a.q \) or \( \lambda x.q \). In particular, in the \( \text{catch}_r[D[V]] \) case, it does not contain any occurrences of \( a \). Similarly, no variable that is bound to some \( \text{cofix}_{b,q} \) in \( D \) can occur in \( V \) since otherwise \( V \) would have \( \nu \) in its type. Hence \( V \) is closed and is a proof of \( A \). Since \( V \) does not contain any catch, nor throw, nor prf, it is in \( HA^\omega \).

In arithmetic, any \( \Sigma^0_1 \)-formula is equivalent to a \( \forall \rightarrow \forall \)-wit-free formula. Hence we have:

Theorem 3 (Conservativity, second version): If \( A \) is \( \Sigma^0_1 \) then \( \vdash p : A \) in \( dPA^\omega \) implies \( \vdash p : A \) in \( HA^\omega \).

This of course implies consistency:

Theorem 4 (Consistency): \( \forall p : \bot \) in \( dPA^\omega \).

III. THE AXIOMS OF COUNTABLE CHOICE AND DEPENDENT CHOICE

Our main result is that \( dPA^\omega \) proves the axiom of countable choice, the axiom of dependent choice, and thus equivalent axioms such as bar induction, open induction and update induction. The main trick is to turn a proof of \( \forall x^2 A(x) \) where possibly the proof of \( A(x) \) is classical into a coinductive conjunction \( A(g(0)) \land A(g(1)) \land A(g(2)) \ldots \) for a suitable law \( g \) of type \( \mathbb{N} \rightarrow A \), so that the coinductive stream can be reduced using a (lazy) call-by-value discipline and the resulting (non-classical) values be shared by calls to the strong existential elimination.

A. The Axiom of Countable Choice

Here, \( A(x) \) is \( \exists y P(x,y) \) and the appropriate stream we want to build is the stream \( A(0) \land A(1) \land A(2) \ldots \), so we consider
the coinductive conjunction \( R_C(n) \equiv v_{f_x}^n(A(x) \land f(S(x)) = 0) \).

The proof is now direct:

\[
AC_n \equiv \lambda a. \text{let } b = \text{cofix}_{n}^0(a n, b(S(n)))\text{ in } (\forall n \forall y. P(n, y) \rightarrow \exists f \forall n P(f(n))
\]

where

\[
\text{nth}_C n : R_C(0) \rightarrow A(n)
\]

\[
\text{nth}_C n \equiv \lambda b. \pi_1(\text{ind} n \text{ of } [b \mid (m,c)].\pi_2(c))
\]

Note that the proof does not use classical logic and holds also in dHA\(^\omega\).

B. The Axiom of Dependent Choice

Here again, \( A(x) \equiv \exists y. P(x,y) \) and the appropriate stream we want to build is the stream \( A(x_0) \land A(g(x_0)) \land A(g^2(x_0)) \ldots \) where \( g \) is the choice function implicit in some proof of \( \forall x \exists y. P(x,y) \). So we consider the coinductive formula \( \text{R}_C(c) \equiv v_{f_x}^n(P(x,y) \land f(y) = 0) \). The proof is now direct:

\[
\text{DC} \equiv \lambda a. \lambda x_0. \text{let } b = \text{sa} x_0 \text{ in } (\forall n \forall y. P(x,y) \rightarrow \exists x_0 \exists y. P(x,y) \rightarrow \forall x_0 \exists f(f(0) = x_0 \land \forall y. P(f(n), f(S(n))))
\]

where

\[
\text{nth}_D n : \exists R_D(x) \rightarrow \exists x R_D(x)
\]

\[
\text{nth}_D n \equiv \lambda b. \text{ind} n \text{ of } [b \mid (m,c)].\pi_1(\text{prf}(\text{prf}(\text{nth}_D n, x_0)))]
\]

\[
\text{sa} x : R_D(x)
\]

\[
\text{sa} x \equiv \text{cofix}_{n}^0(\text{dest} an \text{ as } (y,c) \text{ in } (y,c,by))
\]

Note that this proof too does not use classical logic and holds in dHA\(^\omega\).

C. Bar Induction

To express bar induction, we extend dPA\(^\omega\) with a type constructor for finite sequences:

\[
T ::= \ldots | T^*
\]

\[
t, l ::= \ldots | \emptyset | l \otimes t | \text{rec} l \text{ of } [t|(x,y,z), l]
\]

\[
p ::= \ldots | \text{ind} l \text{ of } [p|(x,y,a),p]
\]

The corresponding reduction, inference and typing rules are canonical and we skip them.

To state bar induction, we also need to define the initial segment of length \( n \) of a function from \( \mathbb{N} \) to \( T \):

\[
f_{f_m} \equiv \text{rec} n \text{ of } [\emptyset | (m, l), l \star f(m)]
\]

We now have all the ingredients to state the standard formulation of bar induction in intuitionistic logic:

\[
BI : \forall f \exists n B(f_n) \rightarrow \forall P \left( \forall l (B(l) \rightarrow P(l)) \land \forall l (\forall x P(l \star x) \rightarrow P(l)) \right) \rightarrow P(\emptyset)
\]

Let us consider a contrapositive variant of BI:

\[
BI_c : v_{g}^0(\neg B(l) \land \exists x g(l \star x) = 0) \rightarrow \exists f \forall n \neg B(f_{f_m})
\]

where we have recognised the negation of the conclusion as a coinductive positive formula. By classical reasoning and the axiom of unique choice, BI and BI\(_C\) are equivalent.

Let us write \( R_{BI}(l) \) for the coinductive formula occurring in the statement of BI\(_C\). The same way as we proved the axiom of dependent choice, we have:

\[
BI_{C} \equiv \lambda a. (\exists l. \text{wit}(\pi_2(\pi_1(\text{prf}(\text{nth}_{BI} n, (l, a)))))) \}
\]

\[
\lambda n. \text{wit}(\text{nth}_{BI} n, (l, a))) \}
\]

\[
\pi_1(\text{prf}(\text{nth}_{BI} n, (l, a)))) \}
\]

\[
e n \}
\]

\[
\text{wit}(\text{nth}_{BI} n, (l, a)) = \}
\]

\[
(\lambda n. \text{wit}(\pi_2(\text{prf}(\text{nth}_{BI} n, (l, a))))))_n
\]

\[
e n \}
\]

\[
\text{ind} n \text{ of } \text{refl} | (m,c) \text{.subst } c \text{.refl}
\]

from which BI\(_C\) directly follows. Then, from BI\(_C\) and Markov’s principle, we get the following weaker form of BI:

\[
\forall l (\forall x (g(l \star x) = 0 \rightarrow g(l) = 0)) \rightarrow g(\emptyset) = 0
\]

Finally, in the special case when \( x \) ranges over \( \mathbb{N} \), the characteristic function \( g \) of any predicate \( P \) over \( \mathbb{N}^* \) can be built classically using the axiom of countable choice. Hence a (classical) proof of BI is obtainable in this case.

IV. DISCUSSION AND RELATION TO OTHER WORKS

f) A constructive intuitionistic logic which proves Markov’s principle, the double negation shift and the axiom of dependent choice: It has been shown that adding delimited classical logic to intuitionistic logic allows to derive weakly classical schemes such as Markov’s principle and the double negation shift while still preserving the disjunction and existence properties that are specific to intuitionistic logic [20], [21]. Adding strong existential elimination to intuitionistic logic with delimited classical logic should provide with a constructive intuitionistic logic that proves Markov’s principle, the double negation shift and the axiom of dependent choice, and that is therefore adequate for intuitionistic analysis.

g) Relation with Berardi, Bezem and Coquand’s realiser of the axiom of countable choice: The computational content of our proof of the countable axiom of choice is slightly different from the one of the realiser given in the paper by Berardi, Bezem and Coquand [6]. First, in our proof, there is no construction of a function with dummy values: when in our proof, the order in which the proofs of BI\(_C\) is slightly different from the one of the case of [6], these proofs are evaluated on demand depending on which n’s
the context that interacts with the realiser of $\exists f \forall n P(n, f(n))$ needs a certification that $P(n, f(n))$ holds. In this sense, our proof seems suboptimal. For instance, if only the content of the proof of $\exists y P(1001, y)$ is needed, it will evaluate all the proofs of $\exists y P(n, y)$ for $n < 1001$ first. Of course, one could be more lazy than we did in our evaluation algorithm and in particular be lazy on the evaluation of each $\exists y P(n, y)$ that is not explicitly required. Still, the stream built will be a stream of length 1001 while in [6], the stream has the same size as the number of $n$'s for which a proof of $\exists y P(n, y)$ is needed.

b) Dependent choice in a logic with quantification over second-order predicates: Our approach is uniform over the type of the codomain of the choice function, so it directly scales to quantification over second-order predicate. Let us call $dPA_2$ and $dHA_2$ the classical and intuitionistic systems obtained by replacing the quantification over functions in finite types with quantification over second order predicates, i.e. the systems obtained from $dPA^w$ and $dHA^w$ by replacing the definition of types with:

$$T, U ::= \mathbb{N} \mid \star \mid \mathbb{N} \rightarrow T$$

where $\star$ denotes the type of propositions. Then, the axiom of countable choice is provable in $dPA_2$ and $dHA_2$, what typically covers the instance

$$\text{AC}^*_\mathbb{N}: \forall n \exists y \forall \bar{u} \forall \bar{v} \forall \bar{w} P(n, y) \rightarrow \exists f \forall n \forall \bar{u} \forall \bar{v} \forall \bar{w} P(n, F n)$$

that we discuss in the next paragraph.

i) Comparison with Krivine’s realiser of the axiom of countable choice: Krivine [23] realises the axiom of countable choice in the context of classical second-order arithmetic using a notion of classical realisability that interprets quantifiers by intersection types and that consequently keeps no trace of the quantifiers in the realiser. A detailed comparison can be found in [20].

j) Functional interpretation and products of selection functions: Products of selection functions have been developed by Escardó and Oliva in the context of functional interpretation to interpret bar recursion [14]. This seems to correspond at the level of realisability to what we are doing at the level of proofs.

V. CONCLUSION

We showed how to slightly restrict strong existential elimination (Martin-Löf’s dependent sum type) so that it becomes compatible with classical reasoning in a computationally sound way. In this restricted framework, we lose the full axiom of choice but keep the axioms of countable choice and dependent choice thanks to a detour via coinductively defined connectives. Because the choice functions we are able to build are paths in coinductive trees, we suspect our framework to exactly capture the strength of the axiom of dependent choice.

The idea here is to reason by induction on the structure of the

$$\text{AC}^*_\mathbb{N}: \exists f \forall n \exists y \forall \bar{u} \forall \bar{v} \forall \bar{w} P(n, y) \rightarrow \forall u \forall v \forall w P(n, Y)$$

which is classically equivalent to $\text{AC}^*_\mathbb{N}$ over the codomain $\mathbb{N} \rightarrow \star$.

13In practise, Krivine realises the axiom

$$\text{CAC}: \exists f \exists n \forall y \exists f^n(P(n, Z n) \rightarrow \forall y f^n P(n, Y))$$

argument of strong existential elimination, but we leave this for future work.

ACKNOWLEDGEMENTS

Concepts coming from the programming languages side were instrumental for this work and I’m grateful to the community that designed them.

On a personal side, I thank Danko Ilik, Paul-André Melliès, Guillaume Munch-Maccagnoni, and Noam Zeilberger for fruitful discussions they shared with me.


