A Constructive Proof of Dependent Choice, Compatible with Classical Logic

(includes erratum to Section III-C – September 2012)

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:

$$\begin{array}{lll} AC_A & \triangleq & \lambda H.(\lambda x.\mathtt{wit}\,(H\,x), \lambda x.\mathtt{prf}\,(H\,x)) \\ & : & \forall x^A \exists y^B\, P(x,y) \to \exists f^{A\to B}\, \forall x^A\, P(x, f(x)) \end{array}$$

where wit and 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 Σ^0_1-statement $\exists z\, g(z) = 0$ 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 callcc and throw control operators[2], we can also compute witnesses from closed proofs of Σ^0_1-statements.

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

$$\begin{array}{l} \mathtt{prf}\,(\mathtt{callcc}_\alpha(t_1, \phi(\mathtt{throw}_\alpha(t_2, p)))) \\ \quad \triangleright\, \mathtt{callcc}_\alpha \mathtt{prf}\,(t_1, \phi(\mathtt{throw}_\alpha \mathtt{prf}\,(t_2, p))) \end{array}$$

since if the continuation α had input type $\exists n\, P(n)$ 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

$$\begin{array}{l} E[\mathtt{prf}\,(\mathtt{callcc}_\alpha(t_1, \phi(\mathtt{throw}_\alpha(t_2, p))))] \\ \quad \triangleright\, \mathtt{callcc}_\alpha E[\mathtt{prf}\,(t_1, \phi(\mathtt{throw}_\alpha E[\mathtt{prf}\,(t_2, p)]))] \\ E[\mathtt{wit}\,(\mathtt{callcc}_\alpha(t_1, \phi(\mathtt{throw}_\alpha(t_2, p))))] \\ \quad \triangleright\, \mathtt{callcc}_\alpha E[\mathtt{wit}\,(t_1, \phi(\mathtt{throw}_\alpha E[\mathtt{wit}\,(t_2, p)]))] \end{array}$$

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 $\mathtt{prf}(\mathtt{callcc}_\alpha(t_1, \phi(\mathtt{throw}_\alpha(t_2, p))))$ 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 $\mathtt{callcc}_\alpha(t_1, \phi(\mathtt{throw}_\alpha(t_2, p)))$), 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

$$AC_{\mathbb{N}} \quad : \quad \forall x^{\mathbb{N}} \exists y^A\, P(x,y) \to \exists f^{\mathbb{N}\to A}\, \forall x^{\mathbb{N}}\, P(x, f(x))$$

and the slightly stronger axiom of dependent choice

$$\begin{array}{lll} DC & : & \forall x^A \exists y^A\, P(x,y) \to \\ & & \forall x_0\, \exists f^{A\to A}\, (f(0) = x_0 \wedge \forall n\, P(f(n), f(S(n)))) \end{array}$$

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 Σ-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 λ-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 some variant of open induction called update induction:

$$\begin{array}{rcl} UI_P & : & \forall f (\forall n (f(n) = \bot \to \forall a\, P(f[n \leftarrow a])) \to P(f)) \\ & & \to \forall f\, P(f) \end{array}$$

for f ranging over $\mathbb{N} \to A_\bot$ for A arbitrary and $A_\bot$ the extension of A with one extra element $\bot$, for a ranging in A and $f[n \leftarrow a]$ denoting the function g defined by $g(n) = a$ and $g(p) = f(p)$ for $p \neq n$, for $P(f)$ open predicate of the form $\forall n\, Q(f_{|n}) \to \exists n\, R(f_{|n})$ assuming that $f_{|n}$ is the sequence $(f(0), ..., f(n-1))$. Otherwise said, Berger reduced the computational content of the axiom of dependent choice to a well-foundedness axiom whose computational content is a simple fixpoint. In practice, this means that we can prove the axioms of countable choice and dependent choice in a logic satisfying cut-elimination by just setting an axiom UI_P whose computational content is a well-founded recursor: $UI_P\, p\, f \triangleright p\, f\, (\lambda n \lambda q \lambda a. UI_P\, p\, f[n \leftarrow a])$.

In 2003, Krivine proposed realisers for the axioms of countable choice and dependent choice [23] in the context of classical realisability for second-order arithmetic. Classical realisability, as developed by Krivine [24], can be seen as the composition of Kleene's realisability [22] with double negation translation and Friedman-Dragalin's A-translation[4] [13], [16]. Alternatively, it can be seen as a form of realisability allowing the use of control operators in realisers. Using our notations, the variant of countable choice realised by Krivine is

$$AC^\star_{\mathbb{N}} \quad : \quad \forall x^{\mathbb{N}} \exists Y^{\mathbb{N}\to\star}\, P(x, Y) \to \exists F^{\mathbb{N}\to\mathbb{N}\to\star}\, \forall x^{\mathbb{N}}\, P(x, F(x))$$

where $\mathbb{N} \to \star$ and $\mathbb{N} \to \mathbb{N} \to \star$ respectively denote the type of predicates and relations over $\mathbb{N}$. Krivine's realiser for $AC^\star_{\mathbb{N}}$ does not use a fixpoint but instead a "quote" function which, informally, maps those Y_x such that $P(x, Y_x)$ into natural numbers that can then be compared so that the Y_x with least "quote" is used to define F on x. The realiser also crucially uses control operators: if some Y_x is found that has lesser quoted value than the Y_x 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_x 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.

[4]See e.g. Berger and Oliva [8] for a notion of realisability obtained by combination of Kleene's realisability and Friedman-Dragalin A-translation and in which $\bot$ is realisable. That Krivine's classical realisability contains A-translation comes from the fact that $\bot$ is not empty but realised by a fixed set of realisers.

c) Call-by-name, call-by-value and call-by-need: Church's λ-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[5] call-by-value counterpart, thanks successively to Plotkin [32], Moggi [27], Sabry and Felleisen [33], Sabry and Wadler [34], etc. Similarly, call-by-need λ-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.

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:

$$\begin{array}{l} AC_{\mathbb{N}} \triangleq \lambda H. (\lambda n.\mathtt{if}\ n = 0\ \mathtt{then}\ \mathtt{wit}\,(H\,0)\ \mathtt{else} \\ \qquad\qquad\quad \mathtt{if}\ n = 1\ \mathtt{then}\ \mathtt{wit}\,(H\,1)\ \mathtt{else}\ ..., \\ \qquad\qquad \lambda n.\mathtt{if}\ n = 0\ \mathtt{then}\ \mathtt{prf}\,(H\,0)\ \mathtt{else} \\ \qquad\qquad\quad \mathtt{if}\ n = 1\ \mathtt{then}\ \mathtt{prf}\,(H\,1)\ \mathtt{else}\ ...) \end{array}$$

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

$$\begin{array}{l} AC_{\mathbb{N}} \triangleq \lambda H.\ \mathtt{let}\ H_0 = H\,0\ \mathtt{in} \\ \qquad\qquad\quad \mathtt{let}\ H_1 = H\,1\ \mathtt{in} \\ \qquad\qquad\quad ... \\ \qquad\qquad\quad (\lambda n.\mathtt{if}\ n = 0\ \mathtt{then}\ \mathtt{wit}\,H_0\ \mathtt{else} \\ \qquad\qquad\qquad \mathtt{if}\ n = 1\ \mathtt{then}\ \mathtt{wit}\,H_1\ \mathtt{else}\ ..., \\ \qquad\qquad\quad \lambda n.\mathtt{if}\ n = 0\ \mathtt{then}\ \mathtt{prf}\,H_0\ \mathtt{else} \\ \qquad\qquad\qquad \mathtt{if}\ n = 1\ \mathtt{then}\ \mathtt{prf}\,H_1\ \mathtt{else}\ ...) \end{array}$$

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

$$\begin{array}{l} AC_{\mathbb{N}} \triangleq \lambda H.\ \mathtt{let}\ s = \mathtt{cofix}^0_{fn}(H\,n, f\,n)\ \mathtt{in} \\ \qquad\qquad\quad (\lambda n.\mathtt{wit}\,(\mathtt{nth}\,n\,s), \lambda n.\mathtt{prf}\,(\mathtt{nth}\,n\,s)) \end{array}$$

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

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

while $\mathtt{nth}\, n\, s$ 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 $\nu^{0}_{Xn}(\exists y\, P(0,y) \wedge X(n+1))$. 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 Σ^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^{ω}: 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^{ω}, 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^{ω}, 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 of proofs. In dPA^{ω}, terms t, u, ... can depend on proofs p, q, ..., and vice versa so that both are defined mutually:

$$
\begin{array}{rcl}
t,u & ::= & x \mid 0 \mid S(t) \mid \mathtt{rec}\ t\ \mathtt{of}\ [t \mid (x,y).t] \\
 & \mid & \lambda x.t \mid t\,t \mid \mathtt{wit}\, p \\
p,q & ::= & a \mid \iota_i(p) \mid (p_1,p_2) \mid (t,p) \mid \lambda a.p \mid \lambda x.p \\
 & \mid & \mathtt{case}\ p\ \mathtt{of}\ [a_1.p_1 \mid a_2.p_2] \\
 & \mid & \mathtt{split}\ p\ \mathtt{as}\ (x,a)\ \mathtt{in}\ q \\
 & \mid & \mathtt{dest}\ p\ \mathtt{as}\ (x,a)\ \mathtt{in}\ q \mid \mathtt{prf}\, p \\
 & \mid & p\,q \mid p\,t \mid \mathtt{exfalso}\ p \\
 & \mid & \mathtt{refl} \mid \mathtt{subst}\, p\, q \\
 & \mid & \mathtt{ind}\ t\ \mathtt{of}\ [p \mid (x,a).q] \\
 & \mid & \mathtt{cofix}^{t}_{bx} p \\
 & \mid & \mathtt{catch}_{\alpha} p \mid \mathtt{throw}_{\alpha} p \\
 & \mid & \mathtt{let}\ a = p\ \mathtt{in}\ q
\end{array}
$$

where f ranges over function symbols, $\vec{t}$ denotes in $f(\vec{t})$ a sequence of terms of length the arity of f, the names x, y, ... range over a set of term variables, a, b, ... over a set of proof variables, α, β, ... over a set of continuation variables. The constructions $\lambda a.p$, $\mathtt{case}\ p\ \mathtt{of}\ [a_1.p_1 \mid a_2.p_2]$, $\mathtt{split}\ p\ \mathtt{as}\ (a_1,a_2)\ \mathtt{in}\ q$, $\mathtt{dest}\ p\ \mathtt{as}\ (x,a)\ \mathtt{in}\ q$ and $\mathtt{ind}\ t\ \mathtt{of}\ [p \mid (x,a).q]$ bind a, a_1 and a_2. The constructions $\lambda x.p$, $\mathtt{dest}\ p\ \mathtt{as}\ (x,a)\ \mathtt{in}\ q$, $\mathtt{dest}\ p\ \mathtt{as}\ (x,a)\ \mathtt{in}\ t$, $\lambda x.t$, $\mathtt{ind}\ t\ \mathtt{of}\ [p \mid (x,a).q]$ and $\mathtt{rec}\ t\ \mathtt{of}\ [t \mid (x,y).t]$ bind x and y. The construction $\mathtt{catch}_{\alpha} p$ binds α. The binders are considered up to the actual name used to represent the binder (α-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 ($\mathtt{case}$) or destruction of proofs ($\mathtt{split}$ and $\mathtt{dest}$). The operator $\mathtt{rec}\ t\ \mathtt{of}\ [t \mid (x,y).t]$ is for recursion in finite types while $\mathtt{ind}\ t\ \mathtt{of}\ [p \mid (x,a).q]$ is for induction. The construction $\mathtt{cofix}^{t}_{bx} p$ is for building coinductive formulae.

Let us say also that the operators $\mathtt{catch}$ and $\mathtt{throw}$ implement classical reasoning. They are similar to the operators of same name in Nakano [28] or Crolard [11]. In terms of Parigot's λ-calculus [30], $\mathtt{catch}_{\alpha} p$ is basically equivalent to $\mu\alpha.[\alpha]p$ and $\mathtt{throw}_{\alpha} p$ to $\mu\delta.[\alpha]p$ for δ not occurring in p.

The abbreviations $\pi_1(p) \triangleq \mathtt{split}\ p\ \mathtt{as}\ (a_1,a_2)\ \mathtt{in}\ a_1$ and $\pi_2(p) \triangleq \mathtt{split}\ p\ \mathtt{as}\ (a_1,a_2)\ \mathtt{in}\ a_2$ 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^{ω} 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:

$$V \quad ::= \quad a \mid \iota_i(V) \mid (V,V) \mid (t,V) \mid \lambda a.p \mid \lambda x.p \mid () \mid \mathtt{refl}$$

To define the operational semantics of dPA^{ω}, 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:

$$
\begin{array}{rcl}
F[\ ] & ::= & \iota_i([\ ]) \mid ([\ ],p) \mid (V,[\ ]) \mid (t,[\ ]) \\
 & \mid & \mathtt{case}\ [\ ]\ \mathtt{of}\ [a_1.p_1 \mid a_2.p_2] \\
 & \mid & \mathtt{split}\ [\ ]\ \mathtt{as}\ (a_1,a_2)\ \mathtt{in}\ q \\
 & \mid & \mathtt{dest}\ [\ ]\ \mathtt{as}\ (x,a)\ \mathtt{in}\ p \mid \mathtt{prf}\,[\ ] \\
 & \mid & [\ ]\,q \mid [\ ]\,t \mid \mathtt{let}\ a = [\ ]\ \mathtt{in}\ q \\
 & \mid & \mathtt{subst}\,[\,]\,p
\end{array}
$$

$$D[\ ] \quad ::= \quad [\ ] \mid D[F[\ ]] \mid \mathtt{let}\ a = \mathtt{cofix}^{t}_{bx} p\ \mathtt{in}\ D[\ ]$$

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 $D[\ ]$.

$$
\begin{array}{lcl}
\texttt{let}\ a = \iota_i(p)\ \texttt{in}\ q & \triangleright & \texttt{let}\ b = p\ \texttt{in}\ q[\iota_i(b)/a] \\
\texttt{let}\ a = (p_1, p_2)\ \texttt{in}\ q & \triangleright & \texttt{let}\ a_1 = p_1\ \texttt{in}\ \texttt{let}\ a_2 = p_2\ \texttt{in}\ q[(a_1, a_2)/a] \\
\texttt{let}\ a = (t, p)\ \texttt{in}\ q & \triangleright & \texttt{let}\ b = p\ \texttt{in}\ q[(t, b)/a] \\
\texttt{let}\ a = \lambda b.p\ \texttt{in}\ q & \triangleright & q[\lambda b.p/a] \\
\texttt{let}\ a = \lambda x.p\ \texttt{in}\ q & \triangleright & q[\lambda x.p/a] \\
\texttt{let}\ a = ()\ \texttt{in}\ q & \triangleright & q[()/a] \\
\texttt{let}\ a = b\ \texttt{in}\ q & \triangleright & q[b/a] \\
\texttt{case}\ \iota_i(p)\ \texttt{of}\ [a_1.p_1 \,|\, a_2.p_2] & \triangleright & \texttt{let}\ a_i = p\ \texttt{in}\ p_i \\
\texttt{split}\ (p_1, p_2)\ \texttt{as}\ (a_1, a_2)\ \texttt{in}\ q & \triangleright & \texttt{let}\ a_1 = p_1\ \texttt{in}\ \texttt{let}\ a_2 = p_2\ \texttt{in}\ q \\
\texttt{dest}\ (t, p)\ \texttt{as}\ (x, a)\ \texttt{in}\ q & \triangleright & \texttt{let}\ a = p\ \texttt{in}\ q[t/x] \\
\texttt{prf}\,(t, p) & \triangleright & p \\
(\lambda a.q)\, p & \triangleright & \texttt{let}\ a = p\ \texttt{in}\ q \\
(\lambda x.p)\, t & \triangleright & p[t/x] \\
\texttt{subst}\ \texttt{refl}\ p & \triangleright & p \\
\texttt{ind}\ 0\ \texttt{of}\ [p \,|\, (x, a).q] & \triangleright & p \\
\texttt{ind}\ S(t)\ \texttt{of}\ [p \,|\, (x, a).q] & \triangleright & q[t/x][\texttt{ind}\ t\ \texttt{of}\ [p \,|\, (x, a).q]/a] \\
\texttt{case}\ \texttt{cofix}^t_{bx} p\ \texttt{of}\ [a_1.p_1 \,|\, a_2.p_2] & \triangleright & \texttt{let}\ c = \texttt{cofix}^t_{bx} p\ \texttt{in}\ \texttt{case}\ c\ \texttt{of}\ [a_1.p_1 \,|\, a_2.p_2] \\
\texttt{split}\ \texttt{cofix}^t_{bx} p\ \texttt{as}\ (a_1, a_2)\ \texttt{in}\ q & \triangleright & \texttt{let}\ c = \texttt{cofix}^t_{bx} p\ \texttt{in}\ \texttt{split}\ c\ \texttt{as}\ (a_1, a_2)\ \texttt{in}\ q \\
\texttt{dest}\ \texttt{cofix}^t_{bx} p\ \texttt{as}\ (x, a)\ \texttt{in}\ q & \triangleright & \texttt{let}\ c = \texttt{cofix}^t_{bx} p\ \texttt{in}\ \texttt{dest}\ c\ \texttt{as}\ (x, a)\ \texttt{in}\ q \\
\texttt{let}\ a = \texttt{cofix}^t_{bx} p\ \texttt{in}\ \texttt{exfalso}\ q & \triangleright & \texttt{exfalso}\ \texttt{let}\ a = \texttt{cofix}^t_{bx} p\ \texttt{in}\ q \\
\texttt{let}\ a = \texttt{cofix}^t_{bx} p\ \texttt{in}\ \texttt{throw}_\alpha q & \triangleright & \texttt{throw}_\alpha \texttt{let}\ a = \texttt{cofix}^t_{bx} p\ \texttt{in}\ q \\
\texttt{let}\ a = \texttt{cofix}^t_{bx} p\ \texttt{in}\ \texttt{catch}_\alpha q & \triangleright & \texttt{catch}_\alpha \texttt{let}\ a = \texttt{cofix}^t_{bx} p\ \texttt{in}\ q \\
\texttt{let}\ a = \texttt{cofix}^t_{bx} p\ \texttt{in}\ D[\texttt{case}\ a\ \texttt{of}\ [a_1.p_1 \,|\, a_2.p_2]] & \triangleright & \texttt{let}\ a = p[\lambda y.\texttt{cofix}^y_{bx} p/b][t/x]\ \texttt{in}\ D[\texttt{case}\ a\ \texttt{of}\ [a_1.p_1 \,|\, a_2.p_2]] \\
\texttt{let}\ a = \texttt{cofix}^t_{bx} p\ \texttt{in}\ D[\texttt{split}\ a\ \texttt{as}\ (a_1, a_2)\ \texttt{in}\ q] & \triangleright & \texttt{let}\ a = p[\lambda y.\texttt{cofix}^y_{bx} p/b][t/x]\ \texttt{in}\ D[\texttt{split}\ a\ \texttt{as}\ (a_1, a_2)\ \texttt{in}\ q] \\
\texttt{let}\ a = \texttt{cofix}^t_{bx} p\ \texttt{in}\ D[\texttt{dest}\ a\ \texttt{as}\ (x, a')\ \texttt{in}\ q] & \triangleright & \texttt{let}\ a = p[\lambda y.\texttt{cofix}^y_{bx} p/b][t/x]\ \texttt{in}\ D[\texttt{dest}\ a\ \texttt{as}\ (x, a')\ \texttt{in}\ q] \\
F[\texttt{let}\ a = \texttt{cofix}^t_{bx} p\ \texttt{in}\ q] & \triangleright & \texttt{let}\ a = \texttt{cofix}^t_{bx} p\ \texttt{in}\ F[q] \\
F[\texttt{exfalso}\ p] & \triangleright & \texttt{exfalso}\ p \\
F[\texttt{throw}_\alpha p] & \triangleright & \texttt{throw}_\alpha p \\
F[\texttt{catch}_\alpha p] & \triangleright & \texttt{catch}_\alpha F[p[F/\alpha]] \\
\texttt{exfalso}\ \texttt{exfalso}\ p & \triangleright & \texttt{exfalso}\ p \\
\texttt{exfalso}\ \texttt{throw}_\beta p & \triangleright & \texttt{throw}_\beta p \\
\texttt{exfalso}\ \texttt{catch}_\beta p & \triangleright & \texttt{exfalso}\ p[\texttt{exfalso}\ [\ ]/\alpha] \\
\texttt{throw}_\beta \texttt{exfalso}\ p & \triangleright & \texttt{exfalso}\ p \\
\texttt{throw}_\beta \texttt{throw}_\alpha p & \triangleright & \texttt{throw}_\alpha p \\
\texttt{throw}_\beta \texttt{catch}_\alpha p & \triangleright & \texttt{throw}_\beta p[\beta/\alpha] \\
\texttt{catch}_\alpha \texttt{throw}_\alpha p & \triangleright & \texttt{catch}_\alpha p \\
\texttt{catch}_\beta \texttt{catch}_\alpha p & \triangleright & \texttt{catch}_\beta p[\beta/\alpha] \\
 & & \\
\texttt{wit}\,(t, p) & \triangleright & t \\
(\lambda x.t)\, u & \triangleright & t[u/x] \\
\texttt{rec}\ 0\ \texttt{of}\ [t_0 \,|\, (x, y).t_S] & \triangleright & t_0 \\
\texttt{rec}\ S(t)\ \texttt{of}\ [t_0 \,|\, (x, y).t_S] & \triangleright & t_S[t/x][\texttt{rec}\ t\ \texttt{of}\ [t_0 \,|\, (x, y).t_S]/y]
\end{array}
$$

Fig. 1. Reduction rules on terms and proofs of dPA^ω

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 α) and where the substitution $p[F/\alpha]$ means replacing subterms of the form $\texttt{throw}_\alpha\, q$ in p by $\texttt{throw}_\alpha\, F[q]$ (including the recursive replacements in q)[6].

We write $\triangleright\!\!\triangleright$ for the reflexive-transitive closure of $\triangleright$. We write $\equiv$ for the reflexive-symmetric-transitive closure of $\triangleright$.

[6]Strictly speaking, the definition of $p[\beta/\alpha]$ and $p[F/\alpha]$ requires also to consider their variants $t[\beta/\alpha]$ and $t[F/\alpha]$ 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:

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

In dPA^ω, we consider implication to be possibly dependent in its antecedent and use the notation $[a : A] \to B$ 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 $[a : \exists x\, P(x)] \to$

$$\frac{(a:A)\in\Gamma}{\Gamma\vdash a:A}\ \textsc{axiom}\qquad\frac{\Gamma\vdash p:A\qquad A\equiv B}{\Gamma\vdash p:B}\ \textsc{conv}$$

$$\frac{\Gamma\vdash p:A_i}{\Gamma\vdash\iota_i(p):A_1\vee A_2}\ \vee_I^i\qquad\frac{\Gamma\vdash p:A_1\vee A_2\qquad\Gamma,a_1:A_1\vdash p_1:B\qquad\Gamma,a_2:A_2\vdash p_2:B}{\Gamma\vdash\mathtt{case}\ p\ \mathtt{of}\ [a_1.p_1\,|\,a_2.p_2]:B}\ \vee_E$$

$$\frac{\Gamma\vdash p_1:A_1\qquad\Gamma\vdash p_2:A_2}{\Gamma\vdash(p_1,p_2):A_1\wedge A_2}\ \wedge_I\qquad\frac{\Gamma\vdash p:A_1\wedge A_2\qquad\Gamma,a_1:A_1,a_2:A_2\vdash q:B}{\Gamma\vdash\mathtt{split}\ p\ \mathtt{as}\ (a_1,a_2)\ \mathtt{in}\ q:B}\ \wedge_E$$

$$\frac{\Gamma\vdash p:A[t/x]\qquad\Gamma\vdash t:T}{\Gamma\vdash(t,p):\exists x^T A}\ \exists_I\qquad\frac{\Gamma\vdash p:\exists x^T A\qquad\Gamma,x:T,a:A\vdash q:B}{\Gamma\vdash\mathtt{dest}\ p\ \mathtt{as}\ (x,a)\ \mathtt{in}\ q:B}\ \exists_E$$

$$\frac{\Gamma\vdash p:\exists x^T A\qquad p\text{ is N-elimination-free}}{\Gamma\vdash\mathtt{prf}\,p:A[\mathtt{wit}\,p/x]}\ \exists_E^{\mathrm{PRF}}$$

$$\frac{\Gamma,a:A\vdash p:B}{\Gamma\vdash\lambda a.p:[a:A]\to B}\ \to_I\qquad\frac{\Gamma\vdash p:[a:A]\to B\qquad\Gamma\vdash q:A\quad a\notin FV(B)\text{ if }q\text{ not N-elimination-free}}{\Gamma\vdash p\,q:B[q/a]}\ \to_E$$

$$\frac{\Gamma,x:T\vdash p:A}{\Gamma\vdash\lambda x.p:\forall x^T A}\ \forall_I\qquad\frac{\Gamma\vdash p:\forall x^T A\qquad\Gamma\vdash t:T}{\Gamma\vdash p\,t:A[t/x]}\ \forall_E$$

$$\frac{}{\Gamma\vdash():\top}\ \top_I\qquad\frac{\Gamma\vdash p:\bot}{\Gamma\vdash\mathtt{exfalso}\ p:C}\ \bot_E$$

$$\frac{\Gamma\vdash t:\mathbb{N}}{\Gamma\vdash\mathtt{refl}:t=t}\ \textsc{refl}\qquad\frac{\Gamma\vdash p:t=u\qquad\Gamma\vdash q:A[t/x]\qquad x\notin Dom(\Gamma)}{\Gamma\vdash\mathtt{subst}\,p\,q:A[u/x]}\ \textsc{subst}$$

$$\frac{\Gamma\vdash t:\mathbb{N}\qquad\Gamma\vdash p:A[0/x]\qquad\Gamma,x:T,a:A\vdash q:A[S(x)/x]}{\Gamma\vdash\mathtt{ind}\ t\ \mathtt{of}\ [p\,|\,(x,a).q]:A[t/x]}\ \textsc{ind}$$

$$\frac{\Gamma\vdash p:A\qquad\Gamma,a:A\vdash q:B\qquad a\notin FV(B)\text{ if }p\text{ not N-elimination-free}}{\Gamma\vdash\mathtt{let}\ a=p\ \mathtt{in}\ q:B[p/a]}\ \textsc{cut}$$

$$\frac{\Gamma\vdash t:T\qquad\Gamma,f:T\to\mathbb{N},x:T,b:\forall y\,f(y)=0\vdash p:A\qquad f\text{ positive in }A}{\Gamma\vdash\mathtt{cofix}^t_{bx}p:\nu^t_{fx}A}\ \nu_I$$

$$\frac{\Gamma,\alpha:A^{\perp\!\!\!\perp}\vdash p:A}{\Gamma\vdash\mathtt{catch}_\alpha\,p:A}\ \textsc{catch}\qquad\frac{\Gamma\vdash p:A\qquad(\alpha:A^{\perp\!\!\!\perp})\in\Gamma}{\Gamma\vdash\mathtt{throw}_\alpha\,p:C}\ \textsc{throw}$$

$$\frac{(x:T)\in\Gamma}{\Gamma\vdash x:T}\qquad\frac{\Gamma,x:U\vdash t:T}{\Gamma\vdash\lambda x.t:U\to T}\qquad\frac{\Gamma\vdash t:U\to T\qquad\Gamma\vdash u:U}{\Gamma\vdash t\,u:T}\qquad\frac{}{\Gamma\vdash 0:\mathbb{N}}\qquad\frac{\Gamma\vdash t:\mathbb{N}}{\Gamma\vdash S(t):\mathbb{N}}$$

$$\frac{\Gamma\vdash t:\mathbb{N}\qquad\Gamma\vdash t_0:U\qquad\Gamma,x:\mathbb{N},y:U\vdash t_S:U}{\Gamma\vdash\mathtt{rec}\ t\ \mathtt{of}\ [t_0\,|\,(x,y).t_S]:U}\qquad\frac{\Gamma\vdash p:\exists x^T A\quad p\text{ is N-elimination-free}}{\Gamma\vdash\mathtt{wit}\,p:T}\ \exists_E^{\mathrm{WIT}}$$

Fig. 2. dPA^ω: Classical arithmetic in finite types with strong existential

$P(\texttt{wit}\, a))$[7]. In addition to the usual connectives and quantifiers, we have coinductive formulae:

$$\begin{array}{lll} A, B & ::= & t = u \mid [a : A] \to B \mid A \vee B \mid A \wedge B \mid \bot \mid \top \\ & \mid & \forall x^T A \mid \exists x^T A \mid \nu^t_{fx} A \end{array}$$

In atoms, P ranges over predicate symbols and $\vec{t}$ is a sequence of terms whose length is the arity of P. Negation $\neg A$ is defined as $A \to \bot$. The construction $\nu^t_{fx} A$ stands for the instance on t of the coinductive predicate built from the monotone functor $\lambda f.\lambda x.A$ where A is made of atoms and positive connectives only (including ν-formulae themselves). In $\forall x\, A$ and $\exists x\, A$, x is bound and freely subject to renaming (α-conversion). In $\nu^t_{fx} A$, x and f are bound term variables. The formulae $\forall x\, A$ and $[a : A] \to B$ are called negative. All other kinds of formulae are called positive.

Formulae are considered modulo the equational theory on terms, as it is common in Martin-Löf's intensional type theory. The equational theory is the one induced by reduction on terms and proofs plus the following reduction rules for equality[8] and coinductive formulae unfolding:

$$\begin{array}{lll} 0 = 0 & \triangleright & \top \\ 0 = S(u) & \triangleright & \bot \\ S(t) = 0 & \triangleright & \bot \\ S(t) = S(u) & \triangleright & t = u \\ \nu^t_{fx} A & \triangleright & A[t/x][\nu^y_{fx} A / f(y) = 0] \end{array}$$

We write $\equiv$ for the resulting[9] reflexive-symmetric-transitive closure of $\triangleright$ on formulae.

When obvious 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^ω, it is convenient to indicate the types of variables in typing contexts. Hence, contexts are defined by:

$$\Gamma \quad ::= \quad \emptyset \mid \Gamma, x : T \mid \Gamma, a : A \mid \Gamma, \alpha : A^{\perp\!\!\!\perp}$$

where $a : A$ stands for an assumption of A and $\alpha : A^{\perp\!\!\!\perp}$ for an assumption of the refutation[10] 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 $Dom(\Gamma)$ for the set of names a and α thus declared in Γ.

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.

[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^ω with dependent implication is conservative over its version without dependent implication.

[8] See e.g. Allali [1] for such a presentation of arithmetic.

[9] 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.

[10] Not to be confused with the notation A^B sometimes used for powerset.

In the rule ν_I, the function f is said to be positive in A if A is built from atoms[11] $f(t) = 0$ using disjunction, conjunction, existential quantification, equality or another coinductive type. That the typing rule ν_I and the reduction rule for coinductive formulae do not extend by themselves the logical strength of HA^ω and PA^ω comes from the equations given in Figure 3 where $\texttt{out}\, a$ implements the reduction rule for $\nu^t_{fx} A$. However, the derived computational content is not the one we want because of the use of non positive connectives in the second-order encoding, what justifies taking ν_I and its associated reduction rule as primitive. Indeed, with a primitive notion of coinductive formula, it becomes syntactically direct to see ν as a constructor of positive formulae. In particular, and this is important later on to prove the axioms of countable choice and dependent choice, strong existential elimination is allowed to descend through coinductive formulae.

Dependent proofs have to be N-elimination-free (negative-elimination-free). N-elimination-freeness is defined by the following rules:

- a, $()$, $\lambda x.p$ and $\lambda a.p$ are N-elimination-free
- if p, q, p_1 and p_2 are N-elimination-free then $\iota_i(p)$, (p_1, p_2), (t, p), $\texttt{case}\ a\ \texttt{of}\ [a_1.p_1 \mid a_2.p_2]$, $\texttt{split}\ q\ \texttt{as}\ (a_1, a_2)\ \texttt{in}\ p$, $\texttt{dest}\ q\ \texttt{as}\ (x, a)\ \texttt{in}\ p$, $\texttt{prf}\, p$, $\texttt{refl}$, $\texttt{subst}\, p\, q$, $\texttt{ind}\ t\ \texttt{of}\ [p_1 \mid (x, a).p_2]$ and $\texttt{let}\ a = p\ \texttt{in}\ q$ are N-elimination-free.

Otherwise said, in N-elimination-free proofs, expressions of the form $p\, q$, $p\, t$, $\texttt{exfalso}\ p$, $\texttt{catch}_\alpha p$ or $\texttt{throw}_\alpha p$ can only occur in the body of a λx or of a λa.

The N-elimination-free condition is what ensures in particular that $\texttt{wit}$ will never be applied to a classical proof, i.e. to a proof starting with $\texttt{catch}_\alpha p$ or $\texttt{throw}_\alpha p$.

The resulting theory is then essentially Troelstra's arithmetic in all finite types HA^ω (with equality on $\mathbb{N}$) extended with classical logic and strong existential elimination. We write dHA^ω for the version of dPA^ω with rules CATCH and THROW removed. Since dPA^ω has classical reasoning, quantification over functional symbols and, as will be shown in Section III, dependent choice, it can simulate quantification over the predicates talking about $\mathbb{N}$ (since from the classical statement $\forall n\, \exists b\, (b = 0 \wedge \phi(n)) \vee (b = 1 \wedge \neg\phi(n))$, we get a characteristic function f for ϕ, i.e. a function that satisfies $\forall n\, (f(n) = 0 \wedge \phi(n)) \vee (f(n) = 1 \wedge \neg\phi(n)))$. However, to get 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

$$\forall x^T\, \exists! n\, P(x, n) \to \exists f\, \forall x^T\, P(x, f(x))$$

and there is no reason to think that this holds.

We are now ready to state the operational and logical properties of dPA^ω.

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

[11] Since we have symbols for functions and not for predicates, we use expressions of the form $f(t) = 0$ to represent arbitrary atoms.

[12] I.e. reification of functional relations into functions.

$$
\begin{array}{lcl}
\nu^t_{fx} A & \triangleq & \exists f\,(f(t) = 0 \wedge \forall x\,(f(x) = 0 \rightarrow A)) \\
\mathtt{cofix}^t_{bx} p & \triangleq & (\lambda x.0, (\mathtt{refl}, \lambda x.\lambda b.p)) \\
\mathtt{out}\, a & \triangleq & \mathtt{dest}\ a\ \mathtt{as}\ (f,b)\ \mathtt{in}\ \mathtt{split}\ b\ \mathtt{as}\ (c,d)\ \mathtt{in}\ \mathtt{mon}^{A[f/f][t/x]}_{f,d}\,(d\,t\,c)
\end{array}
$$

where, for $d : \forall x\, f(x) = 0 \rightarrow A$, and $a : B$, $\mathtt{mon}^B_{f,d}\, a$ is defined inductively:

$$
\begin{array}{lcl}
\mathtt{mon}^B_{f,d} a & : & B[\nu^y_{fx} A / f(y) = 0] \\
\mathtt{mon}^{f(t)=0}_{f,d}\ a & \triangleq & (f, (a, d)) \\
\mathtt{mon}^{B_1 \wedge B_2}_{f,d}\ a & \triangleq & \mathtt{split}\ a\ \mathtt{as}\ (a_1, a_2)\ \mathtt{in}\ (\mathtt{mon}^{B_1}_{f,d}\, a_1, \mathtt{mon}^{B_2}_{f,d}\, a_2) \\
\mathtt{mon}^{B_1 \vee B_2}_{f,d}\ a & \triangleq & \mathtt{case}\ a\ \mathtt{of}\ [a_1.\mathtt{mon}^{B_1}_{f,d}\, a_1 \,|\, a_2.\mathtt{mon}^{B_2}_{f,d}\, a_2] \\
\mathtt{mon}^{\exists x\, B}_{f,d}\ a & \triangleq & \mathtt{dest}\ a\ \mathtt{as}\ (x, a)\ \mathtt{in}\ (x, \mathtt{mon}^B_{f,d}\, a) \\
\mathtt{mon}^{\nu^u_{gy} C}_{f,d}\ a & \triangleq & \mathtt{dest}\ a\ \mathtt{as}\ (g, b)\ \mathtt{in}\ \mathtt{split}\ b\ \mathtt{as}\ (c', d')\ \mathtt{in}\ (g, (c', \lambda x.\lambda a.\mathtt{mon}^C_{f,d}\,(d'\, x\, a)))
\end{array}
$$

Fig. 3. Derivability of introduction and reduction of coinductive formula

Proof: Most reduction rules are standard in a call-by-value setting with control and a simple analysis shows that they preserve the correctness of derivations. The difficulty comes from strong existential elimination. The N-elimination-freeness of p in $\mathtt{prf}\, p$ then ensures that the cases where F is $\mathtt{prf}[\ ]$ in those reduction rules that explicitly mention F cannot happen. In particular, the only rule involving $\mathtt{prf}$ is $\mathtt{prf}\,(t, p) \triangleright p$ which preserves the correctness of derivations. Note also that N-elimination-freeness is stable by substitution of values. ■

We claim normalisation by giving a sketch of proof.

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 $\forall$-$\rightarrow$-ν-$\mathtt{wit}$-free formula then $\vdash p : A$ in dPA^ω implies that there is some V such that $\vdash V : A$ in HA^ω.

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

In arithmetic, any Σ^0_1-formula is equivalent to a $\forall$-$\rightarrow$-ν-$\mathtt{wit}$-free formula. Hence we have:

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

This of course implies consistency:

Theorem 4 (Consistency): $\nvdash p : \bot$ in dPA^ω.

III. The Axioms of Countable Choice and Dependent Choice

Our main result is that dPA^ω 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 $\forall x^T A(x)$ where possibly the proof of $A(t)$ is classical into a coinductive conjunction $A(g(0)) \wedge A(g(1)) \wedge A(g(2)) \ldots$ 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 $A(0) \wedge A(1) \wedge A(2) \ldots$, so we consider the coinductive conjunction $R_C(n) \triangleq \nu^n_{fx}(A(x) \wedge f(S(x)) = 0)$. The proof is now direct:

$$
\begin{array}{lcl}
AC_{\mathbb{N}} & \triangleq & \lambda a.\mathtt{let}\ b = \mathtt{cofix}^0_{bn}(a\,n, b\,(S(n)))\ \mathtt{in} \\
 & & (\lambda n.\mathtt{wit}\,(\mathtt{nth}_C\, n\, b), \lambda n.\mathtt{prf}\,(\mathtt{nth}_C\, n\, b)) \\
 & : & \forall n \exists y\, P(n, y) \to \exists f\, \forall n\, P(n, f(n)) \\
\text{where} & & \\
\mathtt{nth}_C\, n & : & R_C(0) \to A(n) \\
\mathtt{nth}_C\, n & \triangleq & \lambda b.\pi_1(\mathtt{ind}\ n\ \mathtt{of}\ [b \,|\, (m, c).\pi_2(c)])
\end{array}
$$

Note that the proof does not use classical logic and holds also in dHA^ω.

B. The Axiom of Dependent Choice

Here again, $A(x)$ is $\exists y\, P(x, y)$ and the appropriate stream we want to build is the stream $A(x_0) \wedge A(g(x_0)) \wedge A(g^2(x_0)) \dots$ where g is the choice function implicit in some proof of $\forall x\, \exists y\, P(x, y)$. So we consider the coinductive formula $R_D(z) \triangleq \nu^z_{fx} \exists y\, (P(x, y) \wedge f(y) = 0)$. The proof is now direct:

$$
\begin{array}{lcl}
DC & \triangleq & \lambda a.\lambda x_0.\mathtt{let}\ b = \mathtt{s}\, a\, x_0\ \mathtt{in} \\
 & & (\lambda n.\mathtt{wit}\,(\mathtt{nth}_D\, n\,(x_0, b)), \\
 & & (\mathtt{refl}, \lambda n.\pi_1(\mathtt{prf}\,(\mathtt{prf}\,(\mathtt{nth}_D\, n\,(x_0, b)))))) \\
 & : & \forall x \exists y\, P(x, y) \to \\
 & & \forall x_0\, \exists f\, (f(0) = x_0 \wedge \forall n\, P(f(n), f(S(n)))) \\
\text{where} & & \\
\mathtt{nth}_D\, n & : & \exists x\, R_D(x) \to \exists x\, R_D(x) \\
\mathtt{nth}_D\, n & \triangleq & \lambda b.\mathtt{ind}\ n\ \mathtt{of} \\
 & & \quad [b \,|\, (m, c).(\mathtt{wit}\,(\mathtt{prf}\, c), \pi_2(\mathtt{prf}\,(\mathtt{prf}\, c)))] \\
\mathtt{s}\, a\, x & : & R_D(x) \\
\mathtt{s}\, a\, x & \triangleq & \mathtt{cofix}^x_{bn}(\mathtt{dest}\ a\, n\ \mathtt{as}\ (y, c)\ \mathtt{in}\ (y, (c, by)))
\end{array}
$$

Note that this proof too does not use classical logic and holds in dHA^ω.

C. Bar Induction

To express bar induction, we extend dPA^ω with a type constructor for finite sequences:

$$
\begin{array}{lcl}
T & ::= & \dots \,|\, T^* \\
t, l & ::= & \dots \,|\, \langle\rangle \,|\, l \star t \,|\, \mathtt{rec}\ l\ \mathtt{of}\ [t \,|\, (x, y, z).t] \\
p & ::= & \dots \,|\, \mathtt{ind}\ l\ \mathtt{of}\ [p \,|\, (x, y, a).p]
\end{array}
$$

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 f from $\mathbb{N}$ to T:

$$f_{|n} \triangleq \mathtt{rec}\ n\ \mathtt{of}\ [\langle\rangle \,|\, (m, l).\, l \star f(m)]$$

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

$$BI : \forall f\, \exists n\, B(f_{|n}) \to \forall P \begin{pmatrix} \forall l\, (B(l) \to P(l)) \wedge \\ \forall l\, (\forall x\, P(l \star x) \to P(l)) \end{pmatrix} \to P(\langle\rangle)$$

Let us consider a contrapositive variant of BI

$$BI_c : \nu^{\langle\rangle}_{gl}(\neg B(l) \wedge \exists x\, g(l \star x) = 0) \to \exists f \forall n\, \neg B(f_{|n})$$

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:

$$
\begin{array}{lcl}
BI^+_C & \triangleq & \lambda a.(\lambda n.\mathtt{wit}\,(\pi_2\,(\mathtt{prf}\,(\mathtt{nth}_{BI}\, n\,(\langle\rangle, a)))), \\
 & & \quad \lambda n.(\mathtt{wit}\,(\mathtt{nth}_{BI}\, n\,(\langle\rangle, a)), \\
 & & \qquad (\pi_1(\mathtt{prf}\,(\mathtt{nth}_{BI}\, n\,(\langle\rangle, a))), e\, a\, n))) \\
 & : & R_{BI}(\langle\rangle) \to \exists f \forall n\, \exists l\, (\neg B(l) \wedge l = f_{|n}) \\
\text{where} & & \\
\mathtt{nth}_{BI}\, n & : & \exists l\, R_{BI}(l) \to \exists l\, R_{BI}(l) \\
\mathtt{nth}_{BI}\, n & \triangleq & \lambda b.\mathtt{ind}\ n\ \mathtt{of} \\
 & & \quad [b \,|\, (m, c).(\mathtt{wit}\, c \star \mathtt{wit}\,(\pi_2\,(\mathtt{prf}\, c)), \\
 & & \qquad \mathtt{prf}\,(\pi_2\,(\mathtt{prf}\, c)))] \\
e\, a\, n & : & \mathtt{wit}\,(\mathtt{nth}_{BI}\, n\,(\langle\rangle, a)) = \\
 & & (\lambda n.\mathtt{wit}\,(\pi_2\,(\mathtt{prf}\,(\mathtt{nth}_{BI}\, n\,(\langle\rangle, a)))))_{|n} \\
e\, a\, n & \triangleq & \mathtt{ind}\ n\ \mathtt{of}\ [\mathtt{refl} \,|\, (m, c).\mathtt{subst}\, c\, \mathtt{refl}]
\end{array}
$$

from which BI_c directly follows. Then, from BI_c and Markov's principle, we get the following weaker form of BI:

$$
\begin{array}{l}
\forall f\, \exists n\, B(f_{|n}) \to \\
\forall g \left[\begin{pmatrix} \forall l\, (B(l) \to g(l) = 0) \wedge \\ \forall l\, (\forall x\, g(l \star x) = 0 \to g(l) = 0) \end{pmatrix} \to g(\langle\rangle) = 0 \right]
\end{array}
$$

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 the value of a function is needed (typically because some computation with the value ends into a natural number that serves in an induction step), it is directly the (first) value given by the proof of $\forall n \exists y\, P(n, y)$ which is used. Secondly, in our proof, the order in which the proofs of $\forall n \exists y\, P(n, y)$ are evaluated is the natural order, while in 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.

h) 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^ω and dHA^ω by replacing the definition of types with:

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

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

$$AC^{\star}_{\mathbb{N}} : \forall n^{\mathbb{N}} \exists Y^{\mathbb{N}\to\star}\, P(n, Y) \to \exists F^{\mathbb{N}\to\mathbb{N}\to\star}\, \forall n^{\mathbb{N}}\, P(n, F\, n)$$

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

[13]In practise, Krivine realises the axiom

$$CAC : \exists Z^{\mathbb{N}\to\mathbb{N}\to\star} \forall n^{\mathbb{N}} (P(n, Z\, n) \to \forall Y^{\mathbb{N}\to\star}\, P(n, Y))$$

which is classically equivalent to $AC_{\mathbb{N}}$ over the codomain $\mathbb{N} \to \star$.

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.

References

[1] L. Allali, "Algorithmic equality in heyting arithmetic modulo," in *TYPES 2007, Revised Selected Papers*, ser. LNCS, M. Miculan, I. Scagnetto, and F. Honsell, Eds., vol. 4941. Springer, 2007, pp. 1–17.

[2] Z. Ariola and M. Felleisen, "The call-by-need lambda calculus," *J. Funct. Program.*, vol. 7, no. 3, pp. 265–301, 1993.

[3] Z. M. Ariola, P. Downen, H. Herbelin, K. Nakata, and A. Saurin, "Classical call-by-need sequent calculi: The unity of semantic artifacts," in *Fuji International Symposium on Functional and Logic Programming (FLOPS '12), Kobe, Japan, May 23-25, 2012*, 2012, to appear.

[4] Z. M. Ariola, H. Herbelin, and A. Saurin, "Classical call-by-need and duality," in *Typed Lambda Calculi and Applications - 10th International Conference, TLCA 2011, Novi Sad, Serbia, June 1-3, 2011. Proceedings*, ser. Lecture Notes in Computer Science, C.-H. L. Ong, Ed., vol. 6690. Springer, 2011, pp. 27–44.

[5] H. P. Barendregt, *The Lambda Calculus: Its Syntax and Semantics*. Amsterdam: North Holland, 1984.

[6] S. Berardi, M. Bezem, and T. Coquand, "On the computational content of the axiom of choice," *J. Symb. Log.*, vol. 63, no. 2, pp. 600–622, 1998.

[7] U. Berger, "A computational interpretation of open induction," in *Proceedings of LICS 2004*. IEEE Computer Society, 2004, p. 326.

[8] U. Berger and P. Oliva, "Modified bar recursion," BRICS, University of Aarhus, Denmark, Tech. Rep. 02/14, Apr. 2002.

[9] A. Church, "A set of postulates for the foundation of logic," *Annals of Mathematics*, vol. 2, pp. 33, 346–366, 1932.

[10] T. Coquand, "A semantics of evidence for classical arithmetic," *J. Symb. Log.*, vol. 60, no. 1, pp. 325–337, 1995.

[11] T. Crolard, "A confluent lambda-calculus with a catch/throw mechanism," *J. Funct. Program.*, vol. 9, no. 6, pp. 625–647, 1999.

[12] P.-L. Curien and H. Herbelin, "The duality of computation," in *Proceedings of ICFP 2000*, ser. SIGPLAN Notices 35(9). ACM, 2000, pp. 233–243.

[13] A. G. Dragalin, "New kinds of realizability and Markov's rule," *Soviet Mathematical Doklady*, vol. 251, pp. 534–537, 1980.

[14] M. H. Escardó and P. Oliva, "Computational interpretations of analysis via products of selection functions," in *CIE*, ser. Lecture Notes in Computer Science, F. Ferreira, B. Löwe, E. Mayordomo, and L. M. Gomes, Eds., vol. 6158. Springer, 2010, pp. 141–150.

[15] J. H. Fasel, P. Hudak, S. Peyton Jones, and P. W. (editors), "Haskell special issue," *SIGPLAN Notices*, vol. 27, no. 5, May 1992.

[16] H. Friedman, "Classically and intuitionistically provably recursive functions," in *Higher Set Theory*, ser. Lecture Notes in Mathematics, D. S. Scott and G. H. Muller, Eds. Berlin/Heidelberg: Springer, 1978, vol. 669, pp. 21–27.

[17] T. G. Griffin, "The formulae-as-types notion of control," in *Conf. Record of POPL '90*. ACM Press, New York, 1990, pp. 47–57.

[18] H. Herbelin, "C'est maintenant qu'on calcule: au cœur de la dualité," Habilitation thesis, University Paris 11, Dec. 2005.

[19] ——, "On the degeneracy of sigma-types in presence of computational classical logic," in *Proceedings of TLCA 2005*, ser. LNCS, P. Urzyczyn, Ed., vol. 3461. Springer, 2005, pp. 209–220.

[20] ——, "An intuitionistic logic that proves Markov's principle," in *Proceedings of LICS 2010*. IEEE Computer Society, 2010, pp. 50–56.

[21] D. Ilik, "Preuves constructives de complétude et contrôle délimité," PhD, École Polytechnique, 2010.

[22] S. C. Kleene, "On the interpretation of intuitionistic number theory," *The Journal of Symbolic Logic*, vol. 10, no. 4, pp. 109–124, 1945.

[23] J.-L. Krivine, "Dependent choice, 'quote' and the clock," *Theor. Comput. Sci.*, vol. 308, no. 1-3, pp. 259–276, 2003.

[24] ——, “Realizability in classical logic,” *Panoramas et synthèses*, 2004, to appear.

[25] J. Maraist, M. Odersky, and P. Wadler, “The call-by-need lambda calculus,” *J. Funct. Program.*, vol. 8, no. 3, pp. 275–317, 1998.

[26] P. Martin-Löf, “A theory of types,” University of Stockholm, Tech. Rep. 71-3, 1971.

[27] E. Moggi, “Computational lambda-calculus and monads,” Edinburgh Univ., Tech. Rep. ECS-LFCS-88-66, 1988.

[28] H. Nakano, “A constructive formalization of the catch and throw mechanism,” in *Proceedings of LICS 1992*. IEEE Computer Society, 1992, pp. 82–89.

[29] C. Okasaki, P. Lee, and D. Tarditi, “Call-by-need and continuation-passing style,” *Lisp and Symbolic Computation*, vol. 7, no. 1, pp. 57–82, 1994.

[30] M. Parigot, “Free deduction: An analysis of "computations" in classical logic.” in *Proceedings of LPAR*, ser. LNCS, A. Voronkov, Ed., vol. 592. Springer, 1991, pp. 361–380.

[31] ——, “Lambda-mu-calculus: An algorithmic interpretation of classical natural deduction,” in *Proceedings of LPAR ’92*. Springer-Verlag, 1992, pp. 190–201.

[32] G. D. Plotkin, “Call-by-name, call-by-value and the lambda-calculus,” *Theor. Comput. Sci.*, vol. 1, pp. 125–159, 1975.

[33] A. Sabry and M. Felleisen, “Reasoning about programs in continuation-passing style,” *Lisp and Symbolic Computation*, vol. 6, no. 3-4, pp. 289–360, 1993.

[34] A. Sabry and P. Wadler, “A reflection on call-by-value,” *ACM Trans. Program. Lang. Syst.*, vol. 19, no. 6, pp. 916–941, 1997.

[35] C. Spector, “Provably recursive functionals of analysis: A consistency proof of analysis by an extension of principles in current intuitionistic mathematics,” in *Recursive function theory: Proceedings of symposia in pure mathematics*, F. D. E. Dekker, Ed., vol. 5. American Mathematical Society, 1962, p. 1–27.