

A constructive proof of dependent choice, compatible with classical logic

DRAFT

September 22, 2011

Hugo Herbelin
INRIA - PPS
23 avenue d'Italie
F-75214 Paris Cedex 13
Hugo.Herbelin@inria.fr

Abstract—Martin-Löf's type theory has strong existential elimination (dependent sum type) what 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.

Because the proof is constructive, it directly yields the conservativity of classical arithmetic over intuitionistic arithmetic for Σ_1^0 -formulae in the presence of the above axioms.

INTRODUCTION

a) *Scaling Martin-Löf's proof of the axiom of choice to classical logic:* In Martin-Löf's intuitionistic type theory [1], the functional form of the axiom of choice has a simple proof:

$$\begin{aligned} AC_A &\triangleq \lambda H.(\lambda x.\text{wit}(H x), \lambda x.\text{prf}(H x)) \\ &: \forall x^A \exists y^B P(x, y) \rightarrow \exists f^{A \rightarrow B} \forall x^A P(x, f(x)) \end{aligned}$$

where `wit` and `prf` are the first and second projections of a *strong* existential quantifier¹.

The proof is constructive: it is a program which we can compute with in the sense that any closed proof of some Σ_1^0 -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 [2], [3] and by interpreting Peirce's law by means of

¹Also known as Σ -type, dependent sum, or strong sum.

the `callcc` and `throw` control operators², we can also compute witnesses from closed proofs of Σ_1^0 -statements.

Combining the two is however delicate. Reminding that `callcc α p` has type A and binds the continuation variable α of input type A when p has type A while `throw α 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:

$$\begin{aligned} &\text{prf}(\text{callcc}_\alpha(t_1, \phi(\text{throw}_\alpha(t_2, p)))) \\ &\triangleright \text{callcc}_\alpha \text{prf}(t_1, \phi(\text{throw}_\alpha \text{prf}(t_2, p))) \end{aligned}$$

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 discourse³ [4]. This first problem is solved by using higher-level reduction rules such as

$$\begin{aligned} &E[\text{prf}(\text{callcc}_\alpha(t_1, \phi(\text{throw}_\alpha(t_2, p))))] \\ &\triangleright \text{callcc}_\alpha E[\text{prf}(t_1, \phi(\text{throw}_\alpha E[\text{prf}(t_2, p)]))] \\ &E[\text{wit}(\text{callcc}_\alpha(t_1, \phi(\text{throw}_\alpha(t_2, p))))] \\ &\triangleright \text{callcc}_\alpha E[\text{wit}(t_1, \phi(\text{throw}_\alpha E[\text{wit}(t_2, p)]))] \end{aligned}$$

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 `prf(callcc α (t_1 , ϕ (throw α (t_2 , p)))` derived from the same initial proof (and this is precisely what

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

³Failure of subject reduction when combining strong existential quantification and computational classical logic was also observed by P. Blain-Levy (private communication).

would happen in Martin-Löf’s proof of AC_A if the two copies of Hx 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

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

and the slightly stronger axiom of dependent choice

$$DC : \forall x^A \exists y^A P(x, y) \rightarrow \forall x_0 \exists f^{A \rightarrow A} (f(0) = x_0 \wedge \forall n P(f(n), f(S(n))))$$

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 first important step was made in 1961 in the context of Gödel’s functional interpretation (Dialectica) with the definition by Spector [5] 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 [6], was proposed in the context of the arithmetic in finite types by Berardi, Bezem and Coquand [7] 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 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 variant of bar recursion. Then, in 2004, Berger [9] reduced the termination of these realisers to some variant of open induction called update induction:

$$UI_P : \forall f (\forall n (f(n) = \perp \rightarrow \forall a P(f[n \leftarrow a])) \rightarrow P(f)) \rightarrow \forall f P(f)$$

for f ranging over $\mathbb{N} \rightarrow A_\perp$ for A arbitrary and A_\perp the extension of A with one extra element \perp , 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 $\forall n Q(f_n) \rightarrow \exists n R(f_n)$ assuming that f_n is the sequence $(f(0), \dots, f(n-1))$. Otherwise said, Berger reduced the computational content of the axiom of dependent choice to a simple wellfoundedness

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 [10] in the context of classical realisability for second-order arithmetic. Classical realisability, as developed by Krivine [11] can be seen as the composition of Kleene’s realisability [6] with double negation translation and Friedman-Dragalin’s A -translation⁴⁵ [12], [13], up to the point that it is stated not in arithmetic but in the more general framework of second order arithmetic in which being a natural number n is expressed as a second-order predicate $\forall P P(0) \rightarrow \forall m (P(m) \rightarrow P(S(m))) \rightarrow P(n)$ and induction performed by instantiating this predicate, making the propositional part of a formula the only part really useful for computation and making therefore the need for keeping a trace of the quantifiers in the realisers (as Kleene’s realisability does) useless in this context. In second-order arithmetic, the codomain of the axioms of countable choice and dependent choices is a domain of predicates instead of an arbitrary finite type and this fact is apparently important for Krivine’s realisers to work. In addition, the realisers use a “quote” function and no fixpoint, so that they are rather different in style from the ones of Berardi, Bezem and Coquand, and a fortiori from bar recursion.

c) *Call-by-name, call-by-value and call-by-need:*

Church’s λ -calculus [14], [15] is a “call-by-name” calculus and it is so simple to define (three constructors, one evaluation rule, one observational rule) that it is only as a consequence of being used in practise for computing that its more intricate⁶ call-by-value counterpart eventually came to be studied from a more theoretical side, thanks successively to Plotkin [18], Moggi [19], Sabry and Felleisen [20], Sabry and Wadler [21], etc. So is call-by-need, which, in spite of being at the heart of programming languages like Haskell [22], is still in a rudimentary stage of abstract understanding, being for a large part only studied from the point of view of standard head

⁴See e.g. Berger and Oliva for a notion of realisability obtained by combination of Kleene’s realisability and Friedman-Dragalin A -translation and in which \perp is realisable.

⁵That Krivine’s classical realisability contains A -translation comes from the fact that \perp is not empty but realised by a fixed set of realisers.

⁶Though, when looking at λ -calculus from the point of view of sequent calculus instead of from the point of view of natural deduction [16], [17], call-by-value λ -calculus is no more complicated than call-by-name, both having the same - intermediate - level of intrinsic technical complexity.

reduction [23], [24] or untyped continuation-passing-style transformation [24]. 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 Hx 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:

$$AC_{\mathbb{N}} \triangleq \lambda H. (\lambda n. \text{if } n = 0 \text{ then wit}(H0) \text{ else} \\ \text{if } n = 1 \text{ then wit}(H1) \text{ else } \dots, \\ \lambda n. \text{if } n = 0 \text{ then prf}(H0) \text{ else} \\ \text{if } n = 1 \text{ then prf}(H1) \text{ else } \dots)$$

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

$$AC_{\mathbb{N}} \triangleq \lambda H. \text{let } H_0 = H0 \text{ in} \\ \text{let } H_1 = H1 \text{ in} \\ \dots \\ (\lambda n. \text{if } n = 0 \text{ then wit } H_0 \text{ else} \\ \text{if } n = 1 \text{ then wit } H_1 \text{ else } \dots, \\ \lambda n. \text{if } n = 0 \text{ then prf } H_0 \text{ else} \\ \text{if } n = 1 \text{ then prf } H_1 \text{ else } \dots)$$

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 $(H0, H1, \dots)$. This leads to the following proof of the countable axiom of choice:

$$AC_{\mathbb{N}} \triangleq \lambda H. \text{let } s = \text{cofix}_{fn}^0(Hn, fn) \text{ in} \\ (\lambda n. \text{wit}(\text{nth } n \text{ } s), \lambda n. \text{prf}(\text{nth } n \text{ } s))$$

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

$\nu_{Xn}^0(\exists y P(0, y) \wedge X(n+1))$. 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 and studying this restriction in the context of predicate logic will be the purpose of Section I. In Section II, we will extend this first framework into a classical arithmetic in finite types, showing in passing how to define coinductive formulae in this context. By lazy evaluating the coinductive proofs, termination can be proved 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 the works of Berardi, Bezem and Coquand and of Krivine .

I. CLASSICAL PREDICATE LOGIC WITH STRONG EXISTENTIAL

As discussed in the introduction, computational classical logic and strong existential elimination marry badly together if mixed together without care. We show in this section that if we restrict the elimination of strong existential to “negative-elimination-free” expressions, in a sense to be described below, then first-order classical logic extended with strong existential elimination has the normalisation property when evaluated along a call-by-value discipline.

Two extreme approaches are actually possible. In a first approach, only strong existential elimination is added and the usual rules of predicate logic are kept non dependent. In such a framework, the property that any provable statement could be proved using a derivation that mentions only subformulae of the concluding statement up to substitution fails (so-called subformula property). For instance, if one proves $\exists x P(x)$ by case analysis on some formula $A_1 \vee A_2$ and providing different witnesses t_1 and t_2 in each branch, then, if one applies strong existential elimination, the latter cannot commute with the case analysis in order to eventually cancel the detour via the a-priori non-subformula $\exists x P(x)$.

In a second approach, such commutations (so-called commutative cuts) are admissible, and, consequently, the subformula property is attainable. This is the approach we follow in this section and the resulting logic is called *dPL*, standing for *dependent predicate logic*.

A. Proofs and terms

Strong existential elimination forces formulae to be dependent of proofs. The extra support for commutative cuts forces these dependencies to strongly pervades in the syntax of terms. In *dPL*, terms t, u, \dots can depend on proofs p, q, \dots , and vice-versa so that both are defined mutually:

$$\begin{aligned}
t, u & ::= f(\vec{t}) \mid x \mid \text{wit } p \\
& \mid \text{case } p \text{ of } [a_1.t_1 \mid a_2.t_2] \\
& \mid \text{split } p \text{ as } (a_1, a_2) \text{ in } t \\
& \mid \text{dest } p \text{ as } (x, a) \text{ in } t \\
p, q & ::= a \mid \iota_i(p) \mid (p, q) \mid (t, p) \mid \lambda a.p \mid \lambda x.p \mid () \\
& \mid \text{case } p \text{ of } [a_1.p_1 \mid a_2.p_2] \\
& \mid \text{split } p \text{ as } (a_1, a_2) \text{ in } q \\
& \mid \text{dest } p \text{ as } (x, a) \text{ in } q \mid \text{prf } p \\
& \mid p q \mid p t \mid \text{exfalse } p \\
& \mid \text{catch}_\alpha p \mid \text{throw}_\alpha p
\end{aligned}$$

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, \dots range over a set of term variables, a, b, \dots over a set of proof variables, α, β, \dots over a set of continuation variables. The constructions $\lambda a.p$, $\text{case } p \text{ of } [a_1.p_1 \mid a_2.p_2]$, $\text{case } p \text{ of } [a_1.t_1 \mid a_2.t_2]$, $\text{split } p \text{ as } (a_1, a_2) \text{ in } q$, $\text{split } p \text{ as } (a_1, a_2) \text{ in } t$, $\text{dest } p \text{ as } (x, a) \text{ in } q$ and $\text{dest } p \text{ as } (x, a) \text{ in } t$ bind a, a_1 and a_2 . The constructions $\lambda x.p$, $\text{dest } p \text{ as } (x, a) \text{ in } q$ and $\text{dest } p \text{ as } (x, a) \text{ in } t$ bind x . The construction $\text{catch}_\alpha p$ 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 (case) or destruction of proofs (split and dest). Let us say also that the operators catch and throw implement classical reasoning. They are similar to the operators of same name in Nakano [25] or Crolard [26]. In terms of Parigot's λ -calculus [27], $\text{catch}_\alpha p$ is basically equivalent to $\mu\alpha.[\alpha]p$ and $\text{throw}_\alpha p$ to $\mu\delta.[\alpha]p$ for δ not occurring in p .

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

B. Operational semantics

We equip *dPL* 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:

$$V ::= a \mid \iota_i(V) \mid (V, V) \mid (t, V) \mid \lambda a.p \mid \lambda x.p \mid ()$$

To define the operational semantics of *dPL*, we also need to define the class of elementary call-by-value evaluation contexts:

$$\begin{aligned}
F[] & ::= \text{case } [] \text{ of } [a_1.p_1 \mid a_2.p_2] \\
& \mid \text{split } [] \text{ as } (a_1, a_2) \text{ in } q \\
& \mid \text{dest } [] \text{ as } (x, a) \text{ in } p \\
& \mid (\lambda a.p)[] \mid \text{prf}[] \\
& \mid [] q \mid [] t \mid \text{exfalse } [] \mid \text{throw}_\alpha [] \\
& \mid \iota_i([]) \mid ([], p) \mid (V, []) \mid (t, [])
\end{aligned}$$

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 $F[]$. Finally, we need to consider the subclass of neutral negative elimination that is described by the following grammar:

$$p_e ::= x \mid p_e t \mid p_e q$$

We can now define reduction in *dPL* as the congruent closure of the following reductions on proofs and terms:

$$\begin{aligned}
(\lambda a.p)V & \triangleright p[V/a] \\
(\lambda x.p)t & \triangleright p[t/x] \\
\text{case } \iota_i(V) \text{ of } [a_1.p_1 \mid a_2.p_2] & \triangleright p_i[V/a_i] \\
\text{split } (V_1, V_2) \text{ as } (a_1, a_2) \text{ in } p & \triangleright p[V_1/a_1][V_2/a_2] \\
\text{dest } (u, V) \text{ as } (x, a) \text{ in } p & \triangleright p[u/x][V/a] \\
\text{prf } (u, V) & \triangleright V \\
\text{catch}_\alpha \text{throw}_\alpha p & \triangleright \text{catch}_\alpha p \\
\text{catch}_\alpha \text{catch}_\beta p & \triangleright \text{catch}_\alpha p[\alpha/\beta] \\
F[\text{exfalse } p] & \triangleright \text{exfalse } p \\
F[\text{throw}_\alpha p] & \triangleright \text{throw}_\alpha p \\
F[\text{catch}_\alpha p] & \triangleright \text{catch}_\alpha F[p/F[\alpha]] \\
F[(\lambda a.p)p_e] & \triangleright (\lambda a.F[p])p_e \\
F[\text{case } p_e \text{ of } [a_1.p_1 \mid a_2.p_2]] & \triangleright \text{case } p_e \text{ of } [a_1.F[p_1] \mid a_2.F[p_2]] \\
F[\text{split } p_e \text{ as } (a_1, a_2) \text{ in } q] & \triangleright \text{split } p_e \text{ as } (a_1, a_2) \text{ in } F[q] \\
F[\text{dest } p_e \text{ as } (x, a) \text{ in } p] & \triangleright \text{dest } p_e \text{ as } (x, a) \text{ in } F[p] \\
\text{wit } (t, p) & \triangleright t
\end{aligned}$$

where the substitutions $p[V/a]$, $p[u/x]$, $t[V/a]$ and $t[u/x]$ 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 $\text{throw}_\alpha q$ in p by $\text{throw}_\alpha F[q]$ (including the recursive replacements in q).

C. Formulae and inference rules

In *dPL*, we consider implication to be possibly dependent in its antecedent and use the notation $[a : A] \rightarrow B$

$\frac{(a : A) \in \Gamma}{\Gamma \vdash a : A}$ AXIOM	$\frac{\Gamma \vdash p : A \quad A \equiv B}{\Gamma \vdash p : B}$ CONV	$\frac{}{\Gamma \vdash () : \top}$ \top_I	$\frac{\Gamma \vdash p : \perp}{\Gamma \vdash \text{exfalse } p : C}$ \perp_E
$\frac{\Gamma \vdash p_1 : A_1 \quad \Gamma \vdash p_2 : A_2}{\Gamma \vdash (p_1, p_2) : A_1 \wedge A_2}$ \wedge_I		$\frac{\Gamma \vdash p : A_i}{\Gamma \vdash \iota_i(p) : A_1 \vee A_2}$ \vee_I^i	
$\frac{\Gamma \vdash p : A_1 \wedge A_2 \quad \Gamma, a_1 : A_1, a_2 : A_2 \vdash q : B[(a_1, a_2)/a] \quad a \notin FV(B) \text{ if } p \text{ not N-elimination-free} \quad a_1, a_2 \notin FV(B)}{\Gamma \vdash \text{split } p \text{ as } (a_1, a_2) \text{ in } q : B[p/a]}$ \wedge_E			
$\frac{\Gamma \vdash p : A_1 \vee A_2 \quad \Gamma, a_1 : A_1 \vdash p_1 : B[\iota_1(a_1)/a] \quad \Gamma, a_2 : A_2 \vdash p_2 : B[\iota_2(a_2)/a] \quad a \notin FV(B) \text{ if } p \text{ not N-elimination-free} \quad a_1, a_2 \notin FV(B)}{\Gamma \vdash \text{case } p \text{ of } [a_1.p_1 \mid a_2.p_2] : B[p/a]}$ \vee_E			
$\frac{\Gamma, a : A \vdash p : B}{\Gamma \vdash \lambda a.p : [a : A] \rightarrow B}$ \rightarrow_I		$\frac{\Gamma \vdash p : [a : A] \rightarrow B \quad \Gamma \vdash q : A \quad a \notin FV(B) \text{ if } q \text{ not N-elimination-free}}{\Gamma \vdash pq : B[q/a]}$ \rightarrow_E	
$\frac{\Gamma \vdash p : A \quad x \text{ fresh in } \Gamma}{\Gamma \vdash \lambda x.p : \forall x A}$ \forall_I		$\frac{\Gamma \vdash p : \forall x A}{\Gamma \vdash pt : A[t/x]}$ \forall_E	
$\frac{\Gamma \vdash p : A[t/x]}{\Gamma \vdash (t, p) : \exists x A}$ \exists_I		$\frac{\Gamma \vdash p : \exists x \quad p \text{ is N-elimination-free}}{\Gamma \vdash \text{prf } p : A[\text{wit } p/x]}$ \exists_E^{PRF}	
$\frac{\Gamma \vdash p : \exists x A \quad \Gamma, a : A \vdash q : B[(x, a)/b] \quad x \text{ fresh in } \Gamma \quad b \notin FV(B) \text{ if } p \text{ not N-elimination-free} \quad a \notin FV(B)}{\Gamma \vdash \text{dest } p \text{ as } (x, a) \text{ in } q : B[p/b]}$ \exists_E			
$\frac{\Gamma, \alpha : A^\perp \vdash p : A}{\Gamma \vdash \text{catch}_\alpha p : A}$ CATCH		$\frac{\Gamma \vdash p : A \quad (\alpha : A^\perp) \in \Gamma}{\Gamma \vdash \text{throw}_\alpha p : C}$ THROW	

Fig. 1. *dPL*: a classical predicate logic with strong existential

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 $[a : \exists x P(x)] \rightarrow P(\text{wit } a)$ ⁷. The syntax of formulae is otherwise standard:

$$A, B ::= P(\vec{t}) \mid \top \mid \perp \mid [a : A] \rightarrow B \mid A \vee B \mid A \wedge B \mid \forall x A \mid \exists x A$$

where 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 $A \rightarrow \perp$. In $\forall x A$ and $\exists x A$, x is bound and freely subject to renaming (so-called α -conversion). The formulae $\forall x A$ and $[a : A] \rightarrow B$ are called negative. All other kinds of formulae are called positive.

⁷We do not get extra logical strength from this design choice: we will eventually show that *dPL* is conservative over predicate logic.

When known not dependent, we might shorten the writing of $[a : A] \rightarrow B$ into the more conventional $A \rightarrow B$.

Reduction on terms extends canonically to formulae via the atoms: we write $A \triangleright B$ if for some term occurring in A , reducing this term yields B . We write $A \triangleright^* B$ for the reflexive-transitive closure of \triangleright and $A \equiv B$ for its reflexive-symmetric-transitive closure.

Contexts of formulae, written Γ , are defined by:

$$\Gamma ::= \emptyset \mid \Gamma, a : A \mid \Gamma, \alpha : A^\perp$$

where $a : A$ stands for an assumption of A and $\alpha : A^\perp$ for an assumption of the refutation of A (with the objective of obtaining a proof by contradiction). It is assumed that assumptions have distinct variable names and we write $\text{Dom}(\Gamma)$ for the set of names a and α thus declared in Γ .

The inference rules of dPL are given in Figure 1. The main difference with ordinary predicate logic is the strong elimination rule of existential quantification and the appropriate support for formulae depending on proofs.

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 $\text{prf } p$, $\iota_i(p)$, (p_1, p_2) , (t, p) , $\text{case } a \text{ of } [a_1.p_1 | a_2.p_2]$, $\text{dest } q \text{ as } (x, a) \text{ in } p$ and $\text{split } q \text{ as } (a_1, a_2) \text{ in } p$ are N-elimination-free.

Otherwise said, in N-elimination-free proofs, expressions of the form pq , pt , $\text{exfalse } p$, $\text{catch}_\alpha p$ or $\text{throw}_\alpha p$ can only occur in the body of a λx or of a λa .

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

Let us state a few lemmas about dPL .

Lemma 1: If $A[p/a] \equiv A'[p/a]$ for p belonging to the grammar $h ::= c \mid ht \mid hq$, then $A[r/a] \equiv A'[r/a]$ for any proof r .

Proof: Easy by induction on the length of a derivation of the equality, since the form p forces it to be passive in any of the reduction rules. ■

Lemma 2: If $\Gamma, a : A[q/b] \vdash p : B$ and $\Gamma \vdash q : B$ for q obtained from some variable c by using \rightarrow_E or \forall_E , then, for all r such that $\Gamma \vdash q : B$, there is some B' possibly depending on b such that B is $B'[q/b]$ and $\Gamma, a : A[r/b] \vdash p : B'[r/b]$.

Proof: By induction on the derivation of $\Gamma, a : A[q/b] \vdash p : B$. If p is a variable, then the result holds by taking B' to be A . If conv is used, we apply Lemma 1. The other cases are by induction hypothesis. ■

We are now ready to state the main properties of dPL .

Theorem 3 (Subject reduction): If $\Gamma \vdash p : A$ and $p \triangleright q$ then $\Gamma \vdash q : A$.

Proof: Proof variables are only substituted by values and evaluation context variables occur in N-elimination-free proofs only in the scope of a λx or λa , so that the N-elimination-free condition is stable by reduction. Then, this N-elimination-free condition ensures that a catch is never surrounded by a context $F[\]$ whose type depends on the contents of the hole (this comes from the N-elimination-free restrictions on the rule \wedge_E , \forall_E , \rightarrow_E , \exists_E , prf when these ones are dependent). In particular, the reduction of $F[\text{catch}_\alpha p]$ is well-typed.

That the three rules allowing a context to traverse a positive elimination rule or a $(\lambda a.p)q$ redex (commutative rules) are well-typed deserve a bit of explanation. Let us

consider for instance the commutation of some context F over a proof obtained by using \forall_E on a proof p of $A_1 \vee A_2$ and proofs p_1 and p_2 of $A[\iota_1(a_1)/a]$ and $A[\iota_2(a_2)/a]$ respectively. If the context is non dependent and has type B , its commutation with \forall_E has just to be done using B without isolating any dependency a in B . If otherwise F has a dependent type $B([\])$ with hole $[\]$ of type $A[p/a]$ (this can happen with the first five kinds of elementary evaluation contexts), then, we first have to apply Lemma 2 to duplicate the derivation of $F[\]$ into derivations of $B([\])$ with hole now of type $A[\iota_1(a_1)/a]$ and $A[\iota_2(a_2)/a]$ respectively. Then we build derivations $F[p_1]$ $F[p_2]$ that we respectively see as derivations of $B[\text{case } \iota_1(a_1) \text{ of } [a_1.p_1 | a_2.p_2]/b]$ and $B[\text{case } \iota_2(a_2) \text{ of } [a_1.p_1 | a_2.p_2]/b]$ thanks to conversion. It remains then to apply \forall_E so that the reduct is a derivation of $B[\text{case } p \text{ of } [a_1.p_1 | a_2.p_2]/b]$ has expected (see Figure 2). ■

Theorem 4 (Normalisation): If $\Gamma \vdash p : A$ then p is normalisable.

Proof: If one erases the first-order part of the formulae, one obtains a simply-typed $\lambda\mu$ -calculus, whose call-by-value evaluation strategy can be shown terminating by continuation-passing-style embedding into simply-typed λ -calculus. ■

At the end, dependent predicate logic is not stronger than ordinary predicate logic:

Theorem 5 (Conservativity): If $\Gamma \vdash p : A$ and wit does not occur in A , then $\Gamma \vdash q : A$ for some q such that prf does not occur in q and the rule conv is not used.

Proof: (sketch) The set of reduction rules has commutative cuts and is complete for eliminating all cuts and ensuring the subformula property up to the use of conv of prf and of the use of terms with wit in \forall_E and \exists_I . Then, by joining the conv rules, replacing prf by \exists_E and free occurrences of wit by an arbitrary variable, we can make that internal occurrences of wit disappear as soon as the ending judgment is wit -free. ■

Remark: We chose a “multiplicative” elimination rule for conjunction what fits better with call-by-value semantics and with the view of conjunction as a positive connective (a tensor in linear logic [28]). Taking instead an “additive” elimination rule of the form $\pi_1 p$ and $\pi_2 p$ would however work the same, using $\pi_1([\])$ and $\pi_2([\])$ in the definition of F instead of $\text{split} [\]$ as (a_1, a_2) in p .

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

$$\boxed{
\begin{array}{c}
\Gamma, a_i : A_i \vdash p_i : A[u_i(a_i)/a] \\
\hline
\Gamma, a_i : A_i \vdash F[p_i] : B[p_i/b] \\
\hline
\Gamma \vdash p : A_1 \vee A_2 \quad \Gamma, a_i : A_i \vdash F[p_i] : B[\text{case } u_i(a_i) \text{ of } [a_1.p_1 \mid a_2.p_2]/b] \quad \text{CONV (for } i = 1 \text{ and } i = 2) \\
\hline
\Gamma \vdash \text{case } p \text{ of } [a_1.F[p_1] \mid a_2.F[p_2]] : B[\text{case } p \text{ of } [a_1.p_1 \mid a_2.p_2]/b] \quad \vee_E
\end{array}
}$$

Fig. 2. An example of typing commutative cuts in the proof of Theorem 3

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 convenient for proving the axioms of countable choice and dependent choice.

A. Proofs and terms

Terms are extended with natural numbers, a recursor and λ -calculus while proofs are extended with induction, constructors for equality and constructors for coinductive formulae; for convenience, we also add a `let`:

$$\begin{array}{l}
t, u ::= x \mid 0 \mid S(t) \mid \text{rec } t \text{ of } [t \mid (x, y).t] \\
\quad \mid \lambda x.t \mid tt \mid \text{wit } p \\
p, q ::= a \mid u_i(p) \mid (p_1, p_2) \mid (t, p) \mid \lambda a.p \mid \lambda x.p \\
\quad \mid \text{case } p \text{ of } [a_1.p_1 \mid a_2.p_2] \\
\quad \mid \text{dest } p \text{ as } (x, a) \text{ in } q \mid \text{prf } p \\
\quad \mid \text{split } p \text{ as } (x, a) \text{ in } q \\
\quad \mid pq \mid pt \mid \text{exfalse } p \\
\quad \mid \text{refl} \mid \text{subst } pq \\
\quad \mid \text{ind } t \text{ of } [p \mid (x, a).q] \\
\quad \mid \text{cofix}_{bx}^t p \\
\quad \mid \text{catch}_\alpha p \mid \text{throw}_\alpha p \\
\quad \mid \text{let } a = p \text{ in } q
\end{array}$$

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

Because of corecursion, we have potentially infinite values and we do not want any longer to fully reduce proofs using a call-by-value semantics. Therefore, we move to a more incremental reduction semantics which is lazy on the evaluation of corecursive values. Lazy evaluation requires to introduces a new kind of context,

written D , which consists in pending delayed computation of cofixpoints. Evaluation contexts are then defined as follows:

$$\begin{array}{l}
F[] ::= \text{case } [] \text{ of } [a_1.p_1 \mid a_2.p_2] \\
\quad \mid \text{split } [] \text{ as } (a_1, a_2) \text{ in } q \\
\quad \mid \text{dest } [] \text{ as } (x, a) \text{ in } p \\
\quad \mid [] q \mid \text{let } a = [] \text{ in } q \mid \text{prf } [] \\
\quad \mid [] t \mid \text{exfalse } [] \mid \text{throw}_\alpha [] \\
\quad \mid \text{subst } [] p
\end{array}$$

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

The reduction rules, shown in Figure 3 are extended with rules for simplifying recursion, β -reduction, incremental substitution of values and lazy evaluation of cofixpoints. Again, we write \triangleright for the reflexive-transitive closure of \triangleright . We write \equiv for the reflexive-symmetric-transitive closure of \triangleright .

C. Types, formulae and inference rules

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

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

Formulas are as in dPL but with only equality statements over terms of type \mathbb{N} as atoms and including coinductive formulae:

$$\begin{array}{l}
A, B ::= t = u \mid [a : A] \rightarrow B \mid A \vee B \mid A \wedge B \mid \perp \mid \top \\
\quad \mid \forall x^T A \mid \exists x^T A \mid \nu_{fx}^t A
\end{array}$$

where $\nu_{fx}^t A$ 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).

As in dPL , 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 equality⁸ and coinductive

⁸See e.g. Allali [29] for such a presentation of arithmetic.

$(\lambda a.q) p$	\triangleright let $a = p$ in q
let $a = ()$ in q	\triangleright $q[()/a]$
let $a = (p_1, p_2)$ in q	\triangleright let $a_1 = p_1$ in let $a_2 = p_2$ in $q[(a_1, a_2)/a]$
let $a = \iota_i(p)$ in q	\triangleright let $b = p$ in $q[\iota_i(b)/a]$
let $a = (t, p)$ in q	\triangleright let $b = p$ in $q[(t, b)/a]$
$(\lambda x.p) t$	\triangleright $p[t/x]$
case $\iota_i(p)$ of $[a_1.p_1 \mid a_2.p_2]$	\triangleright let $a_i = p$ in p_i
dest (t, p) as (x, a) in q	\triangleright let $a = p$ in $q[t/x]$
split (p_1, p_2) as (a_1, a_2) in q	\triangleright let $a_1 = p_1$ in let $a_2 = p_2$ in q
prf (t, p)	\triangleright p
subst refl p	\triangleright p
ind 0 of $[p \mid (x, a).q]$	\triangleright p
ind $S(t)$ of $[p \mid (x, a).q]$	\triangleright $q[t/x][\text{ind } t \text{ of } [p \mid (x, a).q]/a]$
case cofix $_{bx}^t p$ of $[a_1.p_1 \mid a_2.p_2]$	\triangleright let $c = \text{cofix}_{bx}^t p$ in case c of $[a_1.p_1 \mid a_2.p_2]$
dest cofix $_{bx}^t p$ as (x, a) in q	\triangleright let $c = \text{cofix}_{bx}^t p$ in dest c as (x, a) in q
split cofix $_{bx}^t p$ as (a_1, a_2) in q	\triangleright let $c = \text{cofix}_{bx}^t p$ in split c as (a_1, a_2) in q
let $a = \text{cofix}_{bx}^t p$ in $D[a]$	\triangleright let $a = p[\lambda y.\text{cofix}_{bx}^y p/b][t/x]$ in $D[a]$
$F[\text{let } a = \text{cofix}_{bx}^t p \text{ in } q]$	\triangleright let $a = \text{cofix}_{bx}^t p$ in $F[q]$
$F[\text{exfalse } p]$	\triangleright exfalse p
$F[\text{throw}_\alpha p]$	\triangleright throw $_\alpha p$
$F[\text{catch}_\alpha p]$	\triangleright catch $_\alpha p[F/\alpha]$
catch $_\alpha$ throw $_\alpha p$	\triangleright catch $_\alpha p$
wit (t, p)	\triangleright t
$(\lambda x.t) u$	\triangleright $t[u/x]$
rec 0 of $[t_0 \mid (x, y).t_S]$	\triangleright t_0
rec $S(t)$ of $[t_0 \mid (x, y).t_S]$	\triangleright $t_S[t/x][\text{rec } t \text{ of } [t_0 \mid (x, y).t_S]/y]$

Fig. 3. Reduction rules on terms and proofs of dPA^ω

formulae unfolding:

$0 = 0$	\triangleright \top
$0 = S(u)$	\triangleright \perp
$S(t) = 0$	\triangleright \perp
$S(t) = S(u)$	\triangleright $t = u$
$\nu_{fx}^t A$	\triangleright $A[t/x][\nu_{fx}^y A/f(y) = 0]$

As before, we write \equiv for the resulting⁹ 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 the context. Hence, contexts are now defined by:

$$\Gamma ::= \emptyset \mid \Gamma, x : T \mid \Gamma, a : A \mid \Gamma, \alpha : A^\perp$$

⁹Unfolding 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.

where $x : T$ stands for the declaration of a variable of type T .

Inference rules are given in Figure 4 with the typing rules in the bottom. As the operational semantics of dPA^ω does not include propagating a context F through the elimination rules of positive connectives, as it was the case in dPL , we can simplify the formulation of these elimination rules and avoid considering updating the possible dependencies (this is visible in Figure 4 only for \exists_E but the same simplification could be done for \wedge_E and \vee_E too).

In the rule ν_I , the function f is said to be positive in A if A is built from atoms¹⁰ $f(t) = 0$ 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 5 where out a implements the reduction rule for $\nu_{fx}^t A$. However,

¹⁰Since we have symbols for functions and not for predicates, we use expressions of the form $f(t) = 0$ to represent arbitrary atoms.

The rules `AXIOM`, `CONV`, `⊤I`, `⊥E`, `∧I`, `∧E`, `∨Ii`, `∨E`, `→I`, `→E`, `CATCH`, `THROW` of Figure 1 and the following rules:

$$\begin{array}{c}
\frac{\Gamma, x : T \vdash p : A}{\Gamma \vdash \lambda x.p : \forall x^T A} \forall_I \quad \frac{\Gamma \vdash p : \forall x^T A \quad \Gamma \vdash t : T}{\Gamma \vdash pt : A[t/x]} \forall_E \\
\\
\frac{\Gamma \vdash p : A[t/x] \quad \Gamma \vdash t : T}{\Gamma \vdash (t, p) : \exists x^T A} \exists_I \quad \frac{\Gamma \vdash p : \exists x^T A \quad \Gamma, x : T, a : A \vdash q : B}{\Gamma \vdash \text{dest } p \text{ as } (x, a) \text{ in } q : B} \exists_E \\
\\
\frac{\Gamma \vdash p : \exists x^T A \quad p \text{ is N-elimination-free}}{\Gamma \vdash \text{prf } p : A[\text{wit } p/x]} \exists_E^{\text{PRF}} \\
\\
\frac{\Gamma \vdash t : \mathbb{N}}{\Gamma \vdash \text{refl} : t = t} \text{REFL} \quad \frac{\Gamma \vdash p : t = u \quad \Gamma \vdash q : A[t/x] \quad x \notin \text{Dom}(\Gamma)}{\Gamma \vdash \text{subst } pq : A[u/x]} \text{SUBST} \\
\\
\frac{\Gamma \vdash t : \mathbb{N} \quad \Gamma \vdash p : A[0/x] \quad \Gamma, x : T, a : A \vdash q : A[S(x)/x]}{\Gamma \vdash \text{ind } t \text{ of } [p | (x, a).q] : A[t/x]} \text{IND} \\
\\
\frac{\Gamma \vdash p : A \quad \Gamma, a : A \vdash q : B \quad a \notin \text{FV}(B) \text{ if } p \text{ not N-elimination-free}}{\Gamma \vdash \text{let } a = p \text{ in } q : B[p/a]} \text{CUT} \\
\\
\frac{\Gamma \vdash t : T \quad \Gamma, f : T \rightarrow \mathbb{N}, x : T, b : \forall y f(y) = 0 \vdash p : A \quad f \text{ positive in } A}{\Gamma \vdash \text{cofix}_{bx}^t p : \nu_{fx}^t A} \nu_I
\end{array}$$

$$\begin{array}{c}
\frac{(x : T) \in \Gamma}{\Gamma \vdash x : T} \quad \frac{\Gamma, x : U \vdash t : T}{\Gamma \vdash \lambda x.t : U \rightarrow T} \quad \frac{\Gamma \vdash t : U \rightarrow T \quad \Gamma \vdash u : U}{\Gamma \vdash tu : T} \quad \frac{}{\Gamma \vdash 0 : \mathbb{N}} \quad \frac{\Gamma \vdash t : \mathbb{N}}{\Gamma \vdash S(t) : \mathbb{N}} \\
\\
\frac{\Gamma \vdash t : \mathbb{N} \quad \Gamma \vdash t_0 : U \quad \Gamma, x : \mathbb{N}, y : U \vdash t_S : U}{\Gamma \vdash \text{rec } t \text{ of } [t_0 | (x, y).t_S] : U} \quad \frac{\Gamma \vdash p : \exists x^T A \quad p \text{ is N-elimination-free}}{\Gamma \vdash \text{wit } p : T} \exists_E^{\text{WIT}}
\end{array}$$

Fig. 4. dPA^ω : Classical arithmetic in finite types with strong existential

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.

The condition of being N-elimination-free is now 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 $\text{prf } p, \iota_i(p), (p_1, p_2), (t, p), \text{ind } t \text{ of } [p_1 | (x, a).p_2], \text{let } a = p \text{ in } q, \text{refl}, \text{subst } pq, \text{case } a \text{ of } [a_1.p_1 | a_2.p_2], \text{dest } q \text{ as } (x, a) \text{ in } p \text{ and split } q \text{ as } (a_1, a_2) \text{ in } p$ are N-elimination-free.

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

$v_{fx}^t A$	$\triangleq \exists f (f(t) = 0 \wedge \forall x (f(x) = 0 \rightarrow A))$
$\text{cofix}_{bx}^t p$	$\triangleq (\lambda x.0, (\text{refl}, \lambda x.\lambda b.p))$
$\text{out } a$	$\triangleq \text{dest } a \text{ as } (f, b) \text{ in split } b \text{ as } (c, d) \text{ in } \text{mon}_{f,d}^{A[f/f][t/x]}(d t c)$
where, for $d : \forall x f(x) = 0 \rightarrow A$, and $a : B$, $\text{mon}_{f,d}^B a$ is defined inductively:	
$\text{mon}_{f,d=0}^B a$	$: B[v_{fx}^y A / f(y) = 0]$
$\text{mon}_{f,d}^{f(t)=0} a$	$\triangleq (f, (a, d))$
$\text{mon}_{f,d}^{B_1 \wedge B_2} a$	$\triangleq \text{split } a \text{ as } (a_1, a_2) \text{ in } (\text{mon}_{f,d}^{B_1} a_1, \text{mon}_{f,d}^{B_2} a_2)$
$\text{mon}_{f,d}^{B_1 \vee B_2} a$	$\triangleq \text{case } a \text{ of } [a_1.\text{mon}_{f,d}^{B_1} a_1 \mid a_2.\text{mon}_{f,d}^{B_2} a_2]$
$\text{mon}_{f,d}^{\lambda x B} a$	$\triangleq \text{dest } a \text{ as } (x, a) \text{ in } (x, \text{mon}_{f,d}^B a)$
$\text{mon}_{f,d}^{g, C} a$	$\triangleq \text{dest } a \text{ as } (g, b) \text{ in split } b \text{ as } (c', d') \text{ in } (g, (c', \lambda x.\lambda a.\text{mon}_{f,d}^C (d' x a)))$

Fig. 5. Derivability of introduction and reduction of coinductive formula

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 choice¹¹ 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^ω .

Theorem 6 (Subject reduction): If $\Gamma \vdash p : A$ and $p \triangleright q$ then $\Gamma \vdash q : A$.

Theorem 7 (Normalisation): If $\Gamma \vdash p : A$ then p is normalisable.

Proof: (sketch) We follow the ideas of [30]. If there is an infinite reduction sequence, then it necessarily explore an infinite branch of some coinductive proof since otherwise, we could replace the coinductive proofs by finite approximations of them and reduce the termination to the call-by-value termination of PA^ω . If some coinductive proof is explored infinitely many times, then, we can extract a subsequence that does not “backtrack” any longer in nodes of the coinductive proofs and that explores the same branch of a tree infinitely many often. This subsequence is explored in a row by the same proof which would itself have an infinite branch in its interpretation in infinitary logic what is not possible since proofs are inductive (and well-founded). ■

Theorem 8 (Conservativity, first version): If A is $\forall\text{-}\rightarrow\text{-}\nu\text{-wit}$ -free then $\vdash p : A$ in dPA^ω implies $\vdash p : A$

¹¹I.e. reification of functional relations into functions.

in HA^ω .

Proof: The choice of rules we have makes that any closed proof of a $\forall\text{-}\rightarrow\text{-}\nu\text{-wit}$ -free formula eventually produces an expression $D[\text{catch}_{\alpha_1} \dots \text{catch}_{\alpha_n} p]$ where p starts with a constructor. By iterating the evaluation, one eventually gets $D[\text{catch}_{\alpha_1} \dots \text{catch}_{\alpha_n} V]$ for some value V which contains no $\lambda x.p$ nor $\lambda a.p$ (by the $\forall\text{-}\rightarrow\text{-}\nu\text{-wit}$ -free constraint on A). The context D and the catch_{α} 's can now be removed, since they do not bind any variable in V . The resulting proof is in HA^ω . ■

In arithmetic, any Σ_1^0 -formula is equivalent to a $\forall\text{-}\rightarrow\text{-}\nu\text{-wit}$ -free formula. Hence we have:

Theorem 9 (Conservativity, second version): If A is Σ_1^0 then $\vdash p : A$ in dPA^ω implies $\vdash p : A$ in HA^ω .

This of course implies consistency:

Theorem 10 (Consistency): $\not\vdash p : \perp$ 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 $A(g(0)) \wedge A(g(1)) \wedge A(g(2)) \dots$ 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 $A(0) \wedge A(1) \wedge A(2) \dots$, so we

consider the coinductive conjunction $R_C(n) \triangleq \nu_{fx}^n (A(x) \wedge f(S(x)) = 0)$. The proof is now direct:

$$\begin{aligned} AC_{\mathbb{N}} &\triangleq \lambda a. \text{let } b = \text{cofix}_{bn}^0 (an, b(Sn)) \text{ in} \\ &\quad (\lambda n. \text{wit}(\text{nth}_C n b), \lambda n. \text{prf}(\text{nth}_C n b)) \\ &: \forall n \exists y P(n, y) \rightarrow \exists f \forall n P(n, f(n)) \end{aligned}$$

where

$$\begin{aligned} \text{nth}_C n &: R_C(0) \rightarrow R_C(n) \\ \text{nth}_C n &\triangleq \lambda b. \pi_1(\text{ind } n \text{ of } [b | (m, c). \pi_2(c)]) \end{aligned}$$

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 $R_D(z) \triangleq \nu_{fx}^z \exists y (P(x, y) \wedge f(y) = 0)$. The proof is now direct:

$$\begin{aligned} DC &\triangleq \lambda a. \lambda x_0. \text{let } b = s a x_0 \text{ in} \\ &\quad (\lambda n. \text{wit}(\text{nth}_D n (x_0, b)), \\ &\quad (\text{refl}, \lambda n. \pi_1(\text{prf}(\text{prf}(\text{nth}_D n (x_0, b)))))) \\ &: \forall x \exists y P(x, y) \rightarrow \\ &\quad \forall x_0 \exists f (f(0) = x_0 \wedge \forall n P(f(n), f(S(n)))) \end{aligned}$$

where

$$\begin{aligned} \text{nth}_D n &: \exists x R_D(x) \rightarrow \exists x R_D(x) \\ \text{nth}_D n &\triangleq \lambda b. \text{ind } n \text{ of} \\ &\quad [b | (m, c). (\text{wit}(\text{prf } c), \pi_2(\text{prf}(\text{prf } c)))] \\ s a x &: R_D(x) \\ s a x &\triangleq \text{cofix}_{bn}^x (\text{dest } an \text{ as } (y, c) \text{ in } (y, (c, by))) \end{aligned}$$

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:

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

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 :

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

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

$$BI: \forall f \exists n B(f_n) \rightarrow \forall P \left(\begin{array}{l} \forall l (B(l) \rightarrow P(l)) \wedge \\ \forall l (\forall x P(l \star x) \rightarrow P(l)) \end{array} \right) \rightarrow P(\langle \rangle)$$

Let us consider a contrapositive variant of BI

$$BI_c: \nu_{gl}^\diamond (\neg B(l) \wedge \exists x g(l \star x) = 0) \rightarrow \exists f \forall n \neg B(f_n)$$

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:

$$\begin{aligned} BI_c^+ &\triangleq \lambda a. (\lambda n. \text{wit}(\pi_2(\text{prf}(\text{nth}_{BI} n (\langle \rangle, a))))), \\ &\quad \lambda n. (\text{wit}(\text{nth}_{BI} n (\langle \rangle, a)), \\ &\quad (\pi_1(\text{prf}(\text{nth}_{BI} n (\langle \rangle, a))), e a n)) \\ &: R_{BI}(\langle \rangle) \rightarrow \exists f \forall n \exists l (\neg B(l) \wedge l = f_n) \end{aligned}$$

where

$$\begin{aligned} \text{nth}_{BI} n &: \exists l R_{BI}(l) \rightarrow \exists l R_{BI}(l) \\ \text{nth}_{BI} n &\triangleq \lambda b. \text{ind } n \text{ of} \\ &\quad [b | (m, c). (\text{wit } c \star \text{wit}(\pi_2(\text{prf } c)), \\ &\quad \text{prf}(\pi_2(\text{prf } c)))] \\ e a n &: \text{wit}(\text{nth}_{BI} n (\langle \rangle, a)) = \\ &\quad (\lambda n. \text{wit}(\pi_2(\text{prf}(\text{nth}_{BI} n (\langle \rangle, a))))_n \\ e a n &\triangleq \text{ind } n \text{ of } [\text{refl} | (m, c). \text{subst } c \text{ refl}] \end{aligned}$$

from which BI_c directly follows. Then, from BI_c , we get the following weaker form of BI :

$$\begin{aligned} \forall f \exists n B(f_n) &\rightarrow \\ \forall g \left[\left(\begin{array}{l} \forall l (B(l) \rightarrow g(l) = 0) \wedge \\ \forall l (\forall x g(l \star x) = 0 \rightarrow g(l) = 0) \end{array} \right) \rightarrow g(\langle \rangle) = 0 \right] \end{aligned}$$

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) *Using explicit marks of classical reasoning to recover the full intuitionistic axiom of choice in dPA^ω* : The restriction we gave on the strong existential elimination rule is related to linearity. We imposed it by referring only to proofs of a positive formula and by enforcing a call-by-value discipline of evaluation. Our criterion rejects the proof of AC_A given in the introduction, even though it would be computationally sound when the proofs of $\exists y^B P(x, y)$ are intuitionistic. The coinductive conjunction detour is not possible either in the general case since the domain A is not necessarily countable. We believe that in a logic with explicit marks of non-linearity, such as linear logic [28], the criterion for accepting strong existential elimination could be made more precise so that the full axiom of choice over linear predicate is derivable.

g) *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 [31], [32]. 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.

h) *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 [7]. 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 [7], 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 [7], the stream has the same size as the number of n 's for which a proof of $\exists y P(n, y)$ is needed.

i) *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:

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

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

$$AC_{\mathbb{N}} : \forall n^{\mathbb{N}} \exists Y^{\mathbb{N} \rightarrow \star} P(n, Y) \rightarrow \exists Z^{\mathbb{N} \rightarrow \mathbb{N} \rightarrow \star} \forall n^{\mathbb{N}} P(n, Zn)$$

that we discuss in the next paragraph.

j) *Comparison with Krivine's realiser of the axiom of countable choice:* Krivine [10] realises the axiom of countable choice¹² 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. We shall now reinterpret Krivine's realiser as a proof in (an ad hoc extension of) the non-dependent version PA_2 of dPA_2 and show that the addition by Krivine of a quote operator on top of classical reasoning allows to simulate a form of call-by-value reduction (in the sense of sharing the values originating from the same initial proof) out of call-by-name.

Recall that the evaluation order problem with the axiom of choice is that whenever a classical proof $\text{catch}_{\alpha}.p$ of $\exists Y P(x, Y)$ is used several times in some evaluation context E , then, each time E is bound to α and dispatched around towards the different intuitionistic (though nested) subproofs (Y_i, p_i) of $\exists Y P(x, Y)$, all references to this proof inside each copy of E itself must refer to the very same (Y_i, p_i) subproof.

The quote operator that Krivine introduced bijectively reifies a proof p into a natural number $\lfloor p \rfloor$. The role of the quote operator is to keep track, by a unique identifier, of every time a subproof (Y_i, p_i) of $\exists Y P(x, Y)$ starts to interact with E so that other references to the initial proof $\text{catch}_{\alpha}.p$ of $\exists Y P(x, Y)$ in this copy of E , which at some time can themselves start an interaction with some of the (Y_i, p_i) and subcontext of E can check, at the time of having to interact one against the other, that they have the same identifier and then safely interact together in the same (Y_i, p_i) component, and, if not, restart the computation of the component with highest unique identifier with the data provided by the other component so that if ever the same interaction between subcomponents arrives again, it behaves correctly.

Schematically, here is the representation of what happens in the process of standard head evaluation of an interaction involving Krivine's realiser. Let E_i and E_j both be contexts expecting a proof of $\exists Y P(x, Y)$. Let E'_i be a subcontext part of p_i and p'_j a subproof part of p_j .

¹²In practise, Krivine realises the axiom

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

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

They have to interact together, one being of type $V_i(x, t)$ and the other of type $V_j(x, t)$. Let n_i be the identifier that reified E_i and n_j the identifier that reified E_j :

$$\dots E_i[(V_i, p_i)] \triangleright \dots \triangleright E_j[(V_j, p_j)] \triangleright \dots \triangleright E'_i[p'_j] \triangleright ?$$

If n_i and n_j are the same, V_i and V_j are the same and the interaction can continue:

$$\dots E_i[(V_i, p_i)] \triangleright \dots \triangleright E_j[(V_j, p_j)] \triangleright \dots \triangleright E'_i[p'_j] \triangleright \dots$$

If $n_i < n_j$ then, the computation that generated n_j restarts with the data provided at the time of n_i :

$$\dots E_i[(V_i, p_i)] \triangleright \dots \triangleright E_j[(V_i, p_i)] \triangleright \dots$$

If otherwise $n_i > n_j$ then, the computation that generated n_i restarts with the data provided at the time of n_j :

$$\dots E_i[(V_j, p_j)] \triangleright \dots$$

Of course, in these later cases, the computation might globally be much longer, but since natural numbers are well-founded, such backtracks will be done only a finite number of time.

Morally, what has to be checked in the interaction above is that the types match when a context whose hole has type V_i interacts with a term of type V_j and numerical identifiers are used because the V_k are types and hence not comparable as types. If the language of realisers had carried the types, one could have simply taken the code of V_k as identifier but since the language of realisers for Krivine's classical realisability drops the instances of quantifiers, something else has to be done. What Krivine uses is an operator χ intended to be called each time an interaction $E[(V_i, p_i)]$ (in fact $E[p_i]$ in the language of classical realisability) happens and whose purpose is to quantify E into a number n usable in place of V_i (so that in fact the interaction has the form $E[(n, p_i)]$ in the language of classical realisability). Otherwise said, translated¹³ into PA_2 , the operator χ obeys the following inference rule:

$$\frac{\Gamma \vdash p : \exists X P(X)}{\Gamma \vdash \chi p : \exists n P(\Phi_p(n))}$$

where Φ_p is an adequate choice function.

In Krivine's classical realisability, only closed expressions of type \perp (but for a possibly inhabited type \perp) are

¹³We apply two changes for moving from Krivine's classical second-order arithmetic to PA_2 : quantification over objects that satisfy the induction property are replaced by quantification over natural numbers (using the native induction scheme in place of the property asserting that the object satisfies induction); existential types, which are defined by their second-order encoding in classical second-order arithmetic, are taken primitive in PA_2 . Moreover, as Krivine reasons on universal quantification, what we write Φ is $\neg\Phi$ in [10].

reduced. Otherwise said, only operational rules of the form $E[p] \triangleright E'[p']$ are considered (there is no consideration of these expressions as part of a bigger context). The operational reduction rule for χ , reformulated in our framework, is $E[\chi(X, p)] \triangleright E[(\llbracket E \rrbracket, p)]$ where, for subject reduction, $\Phi_p(\llbracket E \rrbracket)$ gets interpreted by X in the derivation of E . In particular, this requires that no two similar contexts can have a hole typable by two distinct inconsistent types.

The quote operator is not the only ingredient of Krivine's realiser. Classical logic, i.e. control, is also needed to set up fallbacks allowing to restart past interactions involving values with high identifier with new values having smaller identifier whenever a mismatch occurs in an interaction between two independent parts of a common initial proof of $\exists Y P(x, Y)$ as was exemplified in the standard reduction sequences drawn a few paragraphs above. The fallbacks are then built by capturing the initial context with a control operator (catch_α) and by using well-founded recursion (below wf of which we omit the standard proof) to reinstall this context (throw_α) whenever a smaller number is found.

Finally, there is a third technical argument: To check if the interaction between independent parts of a common initial proof of $\exists Y P(x, Y)$ are consistent, a test of the unique identifier has to be inserted around each component of the initial proof liable to interact with another component of the same proof, i.e. at each component of type $Y(t)$ where the Y is not necessarily the same on both sides of the interaction. This is obtained by a recursive descent through the proof of $\exists Y P(x, Y)$, using extensionality (function \uparrow below of which we only list a few cases of the definition).

At the end, rephrased as a constructive proof, and leaving implicit the proofs related to $p < n$, $n < p$ and $p = n$, Krivine's realiser of the axiom of countable choice corresponds to the following term:

$$\begin{aligned} AC_{\mathbb{N}} &\triangleq \lambda a.(U_p, \\ &\quad \lambda x.\text{dest } \chi(a x) \text{ as } (n, b) \text{ in } \text{catch}_\alpha \\ &\quad \quad \text{wf}_x(\lambda n'. \lambda f. \lambda b'. \text{throw}_\alpha(\uparrow_{P(x, Y)}^{n' f b'} b')) n b) \\ &: \forall x^{\mathbb{N}} \exists Y^{\mathbb{N} \rightarrow \star} P(x, Y) \\ &\rightarrow \exists U^{\mathbb{N} \rightarrow \mathbb{N} \rightarrow \star} \forall x^{\mathbb{N}} P(x, U(x)) \end{aligned}$$

where

$$\begin{aligned} V(x, n) &\triangleq \neg P(x, \Phi_P(x, n)) \\ Z(x, n) &\triangleq \forall m < n V(x, m) \rightarrow V(x, n) \\ U_P(x, y) &\triangleq \forall n \neg Z(x, n) \rightarrow \Phi_P(x, n, y) \end{aligned}$$

$$\mathbf{wf}_x \quad : \quad \forall n Z(x, n) \rightarrow \forall n V(x, n)$$

and for $f : \forall m < n V(x, m)$ and $b : P(x, \Phi_P(x, n))$

$$\begin{aligned} \uparrow_A^{nfb} &: A(\Phi_P(x, n)) \rightarrow A(U_P(x)) \\ \uparrow_{Y(t)}^{nfb} c &\triangleq \lambda n'. \lambda k. \mathbf{if} \ n = n' \ \mathbf{then} \ c \ \mathbf{else} \\ &\quad k \ \lambda f'. \lambda b'. \mathbf{if} \ n' < n \ \mathbf{then} \ f \ n' \ b' \\ &\quad \quad \quad \mathbf{else} \ f' \ n \ b \\ \uparrow_{A \wedge B}^{nfb} c &\triangleq (\uparrow_A^{nfb} (\pi_1 c), \uparrow_B^{nfb} (\pi_2 c)) \\ \uparrow_{A \rightarrow B}^{nfb} c &\triangleq \lambda a. (\uparrow_B^{nfb} (c (\uparrow_A^{nfb} a))) \\ \dots & \end{aligned}$$

$$\begin{aligned} \downarrow_A^{nfb} &: A(U_P(x)) \rightarrow A(\Phi_P(x, n)) \\ \downarrow_{Y(t)}^{nfb} c &\triangleq c \ n \ \lambda k. k \ f \ b \\ \downarrow_{A \wedge B}^{nfb} c &\triangleq (\downarrow_A^{nfb} (\pi_1 c), \downarrow_B^{nfb} (\pi_2 c)) \\ \downarrow_{A \rightarrow B}^{nfb} c &\triangleq \lambda a. (\downarrow_B^{nfb} (c (\uparrow_A^{nfb} a))) \\ \dots & \end{aligned}$$

By dualising Krivine's reasoning [10], it seems that χ might reify its argument instead of its evaluation context, what would allow the rule for χ to be usable as a local reduction rule and not only as an operational rule. In a framework that keeps a trace of the instances of quantifiers, the reduction rule for χ could be as well $\chi(U, p) \triangleright ([U], p)$, for U closed predicate, what means that χ and Φ can be defined using (unrestricted) strong existential elimination as follows:

$$\begin{aligned} \chi p &\triangleq ([\mathbf{wit} \ p], \mathbf{prf} \ p) \\ \Phi(n) &\triangleq [n] \end{aligned}$$

where the quote function assigns the same n to all terms equal with respect to \equiv and where $[n]$ reverts the effect of the quote function¹⁴. This leaves room for another way to combine strong existential elimination and classical logic but keeping this time (unrestricted) strong existential elimination.

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

¹⁴Terms have to be closed so as the quote function to be stable by substitution.

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.

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] P. Martin-Löf, "A theory of types," University of Stockholm, Tech. Rep. 71-3, 1971.
- [2] T. G. Griffin, "The formulae-as-types notion of control," in *Conf. Record of POPL '90*. ACM Press, New York, 1990, pp. 47–57.
- [3] M. Parigot, "Lambda-mu-calculus: An algorithmic interpretation of classical natural deduction," in *Proceedings of LPAR '92*. Springer-Verlag, 1992, pp. 190–201.
- [4] H. Herbelin, "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.
- [5] 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. Providence, Rhode Island: American Mathematical Society, 1962, p. 1–27.
- [6] S. C. Kleene, "On the interpretation of intuitionistic number theory," *The Journal of Symbolic Logic*, vol. 10, no. 4, pp. 109–124, 1945.
- [7] 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.
- [8] U. Berger and P. Oliva, "Modified bar recursion," *Math. Struct. in Comp. Sciences*, vol. 16, no. 2, pp. 163–183, 2006.
- [9] U. Berger, "A computational interpretation of open induction," in *Proceedings of LICS 2004*. IEEE Computer Society, 2004, p. 326.
- [10] J.-L. Krivine, "Dependent choice, 'quote' and the clock," *Theor. Comput. Sci.*, vol. 308, no. 1-3, pp. 259–276, 2003.
- [11] —, "Realizability in classical logic," *Panoramas et synthèses*, 2004, to appear.
- [12] 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.
- [13] A. G. Dragalin, "New kinds of realizability and Markov's rule," *Soviet Mathematical Doklady*, vol. 251, pp. 534–537, 1980.
- [14] A. Church, "A set of postulates for the foundation of logic," *Annals of Mathematics*, vol. 2, pp. 33, 346–366, 1932.
- [15] H. P. Barendregt, *The Lambda Calculus: Its Syntax and Semantics*. Amsterdam: North Holland, 1984.

- [16] 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.
- [17] H. Herbelin, "C'est maintenant qu'on calcule: au cœur de la dualité," Habilitation thesis, University Paris 11, Dec. 2005.
- [18] G. D. Plotkin, "Call-by-name, call-by-value and the lambda-calculus," *Theor. Comput. Sci.*, vol. 1, pp. 125–159, 1975.
- [19] E. Moggi, "Computational lambda-calculus and monads," Edinburgh Univ., Tech. Rep. ECS-LFCS-88-66, 1988.
- [20] 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.
- [21] A. Sabry and P. Wadler, "A reflection on call-by-value," *ACM Trans. Program. Lang. Syst.*, vol. 19, no. 6, pp. 916–941, 1997.
- [22] J. H. Fasel, P. Hudak, S. Peyton Jones, and P. W. (editors), "Haskell special issue," *SIGPLAN Notices*, vol. 27, no. 5, May 1992.
- [23] Z. Ariola and M. Felleisen, "The call-by-need lambda calculus," *J. Funct. Program.*, vol. 7, no. 3, pp. 265–301, 1993.
- [24] J. Maraist, M. Odersky, and P. Wadler, "The call-by-need lambda calculus," *J. Funct. Program.*, vol. 8, no. 3, pp. 275–317, 1998.
- [25] H. Nakano, "A constructive formalization of the catch and throw mechanism," in *Proceedings of LICS 1992*. IEEE Computer Society, 1992, pp. 82–89.
- [26] T. Crolard, "A confluent lambda-calculus with a catch/throw mechanism," *J. Funct. Program.*, vol. 9, no. 6, pp. 625–647, 1999.
- [27] 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.
- [28] J.-Y. Girard, "Linear logic," *Theor. Comput. Sci.*, vol. 50, pp. 1–102, 1987.
- [29] 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.
- [30] T. Coquand, "A semantics of evidence for classical arithmetic," *J. Symb. Log.*, vol. 60, no. 1, pp. 325–337, 1995.
- [31] H. Herbelin, "An intuitionistic logic that proves Markov's principle," in *Proceedings of LICS 2010*. IEEE Computer Society, 2010, pp. 50–56.
- [32] D. Ilik, "Preuves constructives de complétude et contrôle délimité," PhD, École Polytechnique, 2010.