The experimental effectiveness of mathematical proof

Alexandre Miquel

1 Introduction

The aim of this paper is twofold. First, it is an attempt to give an answer to the famous essay of Eugene Wigner about the *unreasonable effectiveness of mathematics in the natural sciences* [25]. We will argue that mathematics are not only *reasonably* effective, but that they are also *objectively* effective in a sense that can be given a precise meaning. For that—and this is the second aim of this paper—we shall reconsider some aspects of Popper's epistemology [23] in the light of recent advances of proof theory [8, 20], in order to clarify the interaction between pure mathematical reasoning (in the sense of a formal system) and the use of empirical hypotheses (in the sense of the natural sciences).

The technical contribution of this paper is the proof-theoretic analysis of the problem (already evoked in [23]) of the *experimental modus tollens*, that deals with the combination of a formal proof of the implication $U \Rightarrow V$ with an experimental falsification of V to get an experimental falsification of U in the case where the formulæ U and V express empirical theories in a sense close to Popper's. We propose a practical solution to this problem based on Krivine's theory of classical realizability [20], and describe a simple procedure to extract from a formal proof of $U \Rightarrow V$ (formalized in classical second-order arithmetic) and a falsifying instance of V a computer program that performs a finite sequence of tests on the empirical theory U until it finds (in finite time) a falsifying instance of U.

1.1 In which sense are mathematics effective?

Questioning the effectiveness of mathematics—especially through their interactions with other scientific fields—raises at least three different problems.

The first problem is the problem of the expressiveness of the mathematical language, its ability to formulate the concepts and problems of scientific theories other than mathematics. The massive development of sciences in the past two centuries has shown that the mathematical language is well-suited to formulate the theories and concepts of (at least some of) the natural sciences. But this is much less clear for the human sciences, where the traditional constructions of mathematical logic are too coarse to express subtle statements such as 'A despite B', 'A instead of B' or even 'the behavior X is typical among the population Y'. Considering the problem of expressiveness, natural languages appear to be much more effective than mathematics to express the majority of scientific concepts—and not only the purely technical aspects of astronomy, physics or chemistry. Expressiveness is not the distinctive feature of mathematics.

The second problem—that is purely subjective—is the ability of mathematics and of mathematical reasoning to integrate within the structures of human reasoning, their compatibility with our 'intellectual habits', their ability to stimulate our imagination by taking intuitions and mental representations in our cultural and historical background, and above all: their ability to be easily grasped by our mind. This problem is strongly related to the question of 'aesthetics' in mathematics. Mathematicians take a great care in finding elegant proofs for their theorems, in organizing their proofs into definitions, lemmas, propositions and corollaries, in designing theories with the highest level of generality. If mathematicians prefer conceptual and elegant proofs to more *ad hoc* proofs, this is not because elegant proofs make theorems more true. This is because these proofs (or these theories) are more easily grasped by our mind, so that their constituents are more likely to be assimilated by mathematicians and later reused in other parts of mathematics. Again, this form of effectiveness (and the accompanying aesthetics) is not distinctive of mathematics. One can find it in mythology, in religions, in poetry, in philosophy and in psychoanalysis. And this is surely not a coincidence that mathematics borrows part of its vocabulary to philosophy and religion: 'canonical', 'transcendental', 'faithful', 'universal' (in Greek: $\kappa\alpha\theta o\lambda\iota\kappa\acute{o}\varsigma$) and more recently: 'category'.

The third problem—which is actually the only problem we want to address in this paper—is the *objectivity* of mathematical reasoning, that is: the ability of mathematical reasoning to interact with non-mathematical *objects*, especially through the experimental method. To understand the difference between the subjective effectiveness and the objective effectiveness of mathematics, let us propose the following analogy. Mathematics can be viewed as a tool, like a hammer, a spoon or even a car. Most tools built by and for human beings have two sides: one side that is intended to be grasped by the hand, and another side that is intended to interact with other objects depending on the function of the tool. Without a side that is adapted to our hand, we could not *manipulate* the tool. But without the side that operates on the world, our tool would be useless. The only difference between mathematics and concrete tools is that mathematics are grasped not by our hand but by our mind.

A common view among pure mathematicians is that mathematics are developed 'for the honor of the human spirit' (Jacobi). But looking at how mathematics where used through ages and considering their prominent role in the technology of the twentieth century, it should be clear that mathematics are not only developed for this purpose. Through applied mathematics, the most abstract mathematical theories succeed to find their applications in the everyday life. For instance, the design of the profile of the wing of a plane involves complex mathematical models. But this is not because these models are aesthetic, or because the underlying theories are cleverly organized into definitions, lemmas and corollaries that we use them here. We use them because they help us to determine the profile of the wing that will make the plane fly.[1]

[1]In this example, the corresponding mathematical models are implemented in computer simulations that are able to predict—up to some extent—the behavior of the wing in the atmosphere. Using these simulations, it is possible to correct the most obvious defects of the profile before doing a real test in a wind tunnel, thus reducing the number of such tests. Here, the objective effectiveness of mathematical models is their ability to produce correct predictions (by the only mean of calculations) of what will happen in the wind tunnel.

1.2 The interface with natural sciences

The distinctive feature of applied mathematics is that they combine pure mathematical reasoning with symbols, concepts and hypotheses that are external to pure mathematics[2], such as the notions of time, velocity, magnetic field, intensity, etc. In the language of mathematics, these external notions are expressed by the means of function and predicate symbols that are simply added to the core language of mathematics. This extra symbols also come with hypotheses (equations, relations, etc.) expressing the assumptions of the applied theory, for instance the equations of fluid mechanics or Maxwell's equations. But the important point here is that neither the symbols nor the accompanying hypotheses constitute a definition of these concepts. The real definition of these concepts lies outside mathematics, in the physical (and experimental) interpretation we decide to give to the corresponding symbols.

For instance, Euler's equation of inviscid motion

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\, \mathbf{u} \;=\; -\frac{\nabla P}{\rho}$$

that relates the velocity $\mathbf{u}$ of an uncompressible fluid of density ρ with its pressure P (in some domain $D \subseteq \mathbb{R}^4$) does not constitute a definition of the functions $\mathbf{u} : D \to \mathbb{R}^3$ and $P : D \to \mathbb{R}$ (assuming that we know the quantity ρ)[3]. Here, the real definition of P lies outside mathematics. It lies in the decision to interpret the symbol P in the physical world as the local pressure in function of the position and the time within the domain D (that also needs to be interpreted, for example by expliciting a system of coordinates).

In the same way, the equation above cannot be proved[4] within mathematics, since it contains undefined symbols ($\mathbf{u}$ and P) and since it is obviously not true for every pair of functions $(\mathbf{u}, P)$. But on the other hand, we can *test* the equation above using the physical interpretation of the symbols $\mathbf{u}$ and P as well as the physical interpretation of the derivative. Given a finite set of points in the domain and fixing a precision for our measures, we can measure the quantities above and check whether they are equal or not at the required precision. But such a test is necessarily finite in nature (including the precision) and cannot constitute a proof that the equation holds everywhere in the domain at an arbitrary precision. Nevertheless, the equation above can be falsified by a test showing that both members of the equation differ at some point of the domain. (Typically: by showing that the difference between the values given by the measure exceeds the given precision.)

This example should clarify the nature of the interface between mathematics and the natural sciences. This interface has two sides:

- A mathematical side, that consists of a finite number of new mathematical symbols accompanied with hypotheses describing (some of) the relations between these symbols. Formally, these hypotheses are expressed as formulæ in the language of mathematics extended by the new symbols.

[2] By pure mathematics, we mean mathematics that can be expressed in the language of (say) set theory and that can be proved in the corresponding system of axioms.

[3] It is well known that such an equation has infinitely many solutions in mathematics.

[4] It is actually possible to prove this equation from the particle model of uncompressible fluids. But in this case, our proof relies on new (still undefined) symbols and new assumptions coming from the particle model and from Newtonian mechanics. Doing this, we reduce fluid mechanics to Newtonian mechanics, but the situation is epistemologically the same.

- A non mathematical side, formed by the 'real world' interpretation of the new symbols together with the procedures by which the relations expressed by these symbols can be tested. Let us insist on the fact that this side of the interface is by nature completely informal.

In what follows, we shall assume that the accompanying hypotheses are falsifiable in a sense close to Popper's [23]. Technically, this assumption will be enforced by putting severe restrictions on the kind of formulæ which may be used as empirical hypotheses (cf section 2.4).

1.3 The experimental modus tollens

The interest of expressing a scientific theory using the mathematical language (through an interface such as described above) is that we can benefit of the whole strength of mathematical reasoning to deduce consequences of our extra-mathematical assumptions. Among these consequences, some of them may fulfill the requirement of falsifiability, which means that they can be tested, using the same interpretation as before for the extra-mathematical symbols. In this way, mathematical reasoning appears to be an extraordinary device to extend the empirical basis of a theory: starting from a small set of empirical hypotheses—which may be difficult to test directly—we can use mathematical reasoning to derive a much larger set of empirical formulæ, thus offering more possibilities to test the theory, and more opportunities to falsify it.

The logical rule by which we deduce the falsity of an empirical theory U from the falsity of an empirical theory V that is a mathematical consequence of U is the rule of *modus tollens*: from $U \Rightarrow V$ and $\neg V$, deduce $\neg U$. But if we look closely at how this rule is used in the process of indirect falsification of U through a direct falsification of V, then we see that the corresponding inference combines two premises of a different nature: a *mathematical proof* of the implication $U \Rightarrow V$—that we can think as a formal object expressed in some formal system—and an empirical falsification of V—given by a set of parameters for which the corresponding test fails. The situation is illustrated with the following (tentative) inference rule

$$\frac{\text{math.} \vdash U \Rightarrow V \qquad \text{exp.} \not\models V}{\text{exp.} \not\models U}$$

where mathematical provability (written 'math. $\vdash \cdots$') is explicitly distinguished from experimental falsification (written 'exp. $\not\models \cdots$').

Clearly, the combination of both premises does not constitute a mathematical proof: it contains an object—the experimental falsification of V—that is external to mathematical reasoning. On the other hand, the combination of both premises does not obviously lead to an experimental falsification of U: we have a set of parameters for which the corresponding test on V fails, but how to deduce from it (and from the proof of $U \Rightarrow V$) another set of parameters for which the corresponding test on U will fail? Actually, we do not even know which statement of the theory U is responsible for the failure.

We are thus set in a very uncomfortable epistemological situation in which the falsification of U is purely logical—we know that U is wrong (because we believe in mathematics[5]), but we do not know where. The problem comes

[5] We shall come back to this belief in section 6.

here from the fact that mathematical reasoning seems to destroy the empirical connection between hypotheses and conclusions. The proof of $U \Rightarrow V$ may rely on mathematical abstractions (functional spaces, measurable sets, categories, etc.) as well as on reasoning principles (*reductio ad absurdum*, the axiom of choice) that have no obvious 'real world' interpretation. So, how to keep track of the empirical falsification of V through all these abstractions?

In this situation, Popper notices that we can first remove from the theory U all the statements from which V is independent (that is: all the statements that are not involved in the proof of $U \Rightarrow V$), while insisting on the fact that the remaining sub-system U' is still falsified as a whole, since the indirect falsification of U' gives us no clue on which statement of U' has to be rejected:

> Thus we cannot at first know which among the various statements of the remaining sub-system U' (of which V is not independent) we are to blame for the falsity of V; which of these statements we have to alter, and which we should retain. (I am not here discussing interchangeable statements.) It is often only the scientific instinct of the investigator (influenced, of course, by the results of testing and re-testing) that makes him guess which statements of U' he should regard as innocuous, and which he should regard as being in need of modification. (...) [23, footnote 2 p. 56][6]

However, Popper's analysis of the problem is limited by the fact that it does not take into account what is actually the key ingredient of an effective solution: the formal proof of the implication $U \Rightarrow V$. In the light of the recent developments of proof theory (especially in the theory of realizability), we shall see that this formal proof does not only contain a program (according to the correspondence between proofs and programs [8, 16]), but that this program—when combined with an experimental falsification of V and when executed in a suitable environment—constitutes a procedure of tests of the theory U that eventually reaches an explicit falsification of this theory.

The problem of experimental effectiveness To perform our proof-theoretic analysis of the problem of the experimental modus tollens, we shall focus on an apparently simpler—but actually equivalent—problem, which we call the *problem of experimental effectiveness*. This problem is the following: given an empirical theory U with a mathematical proof of the formula $\neg U$ (expressing that U is contradictory), how to find an experimental falsification of the theory U, that is: a particular instance of a statement of the theory U that is experimentally false?

$$\frac{\text{math.} \vdash \neg U}{\text{exp.} \not\models U}$$

This problem is clearly an instance of the problem of the experimental modus tollens, corresponding to the case where V is the absurd formula $\bot$. (Remember that in logic, the negation $\neg U$ can be defined as a shorthand for $U \Rightarrow \bot$.) Note that the statement expressed by the formula $\bot$ can be considered as an empirical statement, namely, as the statement which is always false, by convention.

[6] We changed the letters used by Popper to denote the initial system and its falsified consequence to agree with our notations in this paper.

Conversely, it is not difficult to see that the principle of experimental effectiveness actually implies the principle of the experimental modus tollens, in the sense that we can transform any procedure solving the problem of experimental effectiveness into a procedure solving the problem of the experimental modus tollens. Since this transformation relies on the particular way empirical statements are represented in the mathematical language (that will be given in section 2.4), we postpone the description of this transformation to section 4.6.

1.4 From proofs to programs

We already noticed that Popper's analysis of the problem of the experimental modus tollens does not take into account the mathematical proof of the implication $U \Rightarrow V$. In Popper's epistemology, mathematical proofs are only formal objects: they are useful to convey truth between premises and conclusions (provided the theory is consistent), but they have no real operational meaning, they are computationally inert.

The proof-theoretic point of view However, the last century—especially its second half—saw a large development of *proof theory* [3, 6, 14, 15, 10, 7, 21] which shed a new light on the *operational contents* of mathematical proofs. The main discovery was that every formal mathematical proof contains a program[7] that embodies the operational contents of the proof. One of the fascinating aspects of this correspondence between proofs and programs—a.k.a. the Curry-Howard correspondence [5, 13, 8]—is that the program contained in the proof is not constructed from the formulæ constituting the proof (which represent the 'idealistic' part of the proof), but from the deduction steps themselves, from the very hinges of the reasoning. For example, if we want to extract the program hidden in the following proof of the syllogism Barbara

$$\dfrac{\dfrac{\dfrac{\dfrac{[\forall x\,(B(x) \Rightarrow C(x))]^{\,g} \qquad \dfrac{[\forall x\,(A(x) \Rightarrow B(x))]^{\,f} \qquad [A(x)]^{\,u}}{A(x) \Rightarrow B(x) \qquad\qquad B(x)}\ \text{apply}}{B(x) \Rightarrow C(x) \qquad\qquad\qquad C(x)}\ \text{apply}}{A(x) \Rightarrow C(x)}\ \lambda u}{\forall x\,(A(x) \Rightarrow C(x))}}{\forall x\,(B(x) \Rightarrow C(x)) \Rightarrow \forall x\,(A(x) \Rightarrow C(x))}\ \lambda g}{\forall x\,(A(x) \Rightarrow B(x)) \Rightarrow \forall x\,(B(x) \Rightarrow C(x)) \Rightarrow \forall x\,(A(x) \Rightarrow C(x))}\ \lambda f$$

(here expressed in natural deduction style), we have to read *between* the lines of the proof, considering each axiom and each inference step as a single programming construct (depicted here on the right hand side of the corresponding rule). Collecting all these programming constructs, we thus get the program

$$\lambda f\,.\,\lambda g\,.\,\lambda u\,.\,g\,(f\,u)$$

(expressed in the λ-calculus [4, 1]) that performs the composition of two functions f and g, which program constitutes the procedural explanation of the proof of transitivity of inclusion.

[7] In the sense of computer science.

From intuitionistic logic to classical logic For a long time, the proofs-as-programs correspondence was confined to intuitionistic (or constructive) mathematics, and was unable to interpret classical reasoning principles such as the law of excluded middle, Peirce's law, or the principle of *reductio ad absurdum*. The breakthrough came in 1990, when Griffin [11] discovered that Peirce's law (that intuitionistically implies the law of excluded middle) could be interpreted in terms of the control operator call/cc ('call with current continuation'). By enriching the λ-calculus with such control operators, it became possible to extend the proofs-as-programs correspondence to classical logic.

Technically, control operators permit to save the current execution context, thus allowing programs to restart computation at a formerly saved point. In practice, programs extracted from classical proofs use this feature to implement the trial and error method: when asked a question, a program can give a first (and possibly incorrect) answer while saving the current context. If the sequel of the execution reaches a contradiction (because the answer was wrong), the program may *backtrack* (by restoring the context that asked the question) and provide a new answer, typically using information given by the computation that reached the contradiction. In this way, the flow of information becomes more symmetric: it does not only go from premises to the conclusion, but it can also go back from the conclusion to premises.[8]

The model-theoretic point of view The traditional notion of a model is based on Tarski's semantics of truth[9]: the interpretation of a formula has only two possible outcomes: 0 (the formula is false) and 1 (the formula is true). A formal system is said to be *sound* (w.r.t. a given class of models) when every formula that is provable in this system is interpreted by the truth value 1 in every model (belonging to the given class).

This simple picture is sufficient to grasp the main limit of traditional models: they are designed in order to interpret *provability*, not to interpret *proofs*. Indeed, a formula that has a complex proof (say: Fermat's last theorem) will be interpreted the very same way as a formula having simple proofs (say: $2+2=4$), that is: by the truth value 1. Traditional models can only distinguish between truth and falsity, and this distinction is definitely too coarse to convey more elaborate invariants, such as the computational contents of proofs.

To analyze the computational contents of intuitionistic proofs, Kleene introduced the notion of *realizability* [14], that can be seen (at a first glance) as a generalization of the notion of a model where the truth values 0 and 1 are replaced by all the possible sets of programs. Through Kleene's realizability interpretation (a.k.a. the Brouwer-Heyting-Kolmogorov interpretation), a formula A is interpreted as a *type*, that is: as a set $|A|$ of programs which share a common computational behavior—such a computational behavior being usually called a *specification*. For instance, the implication $A \Rightarrow B$ is interpreted as the set of all programs (or computable functions) that transform any program belonging to $|A|$ into to a program belonging to $|B|$. Similarly, the conjunction

[8] Of course, there is a striking analogy between Popper's epistemology and the modern understanding of the computational contents of classical reasoning in terms of the trial and error method (which is technically implemented using control operators). One of the aims of this paper is to help to clarify this connection, which is by no means coincidental.

[9] One of the main sources of inspiration of Popper's theory of 'truth as a correspondence'.

$A \wedge B$ is interpreted as the set of all programs computing an ordered pair $\langle t_1, t_2 \rangle$ whose components t_1 and t_2 belong to $|A|$ and $|B|$, respectively.

In realizability, the model-theoretic property of soundness becomes the following property of *adequacy*: if d is a formal proof (i.e. a *derivation*) of a formula A in a suitable formal system, then the program d^* extracted from d (i.e. the computational contents of d) belongs to the set $|A|$. Which means that the program d^* that has been mechanically constructed from the formal proof d precisely satisfies the specification induced by the formula A seen as a type, thus allowing various predictions about the execution of this program. For example, if d is a derivation of the formula $\exists x \in \mathbb{N}\ f(x) = 0$, then we can predict (using the specification associated with the formula $\exists x \in \mathbb{N}\ f(x) = 0$) that the execution of the extracted program d^* will produce an ordered pair whose first component is a natural number n such that $f(n) = 0$.[10]

As shown by the above example, the theory of realizability reveals that mathematical proofs are more than formal descriptions of mathematical entities: they actually give us procedures to compute with such entities. Far from being only descriptive, mathematical proofs are also prescriptive.

From Kleene realizability to Krivine realizability However, the notion of realizability such as introduced by Kleene is technically limited to intuitionistic mathematics[11]. For this reason we shall not consider Kleene's theory in the sequel, but Krivine's theory of classical realizability [20] that was developed in the 90's to take into account Griffin's discovery about the connection between classical logic and control operators. More than an extension of Kleene's theory, the theory of classical realizability is a complete reformulation of the very principles of realizability in order to cope with the full strength of (classical) mathematical reasoning, including several forms of the axiom of choice [19].

Krivine's theory is a powerful tool that is able to deal with very strong mathematical theories such as Zermelo-Fraenkel set theory [18] or the calculus of constructions with universes [22] (the underlying formalism of the Coq proof assistant). To avoid unnecessary complication, we shall only consider Krivine's theory in the framework of second-order Peano arithmetic (PA2), and we will show how the corresponding realizability interpretation can be used to give a procedural solution to the problem of the experimental modus tollens in the case where the proof of $U \Rightarrow V$ is entirely formalized in PA2. Nevertheless, the methodology described in this paper is not limited to this particular formalism, and it remains valid in the case where the proof of $U \Rightarrow V$ is formalized in any of the aforementioned stronger theories.

Outline of the paper

In section 2 we introduce the syntax and deduction rules of the extension of second-order Peano arithmetic (PA2) we shall use throughout the paper. In

[10] This prediction only holds for constructive proofs. The prediction of the computational behavior of programs extracted from classical existential proofs is much more subtle, since these programs are based on the trial and error method.

[11] It follows from the very definition of Kleene realizability that the law of excluded middle cannot be realized. This argument (combined with the property of adequacy) can be used to show that the law of excluded middle is not derivable in most constructive systems.

section 3, we introduce an extension of Krivine's language λ_c with extra instructions to perform experimental tests, and we show how to extract programs written in this language from derivations formalized in PA2. The corresponding realizability model is introduced in section 4, which provides (via the property of adequacy) a model-theoretic justification of the extraction procedures presented in the previous section. From this we deduce procedural ways to solve the problem of experimental effectiveness as well as the problem of the experimental modus tollens. In section 5 we present two execution methods for the corresponding programs and give an application to computer science. We finally conclude in section 6 by discussing several proof-theoretic aspects about the concerned procedures, including the empirical and non empirical assumptions underlying the proposed methodology.

2 An experimental arithmetic

We now present a formalism which we call the *experimental arithmetic*, whose syntax and deduction rules are summarized in Fig. 1. Basically, this formalism is an extension of second-order Peano arithmetic (PA2) with primitive predicate symbols (written p, q, r, etc.) expressing experimentally testable facts. By this, we mean that each atomic formula $p(x, y, z, \ldots)$ associated with an experimental predicate symbol p comes with a 'real world' interpretation such as

> *The animal in box No. x is a cat*

or

> *The concentration of alcohol in tube No. x is comprised between y and z percent*

or even

> *The 2nd-coordinate of the magnetic field in the universe cube No. x has an average value comprised between y/w and z/w (in Tesla).*

Since we work in arithmetic, sets of parameters of an experimental predicate p are naturally expressed as tuples of natural numbers, hence the need for more or less clever codings to address the physical reality.[12] (For instance in the 3rd example, the use of a fixed sequence of universe cubes with discrete bounds to address portions of the space, and the introduction of a precision parameter w to approximate continuous quantities.) On the other hand, experimental predicates come with no special meaning as *mathematical* entities, that is, they come with no specific axiom or computational rule attached to them.

2.1 The language

The language of experimental arithmetic (cf Fig. 1) is defined from the following sets of symbols:

- An infinite set of first-order variables (written x, y, z, etc.), that is: variables denoting individuals (i.e. natural numbers).

[12] We could easily enrich the language to express richer kinds of parameters such as relative integers, rational numbers, lists, trees, etc. However, the parameters of a testable predicate p should always remain discrete, independently from their representation.

- For each arity $k \geq 0$, an infinite set of second-order variables of arity k, (written X, Y, Z, etc.), that is: variables denoting $k$ary relations over individuals.
- For each primitive recursive definition of a function—including zero and successor—a function symbol (written f, g, h, etc.) representing the function described by the given definition. In particular, we respectively denote by 0 and s the constant symbol representing zero and the unary function symbol representing the successor function. We will also freely use well-known symbols such as $+$ (addition), $\times$ (multiplication), etc.
- A finite set of predicate symbols (written p, q, r, etc.) representing experimental predicates, each of them given with its arity.

The language of experimental arithmetic distinguishes two kinds of expressions: *numeric expressions* (written e, e_1, e', etc.) that represent individuals; and *formulæ* (written A, B, C, etc.) that represent facts—possibly true or false—built from the experimental predicates.

Numeric expressions The language of numeric expressions is inductively defined from the following two construction rules:

- If x is a first-order variable, then x is a numeric expression.
- If f is a $k$ary function symbol and if $e_1, \ldots, e_k$ are k numeric expressions, then $f(e_1, \ldots, e_k)$ is a numeric expression.

The set of free variables of a numeric expression e is written $FV(e)$. Given a numeric expression e, a variable x and a numeric expression e', we denote by $e\{x := e'\}$ the numeric expression obtained by replacing in e every occurrence of the variable x by e'.

For each closed numeric expression e, we call the *value of* e and write $\downarrow e$ the natural number computed by the expression e using the computation rules attached to the (primitive recursive) function symbols f contained in e. In model-theoretic terms, the value of e is but the denotation of e in the standard model of arithmetic (i.e. $\mathbb{N}$), interpreting function symbols the obvious way.

Formulæ The language of formulæ is inductively defined from the following five construction rules:

- If p is a $k$ary experimental predicate symbol and if $e_1, \ldots, e_k$ are k numeric expressions, then $p(e_1, \ldots, e_k)$ is a formula.
- If X is a $k$ary second-order variable and if $e_1, \ldots, e_k$ are k numeric expressions, then $X(e_1, \ldots, e_k)$ is a formula.
- If A and B are formulæ, then $A \Rightarrow B$ is a formula.
- If x is a first-order variable and if B is a formula, then $\forall x\, B$ is a formula.
- If X is a $k$ary second-order variable and if B is a formula, then $\forall X B$ is a formula.

The set of free (first- and second-order) variables of a formula A is written $FV(A)$. As usual, formulæ are considered up to α-conversion, regardless from the names of bound variables. Formulæ are equipped with two different operations of substitution:

- (*First-order substitution*) Given a formula A, a variable x and a numeric expression e, we write $A\{x := e\}$ the formula obtained by replacing in A every free occurrence of the variable x by e, taking care of renaming bound first-order variables in A in order to prevent variable captures. More generally, we denote by

  $$A\{x_1 := e_1; \ldots; x_k := e_k\}$$

  the formula obtained by simultaneously replacing in the formula A every free occurrence of a variable x_i (among $x_1, \ldots, x_k$) by the corresponding numeric expression e_i. (Again: one has to take care of renaming bound variables in A when needed.)

- (*Second-order substitution*) Given a formula A, a $k$ary second-order variable X, a list of k variables $x_1, \ldots, x_k$ and a formula B, we denote by

  $$A\{X(x_1, \ldots, x_k) := B\} \qquad (\text{or} \quad A\{X := \hat{x}_1 \cdots \hat{x}_k.B\})$$

  the formula obtained by replacing in A every atomic subformula of the form $X(e_1, \ldots, e_k)$ (corresponding to a free occurrence of X in A) by the formula $B\{x_1 := e_1; \ldots; x_k := e_k\}$, while taking care of renaming bound (first- and second-order) variables when necessary. Intuitively, this operation describes the instantiation of the indefinite predicate X of arity k by the actual predicate $\hat{x}_1 \cdots \hat{x}_k.B$ ('B of $x_1, \ldots, x_k$') of the same arity.

The language of formulæ we actually presented is the language of *minimal* second-order logic: its only nonatomic constructions are implication, first-order universal quantification and second-order universal quantifications for all arities. However, this core language is expressive enough to encode all the other constructions of logic: absurdity, negation, conjunction, disjunction, first- and second-order existential quantifications — and even Leibniz equality — using the so-called 'second-order encodings' recalled in Fig. 1. In the sequel, we shall freely use the constructions $\bot$, $\neg A$, $A \wedge B$, $A \vee B$, $\exists x\, A$, $\exists X\, A$ and $e_1 = e_2$ as abbreviations in the core language.

2.2 Deduction rules and derivations

The language defined above is equipped with the standard notion of provability in second order arithmetic (PA2), using a presentation based on intuitionistic natural deduction extended with Peirce's law to recover classical logic. Formally, we work with asymmetric sequents of the form

$$\Gamma \vdash A \qquad \text{('under the assumptions } \Gamma\text{, } A \text{ holds')}$$

where $\Gamma \equiv A_1, \ldots, A_n$ is a finite list of formulæ used as hypotheses — called the *context* — and where A is a formula.

The notations $FV(\Gamma)$, $\Gamma\{x := e\}$, $\Gamma\{X(x_1, \ldots, x_n) := B\}$, etc. are extended to lists of formulæ $\Gamma = A_1, \ldots, A_n$ by setting:

$$\begin{array}{rcl} FV(A_1, \ldots, A_n) & = & FV(A_1) \cup \cdots \cup FV(A_n) \\ (A_1, \ldots, A_n)\{x := e\} & \equiv & A_1\{x := e\}, \ldots, A_n\{x := e\} \end{array}$$

(And similarly for the notation $\Gamma\{X(x_1, \ldots, x_n) := B\}$.)

The deduction rules of the system are given in Fig. 1. They consist of:

- The deduction rules of minimal intuitionistic second-order logic, namely: the axiom rule, plus the introduction and elimination rules of implication, first- and second-order universal quantification.
- A specific rule for Peirce's law, that entails all the other classical reasoning principles: the excluded middle ($A \vee \neg A$), double negation elimination ($\neg\neg A \Rightarrow A$) and the rule of *reductio ad absurdum*.

The axiom rule is extended with all the axioms of arithmetic, namely:

- The axioms expressing that the successor function is injective (3rd Peano axiom) and that zero is not in its image (4th Peano axiom). Notice that the formulation of these axioms relies on the encoding of equality and negation given in Fig. 1 ('abbreviations').
- The defining equalities of the function symbols f representing primitive recursive definitions of functions. Example of such equalities defining addition, multiplication, exponentiation, the factorial function, etc. are:

$$\begin{array}{rclcrcl} 0+y & = & y & \qquad & 0\times y & = & 0 \\ s(x)+y & = & s(x+y) & & s(x)\times y & = & (x\times y)+y \\ x\uparrow 0 & = & s(0) & & 0! & = & s(0) \\ x\uparrow s(y) & = & (x\uparrow y)\times x & & s(x)! & = & s(x)\times x! \end{array}$$

Once deduction rules have been defined, we can construct proofs by assembling instances of these rules such as in the following example:

$$\dfrac{\dfrac{\dfrac{}{\forall x\,(Y(x)\Rightarrow Z),\ Y(s(0))\ \vdash\ \forall x\,(Y(x)\Rightarrow Z)}}{\forall x\,(Y(x)\Rightarrow Z),\ Y(s(0))\ \vdash\ Y(s(0))\Rightarrow Z} \qquad \dfrac{}{\forall x\,(X(x)\Rightarrow Y(z+x)),\ Y(s(0))\ \vdash\ Y(s(0))}}{\forall x\,(Y(x)\Rightarrow Z),\ Y(s(0))\ \vdash\ Z}$$

Formally, we call a *derivation* (or a *proof*) any finite tree d whose nodes are labelled with sequents, and such that for each node of d labelled with sequent S and whose children are labelled with sequents $S_1,\ldots,S_n$ ($n \geq 0$), there is a deduction rule such that the inference $\frac{S_1\cdots S_n}{S}$ is an instance of this rule.

Given a derivation d, the sequent S that labels the root of d is called the *conclusion* of d, and d is called a *derivation of* S. When a sequent S has a derivation d, we say that S is *derivable*. In particular, when a sequent of the form $\vdash A$ (without assumption, A being a closed formula) is derivable, we say that the formula A is a *theorem* (of second-order arithmetic).

2.3 Arithmetic reasoning

The reader may have noticed that we presented axioms (cf Fig. 1) expressing that the successor function is injective (Peano 3rd axiom) and non-surjective (Peano 4th axiom), but that there is no axiom for the induction principle on natural numbers (Peano 5th axiom). It would be a bad idea to consider this principle in our axioms, since it is impossible to associate a program to the induction principle through program extraction.[13] To circumvent this difficulty, we shall use the following trick (well-known in second-order logic [8, 16, 20]).

[13]The reason for this is that the induction principle has no realizer in Krivine realizability [20]. The extraction mechanism must preserve the invariant that a program extracted from any proof of a formula A should be a realizer of the formula A. (See section 4 for more details.)

Syntax

(Numeric expressions) $e, e' \;::=\; x \mid f(e_1, \ldots, e_k)$

(Formulæ) $A, B \;::=\; p(e_1, \ldots, e_k) \mid X(e_1, \ldots, e_k) \mid A \Rightarrow B \mid \forall x\, A \mid \forall X\, A$

Abbreviations

(Contradiction)	$\bot$	$\equiv$	$\forall Z\, Z$
(Immediate truth)	$\top$	$\equiv$	$\forall Z\, (Z \Rightarrow Z)$
(Negation)	$\neg A$	$\equiv$	$A \Rightarrow \bot$
(Conjunction)	$A \wedge B$	$\equiv$	$\forall Z\, ((A \Rightarrow B \Rightarrow Z) \Rightarrow Z)$
(Disjunction)	$A \vee B$	$\equiv$	$\forall Z\, ((A \Rightarrow Z) \Rightarrow (B \Rightarrow Z) \Rightarrow Z)$
(1st order existence)	$\exists x\, A[x]$	$\equiv$	$\forall Z\, (\forall x\, (A[x] \Rightarrow Z) \;\Rightarrow\; Z)$
(2nd order existence)	$\exists X\, A[X]$	$\equiv$	$\forall Z\, (\forall X\, (A[X] \Rightarrow Z) \;\Rightarrow\; Z)$
(Leibniz equality)	$e_1 = e_2$	$\equiv$	$\forall Z\, (Z(e_1) \Rightarrow Z(e_2))$
	$e_1 \neq e_2$	$\equiv$	$\neg(e_1 = e_2)$

Deduction rules

(Axiom) $$\frac{}{\Gamma \vdash A}\ {\scriptstyle (A \in \Gamma \cup \mathscr{A})}$$

(⇒-intro,-elim) $$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \qquad \frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}$$

($\forall^1$-intro,-elim) $$\frac{\Gamma \vdash B}{\Gamma \vdash \forall x\, B}\ {\scriptstyle (x \notin FV(\Gamma))} \qquad \frac{\Gamma \vdash \forall x\, B}{\Gamma \vdash B\{x := e\}}$$

($\forall^2$-intro,-elim) $$\frac{\Gamma \vdash B}{\Gamma \vdash \forall X\, B}\ {\scriptstyle (X \notin FV(\Gamma))} \qquad \frac{\Gamma \vdash \forall X\, B}{\Gamma \vdash B\{X(x_1, \ldots, x_k) := A\}}$$

(Peirce's law) $$\frac{}{\Gamma \vdash ((A \Rightarrow B) \Rightarrow A) \Rightarrow A}$$

where $\mathscr{A}$ denotes the set of all axioms of arithmetic, that consists of:

- The axiom of injectivity: $\forall x\ \forall y\ (s(x) = s(y) \Rightarrow x = y)$
- The axiom of non confusion: $\forall x\ s(x) \neq 0$
- The defining equalities attached to the function symbols f (that recursively define the corresponding primitive recursive functions)

Figure 1: Language and deduction rules of PA2

Given a numeric expression e, we write

$$\mathsf{Nat}(e) \quad \equiv \quad \forall Z\ (Z(0)\ \Rightarrow\ \forall x(Z(x) \Rightarrow Z(s(x)))\ \Rightarrow\ Z(e))$$

the formula expressing that (the number denoted by) e belongs to the smallest class containing zero and closed under the successor function. It is easy to check that the class delimited by the predicate $\mathsf{Nat}(x)$ contains zero and is closed under the successor function, in the sense that the following formulæ are derivable in PA2:

1. $\mathsf{Nat}(0)$ (Peano 1st axiom)
2. $\forall x(\mathsf{Nat}(x) \Rightarrow \mathsf{Nat}(s(x)))$ (Peano 2nd axiom)

To relativize universal and existential first-order quantifications to the predicate $\mathsf{Nat}(x)$, we introduce the abbreviations

$$\begin{array}{lrcl} & \forall^{\mathsf{N}} x\ A[x] & \equiv & \forall x\ (\mathsf{Nat}(x) \Rightarrow A[x]) \\ \text{and} & \exists^{\mathsf{N}} x\ A[x] & \equiv & \forall Z\,(\forall x\,(\mathsf{Nat}(x) \Rightarrow A(x) \Rightarrow Z)\ \Rightarrow\ Z) \\ & & \Leftrightarrow & \exists x\ (\mathsf{Nat}(x) \wedge A[x]) \end{array}$$

defining what will be called *numeric quantifications* in the sequel—as opposed to first-order quantifications. Using these notations, it is a simple exercise to check that the induction principle holds provided we express it using numeric quantifications instead of first-order quantifications:

$$\forall Z\ \big(Z(0)\ \Rightarrow\ \forall^{\mathsf{N}} x\,(Z(x) \Rightarrow Z(s(y)))\ \Rightarrow\ \forall^{\mathsf{N}} x\ Z(x)\big)$$

Using this principle, we can more generally show that the class delimited by the predicate $\mathsf{Nat}(x)$ is closed under every function symbol of the signature:

Proposition 1 — *For every function symbol f of arity k, the formula*

$$\forall^{\mathsf{N}} x_1 \cdots \forall^{\mathsf{N}} x_k\ \mathsf{Nat}(f(x_1, \ldots, x_k))$$

is derivable in PA2.

Proof. The derivation of the formula above is constructed using the defining equations of the symbol f combined with the induction principle (relativized to the predicate $\mathsf{Nat}(x)$) according to the primitive recursive definition of f. □

As a consequence, a numeric expression e belongs to the class delimited by the predicate $\mathsf{Nat}(x)$ as soon as all its free variables belong to this class:

Proposition 2 — *For every numeric expression $e[x_1, \ldots, x_k]$ of free variables $x_1, \ldots, x_k$, the formula*

$$\forall^{\mathsf{N}} x_1 \cdots \forall^{\mathsf{N}} x_k\ \mathsf{Nat}(e[x_1, \ldots, x_k])$$

is derivable in PA2.

Proof. By structural induction on e. □

Using Prop. 1 and 2, we can mimic full second-order arithmetic — that is: second-order arithmetic with the induction principle — simply by replacing first-order quantifications by numeric quantifications everywhere in formulæ.

Remark. The trick presented above is based on the intuition that numeric expressions such as defined in section 2.1 do not exactly represent natural numbers, but only *individuals* that we can think of natural numbers without the knowledge that they are natural numbers. The purpose of the predicate $\mathsf{Nat}(x)$ is precisely to delimit the class of natural numbers, i.e. the class of individuals that are reached by any form of reasoning by induction. The knowledge that a numeric expression e is a natural number is represented here by a proof of the formula $\mathsf{Nat}(e)$. Such proofs play an important role during program extraction, since they correspond to the programs that compute the value of the corresponding numeric expression in a sense that will be precised in section 4.5.

2.4 Experimental formulæ

Within the language of formulæ, we distinguish three classes of formulæ representing experimental statements, namely: the class of *elementary formulæ*, the class of *testable formulæ* and the class of *testable universal formulæ* (Fig. 2).

Elementary formulæ. We call an *elementary formula* any (open or closed) formula E that has of the following two forms:

(*Experimentation*)	E	$\equiv$	$p(e_1,\ldots,e_k)$	
(*Equality test*)	E	$\equiv$	$e_1 = e_2$ $\equiv$	$\forall X\,(X(e_1) \Rightarrow X(e_2))$

Intuitively, elementary formulæ are atomic formulæ built from experimental predicates in a broader sense that includes the equality test (seen as a purely mathematical experiment).

Testable formulæ. They are inductively built from the following construction rules:

- The formlula $\bot \equiv \forall X X$ is a testable formula.
- If E is an elementary formula, then E is a testable formula.
- If E is an elementary formula and if T is a testable formula, then the formula $E \Rightarrow T$ is a testable formula.
- If E is an elementary formula and if T is a testable formula, then the formula $\neg E \Rightarrow T$ is a testable formula.

In other words, a testable formula is a formula of the form

$$T \quad \equiv \quad \pm E_1 \Rightarrow \cdots \Rightarrow \pm E_k \Rightarrow E_0\,,$$

where $E_1,\ldots,E_k$ are elementary formulæ (here, the notation $\pm E_i$ refers to one of both formulæ E_i or $\neg E_i$ indifferently) and where E_0 is an elementary formula or the formula $\bot$. From the point of view of classical logic, a testable formula is thus nothing but a (disjunctive) clause formed from the elementary formulæ:

$$T \quad \Leftrightarrow \quad \mp E_1 \vee \cdots \vee \mp E_k \vee E_0\,.$$

Closed testable formulæ are called *testable singular formulæ*

Testable universal formulæ. Finally, we call a *testable universal formula* any closed formula of the form

$$U \quad \equiv \quad \forall^{\mathsf{N}} x_1 \cdots \forall^{\mathsf{N}} x_k \; T[x_1, \ldots, x_k]$$

where $T[x_1, \ldots, x_k]$ is a testable formula whose free variables occur among the set of variables $\{x_1, \ldots, x_k\}$. The natural number k is called the *arity* of U. Notice that every testable singular formula (as well as every closed elementary formula) is a testable universal formula with a null arity.

Testable universal formulæ are testable in the sense that all their instances (corresponding to all the possible combinations of parameters in $\mathbb{N}^n$) can be tested separately. Of course, a testable universal formula cannot be verified globally—which would require an infinite number of tests—but it can be falsified with a single test corresponding to a single set of parameters.

Epistemologically, the notion of testable universal formula corresponds to Popper's notion of empirical (or falsifiable) statement [23].

(Elementary formulæ)	E	$::=$	$p(e_1, \ldots, e_k) \;\mid\; e_1 = e_2$
(Testable formulæ)	T	$::=$	$\bot \;\mid\; E \;\mid\; E \Rightarrow T \;\mid\; \neg E \Rightarrow T$
(Testable universal formulæ)	U	$::=$	$\forall^{\mathsf{N}} x_1 \cdots \forall^{\mathsf{N}} x_k \, T[x_1, \ldots, x_k]$

Figure 2: The language of experimental formulæ

2.5 The experimental theory and the tools to test it

The notion of an experimental theory We call an *experimental theory* any finite list $U_1, \ldots, U_\ell$ of testable universal formulæ. In practice, such a theory will be formed with three kinds of statements:

(1) Currently accepted scientific laws (according to a given theory)

(2) Initial conditions (of a particular experiment)

(3) Observed effects (of the experiment)

Typically, statements of the first kind will be formalized as testable universal formulæ whereas statements of the second and third kinds will be formalised as testable singular formulæ.

The situation we are interested in occurs when the experimental theory is contradictory. In practice, the contradiction arises when an observed effect (3) does not match a prediction derived from the initial conditions (2) using the universal laws given by the theory (1). Our aim is to define an effective procedure that selects particular instances of the formulæ $U_1, \ldots, U_\ell$ and test them successively until one of them fails—typically, an instance of one of the universal statements of the accepted theory.

In what follows, we assume given an experimental theory $U_1, \ldots, U_\ell$.

Test functions To test the experimental theory, we assume that each experimental predicate symbol p comes with a test function

$$\mathbf{Val}(p) : \mathbb{N}^k \to \{0;1\}$$

which maps each k-tuple of parameters $(\nu_1, \ldots, \nu_k) \in \mathbb{N}^k$ to a boolean value representing the success (1) or the failure (0) of the experiment associated to the atomic formula $p(\nu_1, \ldots, \nu_k)$.

From an experimental point of view, we assume that the computation of $\mathbf{Val}(p)(\nu_1, \ldots, \nu_k)$ can be achieved using a finite amount of resources for all tuples of parameters $(\nu_1, \ldots, \nu_k) \in \mathbb{N}^k$. In particular, we assume that this computation can always be done (at least potentially) in finite time.

From a mathematical point of view, we make the only assumption that the function $\mathbf{Val}(p) : \mathbb{N}^k \to \{0;1\}$ is total. In particular:

1. We do not assume that the function $\mathbf{Val}(p) : \mathbb{N}^k \to \{0;1\}$ is decidable, or even semi-decidable. Effectiveness has to be understood here in the (informal) sense of physics, not in the (formal) sense of recursion theory. In this work, we never need the assumption that physically computable functions are Turing computable—the physical version of Church's thesis.

2. More important, we do not assume that the set of test functions $\mathbf{Val}(p)$ constitutes a model of the experimental theory, which means that we do not assume that the experimental theory is valid. The extracted program of tests has to be correct independently from the validity of the experimental theory—it must perform correct computations even from wrong experimental assumptions.

Evaluating testable singular formulæ From the test functions $\mathbf{Val}(p)$ we define a generalized test function $T \mapsto \mathbf{Val}(T)$ that associates a boolean value $\mathbf{Val}(T) \in \{0;1\}$ to every testable singular formula T, indicating whether the experiment associated to T succeeds or fails. Formally, the boolean value $\mathbf{Val}(T) \in \{0;1\}$ is recursively defined on the structure of T by the equations

$$\begin{aligned}
\mathbf{Val}(e_1 = e_2) &= \begin{cases} 1 & \text{if } \downarrow e_1 = \downarrow e_2 \\ 0 & \text{otherwise} \end{cases} \\
\mathbf{Val}(p(e_1, \ldots, e_k)) &= \mathbf{Val}(p)(\downarrow e_1, \ldots, \downarrow e_k) \\
\mathbf{Val}(\bot) &= 0 \\
\mathbf{Val}(E \Rightarrow T) &= \sup(1 - \mathbf{Val}(E),\ \mathbf{Val}(T)) \\
\mathbf{Val}(\neg E \Rightarrow T) &= \sup(\mathbf{Val}(E),\ \mathbf{Val}(T))
\end{aligned}$$

Of course, if we assume that the test functions $\mathbf{Val}(p)$ are physically computable, then the evaluation function $T \mapsto \mathbf{Val}(T)$ for testable singular formulæ is physically computable too, using a simple recursive procedure ultimately based on the test functions $\mathbf{Val}(p)$, the equality test, and on the computation rules of primitive recursive functions inside numeric expressions.

In what follows, we will say that a tuple $(\nu_1, \ldots, \nu_k) \in \mathbb{N}^k$ is a *falsification* of a testable universal formula of the form

$$U \quad \equiv \quad \forall^{\mathsf{N}} x_1 \cdots \forall^{\mathsf{N}} x_k \ T[x_1, \ldots, x_k]\,,$$

when $\mathbf{Val}(T[\nu_1, \ldots, \nu_k]) = 0$.

3 Extracting a program from a proof

3.1 The programming language

We use here an extension of Krivine's language λ_c [20] with a small set of extra instructions to test the experimental theory and to report a failure when necessary. This language is defined from three kinds of syntactic entities:

- *Terms* (written t, u, etc.), that represent programs defined independently from their evaluation context. From the point of view of logic, terms are the computational equivalents of proofs.
- *Stacks* (written π, π', etc.), that represent evaluation contexts of programs. Formally, stacks are defined as finite lists of closed terms (representing function arguments) of the form $\pi = u_1 \cdot \ldots \cdot u_n \cdot \diamond$, where $\diamond$ denotes the bottom of the stack. From the point of view of logic, stacks are the computational equivalents of *counter-proofs* (or *refutations*). Notice that stacks are only formed with closed terms, and thus are closed objects (i.e. with no free occurrence of a term variable).
- *Processes*, written $t \star \pi$, that are just formed by putting a term t in front of a stack π. From the point of view of logic, processes represent the interaction (i.e. the 'debate') between a proof and a counter-proof within a contradictory situation.

The formal definition of terms, stacks and processes of the programming language is given in Fig. 3. The set of all closed terms is written Λ whereas the set of all (closed) stacks is written Π. The set of all processes is written $\Lambda \star \Pi$.

Given two terms t, u and a term variable ξ, we write $t\{\xi := u\}$ the term obtained by replacing in t every free occurrence of the term variable ξ by the term u. In practice, we will only use this operation when u is a closed term. Term substitution immediately generalises to a notion of parallel substitution written $t\{\xi_1 := u_1; \ldots; \xi_n := u_n\}$. Notice that when $u_1, \ldots, u_n$ are closed terms, parallel substitution may be computed sequentially by the equation

$$t\{\xi_1 := u_1; \ldots; \xi_n := u_n\} \quad = \quad (\cdots t\{\xi_1 := u_1\} \cdots)\{\xi_n := u_n\}\,.$$

Syntax of terms The language of terms includes all the constructions of Krivine's language λ_c, namely:

- The usual constructions of the λ-calculus: variables (written ξ, ζ, ϕ, etc.), function abstractions ($\lambda\xi\,.\,t$) and function applications (tu). From the point of view of logic, these constructions correspond to intuitionistic reasoning principles, and thus suffice to extract the computational contents of all proofs of HA2, the intuitionistic fragment of PA2.
- Specific instructions to manipulate continuations: the constant cc (*call with current continuation*, or *call/cc*, for short) and for each stack π, a constant k_π representing the corresponding continuation. From the point of view of logic, the constant cc (that generates constants k_π during the evaluation process) implements Pierce's law, from which we recover all classical logic, and thus the full strength of PA2.

Syntax of the language

Terms	t, u	$::=$	$\xi \mid \lambda\xi\,.\,t \mid tu \mid \mathsf{k}_\pi \mid \mathsf{cc} \mid \mathsf{test}_U \mid \mathsf{stop}$	
Stacks	π	$::=$	$\diamond \mid t \cdot \pi$	(t closed)
Processes	p	$::=$	$t \star \pi$	(t closed)

Evaluation rules (test instructions excepted)

(Push)	$tu \star \pi$	$\succ$	$t \star u \cdot \pi$
(Grab)	$\lambda\xi\,.\,t \star u \cdot \pi$	$\succ$	$t\{\xi := u\} \star \pi$
(Save)	$\mathsf{cc} \star t \cdot \pi$	$\succ$	$t \star \mathsf{k}_\pi \cdot \pi$
(Restore)	$\mathsf{k}_\pi \star t \cdot \pi'$	$\succ$	$t \star \pi$
	$\mathsf{stop} \star \pi$	$\not\succ$	

Evaluation rules for test instructions test_i

$$
\begin{aligned}
U_i &\equiv \forall^{\mathsf{N}} x_1 \cdots \forall^{\mathsf{N}} x_k \; T[x_1, \ldots, x_k] \\
T[x_1, \ldots, x_k] &\equiv L_1[x_1, \ldots, x_k] \Rightarrow \cdots \Rightarrow L_n[x_1, \ldots, x_k] \Rightarrow R[x_1, \ldots, x_k] \\
\bar{\nu}_1, \ldots, \bar{\nu}_k &\equiv \text{Krivine numerals associated to the numbers } \nu_1, \ldots, \nu_k
\end{aligned}
$$

1. Case where $\mathbf{Val}(L_j[\nu_1, \ldots, \nu_k]) = 0 \quad (1 \leq j \leq n)$

$$\mathsf{test}_i \star \bar{\nu}_1 \cdots \bar{\nu}_k \cdot \pi \;\succ\; (\lambda\xi_1 \cdots \xi_n \,.\, \xi_j \, \mathbf{I}) \star \pi$$

2. Case where $\mathbf{Val}(R[\nu_1, \ldots, \nu_k]) = 1$

$$\mathsf{test}_i \star \bar{\nu}_1 \cdots \bar{\nu}_k \cdot \pi \;\succ\; (\lambda\xi_1 \cdots \xi_n \,.\, \mathbf{I}) \star \pi$$

3. Case where $\mathbf{Val}(T[\nu_1, \ldots, \nu_k]) = 0$

$$\mathsf{test}_i \star \bar{\nu}_1 \cdots \bar{\nu}_k \cdot \pi \;\succ\; \mathsf{stop} \star \bar{i} \cdot \bar{\nu}_1 \cdot \ldots \cdot \bar{\nu}_k \cdot \pi$$

(writing $\mathbf{I} \equiv \lambda\phi\,.\,\phi$)

Figure 3: The language λ_c with test instructions

This core language is extended with the following extra instructions:

- For each formula U_i of the experimental theory, a constant test_i that tests an instance of U_i (whose parameters are given as arguments) and whose evaluation depends on the outcome of the experiment (cf Fig. 3).
- A constant stop with no associated evaluation rule, and whose only purpose is to report the failure of a test of the experimental theory.

Following the terminology of Krivine, we call a *quasi-proof* any term which is built only from variables, abstractions, applications and the constant cc. In other words, a quasi-proof is a term that contains neither the continuation constants k_π nor the extra constants test_i or stop. (The terminology of a 'quasi-proof' comes from the fact that programs extracted from proofs in pure classical second-order arithmetic are all of this form.)

Krivine numerals To represent natural numbers in the λ_c-calculus, we do not adopt Church's encoding, but a minor variant of it by letting

$$\text{(Krivine numeral } n\text{)} \qquad \bar{n} \;=\; \underbrace{\bar{s}\,(\cdots(\bar{s}}_{n}\;\bar{0})\cdots)$$

for every natural number $n \in \mathbb{N}$, where $\bar{0} = \lambda\xi\phi\,.\,\xi$ and $\bar{s} = \lambda\nu\xi\phi\,.\,\phi\,(\nu\xi\phi)$. Note that Krivine numeral $\bar{n}$ is not in normal form (except when $n = 0$) but still remains β-equivalent to the corresponding Church numeral. The reason for adopting this particular representation is due to the use of storage operators (cf section 4.4) that cannot work with Church's encoding directly [12].

In the sequel, terms of the form $\hat{n}$ (for some $n \in \mathbb{N}$) will be considered as *values*, and during evaluation, sets of parameters will be communicated to the test instructions test_U using this representation of natural numbers.

3.2 The relation of evaluation

The set of all processes is equipped with a binary relation of *(one step) evaluation* written $p \succ p'$, whose rules are given in Fig. 3. These rules comprise the usual evaluations rules of abstraction (Grab), application (Push) and of the constants cc (Save) and k_π (Restore), plus specific evaluation rules for the constants test_i we will describe with more details in section 4.5.

Notice that evaluation is deterministic, at least for a program that does not use test instructions: for each process $t \star \pi$ where t is not a test instruction, there is at most one process $t' \star \pi'$ such that $(t \star \pi) \succ (t' \star \pi')$. (We shall momentarily consider the possible cases where there is no such process.)

On the other hand, the evaluation rules of test instructions test_i such as presented in Fig. 3 are not deterministic (strictly speaking) since the three cases that define the outcome of the evaluation are not mutually exclusive: case 1 may happen for several values of j simultaneously, whereas case 2 may happen simultaneously with case 1. (Only case 3 is exclusive from cases 1 and 2.) However, it is always possible to define a priority between case 1 and case 2 (and inside case 1) in order to make the evaluation process completely deterministic. A possible policy is to give the highest priority to case 1, and inside case 1, to give the highest priority to the smallest index j. We will not discuss further the

different evaluation policies for the instructions test_i: this is simply irrelevant in the realizability model defined in section 4, where all policies are supported indifferently (even the non deterministic one). Nevertheless, we will assume that evaluation is deterministic in the sequel, since it simplifies the analysis of extracted programs and avoids the introduction of extra terminology.

In what follows, we write $\succ^*$ the reflexive-transitive closure of $\succ$. We say that a process p is *terminal* when there is no process p' such that $p \succ p'$. From the definition of the relation of evaluation, a process p is terminal if and only if it is of one of the following forms:

1. $p \equiv \mathsf{stop} \star \pi$. Here, evaluation terminates because the instruction stop has no evaluation rule. Since this is the intended meaning of the instruction stop, we consider this case as the 'normal' case of termination. All the other cases of termination are considered as errors.

2. $p \equiv t \star \diamond$, where t is an abstraction or one of the constants cc or k_π. Here, evaluation terminates since the term t in head position expects an argument in the stack, but the stack is empty. In stack-based languages (Forth, RPL, PostScript, etc.), this situation is known as 'stack underflow', and means that something went wrong during evaluation.

3. $p \equiv \mathsf{test}_U \star \pi$, where π does not start with k values $\bar{\nu}_1, \ldots, \bar{\nu}_k$ that are needed to perform the corresponding experiment (writing k the number of parameters expected by U). This may happen when the stack π contains less than k arguments ('too few arguments'), or when one of its first k arguments is not a Krivine numeral ('bad argument'). Again, this should be considered as an error.

Finally, we say that a process p' is a *terminal state* of a process p when p' is terminal and $p \succ^* p'$. Under the assumption of determinism, the terminal state of a given process, when it exists, is unique.

3.3 Extracting a program from a proof

The extraction mechanism actually consists of two extraction functions:

- A generic extraction function $d \mapsto d^*$ that extracts an *open* term d^* from a derivation d of an arbitrary sequent $\Gamma \vdash A$ in PA2. The term d^* is actually a quasi-proof (containing no test instruction) whose free variables represent the hypotheses in Γ.

- A specific extraction function $d \mapsto \tilde{d}$ (defined from the latter) that extracts a *closed* term $\tilde{d}$ from a derivation of the sequent $U_1, \ldots, U_\ell \vdash \bot$ (where $U_1, \ldots, U_\ell$ is the experimental theory). Unlike the (open) term d^*, the (closed) term $\tilde{d}$ contains test instructions corresponding to the experimental hypotheses that are used to derive the contradiction. This term is ready to be evaluated against the empty stack.

The generic extraction function The generic extraction function $d \mapsto d^*$ is recursively defined on the structure of the derivation using the equations of Fig. 4. To define this function, we first need to label the hypotheses of the sequents that appear in the derivation with term variables in such a way that:

1. The hypotheses $A_1, \dots, A_n$ of a sequent $A_1, \dots, A_n \vdash B$ are labelled with pairwise distinct term-variables $\xi_1, \dots, \xi_n$. In particular, we require that two occurrences of the same formula (in the context $A_1, \dots, A_n$) are labeled with distinct term-variables.

2. The labeling is consistent across deduction steps: if a formula A that appears in the context of the conclusion of a deduction step is labeled with a term-variable ξ, then the corresponding occurrence of A is labeled with the same proof-variable ξ in every premise of the deduction step.

A simple way to define such a labeling is the following: for every sequent $A_1, \dots, A_n \vdash B$, we label the hypotheses $A_1, \dots, A_n$ with the term-variables $\xi_1, \dots, \xi_n$ respectively, where $(\xi_i)_{i \geq 1}$ denotes a fixed enumeration of all term-variables. Note that here, the term-variable associated to an hypothesis only depends on its position in the context (starting from the left).

In Fig. 4, we (ab)use the notation ξ_A to denote the term-variable associated to the considered occurrence of the formula A in the context. In the case of an axiom $\Gamma \vdash A$ (where $A \in \Gamma$), there might be several occurrences of the formula A in the context Γ, so that the chosen variable ξ_A actually depends on the occurrence we focus on. In the case of the introduction of an implication, the term-variable variable ξ_A refers to the rightmost hypothesis in Γ, A.

$$\left(\overline{\Gamma \vdash A}\right)^* = \begin{cases} \xi_A & \text{if } A \in \Gamma \\ \lambda \xi \,.\, \xi & \text{if } A \in \mathscr{A} \setminus \{\text{4th Peano axiom}\} \\ \lambda \xi \,.\, \xi\, \mathbf{I} & \text{if } A = \text{4th Peano axiom} \end{cases}$$

$$\left(\dfrac{\begin{array}{c}\vdots\, d \\ \Gamma, A \vdash B\end{array}}{\Gamma \vdash A \Rightarrow B}\right)^* = \lambda \xi_A \,.\, d^* \qquad \left(\dfrac{\begin{array}{cc}\vdots\, d_1 & \vdots\, d_2 \\ \Gamma \vdash A \Rightarrow B & \Gamma \vdash A\end{array}}{\Gamma \vdash B}\right)^* = d_1^* d_2^*$$

$$\left(\dfrac{\begin{array}{c}\vdots\, d \\ \Gamma \vdash A\end{array}}{\Gamma \vdash \forall x\, B}\right)^* = d^* \qquad \left(\dfrac{\begin{array}{c}\vdots\, d \\ \Gamma \vdash \forall x\, B\end{array}}{\Gamma \vdash \forall B\{x := e\}}\right)^* = d^*$$

$$\left(\dfrac{\begin{array}{c}\vdots\, d \\ \Gamma \vdash A\end{array}}{\Gamma \vdash \forall X\, B}\right)^* = d^* \qquad \left(\dfrac{\begin{array}{c}\vdots\, d \\ \Gamma \vdash \forall X\, B\end{array}}{\Gamma \vdash \forall B\{X(x_1, \dots, x_n) := A\}}\right)^* = d^*$$

$$\left(\overline{\Gamma \vdash ((A \Rightarrow B) \Rightarrow A) \Rightarrow A}\right)^* = \mathsf{cc}$$

Figure 4: Extraction of a term d^* from a derivation d in PA2

The specific extraction function Given a derivation d (in PA2) of the sequent $U_1, \dots, U_\ell \vdash \bot$ expressing that the experimental theory is contradictory,

we let

$$\tilde{d} \quad := \quad d^*\{\xi_1 := M_{k_1}\mathsf{test}_{U_1};\ \ \ldots;\ \xi_\ell := M_{k_\ell}\mathsf{test}_{U_1}\}$$

where

- $\xi_1, \ldots, \xi_\ell$ are the term variables associated to the experimental hypotheses $U_1, \ldots, U_\ell$ in the derivation d;
- $k_1, \ldots, k_\ell$ are the arities of the hypotheses $U_1, \ldots, U_\ell$, respectively;
- For every $k \geq 0$, M_k is a closed quasi-proof, called a *storage operator of arity* k, that we will define and discuss in subsection 4.4.

We can now state the theorem of experimental effectiveness:

Theorem 1 (Experimental effectiveness) — *If d is a derivation of the sequent $U_1, \ldots, U_\ell \vdash \bot$ (in PA2), then the evaluation of the process $\tilde{d} \star \diamond$ eventually reaches a terminal state of the form*

$$\mathsf{stop} \star \bar{i} \cdot \bar{\nu}_1 \cdots \bar{\nu}_{k_i} \cdot \pi$$

where $i \in [1..\ell]$ is the index of an hypothesis U_i of the experimental theory and where $(\nu_1, \ldots, \nu_{k_i})$ is a falsification of U_i.

Under the assumption of determinism (which is not used in the proof), this theorem tells us several things about the evaluation of the process $\tilde{d} \star \diamond$.

First, it tells us that the evaluation of the process $\tilde{d} \star \diamond$ is finite and reaches a terminal state containing all the parameters of an experimental falsification of the theory $U_1, \ldots, U_\ell$ (including the index of the falsified hypothesis). In particular, test instructions (that are present in the term $\tilde{d}$) are only invoked a finite number of times during the evaluation of $\tilde{d} \star \diamond$.

Second, we know that nothing goes wrong during evaluation, since an error (whose possible forms have been analyzed above) would produce a terminal state whose form would be incompatible with the one that is predicted by the theorem. In particular, the test instruction test_i is always invoked in front of a stack starting with (at least) k_i values $\bar{\nu}_1, \ldots, \bar{\nu}_{k_i}$ defining the particular instance of U_i that is to be tested. It is worth to recall that the outcome of the intermediate process $\mathsf{test}_i \star \bar{\nu}_1 \cdots \bar{\nu}_{k_i} \cdot \pi$ does not only depend on whether the set of parameters $(\nu_1, \ldots, \nu_{k_i})$ corroborates or falsifies the hypothesis U_i, but that it also depends on the truth values of the clauses that compose the tested instance of U_i—as well as on the evaluation policy for test instructions.

The next section is devoted to the proof of Theorem 1.

4 The classical realizability model

The two extraction mechanisms depicted in section 3 are justified by the construction of a classical realizability model following the method introduced by Krivine [20]. Our interpretation of second-order classical arithmetic is completely standard, the sole addition being the (quite natural) interpretation of experimental predicates in this framework.

The big picture The idea of realizability is to associate a set of closed terms $|A| \subseteq \Lambda$ to every (closed) formula A by following the formulæ-as-types interpretation (a.k.a. the Curry-Howard correspondence). The elements of $|A|$ are then called the *realizers* of the formula A. In particular:

- The formula $A \Rightarrow B$ is interpreted as the set of all closed terms that compute a realizer of B when applied to an arbitrary realizer of A. Thus $|A \Rightarrow B|$ can be seen as a function space from $|A|$ to $|B|$.
- The formula $\forall x\, A[x]$ is interpreted as the set of all closed terms that realize the formula $A[n]$ for every $n \in \mathbb{N}$. Thus $|\forall x\, A[x]|$ is the intersection of the sets $|A[n]|$ when n ranges over $\mathbb{N}$. Second-order universal quantification $\forall X A[X]$ is interpreted similarly, with this difference that the intersection is taken on a much larger domain (cf section 4.2).
- The formula $\mathsf{Nat}(n)$ is interpreted as the set of all closed terms that compute the value $\bar{n}$ (i.e. Krivine numeral n) in some way. In classical realizability, the relationship between a realizer of $A(n)$ and the value $\bar{n}$ is not immediate, and the extraction of the latter from the former relies on the notion of a storage operator we will introduce in section 4.4.
- The formula $\forall^{\mathsf{N}} x\, A[x] \equiv \forall x\, (\mathsf{Nat}(x) \Rightarrow A[x])$ is thus interpreted as a mixture of an intersection and a function space, namely, as the set of all terms that compute a realizer of $A[n]$ when applied to an arbitrary realizer of $\mathsf{Nat}(n)$ (i.e. a term computing the value $\bar{n}$ in some way) for every $n \in \mathbb{N}$.[14]

The main property of the generic extraction function $d \mapsto d^*$ defined in section 3.3—which we call the property of *adequacy*—is that it transforms any derivation d of a closed formula A (without hypotheses) into a realizer d^* of the formula A, i.e. $d^* \in |A|$. When the derived formula depends on hypotheses, the property of adequacy more generally expresses that the extracted term d^* defines a realizer of the formula A provided we substitute in d^* all the term-variables attached to the hypotheses by arbitrary realizers of these hypotheses:

> If d is a derivation of $B_1, \dots, B_n \vdash A$ and if $u_i \in |B_i|$ for $i \in [1..n]$,
> then $d^*\{\xi_1 := u_1; \dots; \xi_n := u_n\} \in |A|$.

(Assuming that the formulæ $B_1, \dots, B_n, A$ are closed.)

Through the generic extraction function $d \mapsto d^*$, the deduction rules of PA2 (Fig. 1) can thus be seen as *typing rules* for the program d^*. For instance, the introduction and elimination rules of implication can be seen as the typing rules for abstraction and application:

$$\frac{\begin{array}{c}\vdots\ d\\ \Gamma, \xi : A \vdash d^* : B\end{array}}{\Gamma \vdash \lambda\xi\,.\,d^* : A \Rightarrow B} \qquad \frac{\begin{array}{c}\vdots\ d_1\\ \Gamma \vdash d_1^* : A \Rightarrow B\end{array} \qquad \begin{array}{c}\vdots\ d_2\\ \Gamma \vdash d_2^* : A\end{array}}{\Gamma \vdash d_1^*\, d_2^* : B}$$

(using the notation $\xi : A$ to indicate that ξ is attached to the formula A).

In proof theory, the relationship between typing and realizability is exactly the same as between provability and validity in model theory, and the property of adequacy is the proof-theoretic equivalent of the property of soundness, saying that derivable formulæ (in PA2) are valid in the standard model of PA2.

[14] The numeric universal quantification $\forall^{\mathsf{N}} x\, A[x]$ has thus roughly the same meaning as the dependent product $\Pi x : \mathsf{Nat}\,.\, A[x]$ in type theory and in the calculus of constructions.

Taking the point of view of the opponent The informal presentation above actually describes the point of view of intuitionistic realizability (that is: realizability in HA2). Classical realizability in PA2 follows a similar picture, but with a major twist: the interpretation of formulæ is no more defined (at least directly) from the point of view of terms—the 'defenders' of formulæ—but from the point of view of stacks—the 'opponents' of formulæ. This change of point of view is crucial to interpret classical reasoning principles (excluded middle, Peirce's law, etc.) in terms of control primitives (call/cc).

Formally, every closed formula A of PA2 is interpreted as a set of stacks $\|A\| \subseteq \Pi$ which we call here the *falsity value* of A.[15] Intuitively, the falsity value of A defines the set of all possible ways to 'attack' the formula A: the larger the falsity value $\|A\|$, the falser the formula A. Technically, the falsity value $\|A\|$ is defined by induction on the formula A, and the definition (that will be given in section 4.2) follows the same intuitions as for intuitionistic realizability—but transposed on the side of stacks.

The *truth value* $|A| \subseteq \Lambda$ of the formula A is then defined as the set of all closed terms that can 'resist' (in a sense we shall momentarily precise) to all the attacks by the stacks $\pi \in \|A\|$: the smaller the falsity value $\|A\|$, the larger the truth value $|A|$, and the truer the formula A. Formally, the truth value $|A|$ is defined by

$$|A| \;=\; \{t \in \Lambda \;:\; \forall \pi \in \|A\| \;\; (t \star \pi) \in \perp\!\!\!\perp\}$$

where $\perp\!\!\!\perp$ is a set of processes which we call the *pole* of the model. Intuitively, the pole $\perp\!\!\!\perp$ defines the notion of contradiction in the model (or a particular 'challenge' between terms and stacks), and the equality above expresses that a term t realizes a formula A (i.e. $t \in |A|$) if t successfully contradicts any attack to the formula A (i.e. $t \star \pi \in \perp\!\!\!\perp$ for all $\pi \in \|A\|$).

Of course, the structure of the classical realizability model deeply depends on the choice of the pole $\perp\!\!\!\perp$, and there are as many classical realizability models as there are poles. Although the correctness of the generic extraction function $d \mapsto d^*$ (the property of adequacy we already mentioned above) holds in all classical realizability models [20], we shall see that the correctness of the specific extraction function $d \mapsto \tilde{d}$ only holds w.r.t. a specific pole $\perp\!\!\!\perp_0$ that is defined from the notion of experimental falsification.

4.1 Definitions

Saturated sets A set of processes $\perp\!\!\!\perp \subseteq \Lambda \star \Pi$ is said to be *saturated* (or *closed under anti-evaluation*) if both conditions $p \succ p'$ and $p' \in \perp\!\!\!\perp$ imply $p \in \perp\!\!\!\perp$ for all processes p and p'. In what follows, saturated sets of processes will be used as the poles for constructing classical realizability models. (The condition that the pole is saturated is crucial to establish the property of adequacy.)

Given a saturated set $\perp\!\!\!\perp \subseteq \Lambda \star \Pi$ and a set of stacks $S \subseteq \Pi$ (a falsity value), we denote by $S^{\perp\!\!\!\perp}$ the set of closed terms defined by

$$S^{\perp\!\!\!\perp} \;=\; \{t \in \Lambda \;:\; \forall \pi \in S \;\; (t \star \pi) \in \perp\!\!\!\perp\}\,.$$

[15] In Krivine's works, the set $\|A\|$ is called the truth value of A, but we prefer to adopt here the terminology of *falsity value* to keep the intuition that larger falsity values correspond to falser formulæ. The notion of *falsity value* has to be related with Popper's notion of *falsity contents* for an empirical theory [23, ?], which is the set of all possible (finite conjunctions of) singular statements that are contradictory with the theory.

It is clear from the definition that the operation $S \mapsto S^{\perp\!\!\!\perp}$ is antimonotonic, in the sense that $S_1 \subseteq S_2$ implies $S_2^{\perp\!\!\!\perp} \subseteq S_1^{\perp\!\!\!\perp}$ for all $S_1, S_2 \in \mathfrak{P}(\Pi)$.

In what follows, this operation will be used to transform a falsity value into the corresponding truth value, letting $|A| = \|A\|^{\perp\!\!\!\perp}$.

The pole $\perp\!\!\!\perp_0$ A simple way to define a pole is to start from an arbitrary set of processes $P \subseteq \Lambda \star \Pi$ and to close this set under anti-evaluation, letting

$$\perp\!\!\!\perp \;=\; \{p \in \Lambda \star \Pi \;:\; \exists p' \in P \;\; p \succ^* p'\}\,.$$

Intuitively, the pole $\perp\!\!\!\perp$ generated by P is formed by all the processes that fall in the set P after zero, one or several evaluation steps.

The correctness of the specific extraction function $d \mapsto \tilde{d}$ (cf section 3.3) relies on the pole $\perp\!\!\!\perp_0$ that is precisely defined as the pole generated by all the processes of the form

$$\mathsf{stop} \star \bar{i} \cdot \bar{\nu_1} \cdots \bar{\nu}_{k_i} \cdot \pi$$

where $1 \in [1..\ell]$ and $(\nu_1, \ldots, \nu_{k_i})$ is a falsification of the hypothesis U_i, the stack π being arbitrary. Note that the elements of $\perp\!\!\!\perp_0$ are the processes that evaluate to a terminal state containing a falsification of the experimental theory $U_1, \ldots, U_\ell$ in the form prescribed by Theorem 1. Proving Theorem 1 thus amounts to prove that $(\tilde{d} \star \diamond) \in \perp\!\!\!\perp_0$ for every derivation d of the sequent $U_1, \ldots, U_\ell \vdash \perp$. We can also notice that in the case where the experimental theory $U_1, \ldots, U_\ell$ is valid, the pole $\perp\!\!\!\perp_0$ is empty.[16]

Extending the language of formulæ To define the interpretation of formulæ, it is convenient to enrich the language of formulæ by adding a new predicate symbol $\dot{F}$ of arity k for every falsity value function $F : \mathbb{N}^k \to \mathfrak{P}(\Pi)$:

Formulæ $\qquad A, B \;\; ::= \;\; \cdots \;\; | \;\; \dot{F}(e_1, \ldots, e_k) \qquad\qquad (F : \mathbb{N}^k \to \mathfrak{P}(\Pi))$

(While doing this extension, the cardinality of the language of formulæ jumps from the denumerable to the power of continuum.)

Valuations We call a *valuation* any function ρ that associates a natural number $\rho(x) \in \mathbb{N}$ to every first-order variable x and a falsity value function $\rho(X) : \mathbb{N}^k \to \mathfrak{P}(\Pi)$ to every second-order variable X of arity k.

In what follows, we write $\top$ the formula of the enriched language defined by $\top = \dot{\varnothing}$. Notice that the formula $\top$ (whose truth value will be $|\top| = \varnothing^{\perp\!\!\!\perp} = \Lambda$) is different from the formula $\forall X\,(X \Rightarrow X)$ that we shall write $\mathbf{1}$ in the sequel.

Given an open formula A (possibly in the enriched language) and a valuation ρ, we write $A[\rho]$ the closed formula of the enriched language obtained by replacing in the formula A every free occurrence of a first-order variable x by the natural number $\rho(x) \in \mathbb{N}$ (seen as a numeric expression) and every free occurrence of a second-order variable X of arity k by the predicate symbol $\dot{F}$ associated to the function $F = \rho(X) : \mathbb{N}^k \to \mathfrak{P}(\Pi)$.

[16] In which case the corresponding classical realizability model is isomorphic to the standard (full) model of second-order arithmetic [20].

4.2 Interpreting formulæ

Let $\perp\!\!\!\perp$ be a fixed pole. Every closed formula of the enriched language is interpreted as two sets: a falsity value $\|A\| \subseteq \Pi$ and a truth value $|A| \subseteq \Lambda$. The falsity value $\|A\| \subseteq \Pi$ is defined by induction on the size of the (closed) formula A by

$$\begin{aligned}
\|\dot{F}(e_1,\dots,e_k)\| &= F(\downarrow e_1,\dots,\downarrow e_n)\\
\|p(e_1,\dots,e_k)\| &= \begin{cases}\{t\cdot\pi \ :\ t\star\pi\in\perp\!\!\!\perp\} & \text{if}\quad \mathbf{Val}(p)(\downarrow e_1,\dots,\downarrow e_k)=1\\ \Lambda\cdot\Pi & \text{if}\quad \mathbf{Val}(p)(\downarrow e_1,\dots,\downarrow e_k)=0\end{cases}\\
\|A\Rightarrow B\| &= |A|\cdot\|B\| \ =\ \{t\cdot\pi \ :\ t\in|A|,\ \pi\in\|B\|\}\\
\|\forall X A\| &= \bigcup_{F:\mathbb{N}^k\to\mathfrak{P}(\Pi)} \|A\{X:=\dot{F}\}\|
\end{aligned}$$

whereas the truth value $|A| \subseteq \Lambda$ is defined by

$$|A| \ =\ \|A\|^{\perp\!\!\!\perp} \ =\ \{t\in\Lambda \ :\ \forall\pi\in\|A\| \ \ (t\star\pi)\in\perp\!\!\!\perp\}\,.$$

Since the two sets $|A|$ and $\|A\|$ associated to the formula A depend on the pole $\perp\!\!\!\perp$, we will sometimes write them $|A|_{\perp\!\!\!\perp}$ and $\|A\|_{\perp\!\!\!\perp}$ to indicate this dependency explicitly. Given a closed term t and a closed formula A, we say that t *realizes* A and write $t \Vdash A$ when $t \in |A|$ (relatively to a fixed pole $\perp\!\!\!\perp$). In the case where $\perp\!\!\!\perp = \perp\!\!\!\perp_0$, we write $t \Vdash_0 A$ for $t \in |A|_{\perp\!\!\!\perp_0}$.

Finally, we say that a closed term t is a *universal realizer* of a formula A when $t \in |A|_{\perp\!\!\!\perp}$ for every pole $\perp\!\!\!\perp$. We shall see in section 4.3 that the generic extraction function associates a universal realizer d^* of the formula A to every derivation d of the formula A (without hypotheses).

Remarks Putting aside the realizability interpretation of experimental predicates, the definition above is the standard interpretation of the formulæ of second-order arithmetic in classical realizability [20]. In particular:

- The falsity value of the implication $A \Rightarrow B$ is defined as the set of all stacks of the form $t\cdot\pi$, where t defends A and π attacks B. Intuitively, these stacks are the natural opponents of the terms of type $A \to B$, that expect an argument of type A (here: $t \in |A|$) and return a result of type B that will contradict the rest of the stack (here: $\pi \in \|B\|$). From the point of view of logic, the falsity value $\|A \Rightarrow B\| = |A| \cdot \|B\|$ corresponds to the conjunction $A \wedge \neg B$, the negation of the implication $A \Rightarrow B$.

- The falsity value of the formula $\forall x\, A$ is defined as the union of all falsity values $\|A\{x := n\}\|$ where $n \in \mathbb{N}$. In terms of truth values, we have

$$|\forall x\, A| = \Big(\bigcup_{n\in\mathbb{N}} \|A\{x:=n\}\|\Big)^{\perp\!\!\!\perp} = \bigcap_{n\in\mathbb{N}} \|A\{x:=n\}\|^{\perp\!\!\!\perp} = \bigcap_{n\in\mathbb{N}} |A\{x:=n\}|\,,$$

  which is the expected interpretation of first-order universal quantification. The same remark applies for second-order universal quantification.

To understand the interpretation of the atomic formulæ formed from the experimental predicates, let us first notice that

Lemma 3 (Interpretation of Leibniz equality) — *If e_1 and e_2 are two closed numeric expressions, then*

$$\|e_1 = e_2\| \;=\; \begin{cases} \{t\cdot\pi \;:\; t\star\pi\in\perp\!\!\!\perp\} \;=\; \|\mathbf{1}\| & \text{if } \downarrow e_1 = \downarrow e_2 \\ \Lambda\cdot\Pi \;=\; \|\top\Rightarrow\bot\| & \text{if } \downarrow e_1 \neq \downarrow e_2 \end{cases}$$

This lemma expresses that true equalities are interpreted the same way as the (obviously true) formula $\mathbf{1} \equiv \forall X\,(X \Rightarrow X)$ whereas false equalities are interpreted the same way as the (obviously false) formula $\top \Rightarrow \bot$ (where $\top = \dot{\varnothing}$).

The characterization of the falsity values of (true and false) equalities is the key of the realizability interpretation of experimental predicates: true experimental atomic formulæ are interpreted the same way as true equalities whereas false experimental atomic formulæ are interpreted the same way as false equalities. So that we can treat both forms of elementary formulæ (i.e. experimental atomic formulæ and equalities) the very same way in the realizability model:

Fact 4 (Interpretation of elementary formulæ) — *If E is a closed elementary formula, then*

$$\|E\| \;=\; \begin{cases} \{t\cdot\pi \;:\; t\star\pi\in\perp\!\!\!\perp\} \;=\; \|\mathbf{1}\| & \text{if } \mathbf{Val}(E) = 1 \\ \Lambda\cdot\Pi \;=\; \|\top\Rightarrow\bot\| & \text{if } \mathbf{Val}(E) = 0 \end{cases}$$

4.3 Adequacy

We can now state the property of adequacy for the generic extraction function $d \mapsto d^*$ defined in Fig. 4:

Proposition 5 (Adequacy) — *If d is a derivation of $B_1,\dots,B_n \vdash A$ in PA2 (using the rules of Fig. 1), then for all valuations ρ and for all closed terms $u_1,\dots,u_n$ such that $u_1 \Vdash B_1$, …, $u_n \Vdash B_n$, we have*

$$d^*\{\xi_1 := u_1;\dots;\xi_n := u_n\} \Vdash A$$

where $\xi_1,\dots,\xi_n$ are the term-variables attached to the hypotheses $B_1,\dots,B_n$.

Proof. The proof is done by structural induction on the derivation d. The cases corresponding to purely logical rules is standard (see for instance [17, 20]). Regarding the axioms of PA2, it suffices to check that the term $\mathbf{I} = \lambda\xi\,.\,\xi$ is a realizer of Peano's third axiom (injectivity of successor) and of all defining equalities of primitive recursive function symbols, whereas the term $\lambda\xi\,.\,\xi\mathbf{I}$ is a realizer of Peano's fourth axiom (non surjectivity). □

Notice that the property of adequacy does not depend on the choice of the pole $\perp\!\!\!\perp$. In particular, when d is the derivation of a closed formula A (without hypotheses), the closed quasi-proof d^* is a universal realizer of A.

4.4 Storage operators

It is easy to check that for all $n \in \mathbb{N}$, the quasi-proof $\bar{n}$ (Krivine numeral n) is a universal realizer of the formula $\mathsf{Nat}(n)$. A simple way to prove it is to build (by induction on n) a derivation d_n of the formula $\mathsf{Nat}(n)$ in PA2 such that $d_n^* \equiv \bar{n}$,

and to conclude using Prop. 5 [20]. On the other hand, the formula $\mathsf{Nat}(n)$ has many more realizers than the value $\bar{n}$.

To understand how it is possible to extract the value $\bar{n}$ from an arbitrary realizer of $\mathsf{Nat}(n)$, we extend the language of formulæ with a new syntactic construct written $\{e\} \Rightarrow B$ where e is a numeric expression and where B is an arbitrary formula of the language:

Formulæ $\qquad A, B \quad ::= \quad \cdots \quad | \quad \{e\} \Rightarrow B$

We then extend the realizability interpretation of section 4.2 by letting

$$\|\{e\} \Rightarrow B\| \quad = \quad \overline{\downarrow e} \cdot \|B\|$$

in the case where both e and B are closed.

From this definition, it is clear that $\|\{e\} \Rightarrow B\| \subseteq \|\mathsf{Nat}(e) \Rightarrow B\|$ (hence the inclusion of truth values $|\mathsf{Nat}(e) \Rightarrow B| \subseteq |\{e\} \Rightarrow B|$), which means that the formula $\mathsf{Nat}(e) \Rightarrow B$ can be seen as a subtype of the formula $\{e\} \Rightarrow B$. Intuitively, this expresses that any term that is able to produce a realizer of B when applied to an arbitrary realizer of $\mathsf{Nat}(e)$ is also able to produce a realizer of B when applied to the particular realizer $\bar{n} \Vdash \mathsf{Nat}(e)$, where $n = \downarrow e$. Consequently, the identity term $\lambda\phi \,.\, \phi$ is a universal realizer of the implication

$$\forall x \; \forall Z \; ((\mathsf{Nat}(x) \Rightarrow Z) \Rightarrow (\{x\} \Rightarrow Z)) \,.$$

By definition, we call a *storage operator* (of arity 1) any closed quasi-proof M that is a universal realizer of the converse implication

$$\forall x \; \forall Z \; ((\{x\} \Rightarrow Z) \Rightarrow (\mathsf{Nat}(x) \Rightarrow Z)) \,.$$

Intuitively, a storage operator is a function[17] M that takes a realizer of $\{n\} \Rightarrow Z$ (for an arbitrary $n \in \mathbb{N}$) and a realizer of $\mathsf{Nat}(n)$, extracts the value $\bar{n}$ from the latter and then passes this value to the realizer of $\{n\} \Rightarrow Z$ to produce a Z. (In practice the computation can be much more complex than this simple picture due to the presence of continuations in the realizers, but the basic intuition is important to understand how storage operators work.)

More generally, a closed quasi-proof M_k is called a *storage operator of arity k* ($k \geq 0$) if it is a universal realizer of the formula

$$\begin{array}{l} \forall x_1 \cdots \forall x_k \; \forall Z \; \big((\{x_1\} \Rightarrow \cdots \Rightarrow \{x_k\} \Rightarrow Z) \;\Rightarrow \\ \qquad\qquad\qquad\qquad \mathsf{Nat}(x_1) \Rightarrow \cdots \Rightarrow \mathsf{Nat}(x_k) \Rightarrow Z \big) \,. \end{array}$$

It can be shown [17, 20] that such quasi-proofs M_k exist for all arities $k \geq 0$. For instance, we can take

$$\begin{array}{lcll} M_0 & \equiv & \lambda\phi \,.\, \phi & \\ M_1 & \equiv & \lambda\phi\nu \,.\, \nu \, \phi \, (\lambda\psi\xi \,.\, \psi \, (\bar{s} \, \xi)) \, \bar{0} & \\ M_k & \equiv & \lambda\phi \,.\, M_1(\lambda\nu \,.\, M_{k-1}(\phi \, \nu)) & (k \geq 2) \end{array}$$

Lemma 6 — *If M_k is a storage operator of arity k ($k \geq 0$), then for every formula $A[x_1, \ldots, x_k]$ depending on k first-order variables $x_1, \ldots, x_k$ we have*

$$\begin{array}{lcl} M_k & \Vdash & \forall x_1 \cdots \forall x_k \; (\{x_1\} \Rightarrow \cdots \Rightarrow \{x_k\} \Rightarrow A[x_1, \ldots, x_k]) \\ & \Rightarrow & \forall^{\mathsf{N}} x_1 \cdots \forall^{\mathsf{N}} x_k \; A[x_1, \ldots, x_k] \end{array}$$

[17] In programming, such a function is called a 'wrapper'.

Proof. It suffices to notice that the formula that specifies storage operators of arity k is a subtype of the formula above. □

In the sequel, we shall use the lemma above to show that storage operators transform each test instruction test_i into a realizer of the corresponding hypothesis U_i (w.r.t. the pole $\perp\!\!\!\perp_0$).

4.5 Evaluating test instructions

Let us now study the behaviour of the instruction test_i ($i \in [1..\ell]$) associated to an experimental hypothesis $U_i \equiv \forall^{\mathsf{N}} x_1 \cdots \forall^{\mathsf{N}} x_k \; T[x_1, \ldots, x_k]$ where

- $T[x_1, \ldots, x_k] \;\equiv\; L_1[x_1, \ldots, x_k] \Rightarrow \cdots \Rightarrow L_n[x_1, \ldots, x_k] \Rightarrow R[x_1, \ldots, x_k]$.
- $L_j[x_1, \ldots, x_k]$ is an elementary formula or its negation (for $j \in [1..n]$).
- $R[x_1, \ldots, x_k]$ is an elementary formula or the formula $\bot$.

When the instruction test_i is evaluated in front of an arbitrary stack π_0 (cf the corresponding evaluation rules in Fig. 3), it first checks whether the stack π_0 is of the form $\pi_0 \equiv \bar{\nu}_1 \cdots \bar{\nu}_k \cdot \pi$ or not. If this is not the case (either because π_0 contains less than k elements or because one of the first k elements of π_0 is not a Krivine numeral), no evaluation rule applies and the process $\mathsf{test}_i \star \pi_0$ is a terminal state. If π_0 is of the form $\pi_0 \equiv \bar{\nu}_1 \cdots \bar{\nu}_k \cdot \pi$, then either:

1. $\mathsf{test}_i \star \bar{\nu}_1 \cdots \bar{\nu}_k \cdot \pi$ evaluates to $(\lambda \xi_1 \cdots \xi_n \,.\, \xi_j \, \mathbf{I}) \star \pi$ for some $j \in [1..n]$ because $\mathbf{Val}(L_j[\nu_1, \ldots, \nu_k]) = 0$ (falsity of the jth premise);
2. $\mathsf{test}_i \star \bar{\nu}_1 \cdots \bar{\nu}_k \cdot \pi$ evaluates to $(\lambda \xi_1 \cdots \xi_n \,.\, \mathbf{I}) \star \pi$ because $\mathbf{Val}(R[\nu_1, \ldots, \nu_k]) = 1$ (truth of the conclusion); or
3. $\mathsf{test}_i \star \bar{\nu}_1 \cdots \bar{\nu}_k \cdot \pi$ evaluates to $\mathsf{stop} \star \bar{i} \cdot \bar{\nu}_1 \cdots \bar{\nu}_k \cdot \pi$ because $\mathbf{Val}(T[\nu_1, \ldots, \nu_k]) = 0$ (falsity of the implication).

The reason for putting in head position the magic terms $\lambda \xi_1 \cdots \xi_n \,.\, \xi_j \, \mathbf{I}$ (when the jth premise $L_j[\nu_1, \ldots, \nu_k]$ is false) and $\lambda \xi_1 \cdots \xi_n \,.\, \mathbf{I}$ (when the conclusion $R[\nu_1, \ldots, \nu_k]$ is true) is revealed by the following lemma:

Lemma 7 — *Let* $(\nu_1, \ldots, \nu_k) \in \mathbb{N}^k$ *such that* $\mathbf{Val}(T[\nu_1, \ldots, \nu_k]) = 1$.

1. *In the case where* $\mathbf{Val}(L_j[\nu_1, \ldots, \nu_k]) = 0$ *for some* $j \in [1..\ell]$, *the quasi-proof* $\lambda \xi_1 \cdots \xi_n \,.\, \xi_j \, \mathbf{I}$ *is a universal realizer of the formula* $T[\nu_1, \ldots, \nu_k]$.
2. *In the case where* $\mathbf{Val}(R[\nu_1, \ldots, \nu_k]) = 1$, *the quasi-proof* $\lambda \xi_1 \cdots \xi_n \,.\, \mathbf{I}$ *is a universal realizer of the formula* $T[\nu_1, \ldots, \nu_k]$.

Proof. Let $\perp\!\!\!\perp$ be a fixed pole. We distinguish the following two cases:

1. $\mathbf{Val}(L_j[\nu_1, \ldots, \nu_k]) = 0$ for some $j \in [1..n]$. We distinguish two cases depending on whether $L_j[x_1, \ldots, x_k]$ is an elementary formula or the negation of an elementary formula.

- $L_j[x_1,\ldots,x_k] \equiv E[x_1,\ldots,x_k]$ where $E[x_1,\ldots,x_k]$ is an elementary formula. Since $\mathbf{Val}(E[\nu_1,\ldots,\nu_k]) = 0$, we know from Fact 4 that

$$\|L_j[\nu_1,\ldots,\nu_k]\| \;=\; \|E[\nu_1,\ldots,\nu_k]\| \;=\; \|\top \Rightarrow \bot\| \;=\; \Lambda\cdot\Pi\,.$$

  To show that $\lambda\xi_1\cdots\xi_n\,.\,\xi_j\,\mathbf{I}$ realizes the implication $T[\nu_1,\ldots,\nu_k]$, let us consider realizers $u_1 \in |L_1[\nu_1,\ldots,\nu_k]|,\ldots,u_n \in |L_n[\nu_1,\ldots,\nu_k]|$ as well as a stack $\pi \in \|R[\nu_1,\ldots,\nu_k]\|$, and let us prove that the process $(\lambda\xi_1\cdots\xi_n\,.\,\xi_j\,\mathbf{I})\star u_1\cdots u_n\cdot\pi$ belongs to the pole $\perp\!\!\!\perp$. We have

$$(\lambda\xi_1\cdots\xi_n\,.\,\xi_j\,\mathbf{I})\star u_1\cdots u_n\cdot\pi \;\succ^*\; u_j\star\mathbf{I}\cdot\pi\,.$$

  But since $(\mathbf{I}\cdot\pi)\in(\Lambda\cdot\Pi) = \|L_j[\nu_1,\ldots,\nu_k]\|$, we have $u_j\star\mathbf{I}\cdot\pi\in\perp\!\!\!\perp$ and thus $(\lambda\xi_1\cdots\xi_n\,.\,\xi_j\,\mathbf{I})\star u_1\cdots u_n\cdot\pi\in\perp\!\!\!\perp$ by anti-evaluation.

- $L_j[x_1,\ldots,x_k] \equiv \neg E[x_1,\ldots,x_k]$ where $E[x_1,\ldots,x_k]$ is an elementary formula. Since $\mathbf{Val}(E[\nu_1,\ldots,\nu_k]) = 1$, we know from Fact 4 that

$$\|L_j[\nu_1,\ldots,\nu_k]\| \;=\; \|\neg E[\nu_1,\ldots,\nu_k]\| \;=\; \|\mathbf{1}\Rightarrow\bot\| \;=\; |\mathbf{1}|\cdot\Pi\,.$$

  The rest of the proof is similar to the above case, noticing that $\mathbf{I}\in|\mathbf{1}|$.

2. $\mathbf{Val}(R[\nu_1,\ldots,\nu_k]) = 1$. In this case, we know from Fact 4 that

$$\|R[\nu_1,\ldots,\nu_k]\| \;=\; \|\mathbf{1}\|\,.$$

   To show that $\lambda\xi_1\cdots\xi_n\,.\,\mathbf{I}$ realizes the implication $T[\nu_1,\ldots,\nu_k]$, let us consider realizers $u_1\in|L_1[\nu_1,\ldots,\nu_k]|,\ldots,u_n\in|L_n[\nu_1,\ldots,\nu_k]|$ as well as a stack $\pi\in\|R[\nu_1,\ldots,\nu_k]\|$, and let us prove that the process $(\lambda\xi_1\cdots\xi_n\,.\,\mathbf{I})\star u_1\cdots u_n\cdot\pi$ belongs to the pole $\perp\!\!\!\perp$. We have

$$(\lambda\xi_1\cdots\xi_n\,.\,\xi_j\,\mathbf{I})\star u_1\cdots u_n\cdot\pi \;\succ^*\; \mathbf{I}\star\pi\,.$$

   But since $\mathbf{I}\in|\mathbf{1}| = |R[\nu_1,\ldots,\nu_k]|$, we have $\mathbf{I}\star\pi\in\perp\!\!\!\perp$, and the desired result follows from the usual argument of anti-evaluation.

The lemma above shows that in all cases where the implication $T[\nu_1,\ldots,\nu_k]$ is true, the instruction test_i consumes the parameters $\bar\nu_1,\ldots,\bar\nu_k$ on the top of the stack and puts in head position a universal realizer of the implication $T[\nu_1,\ldots,\nu_k]$ (which realizer depends on the way the implication is true). Hence:

Lemma 8 — *For all k-tuples of parameters $(\nu_1,\ldots,\nu_k)\in\mathbb{N}^k$ such that the implication $T[\nu_1,\ldots,\nu_k]$ is true, the instruction test_i is a universal realizer of the formula*

$$\{\nu_1\}\Rightarrow\cdots\Rightarrow\{\nu_k\}\Rightarrow T[\nu_1,\ldots,\nu_k]\,.$$

Proof. Let $\perp\!\!\!\perp$ be a fixed pole. To show that test_i realizes the formula above, consider a stack $\pi\in\|T[\nu_1,\ldots,\nu_k]\|$ and let us show that $\mathsf{test}_i\star\bar\nu_1\cdots\bar\nu_k\cdot\pi\in\perp\!\!\!\perp$. From the assumption that $\mathbf{Val}(T[\nu_1,\ldots,\nu_k]) = 1$ combined with the evaluation rules of the instruction test_i, we know that

$$\mathsf{test}_i\star\bar\nu_1\cdots\bar\nu_k\cdot\pi \;\succ\; t\star\pi$$

for some quasi-proof t that realizes the formula $T[\nu_1,\ldots,\nu_k]$ (Lemma 7). Hence $t\star\pi\in\perp\!\!\!\perp$ and thus $\mathsf{test}_i\star\bar\nu_1\cdots\bar\nu_k\cdot\pi\in\perp\!\!\!\perp$ by anti-evaluation. □

On the other hand, the instruction test_i is *not* a universal realizer of the formula $\{\nu_1\} \Rightarrow \cdots \Rightarrow \{\nu_k\} \Rightarrow T[\nu_1,\ldots,\nu_k]$ when $\mathbf{Val}(T[\nu_1,\ldots,\nu_k]) = 0$.[18] To ensure that the instruction test_i still realizes the desired formula when $\mathbf{Val}(T[\nu_1,\ldots,\nu_k]) = 0$, we have to focus on a particular pole that has been especially tailored for this task: the pole $\perp\!\!\!\perp_0$ defined in section 4.1:

Lemma 9 — *In the realizability model induced by the pole $\perp\!\!\!\perp_0$:*

$$\mathsf{test}_i \ \Vdash_0 \ \{\nu_1\} \Rightarrow \cdots \Rightarrow \{\nu_k\} \Rightarrow T[\nu_1,\ldots,\nu_k]$$

for all k-tuples of parameters $(\nu_1,\ldots,\nu_k) \in \mathbb{N}^k$.

Proof. This is clear from Lemma 8 in the case where $\mathbf{Val}(T[\nu_1,\ldots,\nu_k]) = 1$, so let us assume that $\mathbf{Val}(T[\nu_1,\ldots,\nu_k]) = 0$. To show that test_i realizes the formula above, consider a stack $\pi \in \|T[\nu_1,\ldots,\nu_k]\|$ and let us show that the process $\mathsf{test}_i \star \bar{\nu}_1 \cdots \bar{\nu}_k \cdot \pi$ belongs to the pole $\perp\!\!\!\perp_0$. Since the testable formula $T[\nu_1,\ldots,\nu_k]$ is false (from our assumption), we have

$$\mathsf{test}_i \star \bar{\nu}_1 \cdots \bar{\nu}_k \cdot \pi \ \succ \ \mathsf{stop} \star \bar{i} \cdot \bar{\nu}_1 \cdots \bar{\nu}_k \cdot \pi \ \in \ \perp\!\!\!\perp_0$$

(from the definition of $\perp\!\!\!\perp_0$), hence $\mathsf{test}_i \star \bar{\nu}_1 \cdots \bar{\nu}_k \cdot \pi \in \perp\!\!\!\perp_0$. □

From this lemma we immediately deduce the proposition:

Proposition 10 (Realization of the experimental hypothesis U_i) — *In the realizability model induced by the pole $\perp\!\!\!\perp_0$, the instruction test_i is a realizer of the formula $\forall x_1 \cdots \forall x_k \ (\{x_1\} \Rightarrow \cdots \Rightarrow \{x_k\} \Rightarrow T[x_1,\ldots,x_k])$, hence*

$$M_k\,\mathsf{test}_i \quad \Vdash_0 \quad \forall^{\mathsf{N}} x_1 \cdots \forall^{\mathsf{N}} x_k \ T[x_1,\ldots,x_k] \quad \equiv \quad U_i\,,$$

where M_k is a storage operator of arity k (by Lemma 6).

We can now complete the

Proof of Theorem 1. Consider a derivation d of the sequent $U_1,\ldots,U_\ell \vdash \perp$ (in PA2). From Prop. 5 (Adequacy) we know that

$$\tilde{d} \ \equiv \ d^*\{\xi_1 := M_{k_1}\mathsf{test}_i;\ldots;\xi_\ell := M_{k_\ell}\mathsf{test}_\ell\} \ \Vdash_0 \ \perp$$

since $M_{k_i}\mathsf{test}_i \Vdash_0 U_i$ for all $i \in [1..\ell]$ (by Prop. 10). Since $\diamond \in \|\perp\| = \Pi$, we have $\tilde{d} \star \diamond \in \perp\!\!\!\perp_0$, which means that $\tilde{d} \star \diamond$ eventually reaches a terminal state containing a falsification of the experimental theory $U_1,\ldots,U_\ell$. □

4.6 The experimental modus tollens

Let us now consider an experimental theory formed with testable universal formulæ $U_1,\ldots,U_\ell$ as well as a testable universal formula V that is a consequence of the theory $U_1,\ldots,U_\ell$ in PA2.

[18] Actually, the formula $\{\nu_1\} \Rightarrow \cdots \Rightarrow \{\nu_k\} \Rightarrow T[\nu_1,\ldots,\nu_k]$ has no universal realizer in the case where $\mathbf{Val}(T[\nu_1,\ldots,\nu_k]) = 0$. This is due to the fact that when the pole $\perp\!\!\!\perp$ is empty, the truth value of a formula is inhabited if and only if this formula is true in the full standard model of PA2 [20]. A formlula which is false in the full standard model of PA2 (which is the case of the formula we consider here) has thus no universal realizer.

Theorem 2 (Experimental modus tollens) — *From a derivation (in PA2) of the sequent $U_1, \ldots, U_\ell \vdash V$ and a falsification of the formula V, it is possible to construct a λ_c-term t such that the evaluation of the process $t \star \diamond$ eventually reaches a terminal state containing a falsification of one of the formulæ $U_1, \ldots, U_\ell$ in the sense of Theorem 1.*

Proof. Let us write $V \equiv \forall^{\mathsf{N}} x_1 \cdots \forall^{\mathsf{N}} x_k \; T[x_1, \ldots, x_k]$ where

- $T[x_1, \ldots, x_k] \;\equiv\; L_1[x_1, \ldots, x_k] \Rightarrow \cdots \Rightarrow L_n[x_1, \ldots, x_k] \Rightarrow R[x_1, \ldots, x_k]$;
- $L_i[x_1, \ldots, x_k]$ is either an elementary formula or its negation ($i \in [1..n]$);
- $R[x_1, \ldots, x_k]$ is an elementary formula. (If $R[x_1, \ldots, x_k]$ is the formula $\bot$, we can replace it by the provably equivalent equality $0 = 1$.)

From the derivation of the sequent $U_1, \ldots, U_\ell \vdash V$, we easy build (in PA2) a derivation d of the sequent

$$U_1, \ldots, U_\ell,\; L_1[\nu_1, \ldots, \nu_k],\; \ldots,\; L_n[\nu_1, \ldots, \nu_k],\; \neg R[\nu_1, \ldots, \nu_k] \;\vdash\; \bot$$

Applying Theorem 1 to the above sequent (whose hypotheses are testable universal formulæ), we know that the evaluation of the process $\tilde{d} \star \diamond$ eventually reaches a finite state containing a falsification (in the sense of Theorem 1) of the hypotheses of the sequent, that is

1. either a falsification of one of the formulæ $U_1, \ldots, U_\ell$;
2. either a falsification of one of the formulæ $L_i[\nu_1, \ldots, \nu_k]$ $(1 \leq i \leq n)$;
3. either a falsification of the formula $\neg R[\nu_1, \ldots, \nu_k]$.

But since $\mathbf{Val}(T[\nu_1, \ldots, \nu_k]) = 0$, we know that $\mathbf{Val}(L_i[\nu_1, \ldots, \nu_k]) = 1$ for all $1 \leq i \leq n$ and that $\mathbf{Val}(R[\nu_1, \ldots, \nu_k]) = 0$. Hence cases 2 and 3 are impossible. Therefore the falsification coming from the evaluation of $\tilde{d} \star \diamond$ is a falsification of one of the formulæ $U_1, \ldots, U_\ell$ (case 1). □

5 Applications

5.1 Extraction of an Herbrand tree

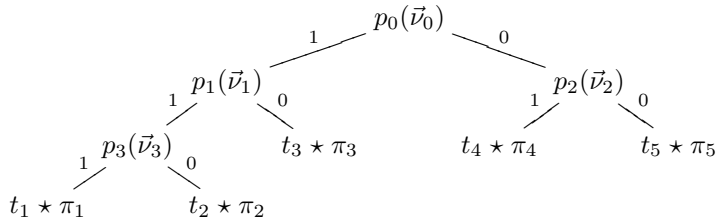
There are essentially two ways of executing the extracted program $\tilde{d}$ such as given by Theorem 1 or Theorem 2: the *sequential mode*, and the *treelike mode*.

Sequential execution In the sequential execution mode, the process $\tilde{d} \star \diamond$ is evaluated linearly, step by step. Each time a test instruction is invoked with a given set of parameters (corresponding to a testable singular formula), the investigator performs a finite sequences of experiments to determine whether the formula under consideration is true or not. Depending on the outcome of the test, the evaluation may stop on an explicit falsification of the theory, or it may continue using the suitable evaluation rule. From Theorem 1 we know that evaluation eventually terminates on an explicit falsification of the theory.

However, this execution mode relies on a strong assumption, which is that all the experiments proposed during the evaluation of the process $\tilde{d} \star \diamond$ can

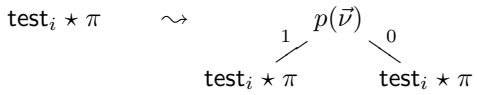
be performed in practice, which may be unrealistic in some circumstances. To circumvent this difficulty, it is possible to perform the execution without doing a single experiment by using an alternative method which is the following.

Treelike execution and construction of an Herbrand tree The idea of the treelike execution mode is to consider all the possible outcomes of the intermediate experiments proposed during the evaluation of the process $\tilde{d} \star \pi$ (without actually performing these experiments) by organizing the computation into a finite binary tree whose inner nodes are labeled with atomic formulæ of the form $p(\vec{\nu})$ (the intermediate experiments) and whose leaves are labelled with processes:



Every process that is attached to a leaf of the tree is evaluated (independently from the other processes) by considering the partial truth value function given by the corresponding path to the root of the tree. In the above picture for instance, the process $t_3 \star \pi_3$ is evaluated by considering the partial truth value function that maps $p_0(\vec{\nu}_0)$ to 1 and $p_1(\vec{\nu}_1)$ to 0, the truth values of all the other atomic formulæ being undefined. Note that all these processes are independent, so that we can evaluate them in any order (or in parallel).

When at a given leaf a test instruction needs to know the truth value of an atomic formula $p(\vec{\nu})$ that is not given by the current path, the current leaf is replaced by an inner node with label $p(\vec{\nu})$ whose children are two copies of the current process:



This mechanism of duplication repeats until the current path contains enough information to determine how to evaluate the current test instruction.[19]

By fully evaluating the process $\tilde{d} \star \diamond$ in this way (seeing this process as a tree with exactly one leaf) we thus get a possibly infinite binary tree, as the limit of the computation. But from Theorem 1 it is clear that this limit tree has no infinite branches, so that it is actually finite (by Koenig's lemma). Moreover, every leaf of this limit tree is labelled by a terminal process that contains the parameters of a falsification of the corresponding branch. As a result of this (finite) treelike computation, we thus get an Herbrand tree falsifying the experimental theory as a whole.

Let us insist on the fact that this way of executing the process $\tilde{d} \star \diamond$ does not require to perform a single experiment. However, the price to pay is that

[19] Considering the way test instructions are evaluated (section 4.5), the complete evaluation of a test instruction may insert up to n inner nodes, where n is the number of atomic formulæ of the form $p(\vec{\nu})$ contained in the closed testable formula that is currently tested.

the result is not given as a single falsification, but as a finite set of potential falsifications attached to the leaves of the resulting Herbrand tree. We can then decide to accept the Herbrand tree itself as a falsification, or we can give this tree to the investigator, who will use it as a binary decision tree to guide her in the choice of crucial experiments.

A remark about efficiency It is interesting to compare the method described above with Herbrand's theorem, that precisely states that a contradictory experimental theory[20] has an Herbrand tree which falsifies it. The standard proof of this result is the following: consider a fixed enumeration $(A_n)_{n\in\mathbb{N}}$ of all atomic formulæ of the form $p(\vec{\nu})$ (i.e. all the possible experiments) and build the infinite binary tree whose inner nodes at depth n are labelled with the formula A_n. Using a similar argument as above, we can cut every infinite branch of this tree at the point where the information given by the current path is sufficient to falsify the theory. In this way, we get a finite binary tree whose leaves correspond to falsifications of the corresponding branches.

However, the tree given by the proof of Herbrand's theorem is based on a blind enumeration of all the possible experiments. Intuitively, this tree corresponds to an investigator that would sequentially perform all the possible experiments in a random order—without taking into account the outcomes of the already performed experiments—until she would acquire enough information to conclude that the theory has been falsified.

Instead, our method proposes to use the *proof* of inconsistency as a guide to chose the sequence of crucial experiments, using the outcomes of the already performed experiments to determine the next experiment. Even in the worst case, we can reasonably think that a method that is directly guided by clever arguments contained in a formal mathematical proof will be not less efficient than the blind method suggested by Herbrand's result.[21]

5.2 An application to software certification

The development of proof assistants such as Coq [24, 2] has shown the practical relevance of formal proofs in the perspective of software certification. The idea is to completely formalize the correctness proof of a piece of software by using a proof assistant, and then to let the system check that the proof is well formed and exhaustive. If we trust the proof checker, then we can trust the software whose correctness proof has been checked by the machine.

However, the certification of a piece of software can never be absolute, since it always relies on explicit or implicit assumptions about the correctness of the low-level components upon which the software is implemented. In the case where a certified piece of software P fails—because one of its low-level components does not meet its specification—we can use our methodology to combine the correctness proof of P (that depends on assumptions $U_1, \ldots, U_\ell$ expressing the correctness of its low-level components) with the falsification of the proved specification V (i.e. a 'bug report' for P) in order to extract a program that produces a series of tests for the low-level components, which is entirely guided

[20] In the sense that it has no model.

[21] We tested our method with several toy examples. For some of them, the method directly produces an Herbrand tree of minimal size. With the same examples, the blind method can produce arbitrarily large trees, depending on the enumeration of all possible experiments.

by the proof. In this way, we get a 'static debugger' that is able to track the defective low-level component without re-executing the certified piece of code. The only requirement for applying our method is that both the global specification V of the software P and the specifications $U_1, \ldots, U_\ell$ of its low-level components should be universal testable formulæ in the sense of section 2.4. On the other hand, the method is especially well-suited to the case where some of the low-level components are black boxes (typically: external devices, processing units, etc.) with no clear mathematical definition.

6 Conclusion

In this paper, we have shown how to transform a formal inconsistency proof of a given experimental theory into a computer program that performs a series of tests on the theory until it reaches an explicit falsification. With little modification the same method can be used to extract a program that solves the problem of the experimental modus tollens discussed in section 1.3.

We presented our results in the case where proofs are formalized in classical second-order arithmetic (PA2), also known as *classical analysis*[22], but the same methodology works when proofs are formalized in stronger theories (Zermelo-Fraenkel set theory or the calculus of constructions with universes), using suitable extensions [18, 22] of the realizability model of section 4.

The existence of the aforementioned procedures strongly argues in favor of what we see as the distinctive feature of mathematical reasoning: the fact that it is effectively accountable for the failure of a (testable) prediction. More than philosophy and religion, the mathematical discourse involves a deep nesting of abstractions tending to produce conceptual entities that seem to be alien to our sensitive experience. But unlike many other forms of reasoning, the intensive use of abstractions in mathematics does not break the empirical connection between premises and conclusions, since it is always possible to keep track of an experimental falsification through an entire mathematical proof.

However, the correctness of the procedure given by Theorem 1 relies on a model-theoretic construction that requires a strong mathematical framework involving programs and infinite sets of programs (see section 4). The fact that the extracted program is correct and terminates as described by Theorem 1 relies on implicit hypotheses about mathematics we now need to determine. In other words, we need to determine the proof-theoretic strength of Theorem 1.

From now on, we take the statement of Theorem 1 as an assumption about the computational behavior of the programs that are extracted from derivations in PA2—an assumption which we call the *hypothesis of experimental effectiveness for PA2*—and investigate some of its consequences.

6.1 Experimental effectiveness and consistency

A mathematical theory is *logically consistent* when the formula $\bot$ representing absurdity cannot be derived in the theory. It is easy to check that:

[22] In PA2, unary second-order variables denote sets of natural numbers that can be used to represent (using standard coding techniques) any kind of continuous entities: real numbers, complex numbers, continuous functions, Borel sets, etc.

Proposition 11 — *The hypothesis of experimental effectiveness for PA2 implies the logically consistency of PA2.*

Proof. Let us assume that the sequent $\vdash \bot$ has a derivation d in PA2. This assumption precisely means that the empty experimental theory (i.e. $\ell = 0$) is inconsistent in PA2. From the hypothesis of experimental effectiveness for PA2, the execution of the process $\tilde{d} \star \diamond$ eventually reaches an experimental falsification of a particular formula U_i ($1 \leq i \leq \ell$) of the experimental theory. But this is impossible since $\ell = 0$, hence the assumption is absurd. □

It is important to understand that the above 'proof' is completely elementary, and that it hardly uses any argument that is specific to mathematics. (In this aspect, it is very different from the proof of Theorem 1 such as presented in section 4.) A similar argument could be used in any scientific field and even in the everyday life. From an epistemologic point of view, we can thus argue that the above proof is not essentially a mathematical proof, but a simple rational argument that the convergence of the procedure described by Theorem 1 implies the absence of any derivation of the sequent $\vdash \bot$ in PA2.

It is well known that the consistency of a theory such as PA2 cannot be proved within the theory itself.[23] From the elementary argument of Prop. 11, it follows that the proof of Theorem 1 has to be developed in a mathematical framework that is strictly stronger than PA2. On the other hand, the consistency of a mathematical theory is a universal statement that can be considered as an empirical statement in the sense of Popper: it can be tested, corroborated, and above all, it can be falsified with a single 'experiment', namely: by producing a (finite) derivation whose conclusion is the sequent $\vdash \bot$. In the sequel, we will argue that the property of logical consistency is crucial to grasp the relationship between mathematics and the natural sciences, provided we do not understand consistency as a purely formal property, but as the very empirical hypothesis that is underlying all mathematical activity, especially when considering applied mathematics. For that, we need to relate the notion of *logical consistency* with the notion of *computational consistency*.

6.2 Logical consistency and computational consistency

A statement such as Fermat's last theorem

$$\forall x, y, z, n \in \mathbb{N}\ (x \neq 0,\ y \neq 0,\ n > 2\ \Rightarrow\ x^n + y^n \neq z^n)$$

is usually understood by mathematicians and non mathematicians as the fact that if we take two natural numbers x and y different from zero, then the sum of the nth powers of x and y (where n is greater than two) is not the nth power of a natural number. As is, this property is an empirical statement: it can be tested, corroborated and falsified by the means of simple calculations. And this is precisely because all the attempts to falsify this property repeatedly failed (and because all the attempts to prove it repeatedly failed too) that mathematicians relentlessly tried to prove the theorem during more than three centuries, until Wiles and Taylor succeeded in 1995.

[23] From Gödel's second incompleteness theorem, stating that a recursive theory containing first-order arithmetic cannot prove its own consistency [9].

But did Wiles and Taylor really prove this property for all x, y and n? Did they take every triple (x, y, n), compute the sum x^n+y^n and check that the result is not the nth power of a natural number? No. What they did is that they wrote a mathematical proof of Fermat's last theorem, that is, a text giving enough explanations to convince the reader that she could build a formal derivation of this formula in a suitable formal system (say: Zermelo-Fraenkel set theory). From the point of view of the natural sciences, they proved the existence of a derivation of Fermat's last theorem. According to Popper's terminology, they established the truth of a particular (i.e. existential) statement, by exhibiting the sketch of what could be a formal derivation (or, if we prefer, by performing the appropriate experiment about derivability).

However, there is a huge epistemological gap between the particular statement 'Fermat's last theorem has a derivation' (which is the only statement actually established by Wiles and Taylor) and the usual interpretation of Fermat's last theorem as a universal empirical statement: 'the sum of two nth powers of positive integers is not the nth power of an integer'. To deduce the latter from the former, we need to set an important hypothesis about mathematics, which is that they are *computationally consistent*, in the sense that:

> *Every computable equality (or inequality) that is derivable in mathematics holds by computation.*

Considering the formal system PA2, this hypothesis becomes:

> *If the sequent $\vdash e_1 = e_2$ is derivable in PA2 (where e_1 and e_2 are two closed numeric expressions), then the expressions e_1 and e_2 have the same value by computation*[24].

The hypothesis of computational consistency is the implicit hypothesis that is underlying all mathematical activity, especially when considering the applications of mathematics to other fields of the human knowledge. Without this hypothesis, nobody would be interested in applying mathematical theorems to physics, astronomy, economics and even to bookkeeping. Without such an hypothesis, there would actually be little epistemologic difference between mathematics and numerology. In a Popperian perspective, the property of computational consistency is a universal empirical statement that can be falsified with a single test, namely, by exhibiting a derivation of a computable (in)equality that appears to be wrong by computation. This property actually constitutes one of the most well corroborated empirical hypotheses of science, since it is daily tested in schools, in bookkeeping, in computers, etc.

We can thus argue (against Popper) that mathematics fulfill the demarcation criterion that makes mathematics an empirical science. The only specificity of mathematics is that the universal empirical hypothesis underlying mathematics is (almost) never stated explicitly. On the other hand, mathematical theorems do not constitute universal empirical statements themselves; they only constitute what Popper calls 'particular statements' expressing the existence of a derivation of a particular formula. This is only by combining the existence of a derivation of a formula (seen as an initial condition) with the hypothesis of computational consistency that we can deduce the universal empirical contents of this formula. Also notice that this combination is only possible for

[24] This notion is not altered by replacing equalities with inequalities.

the theorems expressing universally quantified computable (in)equalities—that is: Π^0_1-formulæ—such as Fermat's last theorem for instance. Most theorems of mathematics are not of this shape, and these theorems cannot be given a universal empirical contents as for Π^0_1-theorems, besides their obvious empirical contents as particular statements about derivability. However, these non Π^0_1-theorems remain useful to derive other Π^0_1-theorems, and thus indirectly contribute to the growth of the empirical basis of mathematics.[25]

It is well known that the property of computational consistency is elementarily equivalent to the property of logical consistency:

Proposition 12 — *A mathematical theory $\mathcal{T}$ that contains (at least) the computation rules of arithmetic (including Peano's 3rd and 4th axioms) is computationally consistent if and only if it is logically consistent.*

Proof. We only do the proof for PA2, but the same argument holds for any formal system that contains the computation rules of arithmetic, including Peano's 3rd and 4th axioms. We actually prove that the logical inconsistency of PA2 is equivalent to the computational inconsistency of PA2. Let us assume that PA2 is logically inconsistent. Then the sequent $\vdash 0 = 1$ is derivable in PA2 (*ex falso quod libet*), so that PA2 is computationally inconsistent. Conversely, let us assume that PA2 is computationally inconsistent. This means that there is a derivation d of $\vdash e_1 = e_2$ where e_1 and e_2 are two closed numeric expressions that have different values n_1 and n_2. Since PA2 contains the computation rules of arithmetic (i.e. the definitional equalities of primitive recursive functions), we can turn the computations showing that e_1 yields n_1 and e_2 yields n_2 into derivations d_1 of $\vdash e_1 = n_1$ and d_2 of $\vdash e_2 = n_2$. And since n_1 and n_2 are different natural numbers, we can build a derivation d_3 of $\vdash n_1 \neq n_2$ (using Peano's 3rd and 4th axioms). By combining the derivations d, d_1, d_2 and d_3 we easily get a derivation of the sequent $\vdash \bot$ in PA2. □

We can thus consider logical consistency and computational consistency as one and the same empirical hypothesis about mathematics. The former formulation is more compact and conceptual, and thus more suited to the needs of proof theory. But the practical interest of the notion of consistency mainly lies in the latter formulation. From an epistemological point of view, the hypothesis of computational consistency is precisely the hypothesis that is needed in order to legitimate the use of mathematical theorems (given by formal derivations) to concrete problems of computing and bookkeeping.

6.3 Experimental effectiveness and 1-consistency

However, the hypothesis of computational consistency alone is not sufficient to describe the connection between mathematical deductions and empirical assumptions when these assumptions deal with non mathematical entities (which is not the case of computations). To describe this connection, we have proposed the hypothesis of experimental effectiveness for PA2 (stated in Theorem 1), that gives an effective solution to the problem of tracking the empirical contents of a falsification through a mathematical derivation built in PA2. And from Prop. 11

[25] This is the case for instance of all the theorems about elliptic curves (usually non Π^0_1) that are involved in Ribet, Wiles and Taylor's proof of Fermat's last theorem.

and 12, it is clear that this hypothesis is at least as strong as the hypothesis of computational consistency of PA2, since it implies it.

It is now time to explain why the hypothesis of experimental effectiveness for PA2 is actually strictly stronger than the consistency of PA2. For that, we need to relate this hypothesis with the 1-consistency of PA2.

Let $\mathcal{T}$ be a mathematical theory that contains the symbols and computation rules of arithmetic (such as PA or PA2 for example). We say that the theory $\mathcal{T}$ is 1-*consistent* when for all primitive recursive function symbols f such that the sequent $\vdash \exists^{\mathsf{N}} x\ f(x) = 0$ is derivable in $\mathcal{T}$, there exists a natural number n such that $f(n)$ equals 0 by computation. In other words, a 1-consistent theory is a theory in which all the derivable Σ^0_1-formulæ are true. It is well known that 1-consistent theories are also consistent (or 0-consistent). The converse is false in general, since it is possible to artificially build theories that are logically consistent but not 1-consistent[26]. In the case of PA2, we easily check that:

Proposition 13 — *The hypothesis of experimental effectiveness for PA2 implies the* 1*-consistency of PA2.*

Proof. Let us assume that the sequent $\vdash \exists^{\mathsf{N}} x\, f(x) = 0$ is derivable in PA2. Then there is a derivation d of the sequent $U \vdash \bot$ in PA2, where U is the universal testable formula $\forall^{\mathsf{N}} x\, (f(x) = 0 \Rightarrow \bot)$. By applying the hypothesis of experimental effectiveness to the experimental theory U, we know that the process $\tilde{d} \star \diamond$ eventually reaches a falsification of U, which means that this process finds a natural number n such that $f(n)$ equals 0 by computation. □

It is actually possible to prove (by elementary means) that the converse implication holds, so that the hypothesis of experimental effectiveness is actually equivalent to the 1-consistency of PA2. We shall not present here the proof of the converse implication, since this proof is based on standard but tricky arithmetization techniques whose level of technicality goes beyond the scope of the present paper. The basic idea is to rephrase the realizability model of section 4 within the language of second-order arithmetic, and to show that given a derivation d of the sequent $U_1, \ldots, U_\ell \vdash \bot$ in PA2, the existence of the finite Herbrand tree whose construction has been described in section 5.1 is derivable in PA2. From the hypothesis of 1-consistency we deduce that this Herbrand tree exists outside PA2, which tree can be used as a certificate indicating that the evaluation of the process $\tilde{d} \star \diamond$ eventually reaches a falsification of the experimental theory $U_1, \ldots, U_\ell$ independently from the truth value functions attached to the experimental predicates of the theory.

According to Popper's terminology [23, ch. 8, § 66], the 1-consistency of PA2 and the hypothesis of experimental effectiveness for PA2 are 'universalized existential statements', that is, statements of the form 'for all x there is some y with the observable property $P(x, y)$'.[27] As such, these (equivalent) hypotheses

[26] For instance, the theory PA + ¬ConsPA (where ConsPA is the formula expressing the consistency of PA) is 0-consistent but not 1-consistent, provided PA itself is consistent.

[27] Provided we understand the hypothesis of experimental effectiveness for PA2 as: 'for all finite sequences of universal testable formulæ $U_1, \ldots U_\ell$ and for all derivations d of the sequent $U_1, \ldots U_\ell \vdash \bot$ (in PA2), there is a number n such that the process $\tilde{d} \star \diamond$ reaches a falsification of the theory $U_1, \ldots U_\ell$ after n evaluation steps'. Note that this statement does not correspond to a Π^0_2-formula of logic, since the observable property ('the process $\tilde{d} \star \diamond$ reaches a falsification of the theory $U_1, \ldots U_\ell$ after n evaluation steps') depends on the experimental interpretation of the predicates p, q, r, etc. mentioned in the experimental theory.

are neither verifiable, nor falsifiable. However, there are good reasons to accept the hypothesis of experimental effectiveness (or the 1-consistency of PA2 that is equivalent), despite the fact that it is not falsifiable, strictly speaking.[28] The first reason is that the hypothesis of experimental effectiveness can be tested, by checking that the extracted program terminates according to the prediction of Theorem 1 for particular experimental theories $U_1, \ldots, U_\ell$ and for particular derivations d of the sequent $U_1, \ldots, U_\ell \vdash \bot$. The second reason is that the hypothesis of experimental effectiveness has a falsifiable consequence, which is the (logical and computational) consistency of PA2.

References

[1] H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and The Foundations of Mathematics*. North-Holland, 1984.

[2] Y. Bertot and P. Castran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Springer, 2004.

[3] L. E. J. Brouwer. Intuïtionistische splitsing van mathematische grondbegrippen. *Nederl. Akad. Wetensch. Verslagen*, 32:877–880, 1923.

[4] A. Church. *The calculi of lambda-conversion*, volume 6 of *Annals of Mathematical Studies*. Princeton, 1941.

[5] H. B. Curry and R. Feys. *Combinatory Logic*, volume 1. North-Holland, 1958.

[6] G. Gentzen. Untersuchungen über das logische Schließen. I. *Mathematische Zeitschrift*, 39(1):176–210, 1935.

[7] J.-Y. Girard. *Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur*. Doctorat d'État, Université Paris VII, Juin 1972.

[8] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge University Press, 1989.

[9] K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931.

[10] K. Gödel. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. *Dialectica*, 12:280–287, 1958.

[11] T. Griffin. A formulae-as-types notion of control. In *Principles Of Programming Languages (POPL'90)*, pages 47–58, 1990.

[12] M. Guillermo. *Jeux de réalisabilité en arithmétique classique*. PhD thesis, Université Paris 7, 2008.

[13] W. A. Howard. The formulae-as-types notion of construction. Privately circulated notes, 1969.

[14] S. C. Kleene. On the interpretation of intuitionistic number theory. *Journal of Symbolic Logic*, 10:109–124, 1945.

[15] G. Kreisel. On the interpretation of non-finitist proofs, part I. *Journal of Symbolic Logic*, 16:241–267, 1951.

[16] J. L. Krivine. *Lambda-calculus, types and models*. Masson, 1993.

[17] J.-L. Krivine. A general storage theorem for integers in call-by-name lambda-calculus. *Th. Comp. Sc.*, 129:79–94, 1994.

[28] Popper faces a similar difficulty [23, ch. 8] when considering the interpretation of probabilistic statements.

[18] J.-L. Krivine. Typed lambda-calculus in classical Zermelo-Fraenkel set theory. *Arch. Math. Log.*, 40(3):189–205, 2001.

[19] J.-L. Krivine. Dependent choice, 'quote' and the clock. *Theor. Comput. Sci.*, 308(1-3):259–276, 2003.

[20] J.-L. Krivine. Realizability in classical logic. *Panoramas et synthèses, Société Mathématique de France*, 2004.

[21] P. Martin-Löf. *Intuitionistic Type Theory.* Bibliopolis, Napoli, 1984.

[22] A. Miquel. Classical program extraction in the calculus of constructions. In Jacques Duparc and Thomas A. Henzinger, editors, *CSL*, volume 4646 of *Lecture Notes in Computer Science*, pages 313–327. Springer, 2007.

[23] K. R. Popper. *The Logic of Scientific Discovery.* Routledge, 1992.

[24] The Coq Development Team. The Coq Proof Assistant Reference Manual – version v8.2. Technical report, INRIA, 2009.

[25] E. Wigner. The unreasonable effectiveness of mathematics in the natural sciences. *Communications in Pure and Applied Mathematics*, 13(1), February 1960.