*Math. Struct. in Comp. Science* (2017), *vol.* 27, *pp.* 1068–1107. © Cambridge University Press 2016
doi:10.1017/S0960129515000559 First published online 18 January 2016

# Classical realizability and arithmetical formulæ

MAURICIO GUILLERMO[†] and ÉTIENNE MIQUEY[‡]

[†]*Universidad de la República, IMERL, Facultad de Ingeniería, Montevideo, Uruguay*
*Email:* mguille@fing.edu.uy
[‡]*Laboratoire PPS, Univ. Paris Diderot, Équipe PiR2, INRIA*
*Universidad de la República, IMERL, Facultad de Ingeniería, Montevideo, Uruguay*
*Email:* etienne.miquey@pps.univ-paris-diderot.fr

*Received 19 February 2014; revised 8 May 2015*

In this paper, we treat the specification problem in Krivine classical realizability (Krivine 2009 *Panoramas et synthèses* **27**), in the case of arithmetical formulæ. In the continuity of previous works from Miquel and the first author (Guillermo 2008 *Jeux de réalisabilité en arithmétique classique*, Ph.D. thesis, Université Paris 7; Guillermo and Miquel 2014 *Mathematical Structures in Computer Science*, Epub ahead of print), we characterize the universal realizers of a formula as being the winning strategies for a game (defined according to the formula). In the first sections, we recall the definition of classical realizability, as well as a few technical results. In Section 5, we introduce in more details the specification problem and the intuition of the game-theoretic point of view we adopt later. We first present a game $\mathfrak{G}^1$, that we prove to be adequate and complete if the language contains no instructions '`quote`' (Krivine 2003 *Theoretical Computer Science* **308** 259–276), using interaction constants to do substitution over execution threads. We then show that as soon as the language contain '`quote`,' the game is no more complete, and present a second game $\mathfrak{G}^2$ that is both adequate and complete in the general case. In the last Section, we draw attention to a model-theoretic point of view and use our specification result to show that arithmetical formulæ are absolute for realizability models.

## 1. Introduction

The so called *Curry–Howard correspondence* constituted an important breakthrough in proof theory, by evidencing a strong connection between the notions of functional programming and proof theory (Curry and Feys 1958; Howard 1980; Girard et al. 1989). For a long time, this correspondence has been limited to intuitionistic proofs and constructive mathematics, so that classical reasonings, that are omnipresent in mathematics, could only be retrieved through negative translations to intuitionistic logic (Friedman 1978) or to linear logic (Girard 2006).

In 1990, Griffin discovered that the control operator `call/cc` (for *call with current continuation*) of the Scheme programming language could be typed by Peirce's law ($((A \to B) \to A) \to A$), this way extending the formuæ-as-types interpretation (Howard 1980). As Peirce's law is known to imply, in an intuitionistic framework, all the other forms of classical reasoning (excluded middle, *reductio ad absurdum*, double negation elimination, etc.), this discovery opened the way for a direct computational interpretation of classical proofs, using control operators and their ability to *backtrack*. Several calculi were born from this idea, such as Parigot's

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

$\lambda\mu$-calculus (Parigot 1997), Barbanera and Berardi's symmetric $\lambda$-calculus (Barbanera and Berardi 1996), Krivine's $\lambda_c$-calculus (Krivine 2009) or Curien and Herbelin's $\bar{\lambda}\mu\tilde{\mu}$-calculus (Curien and Herbelin 2000).

Nonetheless, some difficulties quickly appeared in the analysis of the computational behaviour of programs extracted from classical proofs. One reason for these difficulties was precisely the presence of control operators, whose ability to backtrack breaks the linearity of the execution of programs. More importantly, the formulæ-as-types interpretation suffered from the lack of a theory connecting the point of view of typing with the point of view of computation. Realizability was designed by Kleene to interpret the computational contents of the proofs of Heyting arithmetic (Kleene 1945), and even if it has been extended later to more general frameworks (like intuitionistic set theories (Myhill 1973; Friedman 1973; McCarty 1984)), it is intrinsically incompatible with classical reasoning: the negation of the middle excluded principle is realizable.

### 1.1. *Classical realizibility*

To address this problem, Krivine introduced in the middle of the 90s the theory of *classical realizability* (Krivine 2009), which is a complete reformulation of the very principles of realizability to make them compatible with classical reasoning. (As noticed in several papers (Oliva and Streicher 2008; Miquel 2011a), classical realizability can be seen as a reformulation of Kleene's realizability through Friedman's $A$-translation (Friedman 1978).) Although it was initially introduced to interpret the proofs of classical second-order arithmetic, the theory of classical realizability can be scaled to more expressive theories such as Zermelo–Fraenkel set theory (Krivine 2001) or the calculus of constructions with universes (Miquel 2007).

As in intuitionistic realizability, every formula $A$ is interpreted in classical realizability as a set $|A|$ of programs called the *realizers* of $A$, that share a common computational behaviour dictated by the structure of the formula $A$. This point of view is related to the point of view of deduction (and of typing) via the property of *adequacy*, that expresses that any program extracted from a proof of $A$ – that is: any program of type $A$ – realizes the formula $A$, and thus has the computational behaviour expected from the formula $A$.

However, the difference between intuitionistic and classical realizability is that in the latter, the set of realizers of $A$ is defined indirectly, that is from a set $\|A\|$ of execution contexts (represented as argument stacks) that are intended to challenge the truth of $A$. Intuitively, the set $\|A\|$ – which we shall call the *falsity value of* $A$ – can be understood as the set of all possible counter-arguments to the formula $A$. In this framework, a program realizes the formula $A$ – i.e. belongs to the *truth value* $|A|$ – if and only if it is able to defeat all the attempts to refute $A$ using a stack in $\|A\|$. (The definition of the classical notion of a realizer is also parameterized by a *pole* representing a particular challenge, that we shall define and discuss in Section 4.1.1.)

By giving an equal importance to programs – or terms – that 'defend' the formula $A$, and to execution contexts – or stacks – that 'attack' the formula $A$, the theory of classical realizability is therefore able to describe the interpretation of classical reasoning in terms of manipulation of whole stacks (as first class citizens) using control operators.

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

### 1.2. *Krivine $\lambda_c$-calculus*

The programming language commonly used in classical realizability is Krivine's $\lambda_c$-calculus, which is an extension of Church's $\lambda$-calculus (Church 1941) containing an instruction $\mathfrak{cc}$ (representing the control operator `call/cc`) and *continuation constants* embedding stacks. Unlike the traditional $\lambda$-calculus, the $\lambda_c$-calculus is parameterized by a particular execution strategy – corresponding to the Krivine Abstract Machine (Krivine 2007) – so that the notion of confluence – which is central in traditional $\lambda$-calculi, does not make sense anymore. The property of confluence is replaced by the property of determinism, which is closer from the point of view of real programming languages.

A pleasant feature of this calculus is that it can be enriched with *ad hoc* extra instructions. For instance, a `print` instruction might be added to trace an execution, as well as extra instructions manipulating primitive numerals to do some code optimization (Miquel 2011a). In some situations, extra instructions can also be designed to realize reasoning principles, the standard example being the instruction `quote` that computes the Gödel code of a stack, used for instance to realize the axiom of dependent choice (Krivine 2003). In this paper, we shall consider this instruction together `eq`, that tests the syntactic equality between two $\lambda_c$-terms.

### 1.3. *The specification problem*

A central problem in classical realizability is the *specification problem*, which is to find a characterization for the (universal) realizers of a formula by their computational behaviour. In intuitionistic logic, this characterization does not contain more information than the formula itself, so that this problem has been given little attention. For instance, the realizers of an existential formula $\exists^{\mathsf{N}} x A(x)$ are exactly the ones reducing to a pair made of a witness $n \in \mathbb{N}$ and a proof term realizing $A(n)$ (Krivine 1993).

However, in classical realizability the situation appears to be quite different and the desired characterization is in general much more difficult to obtain. Indeed, owing to the presence of control operators in the language of terms, the realizers have the ability to backtrack at any time, making the execution harder to predict. Considering for instance the very same formula $\exists^{\mathsf{N}} x A(x)$, a classical realizer of it can give as many integers for $x$ as it wants, using backtrack to make another try. Hence, we cannot expect from such a realizer to reduct directly to a witness (for an account of witness extraction techniques in classical realizability, see Miquel's paper (Miquel 2011a)). In addition, as we will see in Section 5.3, giving such a witness might be computationally impossible without backtrack, for example in the case of a formula relying on the Halting Problem. We will treat this particular example in Section 5.3.

Furthermore, as stated in the paper on Peirce's Law (Guillermo and Miquel 2014), the presence of instructions such as `quote` (defined in Section 2.3) makes the problem still more subtle. We will deal with this particular case in Section 7.

### 1.4. *Specifying arithmetical formulæ*

The architecture of classical realizability is centred around the opposition between falsity values (stacks) and truth values (terms). This opposition, as well as the underlying intuition (opponents vs. defenders), naturally leads us to consider the problem in a game-theoretic setting. Such a

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

setting – namely realizability games – was defined by Krivine as a mean to prove that any arithmetical formula which is *universally realized* (i.e. realized for all poles) is true in the ground model (often considered as the *standard full model of second order arithmetics*) (c.f. Theorem 16 of Krivine (2003)). Thereafter, Krivine also proves the converse, which is that every arithmetical formula which is true in the ground model is realized by the term which implements the trivial winning strategy of the game associated to the formula (c.f. Theorem 21 of Krivine (2009)). Those realizability games are largely inspired on the *non-counterexample interpretation* of (Kreisel 1951), Kreisel (1952) and the subsequent development of game semantics for proofs by Coquand (1995).

Our goal is to establish an operational description which characterize *all* the realizers of a given arithmetical formula. In particular, it does not suffice to find a realizer for any true arithmetical formula, but we want to explicit an operational sufficient condition to be a realizer.

In Coquand's games, the only atomic formulæ are $\top$ and $\bot$, therefore a strategy for a true atomic formula does nothing, as the game is already won by the defender. As a consequence, any 'blind' enumeration of $\mathbb{N}^k$ is a winning strategy for every true $\Sigma^0_{2k}$-formulæ. Such a strategy, which is central in Krivine's proof that any true formula in the ground model is realized (Krivine 2009, Theorem 21), has no interesting computational content. Worse, it is not suitable for being a realizer in the general case where we use Leibniz equality. This remark will be discussed more consistently in Section 8.

The game developed by Krivine makes both players to use only constants. If the calculus does not contain instructions incompatible with substitution (like '`quote`'), this game is equivalent to the one we prove that specifies the arithmetical formulæ in the substitutive case. However, Krivine's realizers are eventually intended to contain '`quote`'. In this general case, we prove that the specification is obtained from the first game by a relaxation of the rules of $\exists$.

Thus, both works left open the question of giving a precise specification for arithmetic formulæ in the general case.

In this paper, we will restrict ourself to the setting where arithmetical formulæ have been relativized to *canonical integers*, as we explained in Section 4.7. We will rephrase the game-theoretic framework of the first author Ph.D. thesis (Guillermo 2008), to provide a game-theoretic characterization $\mathbb{G}^1$ that is both complete and adequate, in the particular case where the underlying calculus contains infinitely many *interaction constants*. However, this hypothesis – that is crucial in our proof of completeness – is known to be incompatible with the presence of instructions such as `quote` or `eq` (Guillermo and Miquel 2014), which allow us to distinguish syntactically $\lambda_c$ terms that are computationally equivalent. We exhibit in Section 6.3 a *wild realizer* that uses these instructions and does not suit as a winning strategy for $\mathbb{G}^1$, proving that $\mathbb{G}^1$ is no more complete in this case.

Indeed, as highlighted in the paper on Peirce's Law (Guillermo and Miquel 2014), the presence of such instructions introduces a new – and purely game-theoretic – form of backtrack that does not come from a control operator but from the fact that realizers, using a syntactic equality test provided by `quote`, can check whether a position has already appeared before. We present in Section 7 a second game $\mathbb{G}^2$ that allows this new form of backtrack, and captures the behaviour of our wild realizer. Then, we prove that without any assumption on the set of instructions, this game is both adequate and complete, thus constituting the definitive specification of arithmetical formulæ.

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

### 1.5. *Connexion with forcing*

In addition to the question of knowing how to specify arithmetical formulæ, this paper presents an answer to another question, which is to know whether arithmetical formulæ are *absolute* for realizability models. In set theory, a common technique to prove independence results in theory is to use forcing, that allows us to extend a model and add some specific properties to it. Yet, it is known $\Sigma^1_2$-formulæ are absolute for a large class of models, including those produced by forcing. This constitutes somehow a barrier to forcing, which does not permit to change the truth of formulæ that are below $\Sigma^1_2$ in the arithmetical hierarchy.

If classical realizability was initially designed to be a semantics for proofs of Peano second-order arithmetic, it appeared then to be scalable to build models for high-order arithmetic (Miquel 2011b) or set theory (Krivine 2011). Just like forcing techniques, these constructions rest upon a ground model and allow us to break some formulæ that were true in the ground model, say the continuum hypothesis or the axiom of choice (Krivine 2012). In addition, the absoluteness theorem of $\Sigma^2_1$ does not apply to realizability model. Hence, it seems quite natural to wonder, as for forcing, whether realizability models preserve some formulæ. We will explain in Section 8 how the specification results allow us to show that arithmetical formulæ are absolute for realizability models.

## 2. The language $\lambda_c$

A lot of the notions we use in this paper are the very same as in the paper on Peirce's Law (Guillermo and Miquel 2014). We will recall them briefly, for a more gentle introduction, we advise the reader to refer to this paper.

### 2.1. *Terms and stacks*

The $\lambda_c$-calculus distinguishes two kinds of syntactic expressions: *terms*, which represent programs, and *stacks*, which represent evaluation contexts.

Formally, terms and stacks of the $\lambda_c$-calculus are defined (see Figure 1) from three auxiliary sets of symbols, that are pairwise disjoint:

- A denumerable set $\mathcal{V}_\lambda$ of $\lambda$-variables (notation: $x$, $y$, $z$, etc.)
- A countable set $\mathcal{C}$ of instructions, which contains at least an instruction $\mathbf{cc}$ (denoting 'call/cc', for: *call with current continuation*).
- A non-empty countable set $\mathcal{B}$ of stack constants, also called stack bottoms (notation: $\alpha$, $\beta$, $\gamma$, etc.)

In what follows, we adopt the same writing conventions as in the pure $\lambda$-calculus, by considering that application is left-associative and has higher precedence than abstraction. We also allow several abstractions to be regrouped under a single $\lambda$, so that the closed term $\lambda x\,.\,\lambda y\,.\,\lambda z\,.\,((zx)y)$ can be more simply written $\lambda xyz\,.\,zxy$.

As usual, terms and stacks are considered up to $\alpha$-conversion (Barendregt 1984) and we denote by $t\{x := u\}$ the term obtained by replacing every free occurrence of the variable $x$ by the term $u$ in the term $t$, possibly renaming the bound variables of $t$ to prevent name clashes. The sets of all closed terms and of all (closed) stacks are respectively denoted by $\Lambda$ and $\Pi$.

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

| | | | |
|---|---|---|---|
| **Terms** | $t, u$ | $::= \quad x \mid \lambda x.t \mid tu \mid \mathbf{k}_\pi \mid \kappa$ | $x, \in \mathcal{V}_\lambda, \kappa \in C$ |
| **Stacks** | $\pi$ | $::= \quad \alpha \mid t \cdot \pi$ | $(\alpha \in \mathcal{B}, t$ closed$)$ |
| **Processes** | $p, q$ | $::= \quad t \star \pi$ | $(t$ closed$)$ |

$$
\begin{array}{rcl@{\qquad}rcl}
x\{c := u\} & \equiv & x & x\{\alpha := \pi_0\} & \equiv & x \\
(\lambda x\,.\,t)\{c := u\} & \equiv & \lambda x\,.\,t\{c := u\} & (\lambda x\,.\,t)\{\alpha := \pi_0\} & \equiv & \lambda x\,.\,t\{\alpha := \pi_0\} \\
(t_1 t_2)\{c := u\} & \equiv & t_1\{c := u\}t_2\{c := u\} & (t_1 t_2)\{\alpha := \pi_0\} & \equiv & t_1\{\alpha := \pi_0\}t_2\{\alpha := \pi_0\} \\
\mathbf{k}_\pi\{c := u\} & \equiv & \mathbf{k}_{\pi\{c:=u\}} & \mathbf{k}_\pi\{\alpha := \pi_0\} & \equiv & \mathbf{k}_{\pi\{\alpha:=\pi_0\}} \\
c\{c := u\} & \equiv & u & c\{\alpha := \pi_0\} & \equiv & c \\
c'\{c := u\} & \equiv & c' \quad (\text{if } c' \not\equiv c) & \alpha\{\alpha := \pi_0\} & \equiv & \pi_0 \\
\alpha\{c := u\} & \equiv & \alpha & \alpha'\{\alpha := \pi_0\} & \equiv & \alpha' \quad (\text{if } \alpha' \not\equiv \alpha) \\
(t \cdot \pi)\{c := u\} & \equiv & t\{c := u\} \cdot \pi\{c := u\} & (t \cdot \pi)\{\alpha := \pi_0\} & \equiv & t\{\alpha := \pi_0\} \cdot \pi\{\alpha := \pi_0\}
\end{array}
$$

**Substitution over terms and stacks**

| | | |
|---|---|---|
| **First-order terms** | $e_1, e_2 ::= x \mid f(e_1, \ldots, e_k)$ | $x \in \mathcal{V}_1, f \in \Sigma$ |
| **Formulæ** | $A, B ::= X(e_1, \ldots, e_k) \mid A \Rightarrow B \mid \forall x A \mid \forall X A$ | $X \in \mathcal{V}_2$ |

$$
\begin{array}{rcl@{\qquad}rcl}
\bot & \equiv & \forall Z\, Z & A \Leftrightarrow B & \equiv & (A \Rightarrow B) \wedge (B \Rightarrow A) \\
\neg A & \equiv & A \Rightarrow \bot & \exists x\, A(x) & \equiv & \forall Z\, (\forall x\, (A(x) \Rightarrow Z) \Rightarrow Z) \\
A \wedge B & \equiv & \forall Z\, ((A \Rightarrow B \Rightarrow Z) \Rightarrow Z) & \exists X\, A(X) & \equiv & \forall Z\, (\forall X\, (A(X) \Rightarrow Z) \Rightarrow Z) \\
A \vee B & \equiv & \forall Z\, ((A \Rightarrow Z) \Rightarrow (B \Rightarrow Z) \Rightarrow Z) & e_1 = e_2 & \equiv & \forall W\, (W(e_1) \Rightarrow W(e_2))
\end{array}
$$

**Second-order encodings**

$$
\frac{}{\Gamma \vdash x : A}\ {\scriptstyle (x:A)\in\Gamma} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x\,.\,t : A \Rightarrow B} \qquad \frac{\Gamma \vdash t : A \Rightarrow B \qquad \Gamma \vdash t : A}{\Gamma \vdash tu : B}
$$

$$
\frac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall x A}\ {\scriptstyle x\notin FV(\Gamma)} \qquad \frac{\Gamma \vdash t : \forall x A}{\Gamma \vdash t : A\{x := e\}} \qquad \frac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall X A}\ {\scriptstyle X\notin FV(\Gamma)}
$$

$$
\frac{\Gamma \vdash t : \forall X A}{\Gamma \vdash t : A\{X := P\}} \qquad \frac{}{\Gamma \vdash \mathbf{cc} : ((A \Rightarrow B) \Rightarrow A) \Rightarrow A}
$$

**Typing rules of second-order logic**

Fig. 1. Definitions.

**Definition 1 (Proof-like terms).** We say that a $\lambda_c$-term $t$ is *proof-like* if $t$ contains no continuation constant $\mathbf{k}_\pi$. We denote by PL the set of all proof-like terms.

Finally, every natural number $n \in \mathbb{N}$ is represented in the $\lambda_c$-calculus as the closed proof-like term $\overline{n}$ defined by

$$\overline{n} \equiv \overline{s}^n\overline{0} \equiv \underbrace{\overline{s}(\cdots(\overline{s}}_{n}\,\overline{0})\cdots),$$

where $\overline{0} \equiv \lambda xf\,.\,x$ and $\overline{s} \equiv \lambda nxf\,.\,f(nxf)$ are Church's encodings of zero and the successor function in the pure $\lambda$-calculus. Note that this encoding slightly differs from the traditional encoding of numerals in the $\lambda$-calculus, although the term $\overline{n} \equiv \overline{s}^n\overline{0}$ is clearly $\beta$-convertible to Church's encoding $\lambda xf\,.\,f^n x$ – and thus computationally equivalent. The reason for preferring this modified encoding is that it is better suited to the call-by-name discipline of Krivine's Abstract Machine (KAM) we will now present.

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

### 2.2. *Krivine's Abstract Machine*

In the $\lambda_c$-calculus, computation occurs through the interaction between a closed term and a stack within KAM. Formally, we call a *process* any pair $t \star \pi$ formed by a closed term $t$ and a stack $\pi$. The set of all processes is written $\Lambda \star \Pi$ (which is just another notation for the Cartesian product of $\Lambda$ by $\Pi$).

**Definition 2 (Relation of evaluation).** We call a relation of *one step evaluation* any binary relation $\succ_1$ over the set $\Lambda \star \Pi$ of processes that fulfils the following four axioms:

| | | | |
|---|---|---|---|
| (Push) | $tu \star \pi$ | $\succ_1$ | $t \star u \cdot \pi$ |
| (Grab) | $(\lambda x\,.\,t) \star u \cdot \pi$ | $\succ_1$ | $t\{x := u\} \star \pi$ |
| (Save) | $\mathbf{cc} \star t \cdot \pi$ | $\succ_1$ | $t \star \mathbf{k}_\pi \cdot \pi$ |
| (Restore) | $\mathbf{k}_\pi \star t \cdot \pi'$ | $\succ_1$ | $t \star \pi$ |

The reflexive–transitive closure of $\succ_1$ is written $\succ$.

One of the specificities of the $\lambda_c$-calculus is that it comes with a binary relation of (one step) evaluation $\succ_1$ that is not *defined*, but *axiomatized* via the rules (Push), (Grab), (Save) and (Restore). In practice, the binary relation $\succ_1$ is simply another parameter of the definition of the calculus, just like the sets $\mathcal{C}$ and $\mathcal{B}$. Strictly speaking, the $\lambda_c$-calculus is not a particular extension of the $\lambda$-calculus, but a family of extensions of the $\lambda$-calculus parameterized by the sets $\mathcal{B}$, $\mathcal{C}$ and the relation of one step evaluation $\succ_1$. (The set $\mathcal{V}_\lambda$ of $\lambda$-variables – that is interchangeable with any other denumerable set of symbols – does not really constitute a parameter of the calculus.)

### 2.3. *Adding new instructions*

The main interest of keeping open the definition of the sets $\mathcal{B}$, $\mathcal{C}$ and of the relation evaluation $\succ_1$ (by axiomatizing rather than defining them) is that it makes possible to enrich the calculus with extra instructions and evaluation rules, simply by putting additional axioms about $\mathcal{C}$, $\mathcal{B}$ and $\succ_1$. On the other hand, the definitions of classical realizability (Krivine 2009) as well as its main properties do not depend on the particular choice of $\mathcal{B}$, $\mathcal{C}$ and $\succ_1$, although the fine structure of the corresponding realizability models is of course affected by the presence of additional instructions and evaluation rules.

For the needs of the discussion in Section 6, we shall sometimes consider the following extra instructions in the set $\mathcal{C}$:

- The instruction `quote`, which comes with the evaluation rule

  (Quote) $$\texttt{quote} \star t \cdot \pi \succ_1 t \star \overline{n}_\pi \cdot \pi\,,$$

  where $\pi \mapsto n_\pi$ is a recursive injection from $\Pi$ to $\mathbb{N}$. Intuitively, the instruction `quote` computes the 'code' $n_\pi$ of the stack $\pi$, and passes it (using the encoding $n \mapsto \overline{n}$ described in Section 2.1) to the term $t$. This instruction was originally introduced to realize the axiom of dependent choices (Krivine 2003).
- The instruction `eq`, which comes with the evaluation rule

  (Eq) $$\texttt{eq} \star t_1 \cdot t_2 \cdot u \cdot v \cdot \pi \succ_1 \begin{cases} u \star \pi & \text{if } t_1 \equiv t_2 \\ v \star \pi & \text{if } t_1 \not\equiv t_2 \end{cases}$$

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

Intuitively, the instruction $\mathtt{eq}$ tests the syntactic equality of its first two arguments $t_1$ and $t_2$ (up to $\alpha$-conversion), giving the control to the next argument $u$ if the test succeeds, and to the second next argument $v$ otherwise. In presence of the $\mathtt{quote}$ instruction, it is possible to implement a closed $\lambda_c$-term $\mathtt{eq}'$ that has the very same computational behaviour as $\mathtt{eq}$, by letting

$$\mathtt{eq}' \equiv \lambda x_1 x_2 \,.\, \mathtt{quote}\,(\lambda n_1 y_1 \,.\, \mathtt{quote}\,(\lambda n_2 y_2 \,.\, \mathsf{eq_nat}\, n_1\, n_2)\, x_2)\, x_1\,,$$

where $\mathsf{eq_nat}$ is any closed $\lambda$-term that tests the equality between two numerals (using the encoding $n \mapsto \overline{n}$).

- The instruction $\curlywedge$ ('fork'), which comes with the two evaluation rules

(Fork) $\qquad \curlywedge \star t_0 \cdot t_1 \cdot \pi \succ_1 t_0 \star \pi \qquad$ and $\qquad \curlywedge \star t_0 \cdot t_1 \cdot \pi \succ_1 t_1 \star \pi\,.$

Intuitively, the instruction $\curlywedge$ behaves as a non-deterministic choice operator, that indifferently selects its first or its second argument. The main interest of this instruction is that it makes evaluation non-deterministic, in the following sense:

**Definition 3 (Deterministic evaluation).** We say that the relation of evaluation $\succ_1$ is *deterministic* when the two conditions $p \succ_1 p'$ and $p \succ_1 p''$ imply $p' \equiv p''$ (syntactic identity) for all processes $p$, $p'$ and $p''$. Otherwise, $\succ_1$ is said to be *non-deterministic*.

The smallest relation of evaluation, that is defined as the union of the four rules (Push), (Grab), (Save) and (Restore), is clearly deterministic. The property of determinism still holds if we enrich the calculus with an instruction $\mathtt{eq}\,(\not\equiv \mathfrak{cc})$ together with the aforementioned evaluation rules or with the instruction $\mathtt{quote}\,(\not\equiv \mathfrak{cc})$.

On the other hand, the presence of an instruction $\curlywedge$ with the corresponding evaluation rules definitely makes the relation of evaluation non-deterministic.

### 2.4. *The thread of a process and its anatomy*

Given a process $p$, we call the *thread* of $p$ and write $\mathbf{th}(p)$ the set of all processes $p'$ such that $p \succ p'$:

$$\mathbf{th}(p) \;=\; \{p' \in \Lambda \star \Pi \;:\; p \succ p'\}\,.$$

This set has the structure of a finite or infinite (di)graph whose edges are given by the relation $\succ_1$ of one step evaluation. In the case where the relation of evaluation is deterministic, the graph $\mathbf{th}(p)$ can be either:

- *finite and cyclic from a certain point*, because the evaluation of $p$ loops at some point. A typical example is the process $\mathbf{I} \star \delta\delta \cdot \alpha$ (where $\mathbf{I} \equiv \lambda x \,.\, x$ and $\delta \equiv \lambda x \,.\, xx$), that enters into a two-cycle after one evaluation step:

$$\mathbf{I} \star \delta\delta \cdot \alpha \;\succ_1\; \delta\delta \star \alpha \;\succ_1\; \delta \star \delta \cdot \alpha \;\succ_1\; \delta\delta \star \alpha \;\succ_1\; \cdots$$

- *finite and linear*, because the evaluation of $p$ reaches a state where no more rule applies. For example:

$$\mathbf{II} \star \alpha \;\succ_1\; \mathbf{I} \star \mathbf{I} \cdot \alpha \;\succ_1\; \mathbf{I} \star \alpha\,.$$

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

- *infinite and linear*, because $p$ has an infinite execution that never reaches twice the same state. A typical example is given by the process $\delta'\delta' \star \alpha$, where $\delta' \equiv \lambda x\,.\,x\,x\,\mathbf{I}$:

$$\delta'\delta' \star \alpha \succ_3 \delta'\delta' \star \mathbf{I} \cdot \alpha \succ_3 \delta'\delta' \star \mathbf{I} \cdot \mathbf{I} \cdot \alpha \succ_3 \delta'\delta' \star \mathbf{I} \cdot \mathbf{I} \cdot \mathbf{I} \cdot \alpha \succ_3 \cdots$$

### 2.5. *Interaction constants*

The two examples of extra instructions `quote` and `eq` we gave in Section 2.3 have a strong impact on the potential behaviour of processes. Indeed, they are able to distinguish syntactically different terms that are computationally equivalent, such as the terms $\mathbf{I}$ and $\mathbf{II}$. To better understand the consequence of the presence of such extra instructions in the $\lambda_c$-calculus, we need to introduce the important notion of interaction constant. This definition relies on the notions of substitution over terms and stacks, that are defined in Figure 1. Unlike the traditional form of substitution $t\{x := u\}$ (which is only defined for terms), the substitutions $t\{c := u\}$ and $\pi\{c := u\}$ also propagate through the continuation constants $\mathbf{k}_\pi$.

**Definition 4.** A constant $\kappa \in \mathcal{C}$ is said to be

- *inert* if for all $\pi \in \Pi$, there is no process $p$ such that $\kappa \star \pi \succ_1 p$;
- *substitutive* if for all $u \in \Lambda$ and for all processes $p, p' \in \Lambda \star \Pi$, $p \succ_1 p'$ implies $p\{\kappa := u\} \succ_1 p'\{\kappa := u\}$;
- *non-generative* if for all processes $p, p' \in \Lambda \star \Pi$, $p \succ_1 p'$, the constant $\kappa$ cannot occur in $p'$ unless it already occurs in $p$.

A constant $\kappa \in \mathcal{C}$ that is inert, substitutive and non-generative is then called an *interaction constant*. Similarly, we say that a stack constant $\alpha \in \mathcal{B}$ is:

- *substitutive* if for all $\pi \in \Pi$ and for all processes $p, p' \in \Lambda \star \Pi$, $p \succ_1 p'$ implies $p\{\alpha := \pi\} \succ_1 p'\{\alpha := \pi\}$;
- *non-generative* if for all processes $p, p' \in \Lambda \star \Pi$, $p \succ_1 p'$, the constant $\alpha$ cannot occur in $p'$ unless it already occurs in $p$.

The main observation is that substitutive constants are incompatible with both instruction `quote` and `eq` (see Guillermo and Miquel (2014) for a proof):

**Proposition 1.** If the calculus of realizers contains one of both instructions `quote` or `eq`, then none of the constants $\kappa \in \mathcal{C}$ is substitutive.

The very same argument can be applied to prove the incompatibility of substitutive stack constants with the instruction `quote`. On the other hand, it is clear that if the relation of evaluation $\succ_1$ is only defined from the rules (Grab),(Push),(Save) and (Restore) – and possibly: the rule (Fork) – then all the remaining constants $\kappa$ in $\mathcal{C}$ (i.e. $\kappa \not\equiv \mathbf{cc}, \curlywedge$) are interaction constants (and thus substitutive), whereas all the stack constants in $\mathcal{B}$ are substitutive and non-generative. Substitutive (term and stack) constants are useful to analyse the computational behaviour of realizers in a uniform way. For instance, if we know that a closed term $t \in \Lambda$ is such that

$$t \star \kappa_1 \cdots \kappa_n \cdot \alpha \succ p,$$

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

where $\kappa_1, \ldots, \kappa_n$ are substitutive constants that do not occur in $t$, and where $\alpha$ is a substitutive stack constant that does not occur in $t$ too, then we more generally know that

$$t \star u_1 \cdots u_n \cdot \pi \succ p\{\kappa_1 := u_1, \ldots, \kappa_n := u_n, \alpha := \pi\}$$

for all terms $u_1, \ldots, u_n \in \Lambda$ and for all stacks $\pi \in \Pi$. Intuitively, substitutive constants play in the $\lambda_c$-calculus the same role as free variables in the pure $\lambda$-calculus.

## 3. Classical second-order arithmetic

In Section 2, we delt with the *computing facet* of the theory of classical realizability. In this section, we will now present its *logical facet* by introducing the language of classical second-order logic with the corresponding type system. In section 3.3, we will focus to the particular case of *second-order arithmetic* and present its axioms.

### 3.1. *The language of second-order logic*

The language of second-order logic distinguishes two kinds of expressions: *first-order expressions* representing individuals, and *formulæ*, representing propositions about individuals and sets of individuals (represented using second-order variables as we shall see below).

3.1.1. *First-order expressions.* First-order expressions are formally defined (see Figure 1) from the following sets of symbols:

- A *first-order signature* $\Sigma$ defining *function symbols* with their arities, and considering *constant symbols* as function symbols of arity 0. We assume that the signature $\Sigma$ contains a constant symbol 0 ('zero'), a unary function symbol $s$ ('successor') as well as a function symbol $f$ for every primitive recursive function (including symbols $+$, $\times$, etc.), each of them being given its standard interpretation in $\mathbb{N}$ (see Section 3.3).
- A denumerable set $\mathcal{V}_1$ of *first-order variables*. For convenience, we shall still use the lowercase letters $x$, $y$, $z$, etc. to denote first-order variables, but these variables should not be confused with the $\lambda$-variables introduced in Section 2.

The set $FV(e)$ of all (free) variables of a first-order expression $e$ is defined as expected, as well as the corresponding operation of substitution, that we still write $e\{x := e'\}$.

3.1.2. *Formulæ.* Formulæ of second-order logic are defined (see Figure 1) from an additional set of symbols $\mathcal{V}_2$ of *second-order variables* (or *predicate variables*), using the uppercase letters $X$, $Y$, $Z$, etc. to represent such variables:

$$A, B ::= X(e_1, \ldots, e_k) \mid A \Rightarrow B \mid \forall x A \mid \forall X A. \qquad (X \in \mathcal{V}_2)$$

We assume that each second-order variable $X$ comes with an arity $k \geqslant 0$ (that we shall often leave implicit since it can be easily inferred from the context), and that for each arity $k \geqslant 0$, the subset of $\mathcal{V}_2$ formed by all second-order variables of arity $k$ is denumerable.

Intuitively, second-order variables of arity 0 represent (unknown) propositions, unary predicate variables represent predicates over individuals (or *sets* of individuals) whereas binary predicate variables represent binary relations (or sets of pairs), etc.

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

The set of free variables of a formula $A$ is written $FV(A)$. (This set may contain both first-order and second-order variables.) As usual, formulæ are identified up to $\alpha$-conversion, neglecting differences in bound variable names. Given a formula $A$, a first-order variable $x$ and a closed first-order expression $e$, we denote by $A\{x := e\}$ the formula obtained by replacing every free occurrence of $x$ by the first-order expression $e$ in the formula $A$, possibly renaming some bound variables of $A$ to avoid name clashes.

Last, although the formulæ of the language of second-order logic are constructed from atomic formulæ only using implication and first- and second-order universal quantifications, we can define other logical constructions (negation, conjunction disjunction, first- and second-order existential quantification as well as Leibniz equality) using the so called second-order encodings (cf Figure 1).

3.1.3. *Predicates and second-order substitution.* We call a *predicate of arity k* any expression of the form $P \equiv \lambda x_1 \cdots x_k \,.\, C$, where $x_1, \ldots, x_k$ are $k$ pairwise distinct first-order variables and where $C$ is an arbitrary formula. (Here, we (ab)use the $\lambda$-notation to indicate which variables $x_1, \ldots, x_k$ are abstracted in the formula $C$).

The set of free variables of a $k$-ary predicate $P \equiv \lambda x_1 \cdots x_k \,.\, C$ is defined by $FV(P) \equiv FV(C) \setminus \{x_1; \ldots; x_k\}$, and the application of the predicate $P \equiv \lambda x_1 \cdots x_k \,.\, C$ to a $k$-tuple of first-order expressions $e_1, \ldots, e_k$ is defined by letting

$$P(e_1, \ldots, e_k) \;\equiv\; (\lambda x_1 \cdots x_k \,.\, C)(e_1, \ldots, e_k) \;\equiv\; C\{x_1 := e_1; \ldots; x_k := e_k\}$$

(by analogy with $\beta$-reduction). Given a formula $A$, a $k$-ary predicate variable $X$ and an actual $k$-ary predicate $P$, we finally define the operation of *second-order substitution* $A\{X := P\}$ as follows:

$$\begin{array}{rcll}
X(e_1, \ldots, e_k)\{X := P\} & \equiv & P(e_1, \ldots, e_k) & \\
Y(e_1, \ldots, e_m)\{X := P\} & \equiv & Y(e_1, \ldots, e_m) & (Y \not\equiv X) \\
(A \Rightarrow B)\{X := P\} & \equiv & A\{X := P\} \Rightarrow B\{X := P\} & \\
(\forall x\, A)\{X := P\} & \equiv & \forall x\, A\{X := P\} & (x \notin FV(P)) \\
(\forall X\, A)\{X := P\} & \equiv & \forall X\, A & \\
(\forall Y\, A)\{X := P\} & \equiv & \forall Y\, A\{X := P\} & (Y \not\equiv X,\ Y \notin FV(P))
\end{array}$$

## 3.2. *A type system for classical second-order logic*

Through the formulæ-as-types correspondence (Howard 1980; Girard et al. 1989), we can see any formula $A$ of second-order logic as a type, namely, as the type of its proofs. We shall thus present the deduction system of classical second-order logic as a type system based on a typing judgement of the form $\Gamma \vdash t : A$, where

- $\Gamma$ is a typing context of the form $\Gamma \equiv x_1 : B_1, \ldots, x_n : B_n$, where $x_1, \ldots, x_n$ are pairwise distinct $\lambda$-variables and where $B_1, \ldots, B_n$ are arbitrary propositions;
- $t$ is a proof-like term, i.e. a $\lambda_c$-term containing no continuation constant $\mathbf{k}_\pi$;
- $A$ is a formula of second-order logic.

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

The type system of classical second-order logic is then defined from the typing rules of Figure 1. These typing rules are the usual typing rules of AF2 (Krivine 1993), plus a specific typing rule for the instruction $\mathbf{cc}$ which permits to recover the full strength of classical logic.

Using the encodings of second-order logic, we can derive from the typing rules of Figure 1 the usual introduction and elimination rules of absurdity, conjunction, disjunction, (first- and second-order) existential quantification and Leibniz equality (Krivine 1993). The typing rule for call/cc (law of Peirce) allows us to construct proof-terms for classical reasoning principles such as the excluded middle, *reductio ad absurdum*, de Morgan laws, etc.

### 3.3. *Classical second-order arithmetic (PA2)*

From now on, we consider the particular case of *second-order arithmetic* (PA2), where first-order expressions are intended to represent natural numbers. For that, we assume that every $k$-ary function symbol $f \in \Sigma$ comes with an interpretation in the standard model of arithmetic as a function $[\![f]\!] : \mathbb{N}^k \to \mathbb{N}$, so that we can give a denotation $[\![e]\!] \in \mathbb{N}$ to every closed first-order expression $e$. Moreover, we assume that each function symbol associated to a primitive recursive definition (cf Section 3.1.1) is given its standard interpretation in $\mathbb{N}$. In this way, every numeral $n \in \mathbb{N}$ is represented in the world of first-order expressions as the closed expression $s^n(0)$ that we still write $n$, since $[\![s^n(0)]\!] = n$.

3.3.1. *Induction.* Following Dedekind's construction of natural numbers, we consider the predicate $\mathsf{Nat}(x)$ (Girard et al. 1989; Krivine 1993) defined by

$$\mathsf{Nat}(x) \;\equiv\; \forall Z\,(Z(0) \Rightarrow \forall y\,(Z(y) \Rightarrow Z(s(y))) \Rightarrow Z(x))\,,$$

that defines the smallest class of individuals containing zero and closed under the successor function. One of the main properties of the logical system presented above is that the axiom of induction, that we can write $\forall x\ \mathsf{Nat}(x)$, is not derivable from the rules of Figure 1. As Krivine proved (Krivine 2009, Theorem 12), this axiom is not even (universally) realizable in general. To recover the strength of arithmetic reasoning, we need to relativize all first-order quantifications to the class $\mathsf{Nat}(x)$ of Dedekind numerals using the shorthands for *numeric quantifications*

$$\begin{aligned}\forall^{\mathsf{nat}} x\,A(x) &\;\equiv\; \forall x\,(\mathsf{Nat}(x) \Rightarrow A(x))\\ \exists^{\mathsf{nat}} x\,A(x) &\;\equiv\; \forall Z\,(\forall x(\mathsf{Nat}(x) \Rightarrow A(x) \Rightarrow Z) \Rightarrow Z)\end{aligned}$$

so that the *relativized induction axiom* becomes provable in second-order logic (Krivine 1993):

$$\forall Z\,(Z(0) \Rightarrow \forall^{\mathsf{nat}} x\,(Z(x) \Rightarrow Z(s(x))) \Rightarrow \forall^{\mathsf{nat}} x Z(x))\,.$$

3.3.2. *The axioms of PA2.* Formally, a formula $A$ is a *theorem* of second-order arithmetic (PA2) if it can be derived (using the rules of Figure 1) from the two axioms

- $\forall x\,\forall y\,(s(x) = s(y) \Rightarrow x = y)$ (Peano 3rd axiom)
- $\forall x\,\neg(s(x) = 0)$ (Peano 4th axiom)

expressing that the successor function is injective and not surjective, and from the definitional equalities attached to the (primitive recursive) function symbols of the signature:

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

- $\forall x\,(x + 0 = x), \quad \forall x\,\forall y\,(x + s(y) = s(x + y))$
- $\forall x\,(x \times 0 = 0), \quad \forall x\,\forall y\,(x \times s(y) = (x \times y) + x)$
- etc.

Unlike the non-relativized induction axiom – that requires a special treatment in PA2 – we shall see in Section 4.5 that all these axioms are realized by simple proof-like terms.

## 4. Classical realizability semantics

### 4.1. *Generalities*

Given a particular instance of the $\lambda_c$-calculus (defined from particular sets $\mathcal{B}$, $\mathcal{C}$ and from a particular relation of evaluation $\succ_1$ as described in Section 2), we shall now build a classical realizability model in which every closed formula $A$ of the language of PA2 will be interpreted as a set of closed terms $|A| \subseteq \Lambda$, called the *truth value* of $A$, and whose elements will be called the *realizers* of $A$.

4.1.1. *Poles, truth values and falsity values.* Formally, the construction of the realizability model is parameterized by a *pole* $\perp\!\!\!\perp$ in the sense of the following definition:

**Definition 5 (Poles).** A *pole* is any set of processes $\perp\!\!\!\perp \subseteq \Lambda \star \Pi$ which is closed under anti-evaluation, in the sense that both conditions $p \succ p'$ and $p' \in \perp\!\!\!\perp$ together imply that $p \in \perp\!\!\!\perp$ for all processes $p, p' \in \Lambda \star \Pi$.

We will mainly use one method to define a pole $\perp\!\!\!\perp$. From an arbitrary set of processes $P$, we can define *pole* as the complement set of the union of all threads starting from an element of $P$, that is

$$\perp\!\!\!\perp \;\equiv\; \Big(\bigcup_{p\in P} \mathbf{th}(p)\Big)^{c} \;\equiv\; \bigcap_{p\in P} (\mathbf{th}(p))^{c}\,.$$

It is indeed quite easy to check that $\perp\!\!\!\perp$ is closed by anti-reduction, and it is also the largest pole that does not intersect $P$. We shall say that such a definition is *thread-oriented*.

Let us now consider a fixed pole $\perp\!\!\!\perp$. We call a *falsity value* any set of stacks $S \subseteq \Pi$. Every falsity value $S \subseteq \Pi$ induces a *truth value* $S^{\perp\!\!\!\perp} \subseteq \Lambda$ that is defined by

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

Intuitively, every falsity value $S \subseteq \Pi$ represents a particular set of *tests*, while the corresponding truth value $S^{\perp\!\!\!\perp}$ represent the set of all *programs* that passes all tests in $S$ (w.r.t. the pole $\perp\!\!\!\perp$, that can be seen as the *challenge*). From the definition of $S^{\perp\!\!\!\perp}$, it is clear that the larger the falsity value $S$, the smaller the corresponding truth value $S^{\perp\!\!\!\perp}$, and vice-versa.

4.1.2. *Formulæ with parameters.* In order to interpret second-order variables that occur in a given formula $A$, it is convenient to enrich the language of PA2 with a new predicate symbol $\dot{F}$ of arity $k$ for every *falsity value function* $F$ of arity $k$, that is, for every function $F : \mathbb{N}^k \to \mathfrak{P}(\Pi)$ that associates a falsity value $F(n_1, \ldots, n_k) \subseteq \Pi$ to every $k$-tuple $(n_1, \ldots, n_k) \in \mathbb{N}^k$. A formula of the language enriched with the predicate symbols $\dot{F}$ is then called a *formula with parameters*.

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

Formally, this correspond to the formulæ defined by

$$A, B ::= X(e_1, \ldots, e_k) \mid A \Rightarrow B \mid \forall x A \mid \forall X A \mid \dot{F}(e_1, \ldots, e_k) \qquad X \in \mathcal{V}_2, F \in \mathfrak{P}(\Pi)^{\mathbb{N}^k}.$$

The notions of a *predicate with parameters* and of a *typing context with parameters* are defined similarly. The notations $FV(A)$, $FV(P)$, $FV(\Gamma)$, $\mathrm{dom}(\Gamma)$, $A\{x := e\}$, $A\{X := P\}$, etc. are extended to all formulæ $A$ with parameters, to all predicates $P$ with parameters and to all typing contexts $\Gamma$ with parameters in the obvious way.

### 4.2. *Definition of the interpretation function*

The interpretation of the closed formulæ with parameters is defined as follows:

**Definition 6 (Interpretation of closed formulæ with parameters).** The falsity value $\|A\| \subseteq \Pi$ of a closed formula $A$ with parameters is defined by induction on the number of connectives/quantifiers in $A$ from the equations

$$\begin{aligned}
\|\dot{F}(e_1, \ldots, e_k)\| &= F(\llbracket e_1 \rrbracket, \ldots, \llbracket e_k \rrbracket) \\
\|A \Rightarrow B\| &= |A| \cdot \|B\| \quad = \quad \{t \cdot \pi \ : \ t \in |A|,\ \pi \in \|B\|\} \\
\|\forall x\, A\| &= \bigcup_{n \in \mathbb{N}} \|A\{x := n\}\| \\
\|\forall X\, A\| &= \bigcup_{F : \mathbb{N}^k \to \mathfrak{P}(\Pi)} \|A\{X := \dot{F}\}\|, \qquad \text{(if } X \text{ has arity } k\text{)}
\end{aligned}$$

whereas its truth value $|A| \subseteq \Lambda$ is defined by $|A| = \|A\|^{\perp\!\!\!\perp}$. Finally, defining $\top \equiv \dot{\varnothing}$ (recall that we have $\bot \equiv \forall X.X$), one can check that we have

$$\|\top\| = \varnothing \qquad |\top| = \Lambda \qquad \|\bot\| = \Pi.$$

Since the falsity value $\|A\|$ (resp. the truth value $|A|$) of $A$ actually depends on the pole $\perp\!\!\!\perp$, we shall write it sometimes $\|A\|_{\perp\!\!\!\perp}$ (resp. $|A|_{\perp\!\!\!\perp}$) to recall the dependency. Given a closed formula $A$ with parameters and a closed term $t \in \Lambda$, we say that:

- *$t$ realizes $A$* and write $t \Vdash A$ when $t \in |A|_{\perp\!\!\!\perp}$.
  (This notion is relative to a particular pole $\perp\!\!\!\perp$.)
- *$t$ universally realizes $A$* and write $t \Vdash\!\!\!\Vdash A$ when $t \in |A|_{\perp\!\!\!\perp}$ for all poles $\perp\!\!\!\perp$.

From these definitions, we have

**Lemma 1 (Law of Peirce).** Let $A$ and $B$ be two closed formulæ with parameters:

(1) If $\pi \in \|A\|$, then $\mathbf{k}_\pi \Vdash A \Rightarrow B$.
(2) $\mathfrak{cc} \Vdash\!\!\!\Vdash ((A \Rightarrow B) \Rightarrow A) \Rightarrow A$.

### 4.3. *Valuations and substitutions*

In order to express the soundness invariants relating the type system of Section 3 with the classical realizability semantics defined above, we need to introduce some more terminology.

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

**Definition 7 (Valuations).** A *valuation* is a function $\rho$ 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 $k$.

- Given a valuation $\rho$, a first-order variable $x$ and a natural number $n \in \mathbb{N}$, we denote by $\rho, x \leftarrow n$ the valuation defined by

$$(\rho, x \leftarrow n) \;=\; \rho_{|\,\mathrm{dom}(\rho)\setminus\{x\}} \cup \{x \leftarrow n\}\,.$$

- Given a valuation $\rho$, a second-order variable $X$ of arity $k$ and a falsity value function $F : \mathbb{N}^k \to \mathfrak{P}(\Pi)$, we denote by $\rho, X \leftarrow F$ the valuation defined by

$$(\rho, X \leftarrow F) \;=\; \rho_{|\,\mathrm{dom}(\rho)\setminus\{X\}} \cup \{X \leftarrow F\}\,.$$

To every pair $(A, \rho)$ formed by a (possibly open) formula $A$ of PA2 and a valuation $\rho$, we associate a *closed* formula with parameters $A[\rho]$ that is defined by

$$A[\rho] \;\equiv\; A\{x_1 := \rho(x_1); \ldots; x_n := \rho(x_n); X_1 := \dot{\rho}(X_1); \ldots; X_m := \dot{\rho}(X_m)\},$$

where $x_1, \ldots, x_n, X_1, \ldots, X_m$ are the free variables of $A$, and writing $\dot{\rho}(X_i)$ the predicate symbol associated to the falsity value function $\rho(X_i)$. This operation naturally extends to typing contexts by letting $(x_1 : A_1, \ldots, x_n : A_n)[\rho] \equiv x_1 : A_1[\rho], \ldots, x_n : A_n[\rho]$.

**Definition 8 (Substitutions).** A *substitution* is a finite function $\sigma$ from $\lambda$-variables to closed $\lambda_c$-terms. Given a substitution $\sigma$, a $\lambda$-variable $x$ and a closed $\lambda_c$-term $u$, we denote by $\sigma, x := u$ the substitution defined by $(\sigma, x := u) \equiv \sigma_{|\,\mathrm{dom}(\sigma)\setminus\{x\}} \cup \{x := u\}$.

Given an open $\lambda_c$-term $t$ and a substitution $\sigma$, we denote by $t[\sigma]$ the term defined by

$$t[\sigma] \;\equiv\; t\{x_1 := \sigma(x_1); \ldots; x_n := \sigma(x_n)\},$$

where $\mathrm{dom}(\sigma) = \{x_1, \ldots, x_n\}$. Notice that $t[\sigma]$ is closed as soon as $FV(t) \subseteq \mathrm{dom}(\sigma)$. We say that a substitution $\sigma$ *realizes* a closed context $\Gamma$ with parameters and write $\sigma \Vdash \Gamma$ if:

- $\mathrm{dom}(\sigma) = \mathrm{dom}(\Gamma)$;
- $\sigma(x) \Vdash A$ for every declaration $(x : A) \in \Gamma$.

### 4.4. *Adequacy*

Given a fixed pole $\perp\!\!\!\perp$, we say that:

- A typing judgement $\Gamma \vdash t : A$ is *adequate* (w.r.t. the pole $\perp\!\!\!\perp$) if for all valuations $\rho$ and for all substitutions $\sigma \Vdash \Gamma[\rho]$ we have $t[\sigma] \Vdash A[\rho]$.
- More generally, we say that an inference rule

$$\frac{J_1 \quad \cdots \quad J_n}{J_0}$$

is adequate (w.r.t. the pole $\perp\!\!\!\perp$) if the adequacy of all typing judgements $J_1, \ldots, J_n$ implies the adequacy of the typing judgement $J_0$.

From the latter definition, it is clear that a typing judgement that is derivable from a set of adequate inference rules is adequate too.

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

**Proposition 2 (Adequacy).** The typing rules of Figure 1 are adequate w.r.t. any pole $\perp\!\!\!\perp$, as well as all the judgements $\Gamma \vdash t : A$ that are derivable from these rules.

*Proof.* We will only sketch the proof for the introduction and elimination rules for implication. The rule for $\Pi$ is obvious and the rule for $\mathfrak{cc}$ directly stems from Lemma 1, while introduction and elimination rules for universal quantifiers results from the definition of the corresponding falsity values. A more detailed proof can be found in Krivine's article (Krivine 2003).

- Case $\dfrac{\Gamma \vdash t : A \Rightarrow B \qquad \Gamma \vdash t : A}{\Gamma \vdash tu : B}$:
  we assume that $\Gamma \vdash t : A \Rightarrow B$ and $\Gamma \vdash u : B$ are adequate w.r.t. $\perp\!\!\!\perp$, and pick a valuation $\rho$ and a substitution $\sigma$ such that $\sigma \Vdash \Gamma[\rho]$.. We want to show that $(tu)[\sigma] \Vdash B[\rho]$. It suffices to show that if $\pi \in \|B[\rho]\|$, then $(tu)[\sigma] \star \pi \in \perp\!\!\!\perp$. Applying the Push rule, we get
  $$(tu)[\sigma] \star \pi \succ t[\sigma] \star u[\sigma] \cdot \pi.$$
  By hypothesis, we have $u[\sigma] \Vdash A[\rho]$ (and then $u[\sigma] \cdot \pi \in \|(A \Rightarrow B)[\rho]\|$), and $t[\sigma] \Vdash (A \Rightarrow B)[\rho]$, so that $t[\sigma] \star u[\sigma] \cdot \pi$ belongs to $\perp\!\!\!\perp$. We conclude by anti-reduction.
- Case $\dfrac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \Rightarrow B}$:
  we assume that $\Gamma, x : A \vdash t : B$ is adequate w.r.t $\perp\!\!\!\perp$. This means that for any valuation $\rho$, any $u \Vdash A[\rho]$ and any $\sigma \Vdash \Gamma[\rho]$, denoting by $\sigma'$ the substitution $\sigma, x := u$, we have $t[\sigma'] \Vdash B[\rho]$. Let us pick a valuation $\rho$ and a substitution $\sigma$ such that $\sigma \Vdash \Gamma[\rho]$. We want to show that $(\lambda x.t)[\sigma] \Vdash (A \Rightarrow B)[\rho]$. Let $u \cdot \pi$ be a stack in $\|(A \Rightarrow B)[\rho]\|$. Applying the Grab rule, we have
  $$(\lambda x.t)[\sigma] \star u \cdot \pi \succ t[\sigma, x := u] \star \pi.$$
  By hypothesis, we have $u \Vdash A[\rho]$, and so $t[\sigma, x := u] \Vdash B[\rho]$. Thus, $t[\sigma, x := u] \star \pi$ belongs to $\perp\!\!\!\perp$ and we conclude by anti-reduction. □

Since the typing rules of Figure 1 involve no continuation constant, every realizer that comes from a proof of second-order logic by Prop. 2 is thus a proof-like term.

### 4.5. *Realizing the axioms of PA2*

Let us recall that in PA2, Leibniz equality $e_1 = e_2$ is defined by $e_1 = e_2 \equiv \forall Z\,(Z(e_1) \Rightarrow Z(e_2))$.

**Proposition 3 (Realizing Peano axioms Krivine (2009)).**

1. $\lambda z\,.\,z \;\Vdash\!\!\!\Vdash\; \forall x\,\forall y\,(s(x) = s(y) \Rightarrow x = y)$.
2. $\lambda z\,.\,zu \;\Vdash\!\!\!\Vdash\; \forall x\,(s(x) = 0 \Rightarrow \bot)$ (where $u$ is any term such that $FV(u) \subseteq \{z\}$).
3. $\lambda z\,.\,z \;\Vdash\!\!\!\Vdash\; \forall x_1 \cdots \forall x_k\,(e_1(x_1, \ldots, x_n) = e_2(x_1, \ldots, x_k))$
   for all arithmetic expressions $e_1(x_1, \ldots, x_n)$ and $e_2(x_1, \ldots, x_k)$ such that
   $\mathbb{N} \models \forall x_1 \cdots \forall x_k\,(e_1(x_1, \ldots, x_n) = e_2(x_1, \ldots, x_k))$.

From this, we deduce the main theorem:

**Theorem 1 (Realizing the theorems of PA2).** If $A$ is a theorem of PA2 (in the sense defined in Section 3.3.2), then there is a closed proof-like term $t$ such that $t \;\Vdash\!\!\!\Vdash\; A$.

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

*Proof.* Immediately follows from Prop. 2 and 3. □

### 4.6. *The full standard model of PA2 as a degenerate case*

It is easy to see that when the pole $\perp\!\!\!\perp$ is empty, the classical realizability model defined above collapses to the *full standard model of PA2*, that is: to the model (in the sense of Tarski) where individuals are interpreted by the elements of $\mathbb{N}$ and where second-order variables of arity $k$ are interpreted by all the subsets of $\mathbb{N}^k$. For that, we first notice that when $\perp\!\!\!\perp = \varnothing$, the truth value $S^{\perp\!\!\!\perp}$ associated to an arbitrary falsity value $S \subseteq \Pi$ can only take two different values: $S^{\perp\!\!\!\perp} = \Lambda_c$ when $S = \varnothing$, and $S^{\perp\!\!\!\perp} = \varnothing$ when $S \neq \varnothing$. Moreover, we easily check that the realizability interpretation of implication and universal quantification mimics the standard truth value interpretation of the corresponding logical construction in the case where $\perp\!\!\!\perp = \varnothing$. Writing $\mathcal{M}$ for the full standard model of PA2, we thus easily show that:

**Proposition 4.** If $\perp\!\!\!\perp = \varnothing$, then for every closed formula $A$ of PA2 we have

$$|A| = \begin{cases} \Lambda & \text{if } \mathcal{M} \models A \\ \varnothing & \text{if } \mathcal{M} \not\models A. \end{cases}$$

*Proof.* We more generally show that for all formulæ $A$ and for all valuations $\rho$ closing $A$ (in the sense defined in section 4.2), we have

$$|A[\rho]| = \begin{cases} \Lambda & \text{if } \mathcal{M} \models A[\tilde{\rho}] \\ \varnothing & \text{if } \mathcal{M} \not\models A[\tilde{\rho}], \end{cases}$$

where $\tilde{\rho}$ is the valuation in $\mathcal{M}$ (in the usual sense) defined by

- $\tilde{\rho}(x) = \rho(x)$ for all first-order variables $x$;
- $\tilde{\rho}(X) = \{(n_1, \dots, n_k) \in \mathbb{N}^k : \rho(X)(n_1, \dots, n_k) = \varnothing\}$ for all second-order variables $X$ of arity $k$.

(This characterization is proved by a straightforward induction on $A$.) □

An interesting consequence of the above lemma is the following:

**Corollary 1.** If a closed formula $A$ has a universal realizer $t \Vdash A$, then $A$ is true in the full standard model $\mathcal{M}$ of PA2.

*Proof.* If $t \Vdash A$, then $t \in |A|_{\varnothing}$. Therefore, $|A|_{\varnothing} = \Lambda$ and $\mathcal{M} \models A$. □

However, the converse implication is false in general, since the formula $\forall x\, \mathsf{Nat}(x)$ (cf Section 3.3.1) that expresses the induction principle over individuals is obviously true in $\mathcal{M}$, but it has no universal realizer when evaluation is deterministic (Krivine 2009, Theorem 12).

### 4.7. *Relativization to canonical integers*

We previously explained in Section 3.3.1 that we needed to relativize first-order quantifications to the class $\mathsf{Nat}(x)$. If we have as expected $\bar{n} \Vdash \mathsf{Nat}(n)$ for any $n \in \mathbb{N}$, there are realizers of $\mathsf{Nat}(n)$ different from $\bar{n}$. Intuitively, a term $t \Vdash \mathsf{Nat}(n)$ represents the integer $n$, but $n$ might be present only as a computation, and not directly as a computed value.

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

The usual technique to retrieve $\bar{n}$ from such a term consist in the use of a *storage operator* $T$, which simulates a call-by-value reduction (for integers) on the first argument on the stack. While such a term is easy to define, its use would make our definition of game harder. Rather than that, we define a new asymmetrical implication where the left member must be an integer value (somehow forcing call-by-value reduction on all integers), and the interpretation of this new implication.

| | |
|---|---|
| **Formulæ** | $A, B ::= \ldots \mid \{e\} \Rightarrow A$ |
| **Falsity value** | $\|\|\{e\} \Rightarrow A\|\| = \{\bar{n} \cdot \pi : [\![e]\!] = n \wedge \pi \in \|\|A\|\|\}.$ |

We finally define the corresponding shorthands for relativized quantifications:

$$\forall^{\mathsf{N}} x\, A(x) \equiv \forall x\, (\{x\} \Rightarrow A(x))$$
$$\exists^{\mathsf{N}} x\, A(x) \equiv \forall Z\, (\forall x(\{x\} \Rightarrow A(x) \Rightarrow Z) \Rightarrow Z).$$

It is easy to check that this relativization of first-order quantification is equivalent (in terms of realizability) to the one defined in Section 3.3.1 and that the relativized principle of induction holds.

**Proposition 5.** Let $T$ be a storage operator. The following holds for any formula $A(x)$:

1. $\lambda x.x \Vdash \forall^{\mathsf{N}} x \mathsf{Nat}(x)$;
2. $\lambda x.x \Vdash \forall^{\mathsf{nat}} x.A(x) \Rightarrow \forall^{\mathsf{N}} x.A(x)$;
3. $\lambda x.Tx \Vdash \forall^{\mathsf{N}} x.A(x) \Rightarrow \forall^{\mathsf{nat}} x.A(x)$.

For an arithmetic formula $\Phi$ and $\alpha, \beta \in \{\mathsf{N}, \mathsf{nat}\}$, we denote by $\Phi^{\alpha,\beta}$ the formula $\Phi$ where every existential quantifier $\exists x$ has been relativized to $\exists^{\alpha} x$, and every universal quantifier $\forall y$ to $\forall^{\beta} y$. We let the reader check that if we define the following $\lambda_c$-terms :

$$\xi_0 := \lambda tu.t\,(Tu) \qquad \chi_0 := \lambda tu.t\,(\lambda xy.u\,x\,(Ty))$$
$$\xi_n := \lambda tu.t\,(\lambda xy.(Tu)\,x\,(\xi_{n-1}\,y)) \qquad \chi_n := \lambda tu.t\,(\lambda xy.u\,x\,(T(\chi_{n-1}\,y))$$

we get the following proposition.

**Proposition 6.** Given a pole $\perp\!\!\!\perp$, the following holds for all $n \in \mathbb{N}$ and any $\Sigma^0_{2n}$-formula $\Phi_n$:

1. If $t \Vdash \Phi_n^{\mathsf{N,nat}}$ then $t \Vdash \Phi_n^{\mathsf{N,N}}$.
2. If $t \Vdash \Phi_n^{\mathsf{N,nat}}$ then $t \Vdash \Phi_n^{\mathsf{nat,nat}}$.
3. If $t \Vdash \Phi_n^{\mathsf{nat,nat}}$ then $\xi_n t \Vdash \Phi_n^{\mathsf{N,N}}$.
4. If $t \Vdash \Phi_n^{\mathsf{N,N}}$ then $\chi_n t \Vdash \Phi_n^{\mathsf{nat,nat}}$.

In the case of arithmetical formulæ , we can then easily transform a realizer for one relativization into a realizer for the other one. In the authors' opinion, this enhances the fact that the choice of a relativization is more a question of convenience than a choice having a deep impact on the behaviour of the realizers.

For further details about relativization and storage operators, please refer to Section 2.9 and 2.10.1 of Rieg's Ph.D. thesis (Rieg 2014).

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

4.8. *Leibniz equality*

Before going further, we would like to draw the reader's attention to the treatment that is given to equality, which is crucial in what follows. We recall that the equality of two arithmetical expressions $e_1$ and $e_2$ is defined by the second-order encoding

$$e_1 = e_2 \quad \equiv \quad \forall W(W(e_1) \Rightarrow W(e_2)).$$

Unfolding the definitions of falsity values, we easily get the following lemma:

**Lemma 2.** Given a pole $\perp\!\!\!\perp$, if $e_1, e_2$ are arithmetical expressions, we have

$$\|e_1 = e_2\| = \begin{cases} \|\forall X(X \Rightarrow X)\| & \text{if } \mathcal{M} \models e_1 = e_2 \\ \|\top \Rightarrow \bot\| & \text{if } \mathcal{M} \models e_1 \neq e_2. \end{cases}$$

The following corollaries are straightforward but will be very useful in Sections 5–7, so it is worth mentionning them briefly now.

**Corollary 2.** Let $\perp\!\!\!\perp$ be a fixed pole, $e_1, e_2$ some arithmetical expressions, $u \in \Lambda$ a closed term and $\pi \in \Pi$ a stack such that $u \cdot \pi \in \|e_1 = e_2\|$. If $\mathcal{M} \models e_1 = e_2$ then $u \star \pi \in \perp\!\!\!\perp$.

*Proof.* By Lemma 2, we have

$$u \cdot \pi \in \|\forall X(X \Rightarrow X)\| = \{u \cdot \pi : \exists S \in \mathfrak{P}(\Pi), \pi \in S \wedge u \in |\dot{S}|\}$$

so that $u \in S^{\perp\!\!\!\perp}$ and $u \star \pi \in \perp\!\!\!\perp$. □

**Corollary 3.** Given a pole $\perp\!\!\!\perp$, if $e_1, e_2$ are arithmetical expressions, and $u \in \Lambda$, $\pi \in \Pi$ are such that $u \cdot \pi \notin \|e_1 = e_2\|$, then

1. $\mathcal{M} \models e_1 = e_2$.
2. $u \star \pi \notin \perp\!\!\!\perp$.

*Proof.*

1. By contraposition: if $\mathcal{M} \models e_1 \neq e_2$, by Lemma 2 we have $\|e_1 = e_2\| = \|\top \Rightarrow \bot\| = \Lambda \times \Pi$, hence $u \cdot \pi \in \|e_1 = e_2\|$.
2. By (1) we have $\|e_1 = e_2\| = \|\forall X(X \Rightarrow X)\| = \bigcup_{S \in \mathfrak{P}(\Pi)} \|\dot{S} \Rightarrow \dot{S}\|$, hence $u \cdot \pi \notin \|e_1 = e_2\|$ implies that if $S = \{\pi\}$, $u \cdot \pi \notin \|\dot{S} \Rightarrow \dot{S}\|$, *i.e.* $u \not\Vdash S$, so $u \star \pi \notin \perp\!\!\!\perp$. □

## 5. The specification problem

5.1. *The specification problem*

In the continuity of the work that has been done for Peirce's Law (Guillermo and Miquel 2014), we are interested in the specification problem, which is to give a purely computational characterization of the universal realizers of a given formula $A$. As mentioned in this paper, this problem is much more subtle than in the case of intuitionistic realizability, what could be justified, amongst other things, by the presence of extra instructions that do not exist in the pure $\lambda$-calculus and by the ability of a realizer to backtrack at any time. Some very simple case, as the identity-type

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

($\forall X(X \Rightarrow X)$) or the boolean-type ($\forall X(X \Rightarrow X \Rightarrow X)$), are quite easy to specify, but more interestingly, it turns out that some more complex formulæ, for instance the Law of Peirce, can also be fully specified (Guillermo and Miquel 2014). In the following, we will focus on the generic case of arithmetical formulæ. A premise of this work was done by the first author for the particular case of formulæ of the shape $\exists^{\mathsf{N}} n \forall^{\mathsf{N}} y(f(x,y) = 0)$ (Guillermo 2008). In the general case (that is with a finite alternation of quantifiers) an attempt to characterize the threads of universal realizers is also given in an article of Krivine (Krivine 2003), but in the end it only provides us with the knowing of the final state, whereas we are here interested in a specification of the full reduction process. As in (Guillermo 2008), our method will rely on game-theoretic interpretation of the formulæ. Before going more into details, let us first look at the easiest example of specification.

**Example 1 (Identity type).** In the language of second-order logic, the identity type is described by the formula $\forall X(X \Rightarrow X)$. A closed term $t \in \Lambda$ is said to be *identity-like* if $t \star u \cdot \pi \succ u \star \pi$ for all $u \in \Lambda$ and $\pi \in \Pi$. Examples of identity-like terms are of course the identity function $I \equiv \lambda x.x$, but also terms such as $II$, $\delta I$ (where $\delta \equiv \lambda x.xx$), $\lambda x.\mathbf{cc}(\lambda k.x)$, $\mathbf{cc}(\lambda k.kI\delta k)$, etc.

**Proposition 7 (Guillermo and Miquel 2014).** For all terms $t \in \Lambda$, the following assertions are equivalent:

1. $t \Vdash \forall X(X \Rightarrow X)$.
2. $t$ is identity-like.

The interesting direction of the proof is (1) $\Rightarrow$ (2). We prove it with the methods of threads, that we use later in Section 6. Assume $t \Vdash \forall X(X \Rightarrow X)$, and consider $u \in \Lambda, \pi \in \Pi$. We want to prove that $t \star u \cdot \pi \succ u \star \pi$. We define the pole

$$\perp\!\!\!\perp \equiv (\mathbf{th}(t \star u \cdot \pi))^c \equiv \{p \in \Lambda \star \Pi : (t \star u \cdot \pi \not\succ p)\}$$

as well as the falsity value $S = \{\pi\}$. From the definition of $\perp\!\!\!\perp$, we know that $t \star u \cdot \pi \notin \perp\!\!\!\perp$. As $t \Vdash \dot{S} \Rightarrow \dot{S}$ and $\pi \in \|\dot{S}\|$, we get $u \not\Vdash S$. This means that $u \star \pi \notin \perp\!\!\!\perp$, that is $t \star u \cdot \pi \succ u \star \pi$.

### 5.2. *Arithmetical formulæ*

In this paper, we want to treat the case of first-order arithmetical formulæ, that are $\Sigma^0_n$-formulæ. As we explained in Section 3.3.1, in order to recover the strength of arithmetical reasoning, we will relativize all first-order quantifications to the class $\mathsf{Nat}(x)$. Besides, relativizing the quantifiers make the individuals visible in the stacks: indeed, a stack belonging to $\|\forall^{\mathsf{N}} xA(x)\|$ is of the shape $\overline{n} \cdot \pi$ with $\pi \in \|A(n)\|$, whereas a stack of $\|\forall xA(x)\|$ is of the form $\pi \in \|A(n)\|$ for some $n \in \mathbb{N}$ that the realizers do not have any physical access to.

**Definition 9.** We define inductively the following classes of formulæ:

- $\Sigma^0_0$- and $\Pi^0_0$-formulæ are the formulæ of the form $f(\vec{e}) = 0$, where $f$ is a primitive recursive function and $\vec{e}$ a list of first-order expressions.
- $\Pi^0_{n+1}$-formulæ are the formulæ of the form $\forall^{\mathsf{N}} xF$, where $F$ is a $\Sigma^0_n$-formula.
- $\Sigma^0_{n+1}$-formulæ are the formulæ of the form $\exists^{\mathsf{N}} xF$, where $F$ is a $\Pi^0_n$-formula.

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

In the ground model $\mathcal{M}$, any closed $\Sigma^0_n$- or $\Pi^0_n$-formula $\Phi$ naturally induces a game between two players $\exists$ and $\forall$, that we shall name Eloise and Abelard from now on. Both players instantiate the corresponding quantifiers in turns, Eloise for defending the formula and Abelard for attacking it. The game, whose depth is bounded by the number of quantifications, proceeds as follows:

- When $\Phi$ is $\exists x\Phi'$, Eloise has to give an integer $m \in \mathbb{N}$, and the game goes on over the closed formula $\Phi'\{x := m\}$.
- When $\Phi$ is $\forall y\Phi'$, Abelard has to give an integer $n \in \mathbb{N}$, and the game goes on over the closed formula $\Phi'\{y := n\}$.
- When $\Phi$ is atomic and $\mathcal{M} \vDash \Phi$ ($\Phi$ is true), Eloise wins, otherwise Abelard wins.

We say that a player has a *winning strategy* if (s)he has a way of playing that ensures him/her the victory independently of the opponent moves. It is obvious from Tarski's definition of truth that a closed arithmetical formula $\Phi$ is true in the ground model if and only if Eloise has a winning strategy.

The problem with this too simple definition is that there exists true formulæ whose game only has non-computable winning strategies (as we shall see below), so that they cannot be implemented by $\lambda$-terms. This is why in classical logic, we will need to relax the rules of the above game to allow backtracking.

### 5.3. *The Halting problem or the need of backtrack*

For instance, let us consider one of the primitive recursive functions $f : \mathbb{N}^3 \to \mathbb{N}$ such that

$$f(m, n, p) = 0 \qquad \text{iff } (n > 0 \wedge \mathrm{Halt}(m, n)) \vee (n = 0 \wedge \neg\mathrm{Halt}(m, p)),$$

where $\mathrm{Halt}(m, n)$ is the primitive recursive predicate expressing that the $m$th Turing machine has stopped before $n$ evaluation steps (in front of the empty tape). From this, we consider the game on the formula

$$\Phi_H \equiv \forall^{\mathsf{N}} x \exists^{\mathsf{N}} y \forall^{\mathsf{N}} z (f(x, y, z) = 0)$$

that expresses that any Turing machine terminates or does not terminate. (Intuitively $y$ equals 0 when the machine $x$ does not halt, and it represents a number larger than the execution length of $x$ otherwise.) Yet, there is no pure $\lambda$-term that can compute directly from an $m \in \mathbb{N}$ an integer $n_m$ such that $\forall^{\mathsf{N}} z(f(m, n_m, z) = 0)$ (such a term would break the halting problem). However, $\Phi_H$ could be classically realized, using the $\mathfrak{cc}$ instruction. Let $\Theta$ be a $\lambda$-term such that

$$\Theta \star \overline{m} \cdot \overline{n} \cdot t_0 \cdot t_1 \cdot \pi \succ \begin{cases} t_0 \star \pi & \text{if the } m^{\text{th}} \text{ Turing machine stops before } n \text{ steps} \\ t_1 \star \pi & \text{otherwise} \end{cases}$$

and let $t_H$ be the following term :

$$\begin{aligned} T[m, u, k] &\equiv \lambda pv.\Theta\ m\ p\ (k\ (u\ p\ \lambda pv.v))\ v \\ t_H &\equiv \lambda mu.\mathfrak{cc}\ (\lambda k.u\ \overline{0}\ T[m, u, k]). \end{aligned}$$

If we think of $t_H$ as a strategy for Eloise *with backtrack allowed*, we can analyse its computational behaviour this way:

- First Eloise receives the code $m$ of a Turing machine $\mathscr{M}$, and chooses to play $n = 0$, that is '$\mathscr{M}$ *never stops*'.

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

- Then, Abelard answers a given number of steps $p$, and Eloise checks if $\mathscr{M}$ stops before $p$ steps and distinguishes two cases :
  - either $\mathscr{M}$ is still running after $p$ steps, hence $f(m, 0, p) = 0$ and Eloise wins.
  - either $\mathscr{M}$ does stop before $p$ steps, then Eloise backtracks to the previous position and instead of 0, it plays $p$, that is '$\mathscr{M}$ *stops before* $p$ *steps*', which ensures him victory whatever Abelard plays after.

**Proposition 8.** $t_H \Vdash \Phi_H$

*Proof.* Let us consider a fixed pole $\bot\!\!\!\bot$ and let $m \in \mathbb{N}$ be an integer, $\mathscr{M}$ be the $m$th Turing machine, and a stack $u \cdot \pi \in \|\exists^{\mathsf{N}} y \forall^{\mathsf{N}} z(f(m, y, z) = 0)\|$, and let us prove that $t_H \star \overline{m} \cdot u \cdot \pi \in \bot\!\!\!\bot$. We know that

$$t_H \star \overline{m} \cdot u \cdot \pi \succ u \star \overline{0} \cdot T[\overline{m}, u, \mathbf{k}_\pi] \cdot \pi$$

by anti-reduction, it suffices to prove that $T[\overline{m}, u, \mathbf{k}_\pi] \Vdash \forall^{\mathsf{N}} z(f(m, 0, z) = 0)$. Thus, let us consider $p \in \mathbb{N}$ and a stack $u' \cdot \pi' \in \|f(m, 0, p) = 0\|$. We distinguish two cases:

- $\mathscr{M}$ is still running after $p$ steps (that is $\mathcal{M} \vDash \neg\mathrm{Halt}(m, p)$). In this case, we have $f(m, 0, p) = 0$, and so by Corollary 2, $u' \star \pi' \in \bot\!\!\!\bot$. Furthermore, by definition of $\Theta$, we have

  $$T[\overline{m}, u, \mathbf{k}_\pi] \star \overline{p} \cdot u' \cdot \pi' \succ u' \star \pi' \in \bot\!\!\!\bot$$

  which concludes the case by anti-reduction.
- $\mathscr{M}$ stops before $p$ steps ($\mathcal{M} \vDash \mathrm{Halt}(m, p)$). By definition of $\Theta$, we have in this case

  $$T[\overline{m}, u, \mathbf{k}_\pi] \star \overline{p} \cdot u' \cdot \pi' \succ \mathbf{k}_\pi \star (u\ \overline{p}\ (\lambda pv.v)) \cdot \pi' \succ u \star \overline{p} \cdot \lambda pv.v \cdot \pi$$

  hence it suffices to show that $\lambda pv.v \Vdash \forall^{\mathsf{N}} z(f(m, p, z) = 0)$. But this is clear, as $\mathcal{M} \vDash \mathrm{Halt}(m, p)$, we have for any $s \in \mathbb{N}$, $\mathcal{M} \vDash f(m, p, s) = 0$. Therefore, if we consider any integer $s \in \mathbb{N}$ and any stack $u'' \cdot \pi'' \in \|f(m, p, s) = 0\|$, as in the previous case, from Corollary 2 we get $u'' \star \pi'' \in \bot\!\!\!\bot$ and

  $$\lambda pv.v \star \overline{s} \cdot u'' \cdot \pi'' \succ u'' \star \pi'' \in \bot\!\!\!\bot.$$

□

This leads us to define a new notion of game with backtrack over arithmetical formulæ.

### 5.4. $\mathbb{G}^0_\Phi$: *a first game with backtrack*

From now on, to simplify our work, we will always consider $\Sigma^0_{2h}$-formulæ, that is of the form

$$\exists^{\mathsf{N}} x_1 \forall^{\mathsf{N}} y_1 \ldots \exists^{\mathsf{N}} x_h \forall^{\mathsf{N}} y_h f(\vec{x}_h, \vec{y}_h) = 0,$$

where $h \in \mathbb{N}$ and the notation $\vec{x}_i$ refers to the tuple $(x_1, \ldots, x_i)$ (we will denote the concatenation by $\cdot : \vec{x}_i \cdot x_{i+1} = \vec{x}_{i+1}$). It is clear that any arithmetical formulæ can be written equivalently in that way, adding some useless quantifiers if needed.

Given such a formula $\Phi$, we define a game $\mathbb{G}^0_\Phi$ between Eloise and Abelard whose rules are basically the same as they were before, except that we will keep track of all the former $\exists$-positions, allowing Eloise to backtrack. This corresponds to the definition of Coquand's game (Coquand

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

1995). We call an ∃-*position* of size $i \in [\![0, h]\!]$ a pair of tuple of integers $(\vec{m}_i, \vec{n}_i)$ standing for the instantiation of the variables $\vec{x}_i, \vec{y}_i$, while a ∀-position will be a pair of the form $(\vec{m}_{i+1}, \vec{n}_i)$. We call *history of a game* and note $H$ the set of every former ∃-positions. The game starts with an empty history ($H = \{\varnothing\}$) and proceeds as follows:

- ∃-move: Eloise chooses a position $(\vec{m}_i, \vec{n}_i) \in H$ for some $i \in [\![0, h-1]\!]$, and proposes $m_{i+1} \in \mathbb{N}$, so that $(\vec{m}_{i+1}, \vec{n}_i)$ becomes the current ∀-position.
- ∀-move: Abelard has to answer with some $n_{i+1} \in \mathbb{N}$ to complete the position.

If $i + 1 = h$ and $f(\vec{m}_h, \vec{n}_h) = 0$, then Eloise wins and the game stops. Otherwise, we simply add the new ∃-position $(\vec{m}_{i+1}, \vec{n}_{i+1})$ to $H$, and the game goes on. We say that Abelard wins if the game goes on infinitely, that is if Eloise never wins.

Given a set $H$ of former ∃-positions, we will say that Eloise has a *winning strategy* and write $H \in \mathbb{W}^0_\Phi$ if she has a way of playing that ensures her a victory, independently of future Abelard moves.

Formally, we define the set $\mathbb{W}^0_\Phi$ by induction with the two following rules:

1. If there exists $(\vec{m}_h, \vec{n}_h) \in H$ such that $\mathcal{M} \vDash f(\vec{m}_h, \vec{n}_h) = 0$:

$$\frac{}{H \in \mathbb{W}^0_\Phi}\ (\text{Win})$$

2. For all $i < h$, $(\vec{m}_i, \vec{n}_i) \in H$ and $m \in \mathbb{N}$

$$\frac{H \cup \{(\vec{m}_i \cdot m, \vec{n}_i \cdot n)\} \in \mathbb{W}^0_\Phi \quad \forall n \in \mathbb{N}}{H \in \mathbb{W}^0_\Phi}\ (\text{Play})$$

Given a formula Φ, the difference between this game and the one we defined in Section 5.2 is that this one allows Eloise to make some wrong tries before ending on a winning positio. This means for Abelardthat its winning condition changes from *knowing one position* on which Eloise is not winning (the play ends on a false atomic formula) to always be able to find one (the play never ends), that is *Eloise is not knowing* any.

Clearly, there is a winning strategy for $\mathbb{G}^0_\Phi$ if and only if there was one in the previous game[†]. It is even easy to see that for any formula Φ, we have

**Proposition 9.** $\mathcal{M} \vDash \Phi$ iff $\{\varnothing\} \in \mathbb{W}^0_\Phi$

Given a formula Φ, in both games the existence of a winning strategy is equivalent to the truth in the model, hence such a definition does not carry anything new from an outlook of model theory, the interest of this definition is fundamentally computational. For instance, for the halting problem, this will now allow Eloise to use the strategy we described in the previous section.

Besides, it is worth noting that in general, the match somehow grows among a tree of height $h$, as we shall see in the following example.

**Example 2.** We define the following function

$$g : \begin{cases} \mathbb{N}^2 & \to & \mathbb{N} \\ (x, y) & \mapsto & x + (1 \dot{-} x)y \end{cases},$$

[†] It suffices to remove the 'bad tries' to keep only the winning move.

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

| Start | Eloise move | Abelard | new ∃-position |
|---|---|---|---|
| $\emptyset, \emptyset$ | 0 | 1 | $0, 1$ |
| $\emptyset, \emptyset$ | 1 | 0 | $1, 0$ |
| $1, 0$ | 1 | 1 | $1{\cdot}1, 0{\cdot}1$ |
| $1, 0$ | 2 | 2 | $1{\cdot}2, 0{\cdot}2$ |
| $\emptyset, \emptyset$ | 2 | 0 | $2, 0$ |
| $0, 1$ | 2 | 1 | $0{\cdot}2, 1{\cdot}1$ |
| $0{\cdot}2, 1{\cdot}1$ | Eloise wins | / | / |

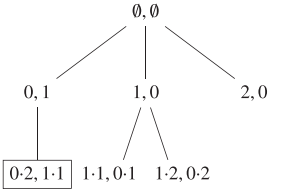



Fig. 2. Example of a match for $\mathbb{G}^0_\varphi$.

where $\dot{-}$ refers to the truncated subtraction. Notice that $g(x, \cdot)$ is clearly bounded if $x \neq 0$. Then, we consider $f$ a function such that

$$f(x_1, y_1, x_2, y_2) = 0 \text{ if and only if } (x_1 = y_1 \vee g(x_1, x_2) > g(y_1, y_2)).$$

Finally, we define the formula $\varphi \in \Sigma^0_4$

$$\varphi \equiv \exists^{\mathsf{N}} x_1 \forall^{\mathsf{N}} y_1 \exists^{\mathsf{N}} x_2 \forall^{\mathsf{N}} y_2 (f(x_1, y_1, x_2, y_2) = 0)$$

which expresses that there exists $x_1$ (in fact 0) such that $g(y_1, \cdot) : z \mapsto g(y_1, z)$ is bounded for every $y_1 \neq x_1$. The shortest strategy for Eloise to win that game would be to give 0 for $x_1$, wait for an answer $m$ for $y_1$, and give $m + 1$ for $x_2$. But we can also imagine that Eloise might try 0 first, receive Abelard answer, and then change her mind, start from the beginning with 1, try several possibilities before going back to the winning position. If we observe the positions Eloise will reach for such a match, we remark it draws a tree (see Figure 2). We shall formalize this remark later, but we strongly advise the reader to keep this representation in mind all along the next section.

## 6. Implementing the game

### 6.1. *Substitutive Game:* $\mathbb{G}^1_\Phi$

Now that we have at our disposal a notion of game that seems to be suitable to capture computational content of classical theorems, we shall adapt it to play with realizers. Considering a formula $\Phi \equiv \exists^{\mathsf{N}} x_1 \forall^{\mathsf{N}} y_1 \ldots \exists^{\mathsf{N}} x_h \forall y_h (f(\vec{x}_h, \vec{y}_h) = 0)$ we will have to consider sub-formulæ of $\Phi$ to write down proofs about $\Phi$. Therefore, we give the following abbreviations that we will use a lot in the following:

$$\begin{aligned} E_i &\equiv \forall X_{i+1}(A_{i+1} \Rightarrow X_{i+1}) && (\forall i \in [\![0, h-1]\!]) \\ A_i &\equiv \forall^{\mathsf{N}} x_i (\forall^{\mathsf{N}} y_i E_i \Rightarrow X_i) && (\forall i \in [\![1, h]\!]) \\ E_h &\equiv \forall W (W(f(\vec{x}_h, \vec{y}_h)) \Rightarrow W(0)) \end{aligned}$$

One can easily check that $E_0 \equiv \Phi$ and that the other definitions correspond to the unfolding of the quantifiers.

In order to play using realizers, we will slightly change the setting of $\mathbb{G}^0_\Phi$, adding processes. One should notice that we only add more information, so that the game $\mathbb{G}^1_\Phi$ is somehow a 'decorated' version of $\mathbb{G}^0_\Phi$.

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

To describe the match, we use ∃-positions – which are just processes – and ∀-positions – which are 4-uples of the shape $(\vec{m}_i, \vec{n}_i, u, \pi) \in \mathbb{N}^{\leqslant h} \times \mathbb{N}^{\leqslant h} \times \Lambda_c \times \Pi$. If $i = h$, we say that the move is *final* or *complete*. In a given time $j$, the set of all ∀-positions reached before is called *the history* and is denoted as $H_j$. At each time $j$, the couple given by the current ∃-position $p_j$ and the history $H_j$ is called the $j$th state. The state evolves throughout the match according to the following rules:

1. Eloise proposes a term $t_0 \in \mathrm{PL}$ supposed to defend $\Phi$ and Abelard proposes a stack $u_0 \cdot \pi_0$ supposed to attack the formula $\Phi$. We say that at time 0, the process $p_0 := t_0 \star u_0 \cdot \pi_0$ is the current ∃-position and $H_0 := \{(\varnothing, \varnothing, u_0, \pi_0)\}$ is the current history. This step defines the initial state $\langle p_0, H_0 \rangle$.
2. Assume $\langle p_j, H_j \rangle$ is the $j^{\text{th}}$ state. Starting from $p_j$ Eloise evaluates $p_j$ in order to reach one of the following situations:
   - $p_j \succ u \star \pi$ for some (final) ∀-position $(\vec{m}_h, \vec{n}_h, u, \pi) \in H_j$. In this case, Eloise wins if $\mathcal{M} \models f(\vec{m}_h, \vec{n}_h) = 0$.
   - $p_j \succ u \star \overline{m} \cdot t \cdot \pi$ for some (not final) ∀-position $(\vec{m}_i, \vec{n}_i, u, \pi) \in H_j$, where $i < h$. If so, Eloise *can* decide to play by communicating her answer $(t, m)$ to Abelard and standing for his answer, and Abelard *must* answer a new integer $n$ together with a new stack $u' \cdot \pi'$. The ∃-position becomes $p_{j+1} := t \star \overline{n} \cdot u' \cdot \pi'$ and we add the ∀-position to the history: $H_{j+1} := H_j \cup \{(\vec{m}_i \cdot m, \vec{n}_i \cdot n, u', \pi')\}$. This step defines the next state $\langle p_{j+1}, H_{j+1} \rangle$.

   If none of the above moves is possible, then Abelard wins.

Intuitively, a state $\langle p, H \rangle$ is *winning* for Eloise if and only if she *can play* in such a way that Abelard *will lose anyway*, independently of the way he might play.

Start with a term $t$ is a 'good move' for Eloise if and only if, proposed as a defender of the formula, $t$ defines an initial winning state (for Eloise), independently from the initial stack proposed by Abelard. In this case, adopting the point of view of Eloise, we just say that $t$ is a *winning strategy* for the formula $\Phi$.

Since our characterization of realizers will be in terms of winning strategies, we might formalize this notion. We define inductively the set of *winning states* – which is a syntactic object – by means of a deductive system:

— if $\exists(\vec{m}_h, \vec{n}_h, u, \pi) \in H$ s.t. $p \succ u \star \pi$ and $\mathcal{M} \vDash f(\vec{m}_h, \vec{n}_h) = 0$ :

$$\frac{}{\langle p, H \rangle \in \mathbb{W}^1_\Phi}\ (\text{Win})\ ;$$

- for every $(\vec{m}_i, \vec{n}_i, u, \pi) \in H$, $m \in \mathbb{N}$ s.t. $p \succ u \star \overline{m} \cdot t \cdot \pi$ :

$$\frac{\langle t \star \overline{n} \cdot u' \cdot \pi', H \cup \{(\vec{m}_i \cdot m, \vec{n}_i \cdot n, u', \pi')\} \rangle \in \mathbb{W}^1_\Phi \quad \forall (n', u', \pi') \in \mathbb{N} \times \Lambda \times \Pi}{\langle p, H \rangle \in \mathbb{W}^1_\Phi}\ (\text{Play})\ .$$

A term $t$ is said to be a winning strategy for $\Phi$ if for any handle $(u, \pi) \in \Lambda \times \Pi$, we have $\langle t \star u \cdot \pi, \{(\varnothing, \varnothing, u, \pi)\} \rangle \in \mathbb{W}^1_\Phi$.

**Proposition 10 (Adequacy).** If $t$ is a winning strategy for $\mathbb{G}^1_\Phi$, then $t \Vdash \Phi$.

*Proof.* We will see a more general game in the following section for which we will prove the adequacy property (Proposition 15) and which admits any winning strategy of this game as a

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

winning strategy (Proposition 14), thus proving the adequacy in the current case. Furthermore, the proof we give for Proposition 15 is suitable for this game too. □

### 6.2. *Completeness of* $\mathbb{G}^1_\Phi$ *in presence of interaction constants*

In this section, we will show the completeness of $\mathbb{G}^1_\Phi$ by substitution over the thread of execution of a universal realizer of $\Phi$. As observed in section 5.4, the successive $\exists$-positions form a tree. We give thereafter a formal statement for this observation, which will allow us to prove the completeness of this game. We shall now give a formal definition of a tree.

**Definition 10.** A (finite) *tree* $\mathscr{T}$ is a (finite) subset[†] of $\mathbb{N}^{<\omega}$ such that if $\tau \cdot c \in \mathscr{T}$ and $c \in \mathbb{N}$, then $\tau \in \mathscr{T}$ and $\forall c' < c, \tau \cdot c' \in \mathscr{T}$, where the $\cdot$ operator denotes the concatenation. If $\tau = c_0 \cdots c_k$, we use the notation $\tau_{|i} = c_0 \cdots c_i$, and we note $\tau \sqsubset \sigma$ ( $\sigma$ *extends* $\tau$) when

$$\tau \sqsubset \sigma \;\equiv\; \sigma_{|k} = c_0 \cdots c_k = \tau.$$

We call *characteristic function* of a tree $\mathscr{T}$ any partial function $\varphi : \mathbb{N} \to \mathcal{P}(\mathbb{N}^{<\omega})$ such that:

1. $\forall n \in \mathrm{dom}(\varphi), \{\varphi(m) : m \leqslant n\}$ is a tree;
2. $\varphi(|\mathscr{T}|) = \mathscr{T}$.

**Lemma 3.** Assume the calculus of realizers is deterministic, and let $t_0$ be a universal realizer of $\Phi \in \Sigma^0_{2h}$. Consider $(n_j)_{j\in\mathbb{N}}$ an infinite sequence of integers, $(\kappa_j)_{j\in\mathbb{N}}$ an infinite sequence of (pairwise distinct) interaction constants that do not occur in $t_0$ and if $(\alpha_j)_{j\in\mathbb{N}}$ is an infinite sequence of substitutive and non-generative stack constants. Then, there exists two integers $f, s \in \mathbb{N}$, two finite sequences $t_0, \ldots, t_f \in \Lambda$ and $m_1, \ldots, m_f \in \mathbb{N}$ as well as a tree characteristic function $\varphi : [\![0, f]\!] \to \mathbb{N}^{<\omega}$ such that

$$\begin{array}{rcl} & t_0 \star \kappa_0 \cdot \alpha_0 \succ \kappa_0 \star \overline{m}_1 \cdot t_1 \cdot \alpha_0 & \\ \forall i \in [\![1, f-1]\!] \quad & t_i \star \overline{n_i} \cdot \kappa_i \cdot \alpha_i \succ \kappa_j \star \overline{m_{i+1}} \cdot t_{i+1} \cdot \alpha_j & \left(\begin{array}{l}\text{with } j \leqslant i \\ \varphi(j) \sqsubset \varphi(i+1)\end{array}\right) \\ & t_f \star \overline{n_f} \cdot \kappa_f \cdot \alpha_f \succ \kappa_s \star \alpha_s, & \end{array}$$

where $|\varphi(s)| = h$ and $\mathcal{M} \vDash f(\vec{m}_{\varphi(s)}, \vec{n}_{\varphi(s)}) = 0$.

**Example 3.** Before doing the proof, let us have a look at an example of such a thread scheme for a formula $\Phi \in \Sigma^0_4$ (as we considered in Example 2) and to the corresponding tree and characteristic function.

We observe that we could actually labelled any node of the tree using its order of apparition in the enumeration of $\mathscr{T}$ with $\varphi$.

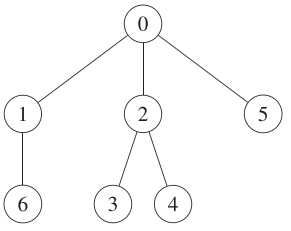
**Definition 11.** Given such a thread scheme and a path $\tau \in \mathscr{T}$, we define $m_\tau = m_{\varphi^{-1}(\tau)}$ (integer $m$ at the node $\tau$), $\vec{m}_\tau = (m_{\tau_{|1}}, m_{\tau_{|2}}, \ldots, m_\tau)$ (integers $m$ along the path) and the substitution along $\tau$ is

$$\sigma(\tau) = \{x_i := m_{\tau_{|i}}\}_{i=1}^{|\tau|}\{y_i := n_{\tau_{|i}}\}_{i=1}^{|\tau|}.$$

[†] Observe that $|\mathscr{T}|$ (the cardinality of $\mathscr{T}$) coincides with the usual definition of the size of $\mathscr{T}$.

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

$$
\begin{aligned}
t_0 \star \kappa_0 \cdot \alpha_0 &\succ \kappa_0 \star \overline{m}_1 \cdot t_1 \cdot \alpha_0\\
t_1 \star \overline{n_1} \cdot \kappa_1 \cdot \alpha_1 &\succ \kappa_0 \star \overline{m_2} \cdot t_2 \cdot \alpha_0\\
t_2 \star \overline{n_2} \cdot \kappa_2 \cdot \alpha_2 &\succ \kappa_2 \star \overline{m_3} \cdot t_3 \cdot \alpha_2\\
t_3 \star \overline{n_3} \cdot \kappa_3 \cdot \alpha_3 &\succ \kappa_2 \star \overline{m_4} \cdot t_4 \cdot \alpha_2\\
t_4 \star \overline{n_4} \cdot \kappa_4 \cdot \alpha_4 &\succ \kappa_0 \star \overline{m_5} \cdot t_5 \cdot \alpha_0\\
t_5 \star \overline{n_5} \cdot \kappa_5 \cdot \alpha_5 &\succ \kappa_1 \star \overline{m_6} \cdot t_6 \cdot \alpha_1\\
t_6 \star \overline{n_6} \cdot \kappa_6 \cdot \alpha_6 &\succ \kappa_4 \star \alpha_4
\end{aligned}
$$




$$
\begin{aligned}
\varphi &: 1 \mapsto 0\\
\varphi &: 2 \mapsto 1\\
\varphi &: 3 \mapsto 1 \cdot 0\\
\varphi &: 4 \mapsto 1 \cdot 1\\
\varphi &: 5 \mapsto 2\\
\varphi &: 6 \mapsto 0 \cdot 0
\end{aligned}
$$

Fig. 3. A thread scheme for $\Phi \in \Sigma_4^0$.

For instance, in Figure 3, for $\tau = 1 \cdot 1$ (wich corresponds to the choosen final position $\kappa_4 \star \alpha_4$), we have

$$\sigma(\tau) \equiv \{x_1 := m_2, x_2 := m_4, y_1 := n_2, y_2 := n_4\}.$$

*Proof of Lemma 3* We build a sequence $(Q_i)_{i\in\mathbb{N}}$ of sets of processes and a sequence of characteristic functions $(\varphi_i)_{i\in\mathbb{N}}$ for some trees $(\mathscr{T}_i)_{i\in\mathbb{N}}$, such that at each step $i \in \mathbb{N}$, $Q_i$ is either empty either of the form $\mathbf{th}(p)$ for some $p \in \Lambda \times \Pi$ :

- $i = 0$ : we set $Q_0 = \mathbf{th}(t_0 \star \kappa_0 \cdot \alpha_0)$ and $\varphi_0 : 0 \mapsto \varnothing$;
- $i \in \mathbb{N}$ : given $Q_i$ and $\varphi_i$, if there exist[†] $j \in \mathbb{N}, m_{i+1} \in \mathbb{N}$ and $t_{i+1} \in \Lambda$ such that $\kappa_j \star \overline{m_{i+1}} \cdot t_{i+1} \cdot \alpha_j \in Q_i$ we set

$$Q_{i+1} := \mathbf{th}(t_i \star \overline{n_{i+1}} \cdot \kappa_{i+1} \cdot \alpha_{i+1}) \qquad \varphi_{i+1} := \begin{cases} k \leqslant i & \mapsto \ \varphi_i(k) \\ i+1 & \mapsto \ \varphi_i(j) \cdot c \end{cases},$$

  where $c := \min\{n \in \mathbb{N} \mid \varphi_i(j) \cdot n \notin \mathscr{T}_i\}$. It is easy to check that if $\varphi_i$ is a characteristic function for $\mathscr{T}_i$, then so is $\varphi_{i+1}$ for $\mathscr{T}_i \cup \{\varphi_i(j) \cdot c\}$;

otherwise $Q_{i+1} := \varnothing$ and $\varphi_{i+1} := \varphi_i$. We define $Q_\infty := \bigcup_{i\in\mathbb{N}} Q_i$, $\perp\!\!\!\perp := Q_\infty^c$ and $\varphi := \lim_{i\in\omega} \varphi_i$. We prove by induction that for any $0 \leqslant i \leqslant h$, the following statement holds:

$$\exists j \in \mathbb{N}, |\varphi(j)| = i \text{ such that } \kappa_j \cdot \alpha_j \notin \|E_i[\sigma(\varphi(j))]\|. \tag{IH$_i$}$$

**IH$_0$:** From the definition of $\perp\!\!\!\perp$, we have $t_0 \star \kappa_0 \cdot \alpha_0 \notin \perp\!\!\!\perp$. Besides, we know that $t_0 \Vdash E_0$, so that $\kappa_0 \cdot \alpha_0 \notin \|E_0\|$.

**IH$_{i+1}$:** Assume we have IH$_i$, for $0 \leqslant i < h$, that is $\exists j_i \in \mathbb{N}, |\varphi(j_i)| = i$ such that

$$\kappa_{j_i} \cdot \alpha_{j_i} \notin \|E_i[\sigma(\varphi(j_i))]\|.$$

Recall that $E_i = \forall X_{i+1}(A_{i+1} \Rightarrow X_{i+1})$, hence $\kappa_{j_i} \nVdash A_{i+1}[X_{i+1} := \dot{\alpha}_\tau][\sigma(\varphi_i(j_i)]$. Therefore, there exists $m \in \mathbb{N}$ and $t \Vdash \forall^{\mathsf{N}} y_{i+1} E_{i+1}[\sigma(\varphi(j_i))]\{x_{i+1} := m\}$ such that $\kappa_{j_i} \star \overline{m} \cdot t \cdot \alpha_{j_i} \notin \perp\!\!\!\perp$. By definition of $\perp\!\!\!\perp$, it means that there is some $j \in \mathbb{N}$ such that this process belong to $Q_j$, so that by definition of $Q_{j+1}$ we have $t_{j+1} = t, m_{j+1} = m$, $\varphi(j+1)_{|i} = \varphi(j_i)$,

$$t_{j+1} \star \overline{n_{j+1}} \cdot \kappa_{j+1} \cdot \alpha_{j+1} \notin \perp\!\!\!\perp.$$

[†] Note that as the calculus is deterministic and the constants $\kappa_j$ inert, if such $j, m_{i+1}, t_{i+1}$ exist, they are unique.

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

Using the fact that $t_{j+1} \Vdash \forall^{\mathsf{N}} y_{i+1} E_{i+1}[\sigma(\varphi(j_i))]\{x_{i+1} := m\}$, we finally get that

$$\kappa_{i+1} \cdot \alpha_{i+1} \notin \|E_{i+1}[\sigma(\varphi(j+1))]\|$$

since $\sigma(\varphi(j+1)) = \sigma(\varphi(j_i))\{x_{i+1} := m_{j+1}; y_{i+1} := n_{j+1}\}$.

We obtain then for $\mathrm{IH}_h$ the following statement :

$$\exists s \in \mathbb{N}, |\varphi(s)| = h \text{ such that } \kappa_s \cdot \alpha_s \notin \|\forall W(f(\vec{m}_{\varphi(s)}, \vec{n}_{\varphi(s)})) \Rightarrow W(0)\|.$$

Applying the Lemma 3, we get that $\mathcal{M} \vDash f(\vec{m_\sigma}, \vec{n_\sigma}) = 0$ and $\kappa_s \star \alpha_s \notin \bot\!\!\!\bot$. Hence, there exists $f \in \mathbb{N}$ such that $\kappa_s \star \alpha_s \in Q_f$, thus

$$t_f \star \overline{n_f} \cdot \kappa_f \cdot \alpha_f \succ \kappa_s \star \alpha_s, \text{ with } \mathcal{M} \vDash f(\vec{m}_\sigma, \vec{n}_\sigma) = 0$$

that is the last line of the expected thread scheme.

Besides, by definition of $Q_f$ and $\varphi_f$, we clearly have that for any $i \in [\![0, f-1]\!]$, there exists $j \in \mathbb{N}$ such that $j \leqslant i$ and

$$t_i \star \overline{n_i} \cdot \kappa_i \cdot \alpha_i \succ \kappa_j \star \overline{m_{i+1}} \cdot t_{i+1} \cdot \alpha_j. \qquad \square$$

Note that, as the constants $\kappa_i$ and $\alpha_i$ are substitutive, the function $\varphi$ and the integers $f$ and $s$ only depend on the sequence $(n_i)_{i\in\mathbb{N}}$. In other words, the threads scheme is entirely defined by this sequence.

**Proposition 11 (Completeness of $\mathfrak{G}^1_\Phi$ in presence of interaction constants).** If the calculus of realizers is deterministic and contains infinitely many interaction constants as well as infinitely many substitutive and non-generative stack constants, then every universal realizer of an arithmetical formula $\Phi \in \Sigma^0_h$ is a winning strategy for the game $\mathfrak{G}^1_\Phi$.

*Proof.* Consider $\Phi \in \Sigma^0_h$ and a closed term $t_0 \Vdash \Phi$. Given any infinite sequence of (pairwise distinct) non-generative constants $(\kappa_i)_{i\in\mathbb{N}}$ that do not occur in $t_0$ and any sequence of stack constants $(\alpha_i)_{i\in\mathbb{N}}$, we have shown that for any sequence $(n_i)_{i\in\mathbb{N}}$ of integers, there exists two integers $f, s \in \mathbb{N}$, two finite sequences of integers $m_0, \ldots, m_f \in \mathbb{N}$ and closed terms $t_0, \ldots, t_f \in \Lambda$ and a finite tree $\mathscr{T}$ whose characteristic function $\varphi$ verifies $|\varphi(s)| = h$:

$$\begin{array}{rcll} & t_0 \star \kappa_0 \cdot \alpha_0 & \succ & \kappa_0 \star \overline{m}_1 \cdot t_1 \cdot \alpha_0 & \\ \forall i \in [\![1, f-1]\!] & t_i \star \overline{n_i} \cdot \kappa_i \cdot \alpha_i & \succ & \kappa_j \star \overline{m_{i+1}} \cdot t_{i+1} \cdot \alpha_j & (\text{with } j \leqslant i \text{ and } \varphi(j) \sqsubset \varphi(i+1)) \\ & t_f \star \overline{n_f} \cdot \kappa_f \cdot \alpha_f & \succ & \kappa_s \star \alpha_s & (\text{with } \mathcal{M} \vDash f(\vec{m}_{\varphi(s)}, \vec{n}_{\varphi(s)}) = 0). \end{array}$$

We assume $t_0$ is not a winning strategy, that is there exists a term $u_0$ and a stack $\pi_0$ such that

$$\langle t_0 \star u_0 \cdot \pi_0, \varnothing \rangle \notin \mathbb{W}^1_\Phi$$

and try to reach a contradiction.

We build by induction four infinite sequences $(n_i)_{i\in\mathbb{N}}, (u_i)_{i\in\mathbb{N}}, (\pi_i)_{i\in\mathbb{N}}, (H_i)_{i\in\mathbb{N}}$ such that for any index $i \in \mathbb{N}$, we have $H_i = \bigcup_{j\leqslant i}\{(\vec{m}_{\varphi_i(j)}, \vec{n}_{\varphi_i(j)}, u_j, \pi_j)\}$ and the following statement:

$$\langle t_i\{\kappa_j := u_j, \alpha_j := \pi_j\}_{j=0}^{i-1} \star \overline{n_i} \cdot u_i \cdot \pi_i, H_i \rangle \notin \mathbb{W}^1_\Phi, \qquad (\mathrm{IH}_i)$$

where $t_i$ is the term taken from the thread scheme we obtain for the sequence $(n_i)_{i\in\mathbb{N}}$.

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

- $\mathbf{IH}_1$ : by substitution over the first line of the scheme, we get

$$t_0 \star u_0 \cdot \pi_0 \succ u_0 \star \overline{m}_1 \cdot t_1\{\kappa_0 := u_0, \alpha_0 := \pi_0\} \cdot \pi_0.$$

  As $\langle t_0 \star u_0 \cdot \pi_0, \varnothing\rangle \notin \mathbb{W}^1_\Phi$, that implies by the second rule of induction that there exists $n_1, u_1, \pi_1$ such that

$$\langle t_1\{\kappa_0 := u_0, \alpha_0 := \pi_0\} \star \overline{n_1} \cdot u_1 \cdot \pi_1, (\varnothing, \varnothing, u_0, \pi_0)\rangle \notin \mathbb{W}^1_\Phi.$$

- $\mathbf{IH}_{i+1}$ : assume we have built $n_j, u_j, \pi_j, H_j$ for all $0 \leqslant j \leqslant i$, such that $\mathrm{IH}_j$ holds. Hence by hypothesis, we have

$$\langle t_i\{\kappa_j := u_j, \alpha_j := \pi_j\}_{j=0}^{i-1} \star \overline{n_i} \cdot u_i \cdot \pi_i, H_i\rangle \notin \mathbb{W}^1_\Phi.$$

  By substitution over the threads scheme, we get an index $j \leqslant i$ such that

$$t_i\{\kappa_j := u_j, \alpha_j := \pi_j\}_{j=0}^{i-1} \star \overline{n_1} \cdot u_i \cdot \pi_i \succ u_j \star \overline{m_{i+1}} \cdot t_{i+1}\{\kappa_j := u_j, \alpha_j := \pi_j\}_{j=0}^{i} \cdot \pi_j.$$

  Furthermore, we know from the hypothesis $\mathrm{IH}_i$ that there is a pair $(\vec{m}_{\varphi_i(j)}, \vec{n}_{\varphi_i(j)})$ such that $(\vec{m}_{\varphi_i(j)}, \vec{n}_{\varphi_i(j)}, u_j, \pi_j) \in H_i$. As the second rule of induction fails, it implies the existence of $n_j, u_j, \pi_j$ such that

$$\langle t_{i+1}\{\kappa_j := u_j, \alpha_j := \pi_j\}_{j=0}^{i} \star \overline{n_{i+1}} \cdot u_{i+1} \cdot \pi_{i+1}, H_{i+1}\rangle \notin \mathbb{W}^1_\Phi,$$

  where taking the very same definition of $\varphi_{i+1}$ we used in the proof of lemma 3, $H_{i+1} = H_i \cup \{((\vec{m}_{\varphi_{i+1}(i+1)}, \vec{n}_{\varphi_{i+1}(i+1)}), u_{i+1}, \pi_{i+1})\}$, so we prove $\mathrm{IH}_{i+1}$.

Now, if we consider the sequence $(n_i)_{i\in\mathbb{N}}$ we built, and define $\varphi = \lim_{i\in\mathbb{N}} \varphi_i$, it is clear that $\varphi$ is the very same function that we obtain by Lemma 3. Moreover, according to this Lemma we know there exists $f, s \in \mathbb{N}$ such that

$$t_f\{\kappa_j := u_j, \alpha_j := \pi_j\}_{j=0}^{f} \star \overline{n_f} \cdot u_f \cdot \pi_f \succ u_s \star \pi_s$$

with $\mathcal{M} \vDash f(\vec{m}_{\varphi(s)}, \vec{n}_{\varphi(s)}) = 0$. As $(\vec{m}_{\varphi(s)}, \vec{n}_{\varphi(s)}, u_s, \pi_s) \in H_f$, the first rule of $\mathfrak{G}^1_\Phi$ applies, and

$$\langle t_f\{\kappa_j := u_j, \alpha_j := \pi_j\}_{j=0}^{f-1} \star \overline{n_f} \cdot u_f \cdot \pi_f, H_f\rangle \in \mathbb{W}^1_\Phi$$

which is obviously a contradiction with $\mathrm{IH}_f$. □

### 6.3. *A wild realizer*

The previous section gives a specification of arithmetical formulæ in the particular case where the language of realizers is deterministic[†] and provides infinitely many interaction constants and infinitely many substitutive and non-generative stack constants. These assumptions are actually incompatible with the presence of instructions such as `eq` or `quote`, as stated by the Proposition 1, since this break the property of substitutivity. It would be pleasing to be able to extend such a characterization to a more general framework that would allow such instructions. Nevertheless,

[†] Actually, this assumption is not necessary, and has been made only for convenience in the proof of Lemma 3. In fact, we could adapt this proof to a non-deterministic case, by defining $Q_{i+1}$ as the union of the threads $\mathbf{th}(t_i \star \overline{n_{i+1}} \cdot \kappa_{i+1} \cdot \alpha_{i+1})$ for all $j \in \mathbb{N}, m_{i+1} \in \mathbb{N}$ and $t_{i+1} \in \Lambda$ such that $\kappa_j \star \overline{m_{i+1}} \cdot t_{i+1} \cdot \alpha_j \in Q_i$. But in this case, the characteristic function of the tree describing the thread scheme is more subtle to construct.

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

we know from (Guillermo and Miquel 2014) that it was not possible for the Law of Peirce, and it is not possible either in this case, for the very same reason: the instruction `eq` (that could be simulated with `quote`, see Section 2.3) allows to define some *wild* realizers for some formulæ, that is realizers of some $\Phi$ that are not winning strategies for the game $\mathbb{G}^1_\Phi$.

If we consider $f_\leqslant : \mathbb{N}^2 \to \mathbb{N}$ such that $\forall x, y \in \mathbb{N}, (f_\leqslant(x,y) = 0 \Leftrightarrow x \leqslant y)$, and the formula $\Phi_\leqslant \equiv \exists^{\mathsf{N}} x \forall^{\mathsf{N}} y (f_\leqslant(x,y) = 0)$, here is an example of such a wild realizer. We define the following terms:

$$\begin{aligned} T_2[y,m] &\equiv \texttt{quote}\ (\lambda nu.\texttt{eq_nat}\ n\ m\ (\texttt{eq}\ u\ (y\ y)\ \mathbf{I}\ u)\ u) \\ T_1[u,m] &\equiv \lambda y.u\ \overline{0}\ T_2[y,m] \\ T_0[u,m] &\equiv T_1[u,m]\ T_1[u,m] \\ t_\leqslant &\equiv \lambda u.\texttt{quote}\ (\lambda m.T_0[u,m]). \end{aligned}$$

From these definitions, we get for all $u \in \Lambda$ and $\pi \in \Pi$:

$$t_\leqslant \star u \cdot \pi \succ T_0[u, \overline{n_\pi}] \star \pi \succ u \star \overline{0} \cdot T_2[T_1[u, \overline{n_\pi}], \overline{n_\pi}] \cdot \pi$$

and moreover, for all $n \in \mathbb{N}$, $u' \in \Lambda$ and $\pi' \in \Pi$:

$$T_2[T_1[u, \overline{n_\pi}], \overline{n_\pi}] \star \overline{n} \cdot u' \cdot \pi' \succ \begin{cases} \mathbf{I} \star \pi' & \text{if } u' \equiv T_0[u, \overline{n_\pi}] \text{ and } \pi \equiv \pi' \\ u' \star \pi' & \text{otherwise} \end{cases}.$$

**Proposition 12.** $t_\leqslant \Vdash \exists^{\mathsf{N}} x \forall^{\mathsf{N}} y (f_\leqslant(x,y) = 0)$.

*Proof.* Let us consider a fixed pole $\bot\!\!\!\bot$ and a stack $u \cdot \pi \in \|\exists^{\mathsf{N}} x \forall^{\mathsf{N}} y (f_\leqslant(x,y) = 0)\|$, that is a falsity value $S$ such that $\pi \in \|\dot{S}\|$ and $u \in |\forall^{\mathsf{N}} x (\forall^{\mathsf{N}} y (f_\leqslant(x,y) = 0) \Rightarrow \dot{S})|$. We distinguish two cases:

- either $T_0[u, \overline{n_\pi}] \star \pi \in \bot\!\!\!\bot$. As we have $t_\leqslant \star u \cdot \pi \succ T_0[u, \overline{n_\pi}] \star \pi$, we get $t_\leqslant \star u \cdot \pi \in \bot\!\!\!\bot$ by anti-evaluation.
- either $T_0[u, \overline{n_\pi}] \star \pi \notin \bot\!\!\!\bot$. In this case, we have $t_\leqslant \star u \cdot \pi \succ u \star \overline{0} \cdot T_2[T_1[u, \overline{n_\pi}], \overline{n_\pi}] \cdot \pi$, hence it suffices to prove that $T_2[T_1[u, \overline{n_\pi}], \overline{n_\pi}] \Vdash \forall^{\mathsf{N}} y (f_\leqslant(0,y) = 0)$. Let us then consider $n \in \mathbb{N}$ and a stack $u' \cdot \pi' \in \|\forall W (W(f_\leqslant(0,n)) \Rightarrow W(0))\|$. First remark that $f_\leqslant(0,n) = 0$, hence by Corollary 3 $u' \star \pi' \in \bot\!\!\!\bot$, thus by assumption, we know that $(u', \pi') \notin (T_0[u, \overline{n_\pi}], \pi)$. Thus, we have $T_2[T_1[u, \overline{n_\pi}], \overline{n_\pi}] \star \overline{n} \cdot u' \cdot \pi' \succ u' \star \pi' \in \bot\!\!\!\bot$, which allows to conclude by anti-evaluation.

□

Notice that the subterm $\mathbf{I}$ that appears in the definition of the term $T_2$ never comes to active position in the proof of Proposition 12, so that we could actually have chosen any other closed $\lambda_c$-term instead. The point is that it can only occur if $(u', \pi') \equiv (T_0[u, \overline{n_\pi}], \pi)$, and when it is the case, we are no more interested in the end of the execution of the process $T_0[u, \overline{n_\pi}] \star \pi$, that is in a way allowed to do anything in the rest of its execution. Before giving a game-theoretic interpretation of this phenomena, we first check that $t_\leqslant$ is not a winning strategy for the game $\mathbb{G}^1_{\Phi_\leqslant}$.

**Proposition 13.** Let us assume that the relation of one step evaluation $\succ_1$ is only defined from the rules (Grab), (Push),(Save),(Restore),(Quote),(Eq). Then, the universal realizer $t_\leqslant$ of $\Phi_\leqslant$ is not a winning strategy for the game $\mathbb{G}^1_{\Phi_\leqslant}$.

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

*Proof.* The following is a valid match for $\mathbb{G}^1_{\Phi_\leqslant}$ that Eloise loses :

- Abelard starts with the initial handle $(\mathbf{I}, \alpha)$ for the empty position, where $\alpha$ is a stack constant.
- The only pair $(m, t)$ such that $t_\leqslant \star \mathbf{I} \cdot \alpha \succ \mathbf{I} \star \overline{m} \cdot t \cdot \alpha$ is $t_1 \equiv T_2[T_1[\mathbf{I}, \overline{n_\alpha}], \overline{n_\alpha}]$ and $m_1 = 0$. Thus, Eloise is forced to play that pair $(0, t_1)$.
- Abelard replies with $n_1 = 0$, $u_1 \equiv T_0[\mathbf{I}, \overline{n_\alpha}]$ and $\pi_1 \equiv \alpha$.
- Then Eloise loses, as the thread $\mathbf{th}(t_1 \star \overline{n_1} \cdot u_1 \cdot \pi_1)$ contains no process of the form $\mathbf{I} \star \overline{m} \cdot t \cdot \alpha$ (to continue to play) or of the form $u_1 \star \pi_1$ (to win the game). □

## 7. Non-substitutive case

### 7.1. $\mathbb{G}^2_\Phi$: *cumulative game*

Despite the wild realizer $t_\leqslant$ of the formula $\Phi_\leqslant$ is not a winning strategy for the corresponding game $\mathbb{G}^1_{\Phi_\leqslant}$, we can still think its computational behaviour in game-theoretic terms as follows. If we observe closely what happens in the match we described in the proof of the previous Proposition, if Abelard starts with $(u, \pi)$, to which Eloise answers $(0, T_2[T_1[u, \overline{n_\pi}], \overline{n_\pi}])$, Eloise then does somehow the distinction between two cases over the next Abelard answer $(n_1, u_1, \pi_1)$.

- if $(u_1, \pi_1) \not\equiv (T_0[u, \overline{n_\pi}], \pi)$, Eloise simply pursues the execution to reach $u_1 \star \pi_1$, which is a final winning position, as $0 \leqslant n_1$.
- if $(u_1, \pi_1) \equiv (T_0[u, \overline{n_\pi}], \pi)$, as no interesting move can be obtained from the current position, Eloise *backtracks* to the former ∃-position $t_\leqslant \star u \cdot \pi$, and now wins since

$$t_\leqslant \star u \cdot \pi \succ T_0[u, \overline{n_\pi}] \star \pi \equiv u_1 \star \pi_1.$$

That is to say that the term $t_\leqslant$ can still be seen as a winning strategy if we give the right to Eloise to compute its move from any former ∃-position. This gives us a new game $\mathbb{G}^2_\Phi$, in which Eloise keeps track of all the previous ∃-positions encountered during the game.

We thus define a $\mathbb{G}^2_\Phi$-state as a pair $\langle P, H\rangle$, where $P$ is now a finite set of processes (intuitively, all ∃-positions, including the current one), and $H$ is exactly as in $\mathbb{G}^1_\Phi$. The set $\mathbb{W}^2_\Phi$ of winning positions is inductively defined as follows:

- if there is $p \in P$ and $(\vec{m}_h, \vec{n}_h, u, \pi) \in H$ such that $p \succ u \star \pi$ and $\mathcal{M} \vDash f(\vec{m}_h, \vec{n}_h) = 0$

$$\frac{}{\langle P, H\rangle \in \mathbb{W}^2_\Phi}\ (\text{Win}).$$

- if there is $p \in P$, $i < h$, $(\vec{m}_i, \vec{n}_i, u, \pi) \in H$ and $m' \in \mathbb{N}$ such that $p \succ u \star \overline{m'} \cdot t \cdot \pi$:

$$\frac{\langle P \cup \{t \star \overline{n'} \cdot u' \cdot \pi'\}, H \cup \{(\vec{m}_i \cdot m', \vec{n}_i \cdot n', u', \pi')\}\rangle \in \mathbb{W}^2_\Phi \quad \forall (n', u', \pi') \in \mathbb{N} \times \Lambda \times \Pi}{\langle P, H\rangle \in \mathbb{W}^2_\Phi}\ (\text{Play}).$$

A term $t$ is say to be a *winning strategy* for $\mathbb{G}^2_\Phi$ if for any handle $(u, \pi) \in \Lambda \times \Pi$, we have $\langle \{t \star u \cdot \pi\}, \{(\varnothing, u, \pi)\}\rangle \in \mathbb{W}^2_\Phi$.

### 7.2. *Adequacy*

**Proposition 14.** A winning strategy for $\mathbb{G}^1_\Phi$ is also a winning strategy for $\mathbb{G}^2_\Phi$.

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

*Proof.* It suffices to prove that for any $\mathbb{G}^1_\Phi$ state $\langle p, H\rangle$, if we have $\langle p, H\rangle \in \mathbb{W}^1_\Phi$, then $\langle\{p\}, H\rangle \in \mathbb{W}^2_\Phi$. We do it by induction on the derivation of $\langle p, H\rangle \in \mathbb{W}^1_\Phi$, observing for the second rules of $\mathbb{G}^2_\Phi$ that if $\langle P, H\rangle \in \mathbb{W}^2_\Phi$ and $P \subset P'$, then $\langle P', H\rangle \in \mathbb{W}^2_\Phi$ (which is also proved by induction). □

**Proposition 15 (Adequacy).** If $t$ is a winning strategy for $\mathbb{G}^2_\Phi$, then $t \Vdash \Phi$.

*Proof.* To make the proof easier, we will use the formulæ $A$ and $E$ that we previously defined in Section 6.1.

Let $\bot\!\!\!\bot$ be a fixed pole, $S_1$ be a falsity value, $u_0 \Vdash \forall^{\mathsf{N}} x_1(E_1 \Rightarrow \dot{S}_1)) \Rightarrow \dot{S}_1$ and $\pi_0 \in S_1$, and let us show that $t \star u_0 \cdot \pi_0 \in \bot\!\!\!\bot$. For that, we more generally prove the following statement:

**Fact 1.** If $\langle P, H\rangle \in \mathbb{W}^2_\Phi$ and $\forall(\vec{m}_i, \vec{n}_i, u_i, \pi_i) \in H, u_i \cdot \pi_i \in \|E_i\{x_j := m_j, y_j := n_j\}^i_{j=1}\|$ then $P \cap \bot\!\!\!\bot \neq \varnothing$.

In particular, we have $\langle\{t \star u_0 \cdot \pi_0\}, \{(\varnothing, \varnothing, u_0, \pi_0)\}\rangle \in \mathbb{W}^2_\Phi$, $u_0 \cdot \pi_0 \in \|E_0\|$, hence we can deduce that $t \star u_0 \cdot \pi_0 \in \bot\!\!\!\bot$. □

*Proof of Fact 1.* We proceed by induction on the derivation of $\langle P, H\rangle \in \mathbb{W}^2_\Phi$, distinguishing two possible cases:

1. $\langle P, H\rangle \in \mathbb{W}^2_\Phi$ because of the first induction rule: there exists $(\vec{m}_h, \vec{n}_h, u, \pi) \in H$ and $p \in P$ such that $p \succ u \star \pi$ and $\mathcal{M} \vDash f(\vec{m}_h, \vec{n}_h) = 0$. If we assume that $u \cdot \pi \in \|E_h\| = \|\forall W(W(f(\vec{m}_h, \vec{n}_h)) \Rightarrow W(0))\|$, as $\mathcal{M} \vDash f(\vec{m}_h, \vec{n}_h) = 0$, we get that $u \star \pi \in \bot\!\!\!\bot$ (Corollary 3) and by anti-reduction, $p \in \bot\!\!\!\bot$.
2. $\langle P, H\rangle \in \mathbb{W}^2_\Phi$ because of the second induction rule : there is some $p_i \in P$, $(\vec{m}_i, \vec{n}_i, u_i, \pi_i) \in H$ and $m \in \mathbb{N}$ such that $p_i \succ u_i \star \overline{m} \cdot \xi \cdot \pi_i$, and for any $(n, u, \pi)$, $\langle P \cup \{\xi \star \overline{n} \cdot u \cdot \pi\}, H \cup \{(\vec{m}_h, \vec{n}_h, u, \pi)\}\rangle \in \mathbb{W}^2_\Phi$. We prove that we cannot have $P \cap \bot\!\!\!\bot = \varnothing$. Indeed, assuming it is the case, we can show that $u_i \star \overline{m} \cdot \xi \cdot \pi_i \in \bot\!\!\!\bot$. Besides, we know by hypothesis that

$$u_i \cdot \pi_i \in \|\forall X_{i+1}(\forall^{\mathsf{N}} x_{i+1}(\forall^{\mathsf{N}} y_{i+1} E_{i+1}\{x_j := m_j, y_j := n_j\}^i_{j=1} \Rightarrow X_{i+1}) \Rightarrow X_{i+1})\|$$

so that it is sufficient to prove that $\xi \Vdash \forall^{\mathsf{N}} y_{i+1} E_{i+1}\{x_j := m_j, y_j := n_j\}^i_{j=1}\{x_{i+1} := m\}$ to conclude. So pick $n \in \mathbb{N}$, $u \cdot \pi \in \|E_{i+1}\{x_j := m_j, y_j := n_j\}^i_{j=1}\{x_{i+1} := m\}\{y_{i+1} := n\}\|$, and let us prove that $\xi \star \overline{n} \cdot u \cdot \pi \in \bot\!\!\!\bot$. We have by hypothesis that

$$\langle P \cup \{\xi \star \overline{n} \cdot u \cdot \pi\}, H \cup \{(\vec{m}_i \cdot m, \vec{n}_i \cdot n, u, \pi)\}\rangle \in \mathbb{W}^2_\Phi$$

from which we deduce by induction (the premises are verified) that

$$(P \cup \{\xi \star \overline{n} \cdot u \cdot \pi\}) \cap \bot\!\!\!\bot \neq \varnothing.$$

As $P \cap \bot\!\!\!\bot = \varnothing$, we get that $\xi \star \overline{n} \cdot u \cdot \pi \in \bot\!\!\!\bot$, which conclude this case.

□

### 7.3. *Completeness of* $\mathbb{G}^2_\Phi$

**Proposition 16 (Completeness of $\mathbb{G}^2_\Phi$).** If $t \Vdash \Phi$ then $t$ is a winning strategy.

*Proof.* Let us reason by contradiction by assuming that there exists a handle $(u_0, \pi_0) \in \Lambda \times \Pi$ such that $\langle t \star u_0 \cdot \pi_0, \{(\varnothing, \varnothing, u_0, \pi_0)\}\rangle \notin \mathbb{W}^2_\Phi$. We will construct an increasing sequence

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

$(\langle P_j, H_j\rangle)_{j\in\mathbb{N}}$ such that for any $j \in \mathbb{N}$, $\langle P_j, H_j\rangle \notin \mathbb{W}^2_\Phi$. For that, let us pick a fixed enumeration $\phi : \mathbb{N} \to \mathbb{N} \times \Lambda$ such that every pair $(m, t)$ appears infinitely many times in the range of $\phi$. The sequence $(\langle P_j, H_j\rangle)$ is then defined as follows:

- We set $P_0 = \{t \star u_0 \cdot \pi_0\}$ and $H_0 = \{(\varnothing, \varnothing, u_0, \pi_0)\}$.
- Assume we have built a state $\langle P_j, H_j\rangle \notin \mathbb{W}^2_\Phi$. Writing $(m, t) = \phi(j)$, we distinguish the two following cases:

  1. Either there exists $p \in P_j$ and $((\vec{m}_i, \vec{n}_i, u, \pi) \in H_j)$ such that $p \succ u \star \overline{m} \cdot t \cdot \pi$. From the second rule of induction, we get the existence of $n \in \mathbb{N}$, $u' \in \Lambda$, $\pi' \in \Pi$ such that $\langle P \cup \{t \star \overline{n} \cdot u' \cdot \pi'\}, H \cup \{(\vec{m}_i \cdot m, \vec{n}_i \cdot n, u', \pi')\}\rangle \notin \mathbb{W}^2_\Phi$. We pick such a tuple $(n, u', \pi')$ and define $P_{j+1} = P_j \cup \{t \star \overline{n} \cdot u' \cdot \pi'\}$ and $H_{j+1} = H_j \cup \{(\vec{m}_i \cdot m, \vec{n}_i \cdot n, u', \pi')\}$.

  2. Either there is no such process, and we set $P_{j+1} = P_j$ and $H_{j+1} = H_j$.

  In both cases, we have construct $P_{j+1}$ and $H_{j+1}$ such that $P_j \subset P_{j+1}$, $H_j \subset H_{j+1}$ and $\langle P_{j+1}, H_{j+1}\rangle \notin \mathbb{W}^2_\Phi$. We set $P_\infty = \bigcup_{j\in\mathbb{N}} P_j$, $Q = \bigcup_{p\in P_\infty} \mathbf{th}(p)$ and $\bot\!\!\bot = Q^c$.

By construction, we have $t \star u_0 \cdot \pi_0 \notin \bot\!\!\bot$, and as $t \Vdash \forall X(\forall^{\mathsf{N}} x_1(\forall^{\mathsf{N}} y_1 E_1 \Rightarrow X) \Rightarrow X)$, we get $u_0 \not\Vdash \forall^{\mathsf{N}} x_1(\forall^{\mathsf{N}} y_1 E_1 \Rightarrow \{\pi_0\})$. Thus, there exists $m_1 \in \mathbb{N}$ and $\xi_1 \Vdash \forall^{\mathsf{N}} y_1 E_1\{x_1 := m_1\}$ such that $u_0 \star \overline{m_1} \cdot \xi_1 \cdot \pi_0 \notin \bot\!\!\bot$, that is exists an index $j \in \mathbb{N}$ and a process $p \in P_j$ such that $p \succ u_0 \star \overline{m_1} \cdot \xi_1 \cdot \pi_0$. Let $k \geqslant j$ be such that $\phi(k) = (m_1, \xi_1)$, then by construction there is some $\overline{n_1}, u_1, \pi_1$ such that $P_{k+1} = P_k \cup \{\xi_1 \star \overline{n_1} \cdot u_1 \cdot \pi_1\}$ and $H_{k+1} = H_k \cup \{((m_1, n_1, u_1, \pi_1)\}$.

As $\xi_1 \Vdash \forall^{\mathsf{N}} y_1 E_1\{x_1 := m_1\} \equiv \forall^{\mathsf{N}} y_1 \forall X((\forall^{\mathsf{N}} x_2 \forall^{\mathsf{N}} y_2 E_2\{x_1 := m_1\} \Rightarrow X) \Rightarrow X)$ and $\xi_1 \star \overline{n_1} \cdot u_1 \cdot \pi_1 \notin \bot\!\!\bot$, we deduce than $u_1 \not\Vdash \forall^{\mathsf{N}} x_2 \forall^{\mathsf{N}} y_2 E_2\{x_1 := m_1, y_1 := n_1\} \Rightarrow \{\pi_1\})$.

Iterating this very same reasoning, we obtain that for every $i \in [\![1, h]\!]$, there exists an index $k_i \in \mathbb{N}$ and a closed term $\xi_i \in \Lambda$, such that $H_{k_i}$ contains a tuple $(\vec{m}_i, \vec{n}_i, u_i, \pi_i)$, with $\xi_i \star \overline{n_i} \cdot u_i \cdot \pi_i \notin \bot\!\!\bot$ and $\xi_i \Vdash \forall^{\mathsf{N}} y_i E_i\{x_j := m_j\}_{j=1}^{i}\{y_j := n_j\}_{j=1}^{i-1}$.

For $i = h$, we get then an index $k_h \in \mathbb{N}$ and a closed term $\xi_h$, such that $H_{k_h}$ contains a tuple $(\vec{m}_h, \vec{n}_h, u_h, \pi_h)$, with $\xi_h \star \overline{n_h} \cdot u_h \cdot \pi_h \notin \bot\!\!\bot$ and $\xi_h \Vdash \forall^{\mathsf{N}} y_h \forall W(W(f(\vec{m}_h, \vec{n}_{h-1} \cdot y_h)) \Rightarrow W(0))$.

If we consider the following predicate

$$\Delta : \begin{cases} \mathbb{N} & \to \mathfrak{P}(\Pi) \\ 0 & \mapsto \{\pi_h\} \\ n \geqslant 1 & \mapsto \varnothing \end{cases},$$

we get in particular that $\xi_h \Vdash \{n_h\} \Rightarrow \Delta(f(\vec{m}_h, \vec{n}_h)) \Rightarrow \Delta(0)$, from which we deduce that $u_h \cdot \pi_h \notin \|\Delta(f(\vec{m}_h, \vec{n}_h)) \Rightarrow \Delta(0)\|$. Obviously $\pi_h \in \|\Delta(0)\|$, so that necessarily we have $u_h \not\Vdash \Delta(f(\vec{m}_h, \vec{n}_h))$. Hence, there exists $\pi \in \|\Delta(f(\vec{m}_h, \vec{n}_h))\|$, which implies that $\pi = \pi_h$ and $\mathcal{M} \vDash f(\vec{m}_h, \vec{n}_h) = 0$, such that $u_h \star \pi_h \notin \bot\!\!\bot$, that is to say there is some $j \in \mathbb{N}$ and $p \in P_j$ such that $p \succ u_h \star \pi_h$. Taking $l = \max(j, k_h)$, this contradicts the fact that $(P_l, H_l) \notin \mathbb{W}^2_\Phi$ because of the first rule of induction. □

**Theorem 2.** If $\Phi$ is an arithmetical formula, there exists $t \Vdash \Phi$ if and only if $t$ implements a winning strategy for $\mathbb{G}^2_\Phi$.

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

## 8. A barrier for realizability models

### 8.1. *A universal realizer for every formulæ*

We show here that if an arithmetic formula $\Phi \equiv \exists^{\mathsf{N}} x_1 \dots \forall^{\mathsf{N}} y_h f(\vec{m}_h, \vec{n}_h) = 0$ is true in the ground model, as soon as we dispose of a term computing $f$, we can implement a winning strategy, hence a universal realizer. The idea of the strategy for Eloise is to enumerate 'smartly' $\mathbb{N}^h$, in the following sense: when playing a tuple $\vec{m}_h$, we first look as deep as possible in the tree of formers positions for the tuple $\vec{m}_i$, and then go with corresponding Abelard answer. In doing so, we ensure that any tuple $\vec{m}_i$ will always be played with the same answers $\vec{n}_i$. Then, it is clear that is $\mathcal{M} \vDash \Phi$, we will reach sooner or later a winning position.

To implement such a strategy, we consider a term computing $f$ on a given position:

$$\Theta_f \star \langle \overline{m} \rangle_h \cdot t_1 \cdot t_2 \cdot \pi \succ \begin{cases} t_1 \star \pi & \text{if } \mathcal{M} \vDash f(\vec{m}, \vec{n}) = 0 \\ t_2 \star \pi & \text{if } \mathcal{M} \vDash f(\vec{m}, \vec{n}) \neq 0 \end{cases},$$

where $\langle \overline{m} \rangle_i$ is a $\lambda_c$-implementation[†] for the tuple $\vec{m}_i$, and that we also have a term `next` acting as a successor for $\mathbb{N}^h$.

$$\texttt{next} \star \langle \overline{m} \rangle_h^i \cdot t \cdot \pi \succ t \star \langle \overline{m} \rangle_h^{i+1} \cdot \pi,$$

where $\vec{m}_h^0 = (0, \dots, 0)$ and the sequence $(\vec{m}_h^i)_{i \in \mathbb{N}}$ is an enumeration of $\mathbb{N}^h$. We also define the relation $\vec{m}_h^i \leqslant_h \langle m \rangle_h^j \equiv i \leqslant j$, which is total on $\mathbb{N}^h$. Furthermore, we assume that we dispose of a $\lambda_c$-implementation of histories as lists of tuples, and for a given history $H$, we will denote by $\hat{H}$ its implementation[‡].

**Definition 12.** We say that a history $H$ is *functional* if for any $\vec{m}_i$, there exists at most one tuple $(\vec{n}_i, u, \pi)$ such that $(\vec{m}_i, \vec{n}_i, u, \mathbf{k}_\pi) \in H$.

Then, we build[§] several $\lambda_c$-terms according to their reductions rules. These terms will all take as parameter a $\lambda_c$-history $\hat{H}$. For $1 \leqslant i < h$, we define a term $T_i$ who is intended to gets Abelard's $i$th answer $(n_i, u_i, \pi_i)$, save it in $\hat{H}$ and plays the next integer with $T_{i+1}$:

$$T_i[\vec{m}_h, \vec{n}_{i-1}, \hat{H}] \star \overline{n}_i \cdot u_i \cdot \pi_1 \succ u_i \star \overline{m}_{i+1} \cdot T_{i+1}[\vec{m}_h, \vec{n}_i, \hat{H}^{(i)}] \cdot \pi_i \,,$$

where $\hat{H}^i \equiv [\vec{m}_1, \vec{n}_i, u_i, \mathbf{k}_{\pi_i}] \cdot \hat{H}$. The term $T_h$ gets Abelard's answer as $T_i$ does, but then computes $f$ to know if it has reached a winning position or should either initiate the next step of enumeration:

$$T_h[\vec{m}_h, \vec{n}_{h-1}, \hat{H}] \star \overline{n}_h \cdot u_h \cdot \pi_h \succ \Theta_f \star \langle m \rangle_h \cdot \langle n \rangle_h \cdot u_h \cdot N[\vec{m}_h, \hat{H}^{(h)}] \cdot \pi_h$$

with $\hat{H}^h \equiv [\vec{m}_h, \vec{n}_h, u_h, \mathbf{k}_{\pi_h}] \cdot \hat{H}$. Then, $N$ computes the next tuple in the enumeration and $L$ looks in the tree for the maximum former partial position similar to an initial segment of this tuple:

$$\begin{aligned} N[\langle m \rangle_h, \hat{H}] \star \pi &\succ \texttt{next} \star \langle m \rangle_h \cdot (\lambda m'_1 \cdots m'_h . L[\langle m' \rangle_h, \hat{H}]) \cdot \pi \\ L[\langle m \rangle_h, \hat{H}] \star \pi &\succ u_i \star \overline{m}_{i+1} \cdot T_{i+1}[\langle m \rangle_h, \langle n \rangle_i, \hat{H}] \cdot \pi_i \end{aligned}$$

[†] We could chose for instance to use a list representation for tuples, in which case $\langle \overline{m} \rangle_i \equiv [\overline{m}_1, \dots, \overline{m}_i]$, but here the data-type would not be relevant, we only pay attention to some 'big' steps of reduction independently of technical representation of data

[‡] $H \cup \{(\vec{m}_i, \vec{n}_i, u, \pi)\}$ will so correspond to $[\langle \overline{m} \rangle_i, \langle \overline{n} \rangle_i), u, k_\pi] \cdot \hat{H}$ .

[§] We let the reader check the existence of such terms, which is a straightforward $\lambda_c$-calculus exercise.

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

with $(\langle m\rangle_i, \langle n\rangle_i, u_i, \mathbf{k}_{\pi_i}) \in \hat{H}$ and $\forall j > i, \forall \vec{n}_j \in \mathbb{N}^j, \forall u \in \Lambda, \forall \pi \in \Pi(\langle m\rangle_j, \langle n\rangle_j, u, \mathbf{k}_\pi) \notin \hat{H}$. Finally, we consider $t_\Phi$ that would be the winning strategy, such that

$$t_\Phi \star u_0 \cdot \pi_0 \succ u_0 \star \overline{0} \cdot T_1[\langle 0\rangle_h, \langle\cdot\rangle, \hat{H}_0] \cdot \pi_0$$

with $H_0 \equiv (\cdot, \cdot, u_0, \mathbf{k}_{\pi_0})$.

**Proposition 17.** If $\mathcal{M} \vDash \Phi$, then $t_\Phi$ is a winning strategy for $\mathfrak{G}^1_\Phi$.

The proof does neither present any conceptual difficulty nor any interest in itself, but still remains quite technical. The idea is to propagate the contradiction along the enumeration of $\mathbb{N}^h$ in order to contradict $\mathcal{M} \vDash \Phi$ at the limit. To do so, we define the proposition $\mathbf{P}(i, \vec{m}_h, H)$ as the following statement :
$\mathbf{P}(i, \vec{m}_h, H)$ :'there exists $\vec{n}_i \in \mathbb{N}^i$, $u_i \in \Lambda$, $\pi_i \in \Pi$ such that

- $\{(\vec{m}_i, \vec{n}_i, u_i, \pi_i)\} \cup H$ is functional;
- $\langle T_i[\langle m\rangle_h, \langle n\rangle_{i-1}, \hat{H}] \star \overline{n}_i \cdot u_i \cdot \pi_i, \{(\vec{m}_i, \vec{n}_i, u_i, \pi_i)\} \cup H\rangle \notin \mathbb{W}^1_\Phi$'

and prove two technical lemmas.

**Lemma 4.** For any $i \in [\![1, h]\!]$, $\vec{m}_h \in \mathbb{N}^h$ and any history $H$, $\mathbf{P}(i, \vec{m}_h, H)$ implies there exists an history $H'$ such that $H \subset H'$ and $\mathbf{P}(h, \vec{m}_h, H')$

*Proof.* It suffices to see that because of the reduction rule defining $T_i$, if $\mathbf{P}(i, \vec{m}_h, H)$ holds then the second rule of $\mathfrak{G}^1_\Phi$ has to fail, hence there exists $n_{i+1}, u_{i+1} \in \Lambda$, $\pi_{i+1} \in \Pi$ such that

$$\langle T_{i+1}[\langle m\rangle_h, \langle n\rangle_i, \hat{H}^i] \star \overline{n}_{i+1} \cdot u_{i+1} \cdot \pi_{i+1}, \{(\vec{m}_{i+1}, \vec{n}_{i+1}, u_{i+1}, \pi_{i+1})\} \cup H^i\rangle \notin \mathbb{W}^1_\Phi,$$

where $H^i \equiv \{\vec{m}_1, \vec{n}_i, u_i, \mathbf{k}_{\pi_i}\} \cup H$, which is still a functional environment. Therefore, we have $\mathbf{P}(i, \vec{m}_h, H) \Rightarrow \mathbf{P}(i+1, \vec{m}_h, H^i)$, and $\mathbf{P}(i, \vec{m}_h, H) \Rightarrow \mathbf{P}(h, \vec{m}_h, H'))$ follows by easy decreasing induction on $i \in [\![1, h]\!]$. □

**Lemma 5.** For any history $H$, $\mathbf{P}(h, \vec{m}^j_h, H)$ implies that

1. there exists $\vec{n}_h \in \mathbb{N}^h$ such that$\mathcal{M} \nvDash f(\vec{m}^j_h, \vec{n}_h) = 0$.
2. there exists a history $H'$ such that $H \subset H'$ and $\mathbf{P}(h, \vec{m}^{j+1}_h, H')$.

*Proof.* Given a history $H$, if $\mathbf{P}(h, \vec{m}^j_h, H)$ holds, then it means that the first rule of induction of $\mathfrak{G}^1$ fails, hence necessarily $\mathcal{M} \nvDash f(\vec{m}^j_h, \vec{n}_h) = 0$ and by definition of $T_h$, using the notations $H^h = \{(\vec{m}^j_h, \vec{n}_h, u_h, \pi_h)\} \cup H$ and $\langle m'\rangle_h = \langle m\rangle^{j+1}_h$, we get that

$$T_h[\langle m\rangle^j_h, \langle n\rangle_{h-1}, \hat{H}] \star \overline{n}_h \cdot u_h \cdot \pi_h \succ u_i \star \overline{m}'_{i+1} \cdot T_{i+1}[\langle m'\rangle_h, \langle n'\rangle_i, \hat{H^h}] \cdot \pi_i$$

with $(\langle m'\rangle_i, \langle n'\rangle_i, u_i, \mathbf{k}_{\pi_i}) \in \hat{H}$ and $\forall j > i, \forall (\vec{n}_j, u, \pi) \in \mathbb{N}^j \times \Lambda \times \Pi(\langle m'\rangle_j, \langle n\rangle_j, u, \mathbf{k}_\pi) \notin \hat{H}$. Note that this condition ensures the functionality of $H^h \cup \{(\vec{m}^{j+1}_h, \vec{n'}_i, u_i, \pi_i)\}$. From $\mathbf{P}(h, \vec{m}^j_h, H)$ once more, we get that the second rule of induction of $\mathfrak{G}^1$ fails too, and so that $\mathbf{P}(i+1, \vec{m}^{j+1}_h, H^h)$. Hence by Lemma 4, we get the existence of $H'$ such that $H \subset H^h \subset H'$ and $\mathbf{P}(h, \vec{m}^{j+1}_h, H')$ holds. □

*Proof of Proposition 17.* By contraposition. We show that if $t_\Phi$ is not a winning strategy, then there exists a growing sequence of history $(H_j)_{j\in\mathbb{N}}$ such that for all $j \in \mathbb{N}$, $\mathbf{P}(h, \vec{m}^j_h, H_j)$ holds.

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

Indeed, assume $t_\Phi$ is not a winning strategy, that is to say there is $u_0 \in \Lambda, \pi_0 \in \Pi$ such that $\langle t_\Phi \star u_0 \cdot \pi_0, \varnothing\rangle \notin \mathbb{W}^1_\Phi$ Then, because of the reduction rule of $t_\Phi$, it means that the second rule of $\mathbb{G}^1_\Phi$ fails, thus there exists $(n_1, u_1, \pi_1 \in \mathbb{N} \times \Lambda \times \Pi$, such that

$$\langle T_1[\langle 0\rangle_h, \langle\cdot\rangle, \hat{H}] \star \overline{n}_1 \cdot u_1 \cdot \pi_1, \{(\vec{m}_1, \vec{n}_1, u_1, \pi_1)\} \cup H\rangle \notin \mathbb{W}^1_\Phi$$

with $H \equiv (\cdot, \cdot, u_0, \pi_0)$, that is $\mathbf{P}(1, \vec{m}^0_h, H)$. Then, by Lemma 4 we get that there exists $H_0$ such that $\mathbf{P}(h, \vec{m}^0_h, H_0)$ holds, and the claim follows by easy induction. Then, we set $\mathcal{H} = \bigcup_{j\in\mathbb{N}} H_j$, that is functional (because each $H_j$ is, and $H_j \subset H_{j+1}$).

Applying the first clause of Lemma 5, we get that for all $j \in \mathbb{N}$, there exists $\vec{n}^j_h$ such that $(\vec{m}^j_h, \vec{n}^j_h, u, \pi) \in \mathcal{H}$ for some $u \in \Lambda$ and $\pi \in \Pi$ and $\mathcal{M} \not\models f(\vec{m}^j_h, \langle n\rangle^j_h) = 0$.

Furthermore, as $\mathcal{H}$ is functional, it easily implies that

$$\forall m_1 \exists n_1 \ldots \forall m_h \exists n_h (\mathcal{M} \not\models f(\vec{m}_h, \vec{n}_h) = 0)$$

and thus we finally get $\mathcal{M} \not\models \Phi$. □

Combining the results we obtained at this point, we get the following theorem:

**Theorem 3.** If $\Phi$ is an arithmetical formula, then $\mathcal{M} \vDash \Phi$ if and only if there exists $t \Vdash \Phi$.

*Proof.* The first direction is a consequence of Propositions 17 and 15, the reverse directly comes from Proposition 4. □

### 8.2. *Leibniz equality vs. primitive non-equality*

Here, we have chosen to consider formulæ based on equalities, and we should wonder what happens if we use instead formulæ based on disequalities:

$$\exists x_1 \forall y_1 \ldots \exists x_n \forall y_n f(\vec{x_n}, \vec{y}_n) \neq 0\,.$$

We know that both definitions are equivalent from a model-theoretic point of view. Indeed, if we define the following function $h$:

$$h = \begin{cases} x \mapsto 1 & \text{if } x = 0 \\ x \mapsto 0 & \text{otherwise} \end{cases},$$

then for all $\vec{x} \in \mathbb{N}^n$, $\mathcal{M} \vDash f(\vec{x}) = 0$ if and only if $\mathcal{M} \vDash (h \circ f)(\vec{x}) \neq 0$. In other words, formulæ based on a non-equality have the same expressiveness, and we also might have chosen it as definition for the arithmetical formulæ (see Definition 9).

In classical realizability the disequality can be a given a simple semantic:

$$\|e_1 \neq e_2\| = \begin{cases} \|\top\| & \text{if } \mathcal{M} \vDash e_1 \neq e_2 \\ \|\bot\| & \text{otherwise} \end{cases}$$

which is equivalent to the negation of equality. Indeed, one can easily check that we have $\lambda xt.(t)x \Vdash e_1 \neq e_2 \Rightarrow \neg(e_1 = e_2)$ and $\lambda t.(t)I \Vdash \neg(e_1 = e_2) \Rightarrow e_1 \neq e_2$.

Yet using these definitions, the rules of the game would have slightly changed. Indeed, if we observe closely what happens at the last level of the game (with every variable already instantiated but the one of the last universal quantifier), that is a formula $\forall^{\mathsf{N}} y(f(y) \neq 0)$, if the

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

formula is true in the model, then the falsity value is empty, so that the opponent cannot give any answer:

$$\|\forall y(f(y) \neq 0)\| = \bigcup_{n\in\mathbb{N}} \|f(n) \neq 0\| = \|\top\| = \varnothing. \qquad (\forall n \in \mathbb{N}, \mathcal{M} \vDash f(n) \neq 0)$$

Hence, Eloise does not have to compute the formula $f$ to know whether she can win or not, she only has to wait for a potential answer of Abelard, and keep on playing if she eventually gets one.

We shall bring the reader to notice two important facts. First, it is clear that as Eloise has no need to compute $f$, she only needs to do somehow a 'blind' enumeration, hence we can build the very same realizer we built in Proposition 17 without using a term computing $f$. In fact, such a realizer would be suitable for any $f$, even not computable, that is:

**Proposition 18 (Krivine 2009).** For all $n \in \mathbb{N}$, there exists $t_n \in \Lambda_c$ such that for any $f : \mathbb{N}^{2n} \to \mathbb{N}$, if $\mathcal{M} \vDash \exists x_1 \forall y_1 \dots \exists x_n \forall y_n f(\vec{x}_n, \vec{y}_n) \neq 0$, then $t_n \Vdash \exists^{\mathsf{N}} x_1 \forall^{\mathsf{N}} y_1 \dots f(\vec{x}_n, \vec{y}_n) \neq 0$.

Second, such a result it obviously false if we use equality instead of non-equality. Going back to the halting problem, if we consider one of the functions $f : \mathbb{N}^2 \to \mathbb{N}$ such that

$$f(m,n) = 0 \qquad \text{iff } (n = 0 \wedge \exists^{\mathsf{N}} p(\mathrm{Halt}(m,p))) \vee (n \neq 0 \wedge \forall^{\mathsf{N}} p(\neg\mathrm{Halt}(m,p)))$$

it is clear that $f$ is not computable and that $\mathcal{M} \vDash \forall y \exists x(f(y,x) = 0)$ (that only says that a Turing machine stops or does not stop). We know by Proposition 18 that there is a term $u \in \Lambda_c$ such that $u \Vdash \forall^{\mathsf{N}} y \exists^{\mathsf{N}} x(h \circ f)(y,x) \neq 0$, but there is no term[†] $t$ such that $t \Vdash \forall^{\mathsf{N}} y \exists^{\mathsf{N}} x f(y,x) = 0$, and thus no term $t'$ such that $t' \Vdash (\forall^{\mathsf{N}} y \exists^{\mathsf{N}} x(h \circ f)(y,x) \neq 0) \Rightarrow (\forall^{\mathsf{N}} y \exists^{\mathsf{N}} x f(y,x) = 0)$. This phenomena is quite strange[‡], as both formulæ were perfectly equivalent in the ground model. As we explained, a game-theoretic interpretation of this fact is based on the idea on the idea that the use of a non-equality leaves the computation to the opponent, and making so the game easier. However, in the authors' opinion this does not furnish a satisfying enough explanation for the model-theoretic point of view, and it might be interesting to deal with this phenomena more deeply.

### 8.3. *Connection with forcing*

In this paper, we only considered the *standard realizability models* of PA2 (following the terminology of Krivine (2011)), that is: the realizability models parameterized on tuples of the form $(\Lambda, \Pi, \succ, \perp\!\!\!\perp)$, where $(\Lambda, \Pi, \succ)$ is a particular instance of the $\lambda_c$-calculus, and where $\perp\!\!\!\perp$ is a pole. The strong separation between the calculus (on one side) and the pole (on the other side) is essential to define the notion of universal realizability, which is at the heart of the specification problem studied in this paper.

However, the definitions of classical realizability can be extended in many different ways. First, we may replace second-order arithmetic (PA2) by Zermelo–Fraenkel set theory (ZF), using a model-theoretic construction (Krivine 2000, 2012) that is reminiscent from the construction of forcing models and of Boolean-valued models of ZF. *Mutatis mutandis*, all the results presented

[†] Otherwise, using a witness extraction method for $\Sigma^1_0$-formulæ (Miquel 2011a), we would be able for all $m \in \mathbb{N}$ to compute $n_m \in \mathbb{N}$ such that $f(m, n_m) = 0$, breaking the halting problem.

[‡] In fact, it already appears when considering the formula $\forall x(x = 0 \Leftrightarrow h(x) \neq 0)$ that is not realized if not relativized to naturals.

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

in this paper remain valid in the framework of classical realizability models of ZF, provided we consider a representation of arithmetic formulæ in the language of set theory that preserves their computational interpretation in the sense of PA2 (see Krivine (2012)).

Second, we may replace the terms and stacks of the $\lambda_c$-calculus by the $\mathcal{A}$-terms and $\mathcal{A}$-stacks of an arbitrary *classical realizability algebra* $\mathcal{A}$, as shown by Krivine Krivine (2011, 2012). Intuitively, classical realizability algebras generalize $\lambda_c$-calculi (with poles) the same way as partial combinatory algebras (Hofstra and Cockett 2006) generalize the $\lambda$-calculus (or Gödel codes for partial recursive functions) in the framework of intuitionistic realizability. This broad generalization of classical realizability – in a framework where terms and stacks are not necessarily of a combinatorial nature – is essential, since it allows us to make the connection between forcing and classical realizability explicit. Indeed, any complete Boolean algebra can be presented as a classical realizability algebra, so that all Boolean-valued models of ZF (or forcing models) can actually seen as particular cases of classical realizability models of ZF. (In this setting, the combination of realizability and forcing presented in Krivine (2011); Miquel (2011b) can be seen as a generalization of the method of iterated forcing.)

In the general framework of classical realizability algebras, the specification problem studied in this paper does not make sense anymore (due to the loss of the notion of universal realizability), but we can still use the $\lambda_c$-terms presented in Section 8.1 to show more generally that every arithmetic formula that is true in the ground model is realized by a proof-like term.

**Theorem 4.** Let $\mathcal{M}$ be a Tarski model of ZFC, $\mathcal{A}$ a classical realizability algebra taken as a point of $\mathcal{M}$, and $\mathcal{M}^{\mathcal{A}}$ the classical realizability model of ZF built from the ground model $\mathcal{M}$ and the classical realizability algebra $\mathcal{A}$. Then, for every closed arithmetical formula $\phi$ (expressed in the language of ZF) such that $\mathcal{M} \models \phi$, there exists a proof-like term $\theta \in \mathcal{A}$ such that $\theta \Vdash_{\mathcal{A}} \phi$.

This shows that arithmetical formulæ remain absolute in the framework of classical realizability models of set theory, which generalizes a well-known property of forcing models to classical realizability. Actually, recent work of Krivine (2015) shows that this result extends to the class of $\Sigma^1_2$- and $\Pi^1_2$-formulæ as well. By discovering the existence of an ultrafilter for the characteristic Boolean algebra $\mathfrak{2}$2 (Krivine 2012) of the realizability model $\mathcal{M}^{\mathcal{A}}$, Krivine succeeded to construct (by quotient and extensional collapse) a proper class $\mathcal{M}' \subseteq \mathcal{M}^{\mathcal{A}}$ that constitutes a transitive model of ZF elementarily equivalent to $\mathcal{M}$, and that contains the same ordinals as $\mathcal{M}^{\mathcal{A}}$. Hence, the Levy–Schoenfield theorem Jech (2013, Theorem 25.20) applies to $\mathcal{M}$, $\mathcal{M}'$ and $\mathcal{M}^{\mathcal{A}}$, thus proving the absoluteness of $\Sigma^1_2$- and $\Pi^1_2$-formulæ.

**Acknowledgements**

The authors wish to thank Alexandre Miquel, who provided valuable assistance to the writing of the connection between the specification results and forcing in Section 8.3, the anonymous reviewers for their constructive comments, as well as Marjolaine Lacombe, who was of great help in correcting numerous english mistakes.

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

## References


Barbanera, F. and Berardi, S. (1996). A symmetric lambda calculus for classical program extraction. *Information and Computation* **125** (2) 103–117.

Barendregt, H. (1984). *The Lambdas Calculus: Its Syntax and Semantics*, Studies in Logic and The Foundations of Mathematics, volume 103, North-Holland, Amsterdam.

Church, A. (1941). *The Calculi of Lambda-Conversion*, Annals of Mathematical Studies, volume 6, issue 4, Princeton 171–172.

Coquand, T. (1995). A semantics of evidence for classical arithmetic. *Journal of Symbolic Logic* **60** (1) 325–337. doi: 10.2307/2275524. url: http://dx.doi.org/10. 2307/2275524.

Curien, P.-L. and Herbelin, H. (September 2000). The duality of computation. In: Odersky, M. and Wadler, P. (eds.) *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP'00)*, ACM, Montreal, Canada 233–243.

Curry, H. B. and Feys, R. (1958). *Combinatory Logic*, volume 1, North-Holland, Amsterdam.

Friedman, H. (1973). *Some Applications of Kleene's Methods for Intuitionistic Systems*, Cambridge Summer School in Mathematical Logic, Springer Lecture Notes in Mathematics, volume 337, Springer-Verlag 113–170.

Friedman, H. (1978). Classically and intuitionistically provably recursive functions. *Higher Set Theory* **669** (1978), 21–28.

Girard, J.-Y. (2006). *Le point aveugle – cours de logique – volume I – vers la perfection*, Hermann.

Girard, J.-Y., Lafont, Y. and Taylor, P. (1989). *Proofs and Types*, Cambridge tracts in theoretical computer science, volume 7, Cambridge University Press.

Guillermo, M. (2008). *Jeux de réalisabilité en arithmétique classique*, Ph.D. thesis, Université Paris 7.

Guillermo, M. and Miquel, A. (2014). Specifying Peirce's law in classical realizability. *Mathematical Structures in Computer Science*, Epub ahead of print, doi: 10.1017/S0960129514000450.

Howard, W.A. (1980). The formulae-as-types notion of construction. In: Hind-ley J.R. and Seldin, J.P. (eds.) *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, Academic Press, New York, 479–490.

Hofstra, P.J.W. and Cockett, J.R.B. (2006). An introduction to partial lambda algebras. Available at: http://mysite.science.uottawa.ca/phofstra/lambda.pdf

Jean-Louis Krivine J.-L. (2015). On the structure of classical realizability models of ZF. In: Herbelin, H., Letouzey, P. and Sozeau, M. (eds.) *20th International Conference on Types for Proofs and Programs (TYPES 2014)*, Leibniz International Proceedings in Informatics (LIPIcs), volume 39, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany.

Jech, T. (2013). *Set Theory: The Third Millennium Edition, Revised and Expanded*, 3rd edition, Springer Monographs in Mathematics, Springer-Verlag Berlin Heidelberg. isbn: 9783642078996. url: http://books. google.com.uy/books?id=70N-cgAACAAJ.

Kleene, S.C. (1945). On the interpretation of intuitionistic number theory. *Journal of Symbolic Logic* **10** (4) 109–124.

Kreisel, G. (1951). On the interpretation of non-finitist proofs–Part I. *Journal of Symbolic Logic* **16** (4) 241–267.

Kreisel, G. (1952). On the interpretation of non-finitist proofs: Part II. Interpretation of number theory. Applications. *Journal of Symbolic Logic* **17** (1) 43–58.

Krivine, J.-L. (1993). *Lambda-Calculus, Types and Models*, Ellis Horwood series in computers and their applications, Masson.

Krivine, J.-L. (2000). The Curry-Howard correspondence in set theory. In: *15th Annual IEEE Symposium on Logic in Computer Science*, IEEE Computer Society, Santa Barbara, CA 307–308.


https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press

Krivine, J.-L. (2001). Typed lambda-calculus in classical Zermelo-Fraenkel set theory. *Archive for Mathematical Logic* **40** (3) 189–205.

Krivine, J.-L. (2003). Dependent choice, 'quote' and the clock. *Theoretical Computer Science* **308** (1–3) 259–276.

Krivine, J.-L. (2007). A call-by-name lambda-calculus machine. *Higher-Order and Symbolic Computation* **20** (3) 199–207.

Krivine, J.-L. (2009). Realizability in classical logic. In interactive models of computation and program behaviour. *Panoramas et synthèses* **27** 197–229.

Krivine, J.-L. (2011). Realizability algebras: A program to well order R. *Logical Methods in Computer Science* **7** (3) 1–47.

Krivine, J.-L. (2012). Realizability algebras II: New models of ZF + DC. *Logical Methods in Computer Science* **8** (1) 1–28.

McCarty, D. (1984). *Realizability and Recursive Mathematics*, Ph.D. thesis, Carnegie-Mellon University.

Miquel, A. (2007). Classical program extraction in the calculus of constructions. In: Proceedings of the Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11–15. *Springer Lecture Notes in Computer Science* **4646** 313–327.

Miquel, A. (2011a). Existential witness extraction in classical realizability and via a negative translation. *Logical Methods for Computer Science* **7** (2) (doi: 10.2168/LMCS-7(2:2)2011).

Miquel, A. (2011b). Forcing as a program transformation. In: *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science*, IEEE Computer Society, Toronto, Ontario, Canada, 197–206.

Myhill, J. (1973). Some properties of intuitionistic Zermelo-Fraenkel set theory. *Lecture Notes in Mathematics* **337** 206–231.

Oliva, P. and Streicher, T. (2008). On Krivine's realizability interpretation of classical second-order arithmetic. *Fundamenta Informaticae* **84** (2) 207–220.

Parigot, M. (1997). Proofs of strong normalisation for second order classical natural deduction. *Journal of Symbolic Logic* **62** (4) 1461–1479.

Rieg, L. (June 2014). *On Forcing and Classical Realizability*, Theses, Ecole normale supérieure de lyon - ENS LYON. url: https://tel.archives-ouvertes.fr/tel- 01061442.

https://doi.org/10.1017/S0960129515000559 Published online by Cambridge University Press