Verificationism and classical realizability

Abstract

This paper investigates the question of whether Krivine's classical realizability can provide a verificationist interpretation of classical logic. We argue that this kind of realizability can be considered an adequate candidate for this semantic role, provided that the notion of verification involved is no longer based on proofs, but on programs. On this basis, we show that a special reading of classical realizability is compatible with a verificationist theory of meaning, insofar as pure logic is concerned. Crucially, in order to remain faithful to a fundamental verificationist tenet, we show that classical realizability can be understood from a single-agent perspective, thus avoiding the usual game-theoretic interpretation involving at least two players.

KEYWORDS: verificationism; realizability semantics; classical logic; untyped proof theory; axiomatic theories.

1 Introduction

Since the Seventies, Michael Dummett and Dag Prawitz proposed basic desiderata that a general verificationist theory of meaning should satisfy. In a successive number of papers and monographs, they tried to show that classical logic fails to meet such desiderata (see, in particular, Dummett 1973 and Prawitz 1977). The outcome of their analysis is that classical operators fail to convey meaning in a verificationist setting and, a fortiori, that classical logic is philosophically flawed.

On the other hand, since intuitionistic logic meets the meaning-theoretic desiderata, Dummett and Prawitz exploited this theory to advocate a form of logical revisionism: a correct linguistic practice should not be based on classical forms of reasoning but rather on intuitionistic ones. In the last decades, several solutions have been proposed to avoid Dummett's and Prawitz's conclusion about the untenability of classical logic from an inferentialist and anti-realist perspective (we can mention, for instance, the works of A. Weir, S. Read, I. Rumfitt, G. Restall, T. Sandqvist).

Yet, with the only exception of Bonnay (2007), these solutions did not take into account some recent developments of the computational theory of

classical logic and almost none of them focused in particular on classical realizability semantics. The aim of this paper is to contribute to fill this lacuna by investigating a way of assessing the philosophical significance of the so-called Krivine's classical realizability. More precisely, we consider the problem of whether classical realizability can provide a verificationist interpretation of classical logic. In order to do this we will analyse realizability semantics not only with respect to the Dummett-Prawitz's verificationism, but also with respect to Hintikka's one. We argue, in particular, that even if classical realizability seems to be much more closer to Hintikka's verificationism, in fact, a special reading of classical realizability can make it compatible also with the Dummett-Prawitz's perspective. Unfortunately, this special reading is too narrow, and it cannot be extended to proper (classical) mathematical theories, something which is possible instead with the standard reading of Krivine's classical realizability.

The paper is organized as follows. In section 2, the notion of realizability semantics is introduced by analyzing Kleene's realizability for intuitionisitic logic. First, its compatibility with the different verificationist approaches it is investigated. Then, a comparison with Krivine's realizability for classical logic is provided. It is shown, in particular, that even if both of these two frameworks consider realizers as computable entities like programs, the latter adopts a wider and richer perspective, allowing to define not only correct programs, but also wrongful ones. In section 3, we introduce some technical concepts and definitions needed to tackle the philosophical question of the relationships between classical realizability and the verificationist theory of meaning. After a brief survey of the syntax of classical realizability, we show how a two-agents based dialogical interpretation can be considered as a natural framework to model the computational behavior of classical proofs. We then compare it with a similar computational setting called "untyped proof theory". In section 4, we investigate the possibility to use classical realizability as a framework for an anti-realist account of classical logic. Some substantial differences occur between classical realizability and Dummett's verificationism. Nonetheless, we are able to show that Krivine's realizability can be made compatible with a single-agent based perspective, not incompatible, in principle, with a Dummettian perspective. In the final section, we show how classical realizability represents a setting flexible enough to provide a computational account of many mathematical axioms and theories. However, our account of classical logic is jeopardized in this extended mathematical setting. We conclude by pointing out some difficulties encountered by our proposal when proper axioms are added to the underlying classical system.

2 The idea of realizability semantics

2.1 Kleene's number realizability

Historically, the notion of realizability was introduced by Kleene (1945) in order to give a formal semantics for intuitionistic logic and intuitionistic arithmetic. The main source of inspiration for this type of semantics was the finitist explanation of mathematical statements, and in particular existential ones, as it was given by Hilbert & Bernays (1934, p. 32). According to this explanation, a statement of the form $\exists x A(x)$ is considered to convey an "incomplete communication", waiting for the exhibition of a witness t. Only when a finitist method for obtaining t is given, the communication is completed and the statement $A(t)$ can then be effectively asserted (cf. Kleene 1973). In particular, Kleene's idea was that finitist methods were nothing but effective algorithmic methods, eventually representable in a formal setting by (partial) recursive functions. The semantics obtained in such way was thus a semantics based on essentially intensional objects – i.e. algorithms –, rather than on explicitly extensional ones, as in the case of standard algebraic or relational semantics, like Kripke models for intuitionistic logic. This is particularly clear when we represent (partial) recursive functions by using Kleene's normal form theorem: a recursive function f is represented in a unique way by coding the way in which it computes, that is its computational tree:

$$f(\vec{x}) \simeq U(\mu y.T(e,\vec{x},y)) \tag{1}$$

where T is the Kleene-predicate expressing that the function f, coded by the Gödel number e, when applied to the list of arguments $\vec{x}$, is computed according to a computational tree, coded by the Gödel number y; the result of the computation, when it exists, is then extracted by the function U, and μ, which is the minimalization operator, guarantees that the computational tree considered is the one having the smallest code.

This way of representing recursive functions allows one to describe them not only with respect to what they do – i.e. with respect to the values that are obtained when the functions are applied to certain arguments – but also with respect to how they compute – i.e. with respect to the steps that are accomplished in order to obtain the expected values. Moreover, recursive functions can enumerated, so that for each natural number n there exists a recursive function f, so that (1) can be rephrased in the following way:

$$\{n\}(\vec{x}) \simeq U(\mu y.T(n,\vec{x},y)).$$

Kleene's realizability consists in associating a formula of intuitionistic logic or arithmetic – proper axioms included – to a natural number n codifying (by a Gödel numbering) a recursive function f which guarantees that A holds. It is for this reason that Kleene's realizability is also known under the name of "number realizability".

The fact that $n \in \mathbb{N}$ realizes, or forces, a formula A is noted by $n \Vdash A$, and inductively defined in the following way (see Sørensen & Urzyczyn 2006, p. 243 and Troelstra & van Dalen 1988, p. 196, with slight modifications):

- $n \Vdash t = s$ iff t and s rewrite to the same numeral $\overline{n}$;[1]
- $n \Vdash A \wedge B$ iff $n = 2^q \cdot 3^r$, where $q \Vdash A$ and $r \Vdash B$;
- $n \Vdash A \vee B$ iff $n = 3^q$ and $q \Vdash A$, or $n = 2 \cdot 3^q$ and $q \Vdash B$;
- $n \Vdash A \to B$ iff for each m, such that $m \Vdash A$, $\{n\}(m)$ is defined and $\{n\}(m) \Vdash B$;
- $n \Vdash \exists x A(x)$ iff $n = 2^m \cdot 3^r$, where $r \Vdash A(\overline{m})$;
- $n \Vdash \forall x A(x)$ iff for all m, $\{n\}(m)$ is defined and $\{n\}(m) \Vdash A(\overline{m})$.[2]

Notice that $2^q \cdot 3^r$ is an injective pairing function and that without loss of generality we can consider the formula $A(x)$ of the two last clauses to contain only x free. Moreover, according to these realizability clauses the set of realizers is consistent, since $0 = 1$ cannot be realized. And since by definition $\bot \equiv 0 = 1$, the following clause holds:

- $n \Vdash \bot$ for no n.

2.2 Realizers as verifiers

What is peculiar to realizers is that they reflect, at the level of functional operations, the syntactic structure of the realized formulas: there thus exists specific realizers for each formula. In this sense, Kleene's realizability seems to offer a finer-grained semantics with respect to usual algebraic or relational ones, where it is the whole structure that renders true or false the whole set of formulas. In particular, when theorems are considered, algebraic and relational semantics assign them the same semantic value[3] – i.e. the top element of an algebra or the set of all possible worlds of a Kripke frame – with

[1] A numeral $\overline{n}$ stands for a term of the language of arithmetic of the form

$$\underbrace{s(\ldots s}_{n \text{ times}}(0)\ldots).$$

In other words, numerals are terms denoting natural numbers, written in a canonical form.

[2] This means that the numbers realizing universal formulas correspond to total recursive functions.

[3] The semantic value of a formula A is that feature allowing one to determine the semantic notions associated to A, like meaning and truth (cf. Dummett 1991, pp. 24, 30-31).

the risk of trivializing, or at least impoverishing, the way in which logical and mathematical theories are understood.[4]

In Kleene's realizability, on the contrary, the semantic value of a formula A corresponds to the set of its realizers – i.e. $|A| = \{t \in \mathbb{N} \mid t \Vdash A\}$ – which provide « witnesses for the constructive truth of existential quantifiers and disjunctions, and in implications [carry] this type of information from premise to conclusion by means of partial recursive operators. In short, realizing numbers "hereditarily" encode information about the realization of existential quantifiers and disjunction » (Troelstra 1998, p. 408). This means, in particular, that the disjunction and the existential properties (for intuitionistic arithmetic) are satisfied (see Sørensen & Urzyczyn 2006, p. 243). Kleene's realizability could then be thought as a formal way of capturing the intuitionistic notion of truth, as corresponding to the possession of a construction, and thus respecting the BHK interpretation. Since this is obtained by interpreting intuitionistic arithmetic over a fragment of arithmetic itself, then Kleene's realizability could be see as a formal and rigorous characterization of the BHK interpretation, namely by using the technique of *inner models*.[5]

The idea that Kleene's realizability can be seen as a definition of the intuitionistic notion of truth, in the same way as Tarski's notion of satisfiability is seen as a definition of the classical notion of truth, seems to be corroborated by the fact that Kleene's realizability allows us to exclude classical principles. For example, by a simple application of the excluded middle, the formula

$$\forall x(\exists y T(x,x,y) \vee \neg \exists y T(x,x,y)) \tag{2}$$

is shown to be provable, and thus valid, in classical arithmetic. On the contrary, according to Kleene's realizability, the validity of (2) corresponds to the existence of a total recursive function deciding $\exists y T(x,x,y)$, which correspond to the possibility of deciding the halting problem. But this is known to be impossible. Thus, there are no realizers for (2). And since there are no realizers for $\bot$ too, by applying a classical reading of the metalanguage expressions in which the realizability clauses are formulated, we can then conclude that the negation of (2) holds.[6]

[4] It is worth noting that the difference we sketched between Kleene's realizability semantics and the algebraic or relational semantics can be taken as a particular instance of the difference between construction-oriented semantics and conditional-oriented semantics studied by Fine (2014, § 1). According to Fine, the first has to be considered as an exact semantics, while the latter an inexact one. The reason is that, in the first case, the semantical entities are wholly, or exactly, relevant for establishing the truth of a given statement. On the contrary, in the second case, the semantical entities are relevant for establishing the truth of a statement only in a loose and inexact way, which is made particularly evident by the fact that in this kind of semantics the monotonicity of the forcing relation holds (see Fine 2014, p. 551).

[5] We due this observation to Göran Sundholm.

[6] The idea is that, according to the standard practice in intuitionistic logic, we consider $\neg A$ as defined by $A \to \bot$.

However, since realizers correspond to recursive functions, they not only seem to serve in order to establish intuitionistic truths, but they also seem to convey an effective method, or procedure, in order to *verify* these truths. Kleene's realizers can then be seen as *verifiers*, and thus, to know the semantic value of a formula corresponds to know how to verify it. In this way, Kleene's realizability can be considered as a framework allowing one to respect a verificationist account of the semantics of linguistic expressions, similar to the one proposed by J. Hintikka (1996, § 10).

2.2.1 A comparison with Hintikka's and Dummett's verificationism

According to Hintikka, in the case of first-order logic, the truth values of a formula are defined with respect to a given domain of objects by the existence of winning strategies for two players games; where one player (the Verifier) tries to verify the formula, while the other one (the Falsifier) tries to falsify it. Differently from usual algebraic or relational semantics, this definition does not simply look at truth as coming out from some kind of structural and « abstract relationship between sentences and facts », but it also gives an operational definition of it (Hintikka 1996, p. 22). In particular, the idea is that linguistic games are constituent of the use of the notion of truth, and since the rules of a game are, in principle, learnable and teachable, the definition of truth which is given is based on an activity of justification of a certain sentence with respect to a certain set (or domain) of objects (cf. Bonnay 2004, p. 107; Boyer & Sandu 2012, p. 822-823). However, the simple existence of a winning strategy does not assure that such a notion of truth can be accessible to human agents. The reason is that, in the case of first-order logic, winning strategies for the Verifier correspond to Skolem functions,[7] and these functions do not necessarily correspond to constructive strategies. More precisely, if one does not want to look only for a definition of truth, but also for its knowability, it is necessary to guarantee the epistemic accessibility to the set of truths. As proposed by Hintikka, this can be achieved by restricting the set of winning strategies to those that can be effectively played by some idealized human agent, namely those that correspond to recursive functions (see Hintikka 1996, p. 214-215).[8] It would be therefore natural to consider

[7] Notice that the strategies adopted by the Falsifier correspond, instead, to Kreisel's counterexamples (Boyer & Sandu 2012, p. 823). In this sense, if we focus on the winning strategies for the Falsifier — instead of the winning strategies for the Verifier — Hintikka's framework can be adapted for justifying a form of falsificationism.

[8] It is worth noting that an idealized human agent is not an agent totally freed from any kind of contingent constraints: on the contrary, she possess the very same epistemic capacities that any other concrete human beings possess, the only difference being that her capacities are perfect. More precisely, like every concrete human being, she can deal with only a finite amount of resources and information, and her actions can be performed only in a finite amount of time and space; however, unlike concrete human beings, her finite capacities are not subject to any fixed bound.

Kleene's realizers as methods of verifications, in Hintikka's sense, guaranteeing – at least in principle – an access to the truth value of a formula. As Hintikka himself says, « the technical interpretation of my [constructivistic] interpretation [of logic and mathematics] does not stray very far from Gödel's *Dialectica* interpretation of first-order elementary arithmetic or from Kleene's realizability interpretation » (Hintikka 1996, p. 235).

Still, Hintikka's verificationism is not the only form of verificationism existing. Intuitionistic principles and theorems are often justified by making appeal to another form of verificationism, i.e. Dummett's verificationism. This proposal results to be more radical that Hintikka's one, since any appeal to a given domain of objects is rejected, and the verification procedures which are allowed are only those acting at a linguistic-inferential level. More precisely, the idea is that a verification for a formula A consists in an effective method for obtaining a *canonical proof* of A, that is a proof terminating with an introduction rule for the principal connective of A. This method, in general, is composed of two steps: i) the exhibition of a (possibly) non-canonical proof of A, and ii) the application of the normalization algorithm to this proof, in order to obtain a (new) proof satisfying the so-called introduction form property (see Dummett 1973, p. 240; Tieszen 1992, p. 72).[9]

The crucial difference of Dummett's verificationism with respect to Hintikka's verificationism is that, by grounding the process of verification on the notion of proof, a verifier cannot transcend human epistemic capacities, since a proof is, by definition, something which is connected to our linguistic capacities. In other words, the idea is that we are always in a position to recognize

[9] A non-canonical proof is called by Dummett a *demonstration*; its relation with a canonical proof is explained in the following manner:

> We [...] require a distinction between a proof proper – a canonical proof – and the sort of argument which will normally appear in a mathematical article or textbook, an argument which we may call a 'demonstration'. A demonstration is just as cogent a ground for the assertion of its conclusion as is a canonical proof, and is related to it in this way: a demonstration of a proposition provides an effective means for finding a canonical proof. (Dummett 1973, p. 240)

According to Dummett, the notion of canonical proof is the semantic key concept of the notion of meaning. More precisely, to know the meaning of a sentence A corresponds to know the conditions for its (direct) assertion, which corresponds, in turn, to know what counts as a canonical proof of A. Thus, grounding the notion of truth on that of canonical proof is a way to assigning priority to the notion of meaning with respect to the notion of truth. Furthermore, as Dummett remarks, the conditions for the truth of a sentence and those for its correct assertion do not, in principle, collapse: possessing an effective method for obtaining a canonical proof does not necessarily mean to be able to *concretely* execute this method and, eventually, get access to this proof (cf. Dummett 1998, p. 122-123). The reason is that human agents could be subject to contingent limitations – e.g. space or time limitations – which do not allow them to terminate the execution of the procedure (e.g. in the case of the normalization, this procedure corresponds to an algorithm of exponential size complexity, which is unfeasible for concrete human agents with limitation of space). Therefore, it is only when idealized human agents are considered that the collapse between the two notion could obtain.

a proof when we see one (Kreisel 1962, p. 202), since when a set of linguistic expressions (i.e. signs) is given, it is possible, by means of mechanical, syntactical, calculation on these expressions alone, to decide whether or not the given set of expressions is a proof of a certain sentence (Sundholm 1994, p. 144).[10] This guarantees in particular the satisfaction of Dummett's *manifestability requirement*: when a sentence is true, we should always be in a position to manifest our recognition of its truth.

A question which naturally arises is to understand if Kleene's realizability is compatible with Dummett's form of verificationism, as well as whether it is compatible with Hintikka's one. But answering this question corresponds, in fact, to answer another question, that is, to understand whether the intuitionistic notion of truth specified by Kleene's realizability is complete with respect to the intuitionistic notion of proof. The answer to this question is negative, since there exist natural numbers realizing formulas which are not provable in intuitionistic logic or arithmetic. In particular, it can be shown that for every closed formula A, either A or $\neg A$ is realizable (see Sørensen & Urzyczyn 2006, pp. 244-245). This result is, in fact, nothing but a generalization of the way in which we showed the negation of (2) to be realizable: either there is a realizer for A or, if there is not, since $\bot$ is never realized, then any arbitrary number can realize $\neg A$. The very same negation of (2) is an example of a sentence which is realizable, but not provable.

But there is also a second aspect which prevents Kleene's realizability from being compatible with Dummett's verificationism. Differently from the case of proofs, it is not possible to decide whether a given number n realizes or not a certain formula A (see Dummett 1977, p. 320; Sørensen & Urzyczyn 2006, p. 244-245). For example, consider the formula $\forall x(x = x)$. This formula is realized by every Gödel number corresponding to a total recursive function. But the set of total recursive function is not enumerable by a total recursive function, and a fortiori not decidable. Hence, Kleene's realizability cannot satisfy Dummett's manifestability requirement, since a formula A could be realized by a certain realizer t and we would not recognize it as such.[11]

[10] It has been argued that the decidability of the notion of proof is in fact an excessively strong assumption. For example, Sundholm (1986, p. 493) argues that the proof relation is only a semi-decidable notion, since « we recognize a proof when we see one, but when we don't see one that does not necessarily mean that there is no proof there. »However, prominent representatives of this form of verificationism, especially Dummett himself, have firmly advocated the decidability of the notion of proof. A quite exhaustive list of places in which Dummett supports this idea can be found in Sundholm (1983, p. 155).

[11] It is worth noting that some authors, like van Atten (2012, § 4.5.2), considers that an essential aspect of the BHK interpretation is that the concepts « that figure in meaning explanations [...] have to do with our cognitive capacities ». In particular, the idea is that the concept of construction which figures in the BHK interpretation should be conceived such that we recognize a construction when we see one. Accepting this reading of the BHK interpretation – which means indeed to assume that Dummett's verification is a declination of it – would then mean to accept that Kleene's realizability is not a formal version of the BHK interpretation, as claimed before.

2.3 From intuitionistic to classical realizability

The previous discussion pointed out the following situation: even if Kleene's realizability has been conceived as a semantics for intuitionistic logic and arithmetic, it contains in fact several classical features as, for instance, the classical reading of its defining clauses – and especially the interpretation of $\bot$ as the absence of any realizer, which induces meta-level reasonings using classical logic (cf. von Plato 2013, p. 103) – or the fact of relying on the notion of recursive function, which is defined with respect to a classical logic background.[12] One can find confirmation of this aspect in the fact that Kleene's realizability allows one to judge as true some principles that are not intuitionistically acceptable, like Markov's principle, i.e.

$$\forall x(P(x) \vee \neg P(x)) \rightarrow (\neg\forall x\neg P(x) \rightarrow \exists x P(x))$$

where P is a predicate on natural numbers, or a limited version of the excluded middle, i.e.

$$A \vee \neg A$$

where A is a closed formula (see Dummett 1977, § 6.2).[13] However, the presence of these classical features is not yet sufficient for a direct and straightforward use of Kleene's realizability as a semantics for classical logic: as we have already seen, with such a semantics it is not possible to realize a classical principle like (2).[14]

Nevertheless, it would be possible to get a realizability semantics for classical logic by using Kleene's realizability (or some little modification of it) in association with a special kind of parametrized negative translation – similar to Friedman's one (see Friedman 1978) – as showed by Oliva & Streicher (2008). However, what we are interested in here is a more direct way of expressing a realizability for classical logic. We will focus our attention on what

[12] Think of the fact that it is possible to define a recursive function by making appeal to the principle of the excluded middle, as for example

$$f(x) =_{df} \begin{cases} 1 & \text{if the Goldbach conjecture is true} \\ 0 & \text{if the Goldbach conjecture is false} \end{cases}$$

However, there are also other, and more critical, aspects of the notion of recursive function which are inherently classical. For example, the regularity condition, which is used in order to define a function f from a relation R by minimization, states that $\forall\vec{x}\exists y R(\vec{x}, y)$. Here, the existential quantifier is understood classically, in the sense that there is no algorithmic procedure for extracting the witness, otherwise the definition of algorithmic procedures via the notion of recursive functions would be circular (cf. Heyting 1962, p. 195). For further details about the non-constructive aspects of the definition of recursive functions see Coquand (2014) and Sundholm (2014).

[13] Note that Kreisel (1973, p. 268) seems to have in mind a very similar situation when he asks if the « (logical) language of the current intuitionistic systems [have been] obtained by *uncritical* transfer from languages which were, tacitly, understood classically ».

[14] This means, in particular, that Kleene's realizability does not allow one to realize the principle of excluded middle for open formulas.

is called Krivine's classical realizability (see Krivine 2009), and we will give it a conceptual analysis, by trying to understand, in particular, its connections with the verificationist approaches sketched before.

At a first sight, Krivine's realizability can be considered to share two fundamental features with Kleene's realizability: i) it makes appeal to a computational-based notion of realizer, and ii) this notion is revealed to be broader than that of proof. Nevertheless, Krivine's realizability cannot be taken as a simple extension of Kleene's realizability. The reason is that the way in which i) and ii) are conceived is radically different from Kleene's realizability.

As it concerns point i), the notion of computation considered by Krivine is a broader notion than that considered by Kleene. In particular, an algorithm is no longer identified with a recursive functions on natural numbers. In Kleene's realizability we have that the computational aspects are mainly focused on the inputs and outputs of a function. This is particularly clear in the case of the implication and the universal quantifier: given a certain input, if there is no output, the condition is not satisfied, and thus the formula not realized. According to this interpretation, a function is then conceived essentially in a set-theoretical way: it is reducible to a set of pairs of natural numbers. In this sense, the domain and the codomain of a computable function are already fixed from the beginning – in both cases they correspond to the set of natural numbers – and thus a computable function has to be considered as a typed entity. On the contrary, in Krivine's classical realizability, algorithms are considered to be entities having a deeper intensional nature. Their behavior is not established in advance by making appeal to a ready-made notion of type, but it is manifested only when the algorithm is executed – or better, tested – within a given *context* (possibly composed by other algorithms). It is only after that its behavior has been manifested that it will be possible to assign a certain type to the algorithm. Moreover, this type will not be assigned in an absolute and unchangeable way, since this assignment depends from the context in which the algorithm is tested.

As it concerns point ii), in Krivine's realizability the number of realizers exceeds the number of proofs. However, differently from what happens in Kleene's realizability, this does not mean that there exist formulas which are realized but not valid in the theory under consideration.[15] It means, instead, that the set of realizers of a formula does not contain only proofs,

[15]This does not mean that Krivine's realizability always guarantees the theory to be complete with respect to the notion of (classical) proof. This depends indeed from the language in which the (classical) theory is presented. If it is a first-order theory, then completeness holds, but if it is a second-order theory, this could no more be the case (as it depends from the way in which the predicate variables are interpreted in the model). Our presentation of Krivine's realizability rests on a second-order theory (see § 3). The possible lack of completeness is then due to the fact that a second-order language is adopted, and not to the way in which the notion of realizability is conceived.

but also other kind of objects. These objects correspond, in particular, to what can be called *tentative proofs*, that is (deductive) *arguments* whose inferential steps are not totally justified, and thus not necessarily logically correct.[16] The presence of this kind of objects is mainly concerned by the attempt of avoiding computationally trivial interpretations of negative formulas.

As we have seen, in Kleene's realizability the formula $\bot$ is never realized. The consequence is that negative formulas $\neg A$ are either never realized, or they are realized by every (partial) recursive function. In both cases, their computational content is lost, since it is completely trivialized, namely it would not be possible to distinguish two negative formulas on the basis of their computational content. In order to avoid this situation, it has to be possible to define a set of realizers also for $\bot$. But since $\bot$ correspond to a(n always) false formula, and verifying a false formula is contradictory, the idea is then to liberalize the notion of realizers, by defining them not only in terms of verifiers, but also in terms of falsifiers. In this sense, Krivine's realizability seems to be much closer to Hintikka's verificationism than Kleene's realizability, since according to Hintikka, the verification-games are defined with respect to two players, the Verifier and the Falsifier (even if the conceptual priority is eventually assigned to the Verifier, since what counts is the definition of the truth of a formula, and this relies on the definition of the winning strategies associated to that formula). In §§ 4.3, 4.4 we will take a step further and we will try to understand if, from the point of view of Krivine's realizability, this two players perspective could become compatible also with Dummett's verificationism.

3 Krivine's classical realizability

3.1 Definitions

The change of perspective induced by Krivine's realizability with respect to the computational account of realizers confers to this framework a greater flexibility than Kleene's realizability. In particular, Krivine's realizability can be applied to the case of classical logic – and even extended to proper mathematical theories, like arithmetic and set theory (see § 5) – by exploiting the fact that to every axiom can be assigned a different term-constant codifying a certain programming operation. For example, if we take Peirce's law

$$((A \to B) \to A) \to A$$

to be the axiom distinguishing classical logic from intuitionistic logic, it is possible to associate it with a program instruction corresponding to the `call-with-current-continuation` control operator of the programming language SCHEME, as shown by Griffin (1990). Similarly, the axiom of countable

[16] Indeed, as Prawitz (2006, p. 511) remarks, « an invalid proof is not really a proof ».

choice is associated to a program instruction akin to the `quote` instruction of the programming language LISP (Krivine 2003). We will come back later to these examples (see §§ 3.2, 5).

The fundamental notion of computation on which Krivine's account rests on three key ingredients: *terms*, *stacks*, and *processes*. From a syntactical point of view, terms are composed by purely λ-terms enriched by two sorts of constants:

i) *instructions*, noted with κ, and ranging over a non-empty set $\mathscr{K}$, containing at least the constant cc which corresponds to the control operator `call-with-current-continuation`.

ii) *continuations*, noted with k_σ, and where σ ranges over the set of stacks.

Stacks are *lists* of closed terms, the last element of which is a stack constant α; we here suppose that the set of possible stack constants is a singleton $\{\diamond\}$, where $\diamond$ somehow stands for the empty stack. Notice however that some models considered by Krivine, among which the *threads model* (Krivine 2012), use several distinct (and even an infinite number of) stack constants. Terms and stacks are defined by mutual induction according to the following Backus-Naur grammar:

Terms	t,u	$::=$	$x \mid \lambda x.t \mid (t)u \mid \kappa \mid k_\sigma$	$(\kappa \in \mathscr{K})$
Stacks	σ	$::=$	$\diamond \mid t \cdot \sigma$	(t closed)

The system obtained in this way is usually called λ_c (see Krivine 1994, 2003, 2009).

It is worth noting that Krivine's classical realizability has been mainly conceived for theories formulated in second-order logic. For this reason, among the set of terms the operators of pair construction, projection, injection, and case analysis do not appear: at the second-order level they become definable (see Sørensen & Urzyczyn 2006, pp. 280-281). As already mentioned, terms correspond to programs for verifying given sentences, as in the case of Kleene's realizability. From the morphological point of view, they can be divided into two categories: those that contain continuations and those that do not. A term containing no continuation constants is called a *proof-like* term. Intuitively, such a term corresponds to a (logically correct) proof, and thus it can be consider as a verifier in Dummett's sense. We will return to this point in §§ 3.3.2, 3.3.

Stacks, on the contrary, correspond to the evaluation contexts of programs, as they are the environments within which programs "react" and exhibit a specific behavior. Finally, processes are obtained by letting (closed) terms and stacks interact. Thus, contexts can be seen as *tests* for programs. Given a (closed) term t and a stack σ, a process is noted by $t \star \sigma$. Computation is then defined by exploiting an evaluation relation on processes, noted with $\rightsquigarrow$, and defined in the following rewrite rules:

$$
\begin{array}{rcllcll}
\lambda x.t & \star & u\cdot\sigma & \rightsquigarrow & t[x:=u] & \star & \sigma & \text{(pop)} \\
(t)u & \star & \sigma & \rightsquigarrow & t & \star & u\cdot\sigma & \text{(push)} \\
\mathrm{cc} & \star & t\cdot\sigma & \rightsquigarrow & t & \star & k_\sigma\cdot\sigma & \text{(grab)} \\
k_{\sigma'} & \star & t\cdot\sigma & \rightsquigarrow & t & \star & \sigma' & \text{(restore)}
\end{array}
$$

An examination of these clauses shows that in order to calculate the result of an evaluation it is not needed to know how the context σ is made, with the only exception of the grab rule. In this case, it is not the form or the structure of the term[17] that determines the computational action which has to be executed, but the form of the context. This means, in particular, that the computational process does not immediately reduce to the computation of a value when an argument is given to a term but it passes through the interaction between the term and the context in which it is asked to be evaluated. We will try to clarify this point in the next section by studying the cc instruction, which is a constant and thus has no proper internal structure: its computational behavior will then depend only from the context in which it is evaluated or tested.

3.2 Dialogues

The way in which realizability semantics operates, and more precisely the way in which processes and their evaluations have to be understood, can be explained in term of dialogues – or better, disputes – between two agents: the *prover* – i.e. the term –, which has to produce a construction of a certain sentence A, and the *skeptic* – i.e. the stack – which doubts of the existence of this construction and thus tries to challenge the prover with respect to A. Consider the following example, involving Peirce law and inspired by Sörensen & Urzyczyn (2006, pp. 144-145).

1. The prover asserts $((A \to B) \to A) \to A$, which corresponds to affirming that a construction cc of $((A \to B) \to A) \to A$ holds thanks to the application of a 0-ary rule.

2. The skeptic does not agree with this assertion, and tries to challenge it by proposing to the prover the following problem: to exhibit a construction of A, given a construction of $(A \to B) \to A$. In order to do this, she provides a term t realizing $(A \to B) \to A$, asks the prover to provide a term a realizing A – using cc and t –, and prepares herself to challenge the fact that a realizes A with a *test* a' for A.

[17] Notice that under the Curry-Howard correspondence for intuitionistic logic a term representing a program corresponds to a proof written in intuitionistic natural deduction. This means that the form of the term reflects the form of the proof, namely the order of application of the inference rules.

This situation corresponds to considering a process $\infty \star t \cdot a' \cdot \diamond$, where $a' \cdot \diamond$ is the *context* of the challenge, that is the set of presuppositions from which the skeptic moves in order to challenge the prover.[18]

3. In order to continue the dispute, the prover makes use of the presuppositions of the skeptic and claims $A \to B$ (for a more detailed justification of this step see p. 23). This claim corresponds to the introduction of a continuation constant $k_{a' \cdot \diamond}$, coming out from the result of the evaluation of the process $\infty \star t \cdot a' \cdot \diamond$ via the grab rule, which gives $t \star k_{a' \cdot \diamond} \cdot a' \cdot \diamond$.

From the point of view of processes, this amounts to saying that it can be given as an argument to the term t, that is, one could consider the application $(t)k_{a' \cdot \diamond}$, which via the push rule reduces to $t \star k_{a' \cdot \diamond} \cdot a' \cdot \diamond$. But this brings us in a situation where we do not know for sure what will happen. Indeed, the way the process $t \star k_{a' \cdot \sigma} \cdot a' \cdot \diamond$ will reduce depends on how the term t is constructed.

Without going into the details, we can notice that t is a realizer of $(A \to B) \to A$. As we will see in § 3.3.1, this means that any process of the form $t \star u \cdot a' \cdot \diamond$, with u a realizer of $A \to B$ and a' a test for A, will win the dispute. In other terms, when given a realizer of $A \to B$ as an argument, the term t reduces to a realizer of A. This can be done in two ways (see Figure 1). First, t could be a term that does not use its given argument, e.g. t "throws away" the argument, that is t is of the form $\lambda x.u$, where x does not appear in u.[19] In that case, the skeptic provides, in the end, a realizer a of A, and the dialogue continues directly at step 5. The second possibility is that t actually uses its argument to compute its output. The dialogue then continues as follows.

4. At some point during the reduction process, one reaches a step of the form $k_{a' \cdot \diamond} \star a \cdot \sigma'$, where a is a realizer of A. The prover claims that $k_{a' \cdot \diamond}$ is indeed a realizer of $A \to B$, but the skeptic considers its use to be unjustified. She then challenges the prover to provide, given a construction a of A, a construction of B.

5. Since the skeptic gives to the prover a construction a of A, the prover comes back to the first challenge, and uses exactly this construction a in order to satisfy it. In case the computation went through step 4. above, this corresponds to the evaluation of $k_{a' \cdot \diamond} \star a \cdot \sigma'$ into $a \star a' \cdot \diamond$ by the

[18] We will try to clarify later what do we mean here for 'presuppositions' (see 4.4). For the time being, it is sufficient to remark that since a context is a list of closed terms, it cannot be a set of hypothesis, as hypotheses correspond to free variables. Moreover, while hypotheses do not presuppose any epistemic attitude towards their truth or falseness, presuppositions are *believed* to be true.

[19] Notice that the condition that x does not appear in u is not necessary for t not to use its argument. In other words, t could not use the argument in the computation, even if x appears in u. For instance, consider the term $t \equiv \lambda x(\lambda y.a)x$, where x and y do not appear in a. Then t is of the form $\lambda x.u$, where x appears in u, but we have the following reduction sequence: $\lambda x.(\lambda y.a)x \star k_{a' \cdot \diamond} \cdot a' \cdot \diamond \rightsquigarrow (\lambda y.a)k_{a' \cdot \diamond} \star a' \cdot \diamond \rightsquigarrow \lambda x.a \star k_{a' \cdot \diamond} \cdot a' \cdot \diamond \rightsquigarrow a \star a' \cdot \diamond$.

If t does not use its argument:

$$\begin{array}{rcll} \mathrm{cc} \star t \cdot a' \cdot \diamond & \rightsquigarrow & t \star k_{a' \cdot \diamond} \cdot a' \cdot \diamond & \text{(grab)} \\ & \rightsquigarrow & \ldots & \\ & \rightsquigarrow & a \star a' \cdot \diamond & \end{array}$$

If t uses its argument:

$$\begin{array}{rcll} \mathrm{cc} \star t \cdot a' \cdot \diamond & \rightsquigarrow & t \star k_{a' \cdot \diamond} \cdot a' \cdot \diamond & \text{(grab)} \\ & \rightsquigarrow & \ldots & \\ & \rightsquigarrow & k_{a' \cdot \diamond} \star a \cdot \sigma' & \\ & \rightsquigarrow & a \star a' \cdot \diamond & \text{(restore)} \end{array}$$

Figure 1: The term cc is a realizer of Peirce's law.

restore rule. In case the term t provided by the skeptic did not use its argument, the computation reaches $a \star a' \cdot \diamond$ without using the restore rule, since $t[x := k_{a' \cdot \diamond}]$ reduces to a term a realizing A.

The prover has thus been able to meet the challenge of the skeptic because, by appealing to the presuppositions held by the skeptic, she has been able to transform an attack of the latter into its own defense. And since in order to do this the prover used only information coming from the skeptic (namely, a and a'), the skeptic cannot but accept them. It is in this sense that we can say that the prover possesses a winning strategy.

However, this possibility of switching the role of a move in a dispute is not the only aspect which characterizes the dialogical account of classical logic. There is in fact another aspect which is linked to the use of contexts, and which essentially distinguishes intuitionistic dialogues from classical ones. When we look at intuitionistic dialogues, we can notice that the prover always replies to the challenge that the skeptic advanced in the immediately previous step. The defense of the prover consists then in setting up a function which returns a value for every argument proposed by the skeptic (see for more details Sörensen & Urzyczyn 2006, § 4.6). When we look at classical dialogues, we can notice, instead, that the prover can move back to a previous challenge and reply to it, by making reference to the context in which this challenge was previously made, thanks to the restore rule.

3.3 Classical realizability and untyped proof theory

We expose here the realizability interpretation of classical logic. This follows a well-known technique consisting in typing terms *a posteriori*, i.e. assigning types to terms according to their interactive behavior, that is, what they

effectively compute. For instance, the lambda term $\lambda x.x$ could be typed by $A \to A$, for any formula A. This induces *subtyping*, which means that a given term can be assigned to several types at the same time.

3.3.1 Realizability interpretation

Let us indicate with Λ the set of terms, and with Σ the set of stacks.

Once the processes and their reductions defined, they are used to interpret formulas. More precisely, we define the realizability relation $t \Vdash A$, where t is a term and A a formula. The definition proceeds by induction, however differently from the definition given in the case of Kleene's realizability, this definition requires two semantic values and not just one. In particular, one defines for each formula A, what can be called its *falsity value* $\|A\| \subset \Sigma$ – corresponding to its set of falsifiers – and its *truth value* $|A| \subset \Lambda$ – corresponding to its set of verifiers. In order to define those sets, one needs to fix once and for all a so-called *pole* which is a subset $\perp\!\!\!\perp \subset \Lambda \star \Sigma$ of processes which is closed under anti-evaluation: if $t \star \sigma \rightsquigarrow u \star \tau$ and $u \star \tau \in \perp\!\!\!\perp$, then $t \star \sigma \in \perp\!\!\!\perp$. This last condition is quite natural since it is meant to ensure that if a process $t \star \sigma$ reduces to an element of $\perp\!\!\!\perp$, the process $t \star \sigma$ is itself an element of $\perp\!\!\!\perp$. We develop further the intuitions behind the pole in the second part of this section. So, once this pole fixed, one defines:

- the set $T^{\perp\!\!\!\perp}$, where T is a subset of Λ, as $\{\sigma \in \Sigma \mid \forall t \in T, t \star \sigma \in \perp\!\!\!\perp\}$;
- the set ${}^{\perp\!\!\!\perp}S$, where S is a subset of Σ, as $\{t \in \Lambda \mid \forall \sigma \in S, t \star \sigma \in \perp\!\!\!\perp\}$.

For convenience, the set of formulas is extended with a predicate symbol $\dot{F}$ for all function $F : \mathbb{N}^k \to \mathscr{P}(\Sigma)$ mapping a k-tuple to a set of stacks. If we want to realize the axioms of Peano arithmetic, we can consider first-order closed terms to be interpreted on natural numbers, i.e. we have a map $[\![\cdot]\!]$ from first-order closed terms to natural numbers. For further details about these definitions we refer to Guillermo & Miquel (2014). The truth value $|A|$ of a formula is defined from the falsity value $\|A\|$ of A by $|A| = {}^{\perp\!\!\!\perp}\|A\|$. The falsity value of a formula is defined by induction (which uses the truth value in the case of the implication).

$$\begin{aligned}
\|\dot{F}(e_1, \ldots, e_n)\| &= F([\![e_1]\!], \ldots, [\![e_n]\!]) \\
\|A \to B\| &= |A| \cdot \|B\| = \{t \cdot \sigma \mid t \in |A|, \sigma \in \|B\|\} \\
\|\forall x\ A\| &= \bigcup_{n \in \mathbb{N}} \|A[x := n]\| \\
\|\forall X\ A\| &= \bigcup_{F : \mathbb{N}^n \to \mathscr{P}(\Sigma)} \|A[X := \dot{F}]\|
\end{aligned}$$

The relation "t realizes A" is then defined by $t \Vdash A \Leftrightarrow t \in |A|$.

Example 1. In order to give better intuitions, let us illustrate this definition for the case of implication. Suppose that one wants to show that a given

term t realizes a formula $A \to B$, i.e. one wants to prove that $t \Vdash A \to B$. This amounts to providing a proof that $t \in |A \to B|$, i.e. $t \in {}^{\perp\!\!\perp}\|A \to B\|$. So, one wants to show that, for any element $\sigma \in \|A \to B\|$, the process $t \star \sigma$ is an element of $\perp\!\!\perp$. Using the above definition, it is in fact possible to know more about σ, namely that σ is of the form $u \cdot \sigma'$, where $u \Vdash A$ and $\sigma' \in \|B\|$. This means that we are trying to prove that $t \star u \cdot \sigma' \in \perp\!\!\perp$. Since $\perp\!\!\perp$ is closed under anti-evaluation, this implies that $(t)u \star \sigma' \in \perp\!\!\perp$, since the latter reduces to $t \star u \cdot \sigma'$ by a push rule. Since this can be deduced for all $\sigma' \in \|B\|$, this means that $(t)u \Vdash B$, i.e. $(t)u$ is a realizer of B. In conclusion, a realizer t of $A \to B$ is a term such that for every realizer u of A, the application $(t)u$ is a realizer of B.

The definition of the realizability interpretation through falsity values reinforces the interpretation of evaluation contexts as falsifiers, that is as *counterexamples* that when opposed to the corresponding verifiers they produce a deadlock, i.e. something corresponding to a sort of antinomic situation.[20] Moreover, it has to be noticed that in analogy with the untyped setting – exposed below, the notion of termination is not an absolute and unchangeable one, but it depends on which processes configurations have been chosen to represent the terminating states, or better, the terminable ones. In other words, the pole $\perp\!\!\perp$ is chosen as an *arbitrary* set of process closed under anti-evaluation.

3.3.2 Classical realizability as an untyped proof theory

In Naibo et al. (2015) the notion of *untyped proof theory* is introduced in order to describe a general framework for dealing with an abstract mathematical (and in particular, geometrical) notion of proof from which it is possible to generate a class of deductive systems suitable both for logical and proper mathematical theories. More precisely, an untyped proof theory represents a very abstract model of computation, based only on two fundamental notions, that of execution and that of termination. In this sense, the idea is to describe a logical or mathematical theory from a computational point of view, in a similar vein as it is done in the theory of constructions of Kreisel (1962, 1965) and Goodman (1970, 1973a, 1973b).[21]

[20] Strictly speaking, these antinomic situations do not imply the incoherence of the system itself. The reason is that, as we already mentioned, verifiers, as well as falsifiers, are only posits. In this sense, it is not astonishing to conceive two logically incompatible situations together: the resulting conflict between these two situations would be only a conflict in principle, not an actual one. On the contrary, a genuine incoherence is obtained when two contrary evidences are present, namely when it is possible to exhibit two proofs of two opposite propositions, respectively (see Miquel 2009a, p. 81). This way of understanding incoherence is the same professed by Hilbert: incoherence is definable only at the level of «concrete objects» (Hilbert 1926, p. 376), i.e. at the level of finitary arithmetic, and not at the level of logic.

[21] Notice that while the notion of execution seems to be (in a form or another) universally accepted as a fundamental ingredient of the notion of computation, the notion of termination

We first recall the general definition of such a framework and then explain to what extent it relates to Krivine's realizability. An *untyped proof theory* is given by:

- A set of untyped paraproofs Π;
- A notion of execution $Ex : \Pi \cdot \Pi \to \Pi$;
- A notion of termination given by a set of untyped proofs Ω.

This definition accounts for a number of so-called *dynamical models* of linear logic such as Geometry of Interaction (Girard 1989), Ludics (Girard 2001), and Interaction Graphs (Seiller 2014). From the notions of termination and execution, one derives a notion of *orthogonality* over the set of untyped paraproofs, i.e. it is possible to define $\perp\!\!\!\perp \subset \Pi \times \Pi$ as the set $\{a \cdot b \mid a, b \in \Pi, Ex(a \cdot b) \in \Omega\}$. From this notion of orthogonality, then one defines the interpretation of formulas in a similar fashion as in the realizability case.[22]

The classical realizability setting is very close to the untyped proof theory framework, except for the loss of symmetry. Where realizability considers two disjoint sets of terms and stacks, untyped proof theory considers a single set of paraproofs. This can be explained by the fact that untyped proof theory is meant for interpreting linear logic formulas, while classical realizability is meant for interpreting classical logic. Linearity allows for the consideration of *one-element stacks* exclusively, that is, those being naturally identified with the term they contain. Modulo this difference, everything works in the same way. In particular, the notion of execution in classical realizability is simply the evaluation of processes. While, as it concerns the notion of termination, even if in classical realizability it is not considered explicitly,[23], the pole of classical realizability corresponds to the induced orthogonality in untyped

needs some explications. In some more specific and "concrete" models of computation, like the one represented by partial recursive functions, termination is not a necessary notion (think precisely of the partiality condition). Following Kreisel (1972), this represents an analysis of *mechanical* effective computability, in the sense that the execution has to be performed by mechanical following a finite list of instruction. However, nothing is said about who has to follow this list of instruction. If it is a human-agent that has to follow it, then the number of steps that she can perform must be finite, since finiteness is a property defining human-agent (see note 8). This means that each execution has to terminate. The notion of termination seems then to be linked to the analysis of what Kreisel calls *human* effective computability. Beside this computational aspect, termination plays a second key role in the present context. From a meaning theoretic point of view, termination ensures us that we are not transcending the capacities of the human agent, thus allowing us to respect the pivotal desideratum of an anti-realist theory of meaning.

[22] Notice, however, that one defines the interpretation of linear logic formulas, and not classical logic formulas, as it will be clarified below.

[23] Although the pole is sometimes defined from a set of processes which could be understood as a notion of termination (see Guillermo & Miquel 2014). For instance, one can take an arbitrary set of processes Ω and then define a pole $\perp\!\!\!\perp_\Omega$ by simply considering the closure of Ω with respect to anti-reduction.

proof theory. In this sense, classical realizability can be understood as an instance of untyped proof theory.

Moreover, classical realizability and untyped proof theory share the possibility of representing incomplete or logically incorrect (deductive) arguments as well as their computational counterparts, that is, wrongful programs.[24] As explained in Naibo et al. (2015), the approach undertaken by untyped proof theory differs from the usual techniques adopted in standard proof theory, as it considers a set of objects – the *paraproofs* – which is much larger than the set of proofs. More precisely, among the set of paraproofs, only a limited subset can be mapped to logically correct and closed derivations,[25] while the others represent aborted or logically incorrect derivations. In other words, this means that the set of logically correct and closed derivations – i.e. the set of proofs – can be interpreted as a specific subset of the set of paraproofs, which are shown to share a specific common property. The set of *winning paraproofs* is then defined as the set of those paraproofs satisfying this property. Although all interpretations of proofs are winning paraproofs, it is not clear in general if a winning paraproof is the interpretation of a proof. In the specific framework of Ludics (Girard 2001), the winning paraproofs are defined as those that do not use a specific non-logical axiom rule named *daimon*, which allows to derive any sequent of the form $\vdash A$ where A is a positive formula. A similar situation appears in classical realizability, since not every term corresponds to a proof. More precisely, the only terms which can be associated to proofs are those that do not contain any continuation constant, that is the so-called proof-like terms (see p. 12): every proof corresponds to a proof-like term, even if the converse is not true in general (a proof-like term need not be typeable[26]). From a more technical point of view, the proof-like terms are those that correspond to winning strategies in a dispute, like the one we described in § 3.2. Proof-like terms are then the realizability analogues of the winning paraproofs. Pushing further this parallel with untyped proof theory, and especially with Ludics, one can think of the continuation constants as those parts of programs, or deductive arguments, which make them incorrect. In other words, continuation constants play a role analogous to the aforementioned daimon rule.

The fact that terms could contain continuation constants plays a crucial

[24] We use the terminology of "wrongful programs" as opposed to "proved programs". Indeed, programs corresponding to proofs in a formal system are *proved* in the sense that they do exactly what they are expected to. More precisely, the corresponding proof can be understood as a certificate that ensures the program is well-behaved (i.e. produces the right type of output when given the right type of input), and terminates. With this idea in mind, a wrongful program is a program which is proved using a incorrect arguments: it is therefore provided with an *unreliable* certificate of well-behaviour and termination.

[25] Notice that in this sense a proof is considered as a closed (logically) valid derivation (or argument), respecting the definition given in Prawitz (2006, p. 511).

[26] For instance, the term $(\lambda x \lambda y.((y)(x)\lambda z.z)(x)\lambda z.\lambda w.z)\lambda z.(z)z$ is not typable in System **F** even though it is strongly normalizable (Giannini & Ronchi Della Rocca 1988).

role, as it means that every formula – even those that are not provable – can be associated to a non-empty set of realizers. In particular, $\bot$ – which in a second-order framework is defined by $\forall X.X$ – can be realized by some terms, not corresponding to proofs, as they contain continuation constants (see, for instance, the term $k_\sigma x$ of the derivation at p. 23). This represents a fundamental aspect differentiating Krivine's realizability from Kleene's one, as it allows one not to trivialize the interpretation of formulas of the form $\neg A$.

Before focusing our analysis on some specific philosophical problems emerging from classical realizability, it is important to point out a major difference between the untyped proof theory approach and classical realizability. Although it is possible to understand classical realizability as an instance of untyped proof theory, there is a *methodological* difference between the two of them. Untyped proof theory somehow works in a top-down fashion: one considers a huge set of paraproofs – i.e. a specific class of mathematical objects – with a homogeneous notion of execution, and then shows what logical/computational principles can be interpreted there. Classical realizability, on the contrary, works in a bottom-up fashion: one extends syntactically the set of paraproofs with in mind a notion of reduction of processes that captures a computational principle, e.g. the `quote` operator (see § 5).

4 Which theory of meaning for classical realizability?

As we have seen in § 3.3.1, Krivine's realizability represents an operational semantics for classical logic, allowing one to assign a computational meaning to classical principles. It becomes than natural to ask if it is also possible to use Krivine's realizability in order to define some kind of anti-realist theory of meaning for classical logic, and in particular a theory of meaning based on the computational uses associated to classical principles and classical operators, rather than their truth-conditions.

4.1 Analogies with the finitist interpretation

Let us first remark that it seems not to be an exaggeration to say that Krivine's proposal respects, in some sense, the spirit of Hilbert's finitist programme, which we have seen to be the starting point of Kleene's realizability interpretation. More precisely, like in Hilbert's account, sentences are meaningful only when they can be associated – or somehow reduced – to concrete objects or to finitist operations defined over these objects. In particular, in the realizability setting, standard Hilbert's strokes – i.e. |, ||, |||, etc. – are replaced by terms and contexts, and since terms and contexts are syntactical objects – that is nothing else but finite configurations of signs – they are also objects

existing in time and space (see Martin-Löf 1970, p. 9). These objects can thus be considered as concrete as Hilbert's strokes are.[27]

In order to show that Krivine's realizability can also recover the notion of finitist operation, the usual identification of this notion with that of primitive recursive function – as proposed by Tait (1981) – has to be abandoned, and it has to be replaced with Kreisel's idea according to which the class of finite operations corresponds to the class of provably recursive functions in arithmetic (Kreisel 1960; see also Zach 2015, § 2.3).

The class of provably recursive functions is exactly the class of functions that Kreisel characterized in establishing his *no-counterexample interpretation* (Kreisel 1951, 1952). Now, the no-counterexample interpretation, as explicitly stated by Krivine (2003, p. 260) and, as we implicitly sketched in §§ 3.2, 3.3, is the method inspiring Krivine's analysis of the computational content of logical and arithmetical theorems: when a sentence is provable, it is shown that is possible to extract an effective procedure capable of falsifying every possible counterexample for that sentence (see Bonnay 2002 for more details). However, this does not mean that Krivine's realizability could be eventually neither reduced to be a simple variant of Kreisel's no-counterexample, nor to be a simple re-interpretation the finitist programme. As we already mentioned, Krivine's realizers are not "ready-typed" functions but untyped programs, the evaluation of which needs the definition of a context. Moreover, Krivine's realizability does not necessarily ask for the extraction of a witness from an existential statement (Rieg 2014, p. 9). This is because the finitist operations are not here carried out at the level of individuals denoted by first-order (closed) terms, but rather at the level of programs for (classically) provable sentences. In other words, the witness which is looked for is not the one for statements of the form $\exists x A(x)$, but the one for judgments of the form $\vdash_{\mathtt{C}} A$, for a certain sentence A and where $\mathtt{C}$ is a derivation system for classical logic.

4.2 Differences with Dummett's verificationism

Since the natural witness for a judgement of the form $\vdash_{\mathtt{C}} A$ is a proof of A – or a proof-term codifying a proof of A –, it would then be natural to look at Dummett's verificationism as the closest theory of meaning with respect to Krivine's realizability semantics. However, it is quite immediate to notice that there is some important differences dividing these two theories.

As we already remarked, the class of terms in classical realizability do not correspond to the class of proofs, being it much bigger. Restricting the attention to the subclass of proof-like terms is still not sufficient for dealing

[27] Actually, as C. Parsons remarked, Hilbert's strokes, as well as syntactical objects in general, are *quasi-concrete* objects: they are not simple tokens, but a particular kind of types, the « intrinsic [property of which is] to have instantiations in the concrete » (Parsons 2008, p. 242).

only with proofs, since, as we said in § 3.3, there is not a perfect correspondence between these two notions. Moreover, in order to assign a type A to a certain proof-like term – and thus making it a realizer – it is necessary to make appeal to a set of contexts. In other words, this means that in classical realizability the understanding of a sentence A is not based on a single semantic notion – that of a proof of A (or better, that of a canonical proof of A) – as it is the case for Dummett's verificationism, but it requires to make appeal to two semantic notions at the same time: i) programs – corresponding in a loose way to proof (when we restrict ourselves to consider only proof-like terms) – and ii) contexts. As we have seen, these two notions are complementary, so that a special kind of *bivalence* is introduced at the semantic level: every well-formed object belonging to the realizability level corresponds either to a program or to a context. Following Dummett (1963), it is exactly the acceptance of a bivalence principle which commits someone to accept a realist approach with respect to semantical notions.

The question then arises if it is possible to avoid such a built-in form of bivalence in classical realizability in order to make it compatible with a verificationist and anti-realist approach. One first step in this direction is represented by the shift from the two-agents dialogical perspective presented in § 3.2 to a single-agent perspective. By providing an account of classical realizability in terms of a single agent, one could hope to avoid, at least in principle, bivalence.

4.3 From a dialogical to a single-agent perspective

As we already mentioned in § 4.1, differently from recursive functions, programs are not reducible to some particular kind of step-by-step set-constructions on natural numbers, but they aim at expressing every kind of actions effectively performable on syntactical objects in general, even on those that would not be considered in principle as well-typed. It is this general and abstract character that allows one to assign a computational content even to those formulas that do not correspond to theorems or axioms, and thus to show it is possible to assign to these formula a specific semantic value. More precisely, the semantic values are set of terms which contain continuation constants. This means that the semantic value of a formula – and in particular its truth value – is obtained under the hypothesis that some counterexample for another formula is given. Let us clarify this point through an example.

According to the usual classical semantics, a formula $A \to B$ can be made true not only when a proof of it is given, but also when a counterexample of A is given. As we have seen in § 3.2, counterexamples can be represented by stacks. Moreover, as we have seen in § 3.1, the information present in a stack σ can be codified by a continuation constant k_σ, and since a continuation constant is a term, then it is plausible to think this term as typeable with $\neg A$. Formally speaking, if $\sigma \in \|A\|$, then k_σ is a realizer of $\neg A$: for any element $\pi \in$

$\|\neg A\|$, by definition $\pi \equiv t \cdot \rho$, with $t \in |A|$, then the process $k_\sigma \star t \cdot \rho$ reduces to $t \star \sigma$ which belongs to $\perp\!\!\!\perp$. In this way, modulo a certain degree of approximation, we can think of $A \to B$ as obtained through the following derivation in a natural deduction setting where formulas are decorated in a Curry-Howard fashion:[28]

$$\dfrac{\dfrac{\dfrac{\dfrac{k_\sigma : \neg A \qquad [x : A]^{(1.)}}{(k_\sigma)x : \perp}\ \to \text{elim}}{(k_\sigma)x : \forall X.X}\ df}{(k_\sigma)x : B}\ \forall^2 \text{ elim}}{\lambda x(k_\sigma)x : A \to B}\ \to \text{intro } (1.)$$

The approximations that we have made in this derivation are mainly two. The first one is to consider that the falsity value of the formula A is not empty. The second one is the result of a mismatch between the point of view of realizability and the point of view of natural deduction. In Krivine's framework, untyped objects are considered, and their interaction – like the application of one to the other – is operated at the level of closed terms.[29] In natural deduction, instead, typed objects are considered, and it is possible to operate also on open terms, i.e. on terms containing free variables. This implies that the typing relation ":" used in the above derivation contains the realizability relation, but it is not equivalent to it. More precisely, the idea is that not every deduction step can be read as expressing a realizability relation between a term and a formula. For example, $x : A$ does not express a realizability relation, since x is not a closed term. The premiss and the conclusion of the derivation, however, can be read as expressing realizability relations. Since $\lambda x(k_\sigma)x$ is the η-expansion of k_σ, and η-equivalent terms can be identified from the computational point of view, then we can consider to be equivalent to work with $\neg A$ instead of $A \to B$.

Notice also that our type assignment to k_σ is not obtained by exploiting the instruction cc, but it is directly extracted from the intuitive reading of the role played by a context σ, i.e. that of a falsifier. However, we can recover Krivine's reading of k_σ once a proof of A is effectively given, i.e. when x is substituted by a closed term u. Looking then from the structural point of view, it should be noticed that $k_\sigma : \neg A$ cannot be considered as a dischargeable hypothesis, since k_σ is by definition a closed term and not a variable. But it

[28] The deduction system adopted here is described in details in Miquel (2009a, p. 85). The idea is that by working in second-order logic we obtain a *polymorphic* type system, that is a system where terms could be associated to more than one type. Since in this paper we adopted the convention to present terms in Curry style, this means that the information concerning types is not present in the terms, and thus polymorphism is not explicitly manifested inside terms – by means of some abstraction operator –, but remains implicit (see Hindley & Seldin 2008, p. 119-120). It is for this reason that the rule $\forall^2$ elim is not associated to any new operation on terms.

[29] As we mentioned at p. 12, processes usually operate on closed terms.

cannot be considered a closed premiss either, since it has not been justified by a proof, but it comes from the "reification" of a context. In a certain sense, the role played by $k_\sigma : \neg A$ is that of a *pretension*, namely the pretension to accept that it is the case that $\neg A$, i.e. to work *as if* $\neg A$ has been proved.

In the dialogical setting, we can interpret the assumption $k_\sigma : \neg A$ as reflecting the prover's attitude to think that the skeptic wants to *refute* her, i.e. to think that the skeptic believes $\neg A$. In turn, this attitude can be seen as the prover's understanding of the "context of the game": from the fact that the skeptic systematically challenges her claims, the prover evaluates that the skeptic does not only doubts about her assertions, but wants also to refute them. However, by interpreting $\neg A$ as a pretension, we open the way to pass from a dialogical, multi-agent position to a single-agent-based perspective. This shift is necessary if one wants to stay as close as possible to a Dummettian theory of meaning. Indeed, the fundamental divergence distinguishing Dummett's and Krivine's settings that we discussed in § 4.2 remains unchanged even if one switches from the computation reading of realizability to its dialogical reading: the distinction between the Verifier – i.e. the prover – and the Falsifier – i.e. the skeptic, provided that we consider negative hypotheses – reintroduces exactly the same form of bivalence that one finds between programs and contexts.

4.4 Negative hypotheses as postulates

The shift operated by working *as if* $\neg A$ has been proved corresponds to consider that $k_\sigma : \neg A$ plays the role of a *postulate*, in an Aristotelian sense, i.e. as « something [which is] not in accordance with the opinion » of the person to whom the discourse is addressed (Aristotle, *Posterior Analytics*, 76b32-34; Barnes 1993, pp. 16, 141-142). From this perspective, the reading of negated assumptions decorated by a continuation constants becomes clear: they are *presuppositions*, which are believed to be true, i.e. special kind of hypotheses to the truth of which an agent commits herself.

One should carefully distinguish this kind of hypotheses from the usual ones, that is, like those used in standard natural deduction. Assuming a sentence to be true can have indeed two different senses: a *potential* and an *actual* one. This distinction is made explicit by Martin-Löf (1991). In the potential sense, assuming that 'A is true' corresponds to assume that the assertion of A is *justifiable*, i.e. that *in principle* there could be found an evidence in favor of it – even if *in fact* it could occur that this evidence is not available, nor it will ever be available. By making such of an assumption we are thus keeping open every kind of possibility: both the existence as well as the non-existence of an evidence in favor of A. On the contrary, in the actual sense, assuming that 'A is true' corresponds to assume that the assertion of A has already been *justified*, i.e. that there is an evidence in favor of it. By making such of an assumption we are thus *pretending* that an evidence in

favor of A effectively exists – even if it is not the case, in the sense that we cannot prove this existence claim.

As we just said, the first kind of hypothesis corresponds to standard hypothesis in natural deduction, which are used for the formation of conditional statements. In order to prove a sentence of the form $A \rightarrow B$, we do not necessarily need to possess a proof of A. For example, we can prove $(A \wedge \neg A) \rightarrow \bot$, even if $A \wedge \neg A$ is not provable at all (cf. Sundholm 1994, p. 163-164). Following the standard notation adopted in the Curry-Howard correspondence between proofs and programs, we will decorate this kind of assumptions using term-variables of the form x, y, z, etc. The proof of $(A \wedge \neg A) \rightarrow \bot$ will be thus decorated in the following way:

$$\dfrac{\dfrac{\dfrac{[x : A \wedge \neg A]^{(1.)}}{p_2(x) : \neg A} \wedge \text{elim}_2 \quad \dfrac{[x : A \wedge \neg A]^{(1.)}}{p_1(x) : A} \wedge \text{elim}_1}{(p_2(x))p_1(x) : \bot} \rightarrow \text{elim}}{\lambda x.(p_2(x))p_1(x) : (A \wedge \neg A) \rightarrow \bot} \rightarrow \text{intro (1.)}$$

where p_1 and p_2 are the first and second projection, respectively.[30]

The second kind of hypothesis corresponds, instead, to those hypotheses used for establishing (proper) admissibility results. In order to establish the validity of an inference rule of the form $\dfrac{A}{B}$, it is asked to assumed that A has been proved, and show that under this assumption B can also be proved. This corresponds to take the sequent $\vdash A$ as an hypothesis, and show that $\vdash B$ can be concluded. But doing this seems to generate two sorts of complementary problems. On the one hand, if we want to discharge this second kind of hypothesis, we have to discharge sequents, which means that we are dealing with some kind of higher-order rules similar to those used by Schroeder-Heister (1984). On the other hand, the conceptual distinction between two kinds of hypothesis risks to be flattened by the following result:

Proposition 2. *Let $\mathscr{B} = \{\vdash A_1, \ldots, \vdash A_n\}$, where the $\vdash A_i$ stand in the position of hypothesis. Then, $\vdash_{NJ+\mathscr{B}} \Gamma \vdash C$ if and only if $\vdash_{NJ} \Gamma, A_1, \ldots, A_n \vdash C$.*[31]

This result concerns the derivability level, while it says nothing about the proof-structure level. But, as we have seen in the case of the definition of canonical proofs, proof-structure is an essential ingredient of Dummett's

[30] Making appeal to projections is for simplicity and shortness of notation. In fact, as we said at p. 12, these operators can by defined in the second-order setting in which Krivine's realizability is conceived.

[31] Notice that we consider to work here with a natural deduction presented in a sequent calculus style. The proof of the proposition can be found in Negri & von Plato (2001, pp. 134-135). In general, in order to prove the 'only if' direction, the idea is to replace every $\vdash A_i$ sequent used in the proof of $\Gamma \vdash C$ with an identity sequent $A_i \vdash A_i$. While in order to prove the 'if' direction, the idea is to apply a cut rule on the A_i, having $\vdash A_i$ and $\Gamma, A_1, \ldots, A_n \vdash C$ as premisses.

verificationism: the key-semantical entities are not proofs in general, but proofs having a particular form, concerning the order of application of inference rules.

We are thus in a situation, in which, on the one hand, we would like to keep track of the distinction between the two kinds of hypothesis we have introduced – so to keep track also of the structural difference between the proofs using one kind of hypothesis or the other – and, on the other, to respect the result stated in the previous proposition – and thus treat the second kind of hypothesis as hypothesis acting on formulas and not on sequents. In order to do this, we can take the second kind of hypothesis as formulas decorated by a different set of variables than the first kind of hypothesis. In particular, we can decorate the second kind of hypothesis using the variables a, b, c, etc., and operating their dischargement using an abstraction operator different from standard λ-abstraction (used for the first kind of hypothesis). The behavior of this operator can be explained by reflecting on the conditions that Dummett's theory of meaning has to satisfy in order to certify his soundness.

4.4.1 A verificationist account of classical logic

Dummett's theory of meaning rests on a what is called a *fundamental assumption*: the assertion of a sentence A, having $\dagger$ as principal connective, should always be performable in a direct way, that is, by means of a $\dagger$-introduction rule (see Dummett 1991, p. 254). In the case of intuitionistic logic, this assumption is respected by considering assertions performed under zero assumptions, which means that only the assertions of theorems are considered. This seems a to be indeed a too narrow interpretation of the fundamental assumption. Satisfying the fundamental assumption when only empty sets of assumptions are considered seems to be just a *minimal* test that the verificationist theory of meaning has to pass in order to be considered as a sound theory. But what about a maximal test? How should it look?

It seems to us that by making appeal to the second kind of hypothesis analyzed in the previous section this maximal test can be defined, and it would take the form of a *robustness test*:

> *There should exist at least a proposition A that can be directly asserted under the worst possible (and non-trivial) assumption, i.e. under the assumption that $\neg A$ is true, in the sense that an evidence for the refutation of A is supposed to effectively exist (and $\neg A$ has not been introduced by a weakening rule).*

The formulation of this test suggests that it is possible to restrict our attention to the use of negative hypotheses, i.e. what we called postulates. Formally speaking, this means that given derivation of the form:

$$\begin{array}{c} a : \neg A \\ \vdots \\ t : A \end{array}$$

it should be possible to rearrange the order of the rules so to conclude using an introduction rule of the principal connective of A.

The problem is that by using again the hypothesis $a : \neg A$, it would be possible to conclude $(a)t : \bot$, which means that under the assumption of the effective possession of a proof of $\neg A$, an inconsistency can be proved. In order to avoid that the addition of the second kind of hypothesis leads to inconsistency, it would be sufficient to deactivate $a : \neg A$ immediately after having derived $t : A$ – so that the hypothesis cannot be used again –, and then continue to assert A. This corresponds to use the following inference rule:

$$\dfrac{\begin{array}{c}[a : \neg A]^{(n.)} \\ \vdots \\ t : A\end{array}}{\nu a.t : A}\ CM\ (n.)$$

where ν is an abstraction operator acting exclusively on the second kind of hypothesis.

This rule corresponds to the principle of *consequentia mirabilis*, i.e.

$$((A \to \bot) \to A) \to A$$

which with respect to intuitionistic logic is equivalent to the Peirce's law (see Ariola et al. 2007, p. 407). This means, in particular, that when *CM* is added to the system of intuitionistic natural deduction (NJ), one obtains a system which allows to recover full classical logic.

A fundamental feature of such a classical system is that it satisfies a form of *quasi-canonicity* of proofs. More precisely, thanks to a result due to Seldin (1989), it is possible to show that if A is a theorem of classical logic, then there exists a proof of it using only one occurrence of the *CM* rule, namely the last one. This means that the conclusion of the penultimate step of the derivation is also A, and that this occurrence of A is intuitionistically derived under the only assumption $\neg A$. But since $\neg A$ is an Harrop formula, then this derivation enjoys the introduction form property, and thus this immediate sub-derivation of A terminates with an introduction rule of the principal connective of A.

The classical system obtain via *CM* does not fully fit in the proofs-as-programs paradigm: providing a direct computational interpretation of *CM* is not a trivial matter. However, a parallel with Kreisel's no-counterexample interpretation can shed light on the implicit computational features of the system. In particular, this should be sufficient in order to guaranteeing that *CM* can recover those computational features of classical logic captured by Krivine's realizability.

The Kreisel's no-counterexample account

(1) Consider $A \equiv \exists x \forall y A_0(x,y)$ and $\neg A \equiv \forall x \exists y \neg A_0(x,y)$, where A is a theorem.

(2) A counterexample to A would consists in a function f such that

$$\forall x \neg A_0(x, f(x)) \qquad (*)$$

(3) Appealing to the consistency of the system guarantees that there exists a functional Φ satisfying the Herbrand normal form of A, i.e.

$$\forall f A_0(\Phi(f), f(\Phi(f)))$$

(4) The functional Φ represents a counterexample to (*), i.e. a counterexample to the existence of the function f.

(5) The computability of the procedure is assured by proving Φ to be recursive. In this particular example, since A is a Σ_1^0 formula, then Φ is even primitive recursive (see Parsons 1972).

The *CM* account

(1′) Consider A and $\neg A$.

(2′) A counterexample to A would consists in a proof of $\neg A$, which means to take

$$a : \neg A$$

(3′) The existence of a derivation $\mathscr{D}$ from $\neg A$ to A, and the subsequent application the rule *CM*, guarantees the consistency of the system, as it allows to transform any alleged proof of $\neg A$ into a proof of A, i.e.

$$\nu a.(t)a : A$$

where ν is an abstraction operator acting on variables used to indicate postulates.

(4′) The proof-term t represents a counterexample to the existence of any possible closed proof-term replacing a.

(5′) The computability of the procedure is assured by the fact that $\mathscr{D}$ uses only intuitionistic means.

Even if the classical system based on the rule *CM* is obtained from an analysis of Krivine's system λ_c, it involves nevertheless some significant differences with the latter. First, it allows one to transform into a rule of inference what in Krivine's is treated like an axiom, that is, the classical principle

expressed by the Peirce's law. This means, at the level of terms, to transform the constant α into a complex term making use of the abstraction operator ν. This transformation, in turn, has been made possible by distinguishing between two kinds of hypothesis, which allows in particular to avoid the use of contexts. The fundamental consequence is that the sneaking of bivalence is blocked: one does no more need to make appeal to a dialogical setting, as the only fundamental semantic concept is the one of proof, as requested by the Dummettian verificationism.

We conclude this section by noting that the ν operator acts similarly to the μ of $\lambda\mu$-calculus (Parigot 1992), with the major difference that in $\lambda\mu$-calculus the μ operator is paired with a second rule allowing the formation of contexts and the evaluation of a term in such a context. In the typed setting, these can be seen as an introduction and an elimination rule for classical negation. On the other hand, in the classical setting we sketched, there is no appeal to contexts: the meaning of logical constants is fixed by the intuitionistic rules and *CM* plays the role of a *structural* rule, in the sense that it acts on the structure of derivations – i.e. at a "global" level – and it allows one to control the discharge of postulates.

5 The computational meaning of axioms

In the previous section, by operating an extension of the notion of hypothesis used in natural deduction systems, we have been able to offer an understanding of the *logical* part of Krivine's realizability in terms of proof-analysis. In other words, by the introduction of a derivation system based on two kinds of hypothesis, we have given a presentation of classical logic which is compatible both with Krivine's realizability and Dummett's verificationism.

However, as we have already anticipated, Krivine's realizability can also be used to give an interpretation — in the sense of assigning a semantical value — to the axioms of proper mathematical theories. But differently from Kleene's realizability, it does not only allow one to interpret the axioms of arithmetic. It covers also the axioms of set theory.

As we already seen in the case of pure (classical) logic, the interpretation given by Krivine's realizability is a computational one. And the notion of computation is, in turn, based on that of execution. The notion of execution is not a stable one, thought. In particular, by adding a new proper axiom to the system, a new realizability model is obtained, because a new constant instruction is added. And when a new instruction is added also the notion of execution has to be extended.

Consider, for example, the *axiom scheme of countable choice*, according to which every countable family of non-empty sets has a choice function. Written in the language of set theory it takes the form:

$$\forall x \in \mathbb{N} \exists y \in S.A(x,y) \rightarrow \exists f \in S^{\mathbb{N}} \forall x \in \mathbb{N}.A(x,f(x)))$$

When we want to add it to second-order arithmetic $\mathbf{PA}^2$, we can simply write:

$$(\mathbf{ACC}) \quad \forall x \exists Y A(x,Y) \to \exists Z \forall x A(x, Z(x))$$

where Y is a k-ary second-order variable, Z a $k+1$-ary second-order variable, and $A(x,Y)$ is any arbitrary formula not containing Z free.

The system $\mathbf{PA}^2 + \mathbf{ACC}$ is a theory adequate enough to formalize analysis. In order to realize **ACC**, and thus construct a realizability model for this theory, a new instruction χ has to be introduced, behaving in the following way (see Krivine 2003, p. 271):

$$\chi \quad \star \quad t \cdot \pi \quad \rightsquigarrow \quad t \quad \star \quad \underline{n_t} \cdot \pi$$

where $\underline{n_t}$ is the Church numeral[32] corresponding to the natural number n_t, which is the number that has been associated to the term t by a (not necessarily recursive) enumeration of the set of closed terms. When this enumeration it is a recursive one, then χ can be implemented by means of the `quote` instruction of LISP.[33]

The introduction of χ induces a modification on program execution, since a new evaluation clause has to be considered, and new contexts of evaluation can be created by exploiting the function enumerating closed terms. But in order to have a full characterization of the execution, it has to be checked whether the new evaluation clause remains compatible with previously defined program transformations, in particular with β-reduction (which corresponds to the reduction defined by the pop and push rules). Actually, this is not the case for the χ operator (see Krivine 2003, p. 271). This means that χ leads up to differentiate terms that otherwise would have been computationally identified. Hence, the identity criterion for computational entities

[32] A Church numeral is a representation in pure λ-calculus of natural numbers, such that a given natural number n corresponds to the λ-term

$$\lambda f \lambda x \underbrace{(f) \ldots (f)}_{n \text{ times}} x$$

For more details see Sørensen & Urzyczyn (2006, p. 20).

[33] Notice that χ does not directly realize **ACC**. What can be proved instead is that there exists a function $F : \mathbb{N}^{k+2} \to \wp(\Sigma)$, with $\wp(\Sigma)$ the power set of the set of stacks Σ, such that:

$$\chi \Vdash \forall x (\forall y (Nat(y) \to A(x, F(x,y))) \to \forall Y (A(x,Y)))$$

where $Nat(y) \equiv \forall X (X(0) \wedge \forall x (X(x) \to X(s(x))) \to X(y))$. It is then easy to show that the term $\lambda z(z)\chi$ realizes what can be called the *intuitionistic countable choice axiom*:

$$(\mathbf{IACC}) \quad \exists U \forall x (\forall y (Nat(y) \to A(x, U(x,y))) \to \forall Y A(x,Y))$$

where Y is a k-ary second-order variable, U a $k+2$-ary second-order variable, and $A(x,Y)$ is any arbitrary formula not containing U free. In order to realize **ACC** it is sufficient to show that **ACC** can be obtained from **IACC** by means of i) logical equivalences, ii) the least number principle, and iii) the principle of extensionality for functions (see Miquel 2009b, §§8.1, 8.2.). It is by performing these deductive steps that an essential appeal to classical logic is made.

not only rests on their behavior – rather than on some pre-fixed features, like their nature or form –, but it also strictly depends from the situations in which this behavior is manifested. More precisely, when new instructions are introduced, and the context of computation changes, the behavior of terms could change as well.

From a philosophical point of view, this phenomenon seems to support a sort of *anti-essentialist* point of view, according to which an entity is recognized to belong to a certain category of objects not because it possess a fixed set of defining characteristic properties, but because we can use it for performing certain kind of operations. From the technical point of view, we know instead that the Curry-Howard correspondence establishes a precise connection between the notion of β-reduction and that of proof-normalization. The incompatibility of the β-reduction with certain kinds of instructions suggests then that the proof-normalization does not play any central role within realizability framework. But since the proof-normalization is at the core of the possibility of obtaining canonical proofs (see p. 7), this seems to represent a conclusive evidence in favor of the idea that Krivine's realizability is conceptually distinct from Dummett's verificationism.[34]

However, this is not a completely astonishing situation. From the verificationist point of view, assigning a semantic value to proper axioms is a discouraging task, because assigning a set of canonical proofs to proper axioms is a sort of a counter sense, since by definition proper axioms are sentences which are accepted (as true) without any specific proofs to be exhibited. On the contrary, in Krivine's account, the meaning of a proper mathematical axiom is not given on the basis of its inferential behavior, but on the basis of its computational behavior. The latter not being defined in absolute terms, but with reference to a given set of contexts, that is, to those situations which oppose to the axiom and try to falsify it. The idea is thus to identify those entities which realize the axiom in spite of all possible attempts made to refute it. And even if by definition those cannot be proofs, they remains nonetheless accessible to human agent, since they correspond to operations consisting in an algorithmic manipulation of a given set of syntactical objects, where the latter represent the context of evaluation.

Certainly, it could be objected that it would be possible to transform the mathematical axioms into some kind of inference rules, as we have done for the Peirce law in the previous section. The problem is that in order to render this transformation faithful with respect to Krivine's realizability we have to show that the so-obtained inference rules possess a computational content, and this is not a trivial question. For example, when axioms are transformed into inference rules following the method proposed by Negri & von

[34] A similar kind of blindness with respect to proof-structure is advocated by Kreisel (1951, pp. 155-156, note 1) when he compares his unwinding program with Brouwer's constructivism.

Plato (2001, § 6; 2011), this seems not to be possible. On the contrary, when axioms are transformed into rewrite rules (acting on the formulas involved in the logical inference rules), as proposed by Dowek et al. (2003), it seems to be possible to preserve the computational content (see Dowek & Werner 2005; Dowek & Miquel 2007). However, rewrite rules are not inference rules, and thus it is not clear to which extent this approach is compatible with Dummett's verificationist.[35] We must then conclude that we lack a general method for assigning a verification interpretation to Krivine's realizability, when it is applied to mathematical theories going beyond pure logic.

6 Conclusion

We have exposed how the compatibility of Krivine's classical realizability with Dummett's verificationism can be obtained only when it is possible to give an inferential treatment of Krivine's computational clauses. This seems to be the case for pure classical logic, as shown in § 4, but not for proper mathematical theories. Krivine's approach allows indeed to give a computational meaning to proper mathematical axioms by assigning them a specific program instruction, while Dummett's verificationism seems not to be a suitable framework for dealing with proper axioms, since axioms are traditionally considered as sentences which are accepted without asking for a proof of them.

However, it should be noticed that the sensibility of the execution operation to the addition of new program instructions seems to prevent classical realizability from being a *uniform* framework for the treatment of axioms. This represents a major difference with respect to the untyped framework presented in § 3.3.2, where the presence of two peculiar ingredients contribute to guarantee the possibility of working with a general and unique notion of execution. On the one hand, the appeal to the *daimon* rule (see p. 19) allows one to define a general notion of axiom, subsuming all kinds of axioms, being them proper axioms or logical ones. On the other had, no real distinction is made between terms and sacks: from the point of view of untyped proof theory, both of these entities correspond to paraproofs, and are thus treated in an homogeneous way.

It would be then interesting to compare these two frameworks in detail, in order to understand whether they belong to different philosophical projects or not. In Naibo et al. (2015) it is claimed, indeed, that the untyped theory framework is not compatible with a verificationist approach, and this claim is based on the treatment of the notion of proper axiom. It would be interesting to understand if the same arguments can be applied also to the case of Krivine's realizability.

[35] For a detailed discussion of these questions see Naibo (2013, in part. chap. 9).

Acknowledgments

We would like to thank Marco Panza for the interest he manifested in our work and Luiz Carlos Pereira for the valuable discussions about the verificationist aspects of classical logic. This work has been partially funded by the French-German ANR-DFG project *BeyondLogic* (ANR-14-FRAL-0002).

References

ARIOLA, Z., HERBELIN, H. & SABRY, A. (2007). "A proof-theoretic foundation of abortive continuations". *Higher-Order and Symbolic Computation*, 20: 403–429.

VAN ATTEN, M. (2014). "The development of intuitionistic logic". In E.N. Zalta (ed.), *The Stanford Encyclopedia of Philosophy*, <http://plato.stanford.edu/archives/spr2014/-entries/intuitionistic-logic-development/>.

BARNES, J. (1993). "Commentary to Aristotle, *Posterior Analytics*", pp. 81–271. Oxford: Clarendon Press.

BONNAY, D. (2002). *Le contenu computationnel des preuves:* No-counterexemple interpretation *et spécification des théorème de l'arithmétique*. Master thesis, Université Paris 7 Paris Diderot

BONNAY, D. (2004). "Preuves et jeux sémantiques". *Philosophia Scientiæ*, 8: 105–123.

BONNAY, D. (2007). "Règles et signification: le point de vue de la logique classique." In J.B. Joinet (ed.), *Logique, Dynamique et Cognition*, pp. 213–231. Paris: Publications de la Sorbonne.

BOYER, J. & SANDU, G. (2012). "Between Proof and Truth." *Synthese*, 187: 821–832.

COQUAND, T. (2014). "Recursive functions and constructive mathematics". In M. Bourdeau and J. Dubucs (eds.), *Constructivity and Computability in Historical and Philosophical Perspective*, p. 159–167. Berlin: Springer.

DOWEK, G. & MIQUEL, A. (2007). "Cut elimination for Zermelo set theory", manuscript.

DOWEK, G. & WERNER, B. (2005). "Arithmetic as a theory modulo". In J. Giesel (ed.), *Term Rewriting and Applications*, Lecture Notes in Computer Science, Vol. 3467, p. 423–437. Berlin: Springer.

DOWEK, G., HARDIN, T. & KIRCHNER, C. (2003). "Theorem proving modulo". *Journal of Automated Reasoning*, 31: 33–72.

DUMMETT, M. (1963). "Realism". In M. Dummett, *Truth and Other Enigmas* (1978), pp. 145–165. London: Duckworth.

DUMMETT, M. (1973). "The philosophical basis of intuitionistic logic." In M. Dummett, *Truth and Other Enigmas* (1978), pp. 215–247. London: Duckworth.

DUMMETT, M. (1977) *Elements of Intuitionism*, Oxford: Clarendon Press.

DUMMETT, M. (1991). *The Logical Basis of Metaphysics*, London: Duckworth.

DUMMETT, M. (1998). "Truth from the constructive standpoint". *Theoria*, 64: 122–138.

FINE, K. (2014). "Truth-maker semantics for intuitionistic logic". *Journal of Philosophical Logic*, 43: 549–577.

FRIEDMAN, H. (1978). "Classically and intuitionistically provably recursive functions". In G.H. Müller and D. Scott (eds.), *Higher Set Theory*, pp. 21–27. Berlin: Springer.

GIANNINI, P. & RONCHI DELIA ROCCA, S. (1988). "Characterization of typings in polymorphic type discipline". In *Logic in Computer Science*, pp. 61–70. IEEE Computer Society Press.

GIRARD, J.-Y. (1989). "Geometry of Interaction I: Interpretation of system F". In R. Ferro, C. Bonotto, S. Valentini and A. Zanardo (eds.), *Logic Colloquium '88*, pp. 221-260. Amsterdam: North-Holland.

GIRARD, J.-Y. (2001). "Locus Solum: From the rules of logic to the logic of rules". *Mathematical Structures in Computer Science*, 11: 301–506.

GOODMAN, N. (1970). "A theory of constructions equivalent to arithmetic". In A. Kino, J. Myhill and R.E. Vesley (eds.), *Intuitionism and Proof Theory*, pp. 101—120. Amsterdam: North-Holland.

GOODMAN, N. (1973a). "The faithfulness of the interpretation of arithmetic in the theory of constructions". *The Journal of Symbolic Logic*, 38: 453—459.

GOODMAN, N. (1973b). "The arithmetic theory of constructions". In A.R.D. Mahtias and H. Rogers (eds.), *Cambridge Summer School in Mathematical Logic*, pp. 274—298. Berlin: Springer.

GUILLERMO, M. & MIQUEL, A. (2014). "Specifying Peirce's law in classical realizability". *Mathematical Structures in Computer Science*, Online First, <DOI: http://dx.doi.org/10.1017/S0960129514000450>.

GRIFFIN, T. (1990). "A formulae-as-types notion of control." In *Proceedings of the 17th ACM Symposium on Principles of Programming Languages*, pp. 47–58. ACM Press.

HEYTING, A. (1962). "After thirty years". In E. Nagel, P. Suppes and A. Tarski (eds.), *Logic, Methodology and Philosophy of Science. Proceedings of the 1960 international congress*, pp. 194–197. Stanford: Stanford University Press.

HILBERT, D. & BERNAYS, P. (1934). *Grundlagen der Mathematik I*, Berlin: Springer.

HINDELY, J.R. & SELDIN, J.P. (2008). *Lambda-Calculus and Combinators: An Introduction*. Cambridge: Cambridge University Press.

HINTIKKA, J. (1996). *The Principles of Mathematics Revisited*, Cambridge: Cambridge University Press.

KLEENE, S. (1945). "On the interpretation of intuitionistic number theory." *Journal of Symbolic Logic*, 10: 109–124.

KLEENE, S. (1973). "Realizability: A retrospective survey". In A.R.D. Mathias and H. Rogers (eds.), *Cambridge Summer School in Mathematical Logic*, pp. 95–112.

Berlin: Springer.

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

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

KREISEL, G. (1960). "Ordinal logics and the characterization of informal notions of proof". In J.A. Todd (ed.), *Proceedings of the International Congress of Mathematicians. Edinburgh, 14-21 August 1958*, pp. 289-299. Cambridge: Cambridge University Press.

KREISEL, G. (1962). "Foundations of intuitionistic logic". In T. Nagel, P. Suppes and A. Tarski (eds.), *Logic, Methodology and Philosophy of Science*, pp. 98–210, Stanford: Stanford University Press.

KREISEL, G. (1965). "Mathematical logic". In T. Saaty (ed.), *Lectures on Modern Mathematics*, Vol. 3, pp. 95—195. New York: Wiley.

KREISEL, G. (1972). "Which number theoretic problems can be solved in recursive progressions on Π^1_1-paths through O?". *The Journal of Symbolic Logic*, 37: 311–334.

KREISEL, G. (1973). "Perspectives in the philosophy of pure mathematics". In P. Suppes, L. Henkin, A. Joja and G. C. Moisil (eds.), *Logic, Methodology and Philosophy of Science IV: Proceedings of the Fourth International Congress for Logic, Methodology and Philosophy of Science, Bucharest, 1971*, pp. 255–277. Amsterdam: North-Holland.

KRIVINE, J.-L. (1994). "Classical logic, storage operators and second-order lambda calculus." *Annals of Pure and Applied Logic*, 68: 53–78.

KRIVINE, J.-L. (2003). "Dependent choice, 'quote' and the clock." *Theoretical Computer Science*, 308: 259–276.

KRIVINE, J.-L. (2009). "Realizability in classical logic". *Panoramas et synthèses*, 27: 197–229.

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

MARTIN-LÖF, P. (1970). *Notes on Constructive Mathematics*. Stockholm: Almqvist & Wiksell.

MARTIN-LÖF, P. (1991). "A path from logic to metaphysics." In G. Corsi and G. Sambin (eds.), *Atti del Congresso "Nuovi problemi della logica e della filosofia della scienza"*, Vol. 2, pp. 141–149. Bologna: CLUEB.

MIQUEL, A. (2009a). *De la formalisation des preuves à l'extraction de programmes*. Habilitation thesis, Université Paris 7 Paris Diderot.

MIQUEL, A. (2009b). "Classical realizability with forcing and the axiom of countable choice", manuscript.

NAIBO, A. (2013). *Le statut dynamique des axiomes. Des preuves aux modèles*. PhD Thesis, Université Paris 1 Panthéon-Sorbonne.

NAIBO, A., PETROLO, M. & SEILLER, T. (2015). "On the computational meaning

of axioms". In A. Napomuceno, O. Pombo and J. Redmond (eds.), *The Dynamic of Knowledge: From Reasoning to Epistemology and Back*. Berlin: Springer. In press.

NEGRI, S. & VON PLATO, J. (2001) *Structural Proof Theory*. Cambridge: Cambridge University Press.

NEGRI, S. & VON PLATO, J. (2011). *Proof Analysis: A Contribution to Hilbert's Last Problem*, Cambridge University Press.

OLIVA, P. & STREICHER, T. (2008). "On Krivine's realizability interpretation of classical second-order arithmetic". *Fundamenta Informaticæ*, 84: 207–220.

PARIGOT, M. (1992). "$\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction." *Logic Programming and Automated Deduction*, 624: 190–201.

PARSONS, C. (1972). "On n-quantifier induction". *The Journal of Symbolic Logic*, 37: 466—482.

PARSONS, C. (2008). *Mathematical Thought and Its Objects*. Cambridge: Cambridge University Press.

VON PLATO, J. (2013). *Elements of Logical Reasoning*. Cambridge: Cambridge University Press.

PRAWITZ, D. (1977). "Meaning and proofs: On the conflict between classical and intuitionistic logic", *Theoria* 43: 2–40.

PRAWITZ, D. (2006). "Meaning approached via proofs." *Synthese*, 148: 507–524.

RIEG, L. (2014). *On Forcing and Classical Realizability*, PhD Thesis, École Normale Supérieure de Lyon, <tel-01061442».

SCHROEDER-HEISTER, P. (1984). "A natural extension of natural deduction". *Journal of Symbolic Logic*, 49: 1284–1300.

SEILLER, T. (2014). "Interaction graphs: Graphings", arXiv:1405.6331.

SELDIN, J. (1989). "Normalization and excluded middle I." *Studia Logica*, 48: 193–217.

SØRENSEN, M.H. & URZYCZYN, P. (2006). *Lectures on the Curry-Howard Isomorphism*. Amsterdam: Elsevier.

SUNDHOLM, G. (1983). "Constructions, proofs and the meaning of logical constants." *Journal of Philosophical Logic*, 12: 151–172.

SUNDHOLM, G. (1986). "Proof theory and meaning". In D. Gabbay and F. Guenthner (eds.), *Handbook of Philosophical Logic*, Vol. 3, pp. 471–506. Dordrecht : Reidel.

SUNDHOLM, G. (1994). "Vestiges of realism". In B. McGuinness and G. Oliveri (eds.), *The Philosophy of Michael Dummett*, pp. 137-165. Dordrecht: Kluwer.

SUNDHOLM, G. (2014). "Constructive recursive functions, Church's thesis, and Brouwer's theory of the creating subject: Afterthoughts on a Parisian joint session". In M. Bourdeau and J. Dubucs (eds.), *Constructivity and Computability in Historical and Philosophical Perspective*, p. 1–35. Berlin: Springer.

TAIT, W.W. (1981). "Finitism". *The Journal of Philosophy*, 78: 524–546.

TIESZEN, R. (1992). "What is a proof?". In M. Detlefsen (ed.), *Proof, Logic and Formalization*, pp. 57–76. London: Routledge.

TROELSTRA, A.S. (1998). "Realizability". In S.R. Buss (ed.), *Handbook of Proof Theory*, pp. 407–473. Amsterdam: Elsevier.

TROELSTRA, A.S. & VAN DALEN, D. (1988). *Constructivism in Mathematics*, Vol. 1. Amsterdam: North-Holland.

ZACH, R. (2015). "Hilbert's program". In E.N. Zalta (ed.), *The Stanford Encyclopedia of Philosophy*, <http://plato.stanford.edu/archives/spr2015/entries/hilbert-program/>.