Introduction to clarithmetic III

Giorgi Japaridze

Abstract

The present paper constructs three new systems of clarithmetic (arithmetic based on computability logic): CLA8, CLA9 and CLA10. System CLA8 is shown to be sound and extensionally complete with respect to PA-provably recursive time computability. This is in the sense that an arithmetical problem \( A \) has a \( \tau \)-time solution for some PA-provably recursive function \( \tau \) iff \( A \) is represented by some theorem of CLA8. System CLA9 is shown to be sound and intensionally complete with respect to constructively PA-provable computability. This is in the sense that a sentence \( X \) is a theorem of CLA9 iff, for some particular machine \( M \), PA proves that \( M \) computes (the problem represented by) \( X \). And system CLA10 is shown to be sound and intensionally complete with respect to not-necessarily-constructively PA-provable computability. This means that a sentence \( X \) is a theorem of CLA10 iff PA proves that \( X \) is computable, even if PA does not “know” of any particular machine \( M \) that computes \( X \).

MSC: primary: 03F50; secondary: 03F30; 03D75; 68Q10; 68T27; 68T30

Keywords: Computability logic; Interactive computation; Game semantics; Peano arithmetic; Constructive theories

1 Introduction

Being a continuation of [3] and [4], this article relies on the terminology, notation, conventions and technical results of its predecessors, with which the reader is assumed to be well familiar. While the present paper is not self-contained, the entire “Introduction to clarithmetic” series is so, and can be read without prior familiarity with computability logic (CoL), which serves as a logical basis for all theories elaborated in the series.

The previously constructed systems CLA4, CLA5, CLA6 and CLA7 form a sequence of incrementally powerful theories, sound and extensionally complete with respect to polynomial time computability, polynomial space computability, elementary recursive time (=space) computability, and primitive recursive time (=space) computability, respectively. Continuing that pattern, the present paper introduces three new, incrementally strong (and stronger than their predecessors) theories CLA8, CLA9 and CLA10.

A natural extreme beyond primitive recursive time is PA-provably recursive time (which can be easily seen to be equivalent to PA-provably recursive space). That means considering PA-provably recursive functions instead of primitive recursive functions as time complexity bounds for computational problems. Our present theory CLA8 turns out to be sound and complete with respect to PA-provably recursive time computability in the same sense as CLA7 is sound and complete with respect to primitive recursive time computability. Remember that, on top of the standard Peano axioms, CLA7 had the single extra-Peano axiom \( \forall x \exists y (y = x + 1) \), and its only nonlogical rule of inference, termed “CLA7-Induction”, was

\[
\begin{align*}
F(0) & \quad F(x) \to F(x+1) \\
\hline
F(x) 
\end{align*}
\]

with no restrictions on \( F(x) \). CLA8 augments CLA7 through the following single additional rule

\[
\begin{align*}
F(x) \sqcup \neg F(x) & \quad \exists x F(x) \\
\hline
\Diamond x F(x) 
\end{align*}
\]

where \( F(x) \) is elementary. A justification for this rule is that, if we know how to decide the predicate \( F(x) \) (the left premise), and we also know that the predicate is true of at least one number (the right premise),
then we can apply the decision procedure to $F(0), F(1), F(2), \ldots$ until a number $n$ is hit such that the procedure finds $F(n)$ true, after which the conclusion $\Box x F(x)$ can be solved by choosing $n$ for $x$ in it.

The story does not end with provably recursive time computability, though. Not all computable problems have recursive (let alone provably so) time complexity bounds. In other words, not all computable problems are **recursive time computable**. An example is

$$\forall x (\exists y p(x, y) \implies \Box y p(x, y)), \tag{2}$$

where $p(x, y)$ is a decidable binary predicate such that the unary predicate $\exists y p(x, y)$ is undecidable (for instance, $p(x, y)$ means “Turing machine $x$ halts within $y$ steps”). Problem (2) is solved by the following effective strategy: Wait till Environment chooses a value $m$ for $x$. After that, for $n = 0, 1, 2, \ldots$, figure out whether $p(m, n)$ is true. If and when you find an $n$ such that $p(m, n)$ is true, choose $n$ for $y$ in the consequent and retire. On the other hand, if there was a recursive bound $\tau$ for the time complexity of a solution $M$ of (2), then the following would be a decision procedure for (the undecidable) $\exists y p(x, y)$: Given an input $m$ (in the role of $y$), run $M$ for $\tau(|m|+1)$ steps in the scenario where Environment chooses $m$ for $x$ at the very beginning of the play of (2), and does not make any further moves. If, during this time, $M$ chooses a number $n$ for $y$ in the consequent of (2) such that $p(m, n)$ is true, accept; otherwise reject.

A next natural step on the road of constructing incrementally strong clarithmetical theories for incrementally weak concepts of computability is to go beyond PA-provably recursive time computability and consider the weaker concept of **constructively PA-provable computability** of (the problem represented by) a sentence $X$. The latter means existence of a machine $M$ such that PA proves that $M$ computes $X$, even if the running time of $M$ is not bounded by any recursive function. System CLA9 turns out to be sound and complete with respect to this sort of computability. That is, a sentence $X$ is provable in CLA9 if and only if it is constructively PA-provably computable. Deductively, CLA9 differs from CLA8 in that, instead of (1), it has the following, stronger, rule:

$$\frac{F(x) \cup \neg F(x)}{\exists x F(x) \rightarrow \Box x F(x)}, \tag{3}$$

where $F(x)$ is elementary. Note that (3) merely “modifies” (1) by changing the status of $\exists x F(x)$ from being a premise of the rule to being an antecedent of the conclusion. A justification for (3) is that, if we know how to decide the predicate $F(x)$, then we can apply the decision procedure to $F(0), F(1), F(2), \ldots$ until (if and when) a number $n$ is hit such that the procedure finds $F(n)$ true, after which the conclusion can be solved by choosing $n$ for $x$ in its consequent. Note that, unlike the earlier-outlined strategy for (1), the present strategy may look for $n$ forever, and thus never make a move. This, however, only happens when $\exists x F(x)$ is false, in which case the conclusion is automatically won.

A further weaker concept of (simply) PA-provable computability is obtained from that of constructively PA-provable computability by dropping the “constructiveness” condition. Namely, PA-provable computability of a sentence $X$ means that PA proves that a machine $M$ solving $X$ exists, yet without necessarily being able to prove “$M$ solves $X$” for any particular machine $M$. An example of a sentence that is PA-provably computable yet not constructively so is $S \cup \neg S$, where $S$ is an elementary sentence with PA $\not\vdash S$ and PA $\not\vdash \neg S$, such as Gödel’s sentence. Let $L$ be a machine that chooses the left disjunct of $S \cup \neg S$ and retires. Similarly, let $R$ be a machine that chooses the right disjunct and retires. One of these two machines is a solution of $S \cup \neg S$, and, of course, PA “knows” this. Yet, PA does not “know” which one of them is a solution (otherwise either $S$ or $\neg S$ would be provable); nor does it have a similar sort of “knowledge” for any other particular machine.

A system sound and complete with respect to PA-provable computability is CLA10. It augments CLA9 through the following additional rule:

$$\frac{\exists x F(x)}{\Box x F(x)} \tag{4}$$

where the premise is an elementary sentence. The admissibility of this rule, simply allowing us to change $\exists x$ to $\Box x$, is obvious in view of the restriction that $\exists x F(x)$ is a sentence (that is, $F(x)$ contains no free variables

---

1An alternative solution: Figure out whether there is a number $n$ with $|n| \leq \tau(|m|+1)$ such that $p(m, n)$ is true. If yes, accept; otherwise reject.
other than \( x \). Indeed, if an \( x \) satisfying \( F(x) \) exists, then it can as well be “computed” (generated), even if we do not know what particular machine “computes” it. As we remember, systems CLA4–CLA7 are sound in a strong, constructive sense. Specifically, there is an effective (in fact, efficient) procedure for extracting solutions from proofs. The same strong form of soundness holds for CLA8 and CLA9 as well. CLA10 stands out as the only system whose soundness theorem is not (and cannot be) constructive. Namely, while CLA10-provability of a sentence \( X \) implies that \( X \) has an algorithmic solution, generally there is no effective (let alone efficient) way to extract a particular solution from a proof of \( X \).

2 Technical preliminaries

All terminology and notation not redefined in this paper has the same meaning as in \([3, 4]\). And all of our old conventions from \([3, 4]\) extend to the present context as well. Namely, as in \([4]\), a “sentence” always means a sentence (closed formula) of the language of CLA4. Similarly for “formula”, unless otherwise specified or suggested by the context. Also, where \( n \) is a natural number, the standard term for \( n \) means 0 followed by \( n \) "s (e.g., 0′′′ is the standard term for 3). We may not always be very careful about terminologically or notationally differentiating between a number and the standard term for it.

As a binary predicate of the variable \( X \) over HPMs and the variable \( X \) over sentences, “\( X \) wins \( X \)” is not arithmetical (is not expressible in the language of PA), for otherwise so would be the truth predicate for elementary sentences: such a sentence is true iff it is won by an HPM that makes no moves. Remember from Section 14.3 of \([3]\) that, on the other hand, for any particular sentence \( X \), the (now unary) predicate “\( X \) wins \( X \)” is arithmetical. Throughout this paper, for each sentence \( X \), we assume the presence of a fixed elementary formula \( \mathbb{W}^X(x) \) naturally representing such a predicate, and when we say something like “PA proves that \( X \) wins \( X \)”, what we precisely mean is that \( \text{PA} \vdash \mathbb{W}^X(\langle X \rangle) \), where \( \langle X \rangle \) is the standard term for the code of \( X \). Or, if we say “PA proves that \( X \) is computable”, what we precisely mean is that \( \text{PA} \vdash \exists x \mathbb{W}^X(x) \).

The soundness proofs found in this article rely on the following lemma. In it, a clarithmetical sequent means a sequent \( E_1, \ldots, E_n \vdash E_0 \) where each \( E_i \) (\( 0 \leq i \leq n \)) is a sentence (of the language of CLA4).

**Lemma 2.1** There is an efficient procedure that takes an arbitrary CLA12-proof of an arbitrary clarithmetical sequent \( E_1, \ldots, E_n \vdash F \) and constructs an \( n \)-ary GHPM \( M \) such that \( \text{PA} \) proves that, for any \( n \)-ary GHPMs \( N_1, \ldots, N_n \), if each \( N_i(\langle N_1 \rangle, \ldots, \langle N_n \rangle) \) (\( 1 \leq i \leq n \)) is a solution of \( E_i \), then \( M(\langle N_1 \rangle, \ldots, \langle N_n \rangle) \) is a solution of \( F \).

**Proof.** If the phrase “PA proves that” is deleted in the present lemma, we get nothing but a weak/simplified version of Theorem 10.1 of \([3]\), which, in turn, is a reproduction of Theorem 10.5 of \([2]\). An analysis of the proof of the latter, combined with some basic experience in working with PA, reveals that the latter can be formalized in PA in the form required by our present lemma. ■

3 CLA8, a theory of PA-provably recursive time computability

The language of each of the theories CLA8, CLA9 and CLA10 introduced in this paper is the same as that of any other system of clarithmetic constructed in the present series of articles — that is, it is an extension of the language of PA through the additional binary connectives \( \land, \lor \) and quantifiers \( \forall, \exists \).

The axiomatization of CLA8 is obtained from that of CLA7 by adding the single new rule of inference, which we call Finite Search (FS):

\[
\frac{\bot(F(x) \lor \neg F(x)) \quad \bot \exists x F(x)}{\bot \exists x F(x)},
\]

where \( F(x) \) is any elementary formula\(^2\)

\(^2\)The forthcoming soundness theorem for CLA8 would just as well go through without the requirement that \( F(x) \) is elementary. But why bother: completeness can be achieved even with the present, restricted, form of FS.
To summarize, the nonlogical axioms of CLA8 are those of PA (Axioms 1-7 from Section 11 of [3]) plus one single additional axiom \( \forall x \exists y (y = x') \) (Axiom 8). There are no logical axioms. The only logical rule of inference is Logical Consequence (LC) as defined in Section 10 of [3], and the only nonlogical rules of inference are FS and CLA7-Induction

\[
\begin{array}{c}
\forall F(0) \quad \forall (F(x) \rightarrow F(x')) \\
\hline
\forall F(x)
\end{array}
\]

We fix

\[ \mathcal{T}(x, y, z, t) \]

as a standard formula of the language of PA saying that Turing machine (encoded by) \( x \), on input \( y \), at computation step \( z \), halts with output \( t \). By an explicit PA-provably recursive function we mean a natural number \( \tau \) such that, where \( \tau^\uparrow \) is the standard term for it, \( \text{PA} \vdash \forall y \exists z \exists t ((\tau^\uparrow, y, z, t) \text{). Context permitting, we usually identify such a number \( \tau \) or term \( \tau^\uparrow \) with the (unary) function computed by the Turing machine encoded by \( \tau \). When \( \tau \) is an explicit PA-provably recursive function and a given HPM \( M \) runs in time \( \tau \), we say that \( \tau \) is an explicit PA-provably recursive bound for the time complexity of \( M \); whenever such a \( \tau \) exists, we say that \( M \) is a PA-provably recursive time machine.

**Theorem 3.1** An arithmetical problem has a PA-provably recursive time solution iff it is provable in CLA8.

Furthermore, there is an efficient procedure that takes an arbitrary extended CLA8-proof of an arbitrary sentence \( X \) and constructs a solution of \( X \) (of \( X^\uparrow \), that is) together with an explicit PA-provably recursive bound for its time complexity.

### 3.1 Proof of the soundness part of Theorem 3.1

Consider any sentence \( X \) with a fixed extended CLA8-proof. Our goal is to construct a PA-provably recursive time HPM \( M \) such that \( M \) wins \( X \) and, furthermore, PA proves that \( M \) wins \( X \). We shall not explicitly address the question of the efficiency of our construction, because it is achieved the same way as in all previous soundness proofs. That is, as in the similar proofs of [4], we will limit ourselves to verifying the pre-“furthermore” part of the theorem.

We proceed by induction on the length of the proof of \( X \). In each case of our induction, we show how to construct the above-mentioned HPM \( M \), together with a recursive function \( \tau \), and present an informal proof of the fact that \( M \) solves \( X \) in time \( \tau \). A reader sufficiently familiar with PA will immediately see that such a proof can be reproduced in PA.

The case of \( X \) being an axiom is simple and is handled in the same way as in the earlier soundness proofs. So is the case of \( X \) being derived by LC, only now it relies on Lemma 27 instead of the earlier relied-upon Theorem 10.1 of [3] (= Theorem 10.5 of [2]). The case of \( X \) being obtained by CLA7-Induction is essentially handled in the way as in the soundness proof for CLA7 found in [4], only with the words “primitive recursive” replaced by “PA-provably recursive”.

So, the only case worth considering is that of \( X \) being derived by FS. Assume \( X \) is (the \( \exists \)-closure) of \( \forall x F(x) \), and thus its premises are (the \( \exists \)-closures of) \( F(x) \cup \neg F(x) \) and \( \exists x F(x) \). By the induction hypothesis, there is an HPM \( \mathcal{N} \) that solves \( F(x) \cup \neg F(x) \). Similarly for the other premise \( \exists x F(x) \) but, since the latter is elementary, its solvability (that is, the solvability of \( \exists x F(x) \)) simply means that \( \forall x F(x) \) is true.

To describe our purported solution \( M \) of \( \forall x F(x) \), assume \( x, \bar{v} \) are exactly the free variables of \( F(x) \), so that \( F(x) \) can be rewritten as \( F(x, \bar{v}) \). For simplicity, we rule out the trivial case of \( F(x) \) having no free occurrences of \( x \). At the beginning, \( M \) waits for Environment to choose constants for the free variables \( \bar{v} \) of \( \forall x F(x, \bar{v}) \). Assume \( \bar{c} \) are the constants chosen for \( \bar{v} \). From now on, we shall write \( F'(x) \) for \( F(x, \bar{c}) \). Further, where \( i \) is a natural number, we shall write \( N_i \) for the machine that works just like \( \mathcal{N} \) does in the scenario where the adversary, at the beginning of the play, has chosen the constant \( i \) for the variable \( x \) and the constants \( \bar{c} \) for the variables \( \bar{v} \). So, \( N_i \) wins the constant game \( F'(i) \cup \neg F'(i) \).

Environment’s initial moves bring the original \( \forall \exists x F(x) \) down to \( \exists x F'(x) \). The goal of \( M \) now is to win \( \exists x F'(x) \). It achieves this goal by creating a record \( i \), initializing it to 0, and then acting as prescribed by the following procedure:
3.3 The intensional strength of CLA8

Procedure LOOP: Simulate \( N_i \) until it chooses one of the two \( \sqcup \)-disjuncts of \( F'(i) \sqcup \neg F'(i) \). If the right disjunct is chosen, increment \( i \) by 1 and repeat LOOP. Otherwise, if the left disjunct is chosen, specify \( x \) as \( i \) in the (real) play of \( \sqcup x F'(x) \), and retire.

Since \( \exists x F'(x) \) is true, sooner or later the above procedure hits an \( i \) such that the simulated \( N_i \) chooses the left disjunct of \( F'(i) \sqcup \neg F'(i) \), meaning that \( F'(i) \) is true. This guarantees that \( \mathcal{M} \) wins. A bound \( \tau \) for the time complexity of \( \mathcal{M} \) is computed by a Turing machine that follows the work of \( \mathcal{M} \) and counts the steps that it performs before making a move. Such a function \( \tau \) is PA-provably recursive because, as already noted, our entire argument can be reproduced in PA.

3.2 Proof of the completeness part of Theorem 3.1

Consider an arbitrary sentence \( X \), an arbitrary HPM \( \mathcal{X} \), and an arbitrary explicit PA-provably recursive function \( \chi \) such that \( \mathcal{X} \) is a \( \chi \) time solution of \( X \). Let \( \langle \chi \rangle \) be the standard term for the code of \( \chi \). We fix some enumeration of pairs of natural numbers and, where \( a \) is a natural number, denote the first (resp. second) element of the \( a \)th pair by \( (a)_1 \) (resp. \( (a)_2 \)). We treat \( (x)_1 \), \( (x)_2 \) as pseudoterms and assume that the above enumeration is “standard enough”, so that the functions \( (x)_1 \) and \( (x)_2 \) are primitive recursive and PA proves

\[
\forall \exists z_1 \exists z_2 \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \exists \3.3 The intensional strength of CLA8

We say that PA constructively proves the PA-provably recursive time computability of a sentence \( X \) iff, for some particular HPM \( \mathcal{X} \) and some particular explicit PA-provably recursive function \( \chi \), PA proves that \( \mathcal{X} \) is a \( \chi \)-time solution of \( X \).

The following theorem holds for virtually the same reasons as the similar Theorem 16.2 of [3] for CLA4 or Theorem 11.2 of [4] for CLA5, CLA6 and CLA7:

**Theorem 3.2** Let \( X \) be any sentence such that PA constructively proves the PA-provably recursive time computability of \( X \). Then CLA8 proves \( X \).

**Remark 3.3** From our soundness proof for CLA8 it is immediately clear that the above theorem, in fact, holds in the stronger, “if and only if” form. The same can be seen to be the case for Theorem 16.2 of [3] and Theorem 11.2 of [4].
4 CLA9, a theory of constructively PA-provable computability

Deductively, CLA9 only differs from CLA8 in that, instead of the FS rule of the latter, CLA9 has the following rule, which we call Infinite Search (IS):

\[ \exists x (F(x) \lor \neg F(x)) \]

\[ \exists x \neg F(x) \rightarrow \exists x F(x) \]

where \( F(x) \) is any elementary formula.

Let \( X \) be a sentence. We say that PA constructively proves the computability of \( X \) iff, for some HPM \( \mathcal{X} \), PA proves that \( \mathcal{X} \) wins \( X \).

Notice that, while our earlier defined concepts of polynomial, elementary recursive, primitive recursive and PA-provably recursive time (or space) computabilities are extensional in their nature, the concept of constructively PA-provable computability is intensional. To be precise, the former are properties of computational problems while the latter is a property of sentences. An extensional version of this concept could be defined by saying that a computational problem \( A \) is constructively PA-provably computable in the extensional sense iff there is a sentence \( X \) with \( X^1 = A \) (\( X \) “represents” \( A \)) such that \( X \) is constructively PA-provably computable in the intensional sense. The forthcoming Theorem 4.1 can be easily seen to continue to hold after replacing the intensional concept of constructively PA-provable computability by its extensional counterpart. Yet, doing so would significantly and unnecessarily weaken the theorem. This is the reason why, in the present context, we have opted for only considering the intensional concept. A similar comment applies to the concept of (not-necessarily-constructively) PA-provable computability defined later in Section 5.

**Theorem 4.1** For any sentence \( X \), CLA9 proves \( X \) iff PA constructively proves the computability of \( X \).

Furthermore, there is an efficient procedure that takes an arbitrary extended CLA9-proof of an arbitrary sentence \( X \) and constructs an HPM \( \mathcal{X} \) such that PA proves that \( \mathcal{X} \) is a solution of \( X \) (of \( X^1 \), that is).

4.1 Proof of the soundness part of Theorem 4.1

As in the earlier soundness proofs, we will limit ourselves to verifying the pre-“furthermore” part. Consider any sentence \( X \) with a fixed extended CLA9-proof. We proceed by induction on the length of the proof of \( X \). In each case of the induction, we (show how to) construct an HPM \( \mathcal{M} \) and present an informal proof of the fact that \( \mathcal{M} \) solves \( X \). It will be immediately clear that such a proof can be reproduced in PA.

The case of \( X \) being an axiom is simple and is handled as in the soundness proof for CLA8. So is the case of \( X \) being derived by LC. So is the case of \( X \) being derived by CLA7-Induction — namely, it is essentially handled in the same way as in the soundness proof for CLA7 found in [4], but is, in fact, simpler, because we no longer need to care about complexity.

Finally, assume \( X \) is (the \( \exists \)-closure of) \( \exists x F(x) \rightarrow \exists x F'(x) \), obtained by IS from (the \( \exists \)-closure of) \( F(x) \lor \neg F(x) \). By the induction hypothesis, we know how to construct a solution \( \mathcal{N} \) of \( F(x) \lor \neg F(x) \). Here we assume that \( x \) occurs free in \( F(x) \) (otherwise the case is straightforward), and that \( \bar{v} \) are all the additional free variables of \( F(x) \). So, \( F(x) \) can be rewritten as \( F(x, \bar{v}) \). We let \( \mathcal{M} \) — the purported solution of \( \exists x F(x) \rightarrow \exists x F'(x) \) — be a machine that, at the beginning of the play, waits till Environment chooses constants \( \bar{c} \) for the variables \( \bar{v} \). Let \( F'(x) \) stand for \( F(x, \bar{c}) \) and, for each natural number \( i \), let \( \mathcal{N}_i \) be an HPM that plays \( F(x) \lor \neg F(x) \) just as \( \mathcal{N} \) does in the scenario where, at the very beginning of the play, the adversary chose the constant \( i \) for the variable \( x \) and the constants \( \bar{c} \) for the variables \( \bar{v} \).

After the above event of Environment having chosen constants for all free variables, thus having brought the original game \( \exists x F(x) \lor \exists x F'(x) \) down to \( \exists x F'(x) \rightarrow \exists x F'(x) \), \( \mathcal{M} \) creates a record \( i \), initializes it to 0, and then acts as prescribed by the following procedure:

**Procedure LOOP**: Simulate \( \mathcal{N}_i \) until it chooses one of the two \( \lor \)-disjuncts of \( F'(i) \lor \neg F'(i) \). If the right disjunct is chosen, increment \( i \) by 1 and repeat LOOP. Otherwise, if the left disjunct is chosen, specify \( x \) as \( i \) in the consequent of \( \exists x F'(x) \rightarrow \exists x F'(x) \), and retire.

\[^3\]On the other hand, Theorem 5.1 and similar theorems from [3][4] — namely, their completeness parts — would fail with the intensional counterparts of the corresponding extensional concepts of computability.
If \(\exists x F'(x)\) is false, \(\exists x F'(x) \rightarrow \sqcup x F'(x)\) is (automatically) won by \(M\). And if \(\exists x F'(x)\) is true, sooner or later the above procedure hits an \(i\) such that the simulated \(N_i\) chooses the left disjunct of \(F'(i) \sqcup \neg F'(i)\); then, again, \(M\) wins, because it brings the consequent of \(\exists x F'(x) \rightarrow \sqcup x F'(x)\) down to the true \(F'(i)\).

### 4.2 Proof of the completeness part of Theorem 4.1

Consider an arbitrary sentence \(X\) and an arbitrary HPM \(\mathcal{X}\). Let \(L\) be an elementary sentence saying “\(\mathcal{X}\) does not win \(X\)”, so that \(\neg L\) says “\(\mathcal{X}\) wins \(X\)”. Our intermediate — and main — goal is to show that

\[
\text{CLA9} \vdash \neg L \rightarrow X \quad \text{(Lemma 4.3, from which the desired completeness of CLA9 follows almost immediately.}
\]

For the purposes of the subsequent section, it is important to note that, at this point, we are not making any assumptions about \(X\) and \(\mathcal{X}\). In particular, we are not assuming that \(\mathcal{X}\) wins \(X\), let alone that \(\text{PA}\) proves so; such an assumption will be made only later, after Lemma 4.4 is proven.

By a **computation history** we shall mean a finite initial segment of some computation branch of \(\mathcal{X}\). The way we encode configurations of \(\mathcal{X}\) is described in Appendix A.1 of [3]. That encoding extends to computation histories as finite sequences of configurations in a standard way. For readability, we will often identify configuration histories with their codes and say something like “\(a\) is a computation history” when what is precisely meant is “\(a\) is the code of a computation history”. We may further identify such an \(a\) with the standard term for it.

Remember from Section 14.5 of [3] that, where \(\Phi\) is a legal position of \(X\), the **yield** of \(\Phi\) means the game \(\langle \Phi \rangle X\). So, the type of the “yield” function in [3] is \(\{\text{positions}\} \times \{\text{games}\} \rightarrow \{\text{games}\}\). Here, for safety, we need an “intensional” version of this concept/function, whose type is \(\{\text{positions}\} \times \{\text{CL12-formulas}\} \rightarrow \{\text{CL12-formulas}\}\). Namely, let \(F\) be a closed CL12-formula containing no predicate letters other than \(=\), and no function letters other than \(', +, \times\) (\(F\) is not necessarily a sentence of the language of CLA9 because it may contain constants other than \(0\)). And let \(\Phi\) be a legal position of \(F\) (of \(F\)'s, that is). Then, in the context of \(F\), the intensional version of the yield of \(\Phi\), denoted by \(\langle \Phi \rangle!F\), is defined inductively as follows:

- For any nonempty legal position \(\langle \lambda, \Psi \rangle\) of \(F\) (where \(\lambda\) is a labmove and \(\Psi\) is a sequence of labmoves):
  - If \(\lambda\) signifies a choice of a component \(G_i\) in an occurrence of a subformula \(G_0 \sqcup G_1\) or \(G_0 \sqcap G_1\) of \(F\), and \(F'\) is the result of replacing that occurrence by \(G_i\) in \(F\), then \(\langle \lambda, \Psi \rangle!F = \langle \Psi \rangle!F'\). For any nonempty legal position \(\langle \lambda, \Psi \rangle\) of \(F\) (where \(\lambda\) is a labmove and \(\Psi\) is a sequence of labmoves):
  - If \(\lambda\) signifies a choice of a component \(c\) for a variable \(x\) in an occurrence of a subformula \(\sqcup x G(x)\) or \(\sqcap x G(x)\) of \(F\), and \(F'\) is the result of replacing that occurrence by \(G(c)\) in \(F\), then \(\langle \lambda, \Psi \rangle!F = \langle \Psi \rangle!F'\).

Let \(E(\vec{s})\) be a formula all of whose free variables are among \(\vec{s}\) (but not necessarily vice versa), and let \(z\) be a variable not among \(\vec{s}\). We will write \(\bar{E}(z, \vec{s})\) to denote an elementary formula whose free variables are \(z\) and those of \(E(\vec{s})\), and which is a natural arithmetization of the predicate that, for any constants \(a, \bar{c}\) in the roles of \(z, \vec{s}\), holds (that is, \(\bar{E}(a, \bar{c})\) is true) if \(a\) is a computation history and, where \(\Phi\) is the position spelled on the run tape of the last configuration of that history, \(\Phi\) is a legal position of \(X\) with \(\langle \Phi \rangle X = E(\vec{c})\).

Let \(y\) be a variable and \(E\) be a formula not containing \(y\). As in Section 14.6 of [3], we say that a formula \(H\) is a \((\bot, y)\)-development of \(E\) iff \(H\) is the result of replacing in \(E\):

- either a surface occurrence of a subformula \(F_0 \sqcap F_1\) by \(F_i\) \((i = 0\) or \(i = 1\)),
- or a surface occurrence of a subformula \(\sqcup x F(x)\) by \(F(y)\).

\((\top, y)\)-development is defined in the same way, only with \(\sqcup, \sqcap\) instead of \(\sqcap, \sqcup\).

**Lemma 4.2** Assume \(E(\vec{s})\) is a formula all of whose free variables are among \(\vec{s}\), and \(y, z\) are variables not occurring in \(E(\vec{s})\). Then:

(a) CLA9 proves \(\bar{E}(z, \vec{s}) \sqcup \neg \bar{E}(z, \vec{s})\).

(b) For every \((\bot, y)\)-development \(H_i(y, \vec{s})\) of \(E(\vec{s})\), CLA9 proves \(\bar{E}(z, \vec{s}) \rightarrow \sqcup u \bar{H}_1(u, y, \vec{s})\).

(c) Where \(H_1(y, \vec{s}), \ldots, H_n(y, \vec{s})\) are all of the \((\top, y)\)-developments of \(E(\vec{s})\), CLA9 proves

\[
\neg \sqcup \land \bar{E}(z, \vec{s}) \land \neg \|E(\vec{s})\| \rightarrow \sqcup \sqcup \sqcup \sqcup \bar{H}_1(u, y, \vec{s}) \sqcup \ldots \sqcup \sqcup \sqcup \bar{H}_n(u, y, \vec{s}).
\]
Proof. For clause (a), observe that the predicate $E(z, \bar{s})$ is primitive recursive and, furthermore, PA obviously constructively proves the primitive recursive time computability of $E(z, \bar{s}) \cup \neg E(z, \bar{s})$. Hence, by Theorem 11.2 of [4], CLA7 proves $E(z, \bar{s}) \cup \neg E(z, \bar{s})$. Therefore, of course, so does CLA9.

For clause (b), consider any $(\perp, y)$-development $H_i(y, \bar{s})$ of $E(\bar{s})$ and assume $E(z, \bar{s})$. For any value of $y$, $H_i(y, \bar{s})$ is the game to which $E(\bar{s})$ is brought down by a certain labmove $\perp \alpha$. To solve $\sqcup uH_i(u, y, \bar{s})$ — i.e., to make $H_i(u, y, \bar{s})$ true — we can choose $u$ to be the result of appending such a labmove $\perp \alpha$ to the run-tape content of the last configuration of $z$. This argument can obviously be reproduced in CLA9.

For clause (c), assume $\neg \perp \sqcup E(z, \bar{s})$ and $\neg \| E(\bar{s}) \|$. Consider the scenario where Environment does not move in any configuration starting from the last configuration of $z$. If $\chi$ does not move either, it can be seen to lose, because $E(\bar{s})$ is the final position reached in the play and its elementarization $\| E(\bar{s}) \|$, by our assumption, is false. But our assumption $\neg \perp \sqcup$ means that $\chi$ cannot lose. So, sooner or later, $\chi$ will move. We can trace, within CLA9, the work of $\chi$ (compute the continuously lengthening computation history) for as long as it takes, until $\chi$ makes a move $\alpha$. This $\alpha$ must be legal. Then it brings $E(\bar{s})$ down to one of (the instances of) its $(\top, y)$-developments $H_i(y, \bar{s})$ for a certain value of $y$. We choose the corresponding $\sqcup$-disjunct $\sqcup u\sqcup yH_i(u, y, \bar{s})$ in the consequent of (10); in it, we specify $u$ as the computation history that we have traced, and specify $y$ as the above-mentioned “certain value of $y$”, which can be easily found from the last configuration of the history. Again, the entire argument can be reproduced in CLA9.

Lemma 4.3 Assume $z, \bar{s}$ are pairwise distinct variables, and $E(\bar{s})$ is a formula all of whose free variables are among $\bar{s}$. Then

CLA9 $\vdash \neg \perp \sqcup E(z, \bar{s}) \rightarrow E(\bar{s})$.

Proof. We prove this lemma by induction on the complexity of $E(\bar{s})$. Pick a fresh variable $y$. By the induction hypothesis, for any $(\perp, y)$- or $(\top, y)$-development $H_i(y, \bar{s})$ of $E(\bar{s})$ (if there are any), CLA9 proves

$$\neg \perp \sqcup H_i(y, \bar{s}) \rightarrow H_i(y, \bar{s}).$$

(11)

Argue in CLA9 to justify $\neg \perp \sqcup E(z, \bar{s}) \rightarrow E(\bar{s})$. Consider any values (constants) $b$ and $\bar{a}$ chosen by Environment for $z$ and $\bar{s}$, respectively. Throughout the rest of this argument, assume both $\neg \perp \sqcup$ and $E(b, \bar{a})$ are true (otherwise we win). We need to see how to win $E(\bar{a})$.

To solve $E(\bar{a})$, we bring the resource (11) down to

$$\neg \perp \sqcup E(b, \bar{a}) \land \neg \| E(\bar{a}) \| \rightarrow \sqcup u\sqcup yH_1(u, y, \bar{a}) \sqcup \ldots \sqcup \sqcup u\sqcup yH_n(u, y, \bar{a}).$$

Since the $\neg \perp$ and $E(b, \bar{a})$ components of the above are true, in fact, the following resource is at our disposal:

$$\neg \| E(\bar{a}) \| \rightarrow \sqcup u\sqcup yH_1(u, y, \bar{a}) \sqcup \ldots \sqcup \sqcup u\sqcup yH_n(u, y, \bar{a}).$$

(12)

We wait till one of the following two events takes place:

1. Event 1: Environment makes a move $c$ in $E(\bar{a})$. We may assume that this move is legal. Then, for one of the $(\perp, y)$-developments $H_i(y, \bar{s})$ of $E(\bar{s})$ and some constant $c$, the labmove $\perp \alpha$ brings $E(\bar{a})$ down to $H_i(c, \bar{a})$.
2. Event 2: The provider of (11) brings it down to

$$\neg \| E(\bar{a}) \| \rightarrow H_i(d, c, \bar{a}).$$

(13)

for one of $i \in \{1, \ldots, n\}$ and some constants $d, c$. Using clause (a) of Lemma 12 we check whether $H_i(d, c, \bar{a})$ is true. If not, $\| E(\bar{a}) \|$ is guaranteed to be true (otherwise [13] would be lost by its provider), and we continue waiting for Event 1 to occur (never again react to Event 2): if such an event never occurs, the truth of $\| E(\bar{a}) \|$ obviously means that we win. Suppose now $H_i(d, c, \bar{a})$ is true. Then, just as in the case of Event 1, (11)

\footnote{Here, unlike the earlier followed practice, for safety, we are reluctant to use the names $z$ and $\bar{s}$ for those constants.}
provides a way to win $H_i(c,a)$. We make the move that brings $E(a)$ down to $H_i(c,a)$, and follow the just-mentioned way.

If neither event happens, then $\|E(a)\|$ is true (otherwise (12) would be lost by its provider) and, again, we win. ■

**Lemma 4.4** $\text{CLA9} \vdash \neg L \rightarrow X$.

**Proof.** Let $a$ be the code of the empty computation history, and $\hat{a}$ be the standard term for $a$. Of course, $\text{PA}$ and hence $\text{CLA9}$ proves $\tilde{X}(\hat{a})$. In view of Fact 12.6 of [3], $\text{CLA9}$ proves $\sqcup z (\neg L \land \tilde{X}(z) \rightarrow X)$. These three can be seen to imply $\neg L \rightarrow X$ by LC. ■

Now we are ready to claim the target result of this section. Suppose $\text{PA}$ constructively proves the computability of $X$. We may assume that it proves “$X$ solves $X$” for the earlier-fixed (yet arbitrary) HPM $X$. In other words, $\text{PA} \vdash \neg L$. Hence $\text{CLA9} \vdash \neg L$. Then, in view of Lemma 4.4, $\text{CLA9} \vdash X$, as desired.

5 CLA10, a theory of PA-provable computability

CLA10 only differs from CLA9 in that it has the following additional rule of inference, called Constructivization:

$$\frac{\exists x F(x)}{\Box x F(x)}$$

where $F(x)$ is any elementary formula containing no free variables other than $x$.

Let $X$ be a sentence. We say that $\text{PA}$ proves the computability of $X$ iff $\text{PA}$ proves that there is ($\exists$) an HPM that wins $X$.

**Theorem 5.1** For any sentence $X$, $\text{CLA10}$ proves $X$ iff $\text{PA}$ proves the computability of $X$.

5.1 Proof of the soundness part of Theorem 5.1

As always, this part of the theorem is verified by induction on the length of the proof of $X$. The cases where $X$ is an axiom, or is derived by LC or IS, are handled in the same way as in the soundness proof for CLA9. Here we shall only look at the case of $X$ being derived by Constructivization. So, assume $X$ is $\sqcup x F(x)$, where $F(x)$ is an elementary formula not containing any free variables other than $x$, and $\sqcup x F(x)$ is derived from $\exists x F(x)$ by Constructivization. By the induction hypothesis, $\text{PA}$ proves that $\exists x F(x)$ is “computable” which, as $F(x)$ is elementary, simply means that $\exists x F(x)$ is true. So, $\text{PA} \vdash \exists x F(x)$.

Argue in $\text{PA}$. Since $\exists x F(x)$ is true, there is a number $a$ such that $F(a)$ is true. Then the target $\sqcup x F(x)$ is solved by an HPM $M$ that makes $a$ as its only move in the play and retires. We ($\text{PA}$, that is) cannot name such an $M$ because we do not know what number $a$ exactly is; yet we know that $M$ exists. In other words, we know that $X$ is computable.

5.2 Proof of the completeness part of Theorem 5.1

Consider an arbitrary sentence $X$. Let $\ell(x)$ be a natural formalization of the predicate “$x$ is (the code of) an HPM which does not win $X$”. Remember that, in Section 4.2, $X$ was a fixed yet arbitrary HPM. And note that the sentence $L$ of Section 4.2 was nothing but what we can now write as $\ell(\tilde{X})$. So, Lemma 4.4 can now be re-stated as

$$\text{CLA9} \vdash \neg \ell(\tilde{X}) \rightarrow X.$$  (14)

5 Just as in the case of CLA8, the requirement that $F(x)$ is elementary can be dropped here without affecting the soundness of the system.
Further observe that, while $X$ was fixed in Section 4.2, the proof of Lemma 4.4 given there goes through with $X$ as a ($\sqcup$-quantified) variable; more precisely, (14), in fact, holds in the following, stronger form:

$$\text{CLA9} \vdash \forall x (\neg L(x) \to X).$$

(15)

By LC, (15) implies

$$\text{CLA9} \vdash \exists x \neg L(x) \to X.$$

(16)

Assume now $\text{PA}$ proves that $X$ is computable. In other words, $\text{PA} \vdash \exists x \neg L(x)$. Then, by Constructivization, $\text{CLA10} \vdash \exists x \neg L(x)$. But then, in view of (16), $\text{CLA10} \vdash X$, as desired.

References

[1] G. Japaridze. *Towards applied theories based on computability logic*. Journal of Symbolic Logic 75 (2010), pp. 565-601.

[2] G. Japaridze. *A logical basis for constructive systems*. Journal of Logic and Computation (under review). Preprint is available at [http://arxiv.org/abs/1003.0425](http://arxiv.org/abs/1003.0425)

[3] G. Japaridze. *Introduction to clarithmetic I*. This journal (under review). Preprint is available at [http://arxiv.org/abs/1003.4719](http://arxiv.org/abs/1003.4719)

[4] G. Japaridze. *Introduction to clarithmetic II*. This journal (under review). Preprint is available at [http://arxiv.org/abs/1004.3236](http://arxiv.org/abs/1004.3236)