Introduction to clarithmetic II

Giorgi Japaridze

Abstract

The earlier paper “Introduction to clarithmetic I” constructed an axiomatic system of arithmetic based on computability logic, and proved its soundness and extensional completeness with respect to polynomial time computability. The present paper elaborates three additional sound and complete systems in the same style and sense: one for polynomial space computability, one for elementary recursive time (and/or space) computability, and one for primitive recursive time (and/or space) computability.

MSC: primary: 03F50; secondary: 03F30; 03D75; 03D15; 68Q10; 68T27; 68T30

Keywords: Computability logic; Interactive computation; Implicit computational complexity; Game semantics; Peano arithmetic; Bounded arithmetic; Constructive logics

1 Introduction

Being a continuation of [8], this article fully and heavily relies on the terminology, notation, conventions and technical results of its predecessor, with which the reader is assumed to be well familiar (the good news, however, is that, [8], in turn, is self-contained).

Remember, from [8], the system CLA4 of arithmetic, both semantically and syntactically based on computability logic (CoL). Its language was that of Peano arithmetic (PA) augmented with the choice conjunction ⊓, choice disjunction ⊔, choice universal quantifier ⊓ and choice existential quantifier ⊔. On top of the standard Peano axioms, CLA4 had two extra-Peano axioms: ⊓x⊔y(y=x+1) and ⊓x⊔y(y=2x), one saying that the function x+1 is computable, and the other saying the same about the function 2x. The only logical rule of CLA4 was Logical Consequence (LC), meaning that the logical basis for the system was the (sound and complete) fragment CL12 of CoL. And the only nonlogical rule of inference was the induction rule

\[
\frac{F(0) \quad F(x) \rightarrow F(2x) \quad F(x) \rightarrow F(2x+1)}{F(x)}
\]

with \(F(x)\) — that is, its choice quantifiers \(\sqcap, \sqcup\) — required to be polynomially bounded. The system was proven in [8] to be sound and extensionally (representationally) complete with respect to polynomial time computability.

The present paper constructs three new CL12-based systems: CLA5, CLA6, CLA7 and proves their soundness and extensional completeness with respect to polynomial space computability, elementary recursive time (and/or space) computability and primitive recursive time (and/or space) computability, respectively. While CLA4 was already simple enough, the above three systems are even more so. All of them only need \(\sqcap x \sqcup y (y=x+1)\) as a single extra-Peano axiom. As before, the only logical rule is LC. And the induction rule (the only nonlogical rule) of each of these systems is

\[
\frac{F(0) \quad F(x) \rightarrow F(x+1)}{F(x)}
\]

The three systems differ from each other only in what (if any) conditions are imposed on the formula \(F(x)\) of induction. In CLA5, as in CLA4, \(F(x)\) is required to be polynomially bounded. CLA6 relaxes this requirement and allows \(F(x)\) to be an exponentially bounded formula. CLA7 takes this trend towards relaxation to an extreme and imposes no restrictions on \(F(x)\) whatsoever. This way, unlike CLA4, CLA5 and CLA6, theory CLA7 is no longer in the realm of bounded arithmetics.
The simplicity and elegance of these systems is additional evidence for the naturalness and productiveness of the idea of basing complexity-oriented systems and bounded arithmetic in particular on CoL instead of classical logic, even when one is only concerned with functions rather than the more general class of all interactive computational problems. In \[1\], achieving representational completeness with respect to polynomial space computable functions required considering a second-order extension of classical-logic-based bounded arithmetic (similarly in \[2\] for certain other complexity classes). In our case, on the other hand, a transition from polynomial time (CLA4) to polynomial space (CLA5) in remarkably smooth with no need for any changes in the underlying language or logic, and with only minimal syntactic changes in the nonlogical part (induction rule) of the system. Among the virtues of CoL is that, as a logic, it remains the same regardless of for what purposes (polynomial time computability, polynomial space computability, computability-in-principle, . . .) it is used. CoL does not have variations, but rather has various (conservative) fragments depending on what part of its otherwise very expressive language is considered. The fragment dealt with in the present paper, as well as in its predecessor \[8\], as well as in its even earlier predecessors \[6, 7\], is the same: logic CL12.

1.1 Technical notes

All terminology and notation not redefined in this paper has the same meaning as in \[8\]. And all of our old conventions from \[8\] extend to the present context as well.

Additionally we agree that, in the sequel, 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, in the context of a given play (computation branch) of an HPM \(M\), by the spacecost of a given clock cycle \(c\) we shall mean the number of cells ever visited by the work-tape head of \(M\) by time \(c\). We extend the usage of this term from clock cycles to the corresponding configurations as well.

2 CLA5, a theory of polynomial space computability

The language of theory CLA5 is the same as that of CLA4 — that is, it is an extension of the language of PA through the additional binary connectives \(\land, \lor\) and quantifiers \(\exists, \forall\). And the axiomatization of CLA5 is obtained from that of CLA4 by deleting Axiom 9 (which is now redundant) and replacing the CLA4-Induction rule by the following rule, which we call CLA5-Induction:

\[
\frac{\exists(F(0)) \land (F(x) \rightarrow F(x'))}{\exists(F(x))},
\]

where \(F(x)\) is any polynomially bounded formula. Here we shall say that \(\exists(F(0))\) is the basis of induction, and \(\exists(F(x) \rightarrow F(x'))\) is the inductive step.

To summarize, the nonlogical axioms of CLA5 are those of PA (Axioms 1-7) plus one single additional axiom \(\exists x \exists y (y = x')\) (Axiom 8). There are no logical axioms. The only logical inference rule is Logical Consequence (LC) as defined in Section 10 of \[8\], and the only nonlogical inference rule is CLA5-Induction.

The following fact establishes that the old Axiom 9 of CLA4 would indeed be redundant in CLA5:

**Fact 2.1** CLA5 \(\vdash \exists x \exists y (y = x0)\).

**Proof.** Argue in CLA5. First, by CLA5-Induction on \(x\), we want to show

\[
\exists z (|z| \leq |x| + |y| \land z = x + y).
\]  

(1)

The basis \(\exists z (|z| \leq 0 + |y| \land z = 0 + y)\) is obviously solved by choosing the value of \(y\) for the variable \(z\). To solve the inductive step

\[\exists z (|z| \leq |x| + |y| \land z = x + y) \rightarrow \exists z (|z| \leq |x'| + |y| \land z = x' + y),\]

\(1\)Including what has been termed “intuitionistic computability logic” (studied in \[3, 4, 5\]), contrary to what this name may suggest. Unlike, say, intuitionistic linear logic, which is indeed a variation of (classical) linear logic, intuitionistic computability logic is merely a conservative fragment of CoL, obtained by restricting its logical vocabulary to the choice operators and the ultimate reduction operator.
we wait till Environment selects a value \( a \) for \( z \) in the antecedent. Then, using Axiom 8, we calculate the value \( b \) of \( a' \), and choose \( b \) for \( z \) in the consequent. The resulting position shown below is true by \( \text{PA} \), so we win:

\[
|a| \leq |x| + |y| \land a = x + y \rightarrow |a'| \leq |x'| + |y| \land a' = x' + y.
\]

By LC, from (the \( \sqcap \)-closure of) \( 1 \) we immediately get \( \sqcap x \sqcup y \sqcup z (z = x + x) \); the latter, in turn, again by LC, implies \( \sqcap x \sqcup y (y = x + x) \), whence, together with the \( \text{PA} \)-provable \( \forall x (x + x = x0) \), by LC, we get the target \( \sqcap x \sqcup y (y = x0) \).

**Fact 2.2** \( \text{CLA5} \vdash \sqcap x \sqcup y (x = y0 \sqcup x = y1) \).

**Proof.** Argue in \( \text{CLA5} \). By \( \text{CLA5} \)-Induction on \( x \), we want to show \( \sqcup y (|y| \leq |x| \land (x = y0 \sqcup x = y1)) \), which immediately implies the target \( \sqcap x \sqcup y (x = y0 \sqcup x = y1) \) by LC.

The basis \( \sqcup y (|y| \leq 0 \land (0 = y0 \sqcup 0 = y1)) \) is solved by choosing \( 0 \) for \( y \) and then choosing the left \( \sqcup \)-disjunct.

To solve the inductive step \( \sqcup y (|y| \leq |x| \land (x = y0 \sqcup x = y1)) \rightarrow \sqcup y (|y| \leq |x'| \land (x' = y0 \sqcup x' = y1)) \), we wait till Environment chooses a constant \( a \) for \( y \) in the antecedent, and also chooses one of the two \( \sqcup \)-disjuncts there.

Suppose the left \( \sqcup \)-disjunct is chosen in the antecedent. So, by now, the game has been brought down to \( |a| \leq |x| \land x = a0 \rightarrow \sqcup y (|y| \leq |x'| \land (x' = y0 \sqcup x' = y1)) \). Then we choose the same \( a \) for \( y \) in the consequent, and further choose the right disjunct there. The resulting position \( |a| \leq |x| \land x = a0 \rightarrow |a| \leq |x'| \land x' = a1 \) is true (by \( \text{PA} \)), so we win.

Now suppose the right \( \sqcup \)-disjunct is chosen in the antecedent. So, by now, the game has been brought down to \( |a| \leq |x| \land x = a1 \rightarrow \sqcup y (|y| \leq |x'| \land (x' = y0 \sqcup x' = y1)) \). Then we, using Axiom 8, compute the value of \( a' \), choose that value for \( y \) in the consequent, and further choose the left \( \sqcup \)-disjunct there. The game will be brought down to the true \( |a| \leq |x| \land x = a1 \rightarrow |a| \leq |x'| \land x' = (a')0 \), so, again, we win.

In the sequel we will heavily yet usually only implicitly rely on the following fact, which allows us to automatically transfer to \( \text{CLA5} \) all \( \text{CLA4} \)-provability results established in [8].

**Fact 2.3** Every \( \text{CLA4} \)-provable sentence is also \( \text{CLA5} \)-provable.

**Proof.** From Fact 2.1 we know that \( \text{CLA5} \) proves the only axiom (Axiom 9) of \( \text{CLA4} \) not present in \( \text{CLA5} \). So, \( \text{CLA5} \) proves all axioms of \( \text{CLA4} \). And the rule of LC is the same in the two theories. Therefore, it only remains to show that \( \text{CLA5} \) is closed under the rule of \( \text{CLA4} \)-Induction. So, assume \( F(x) \) is a polynomially bounded formula, and \( \text{CLA5} \) proves (the \( \sqcap \)-closures of) each of the following three premises of \( \text{CLA4} \)-Induction:

\[
\begin{align*}
F(0); \\
F(x) \rightarrow F(x0); \\
F(x) \rightarrow F(x1).
\end{align*}
\]

Our goal is to show that \( \text{CLA5} \) proves (the \( \sqcap \)-closure of) \( F(x) \), the conclusion of \( \text{CLA4} \)-Induction.

Argue in \( \text{CLA5} \). By \( \text{CLA5} \)-Induction on \( x \), we want to prove

\[
\sqcap y (|y| \leq |x| \rightarrow F(y)).
\]

The basis \( \sqcap y (|y| \leq 0 \rightarrow F(y)) \) is obviously taken care of by (2), according to which, after choosing \( 0 \) for \( y \), we know how to solve \( F(0) \). To solve the inductive step

\[
\sqcap y (|y| \leq |x| \rightarrow F(y)) \rightarrow \sqcap y (|y| \leq |x'| \rightarrow F(y)),
\]

we wait till Environment chooses a constant \( a \) for \( y \) in the consequent. Then, using Fact 2.2, we find the binary predecessor \( b \) of \( a \), and also figure out whether \( a = b0 \) or (\( \sqcup \)) \( a = b1 \). In either case, we specify \( y \) as \( b \) in the antecedent.

If \( a = b0 \), by now (6) is brought down to

\[
(|b| \leq |x| \rightarrow F(b)) \rightarrow (|b0| \leq |x'| \rightarrow F(b0)).
\]
From (3), we also know how to win \( F(b) \rightarrow F(b0) \). By applying copycat between the two \( F(b) \)s and two \( F(b0) \)s, we win (7).

The case of \( a = b1 \) is similar, only relying on (4) instead of (3). Thus, (5) is proven.

Now, the target \( F(x) \) can be easily seen to be a logical consequence of (5) and the \( PA \)-provable \( \forall x(|x| \leq |x|) \). ■

**Theorem 2.4** An arithmetical problem has a polynomial space solution iff it is provable in CLA5.

Furthermore, there is an efficient procedure that takes an arbitrary extended CLA5-proof of an arbitrary sentence \( X \) and constructs a solution of \( X \) (of \( X^\dagger \), that is) together with an explicit polynomial bound for its space complexity.

**Proof.** The soundness (“if”) part of this theorem will be proven in Section 3 and the completeness (“only if”) part in Section 4. ■

## 3 The soundness of CLA5

This section is devoted to proving the soundness part of Theorem 2.4. We will only focus on showing that any CLA5-provable sentence has a polynomial space solution. The “furthermore” clause of the theorem also claims that such a solution, together with an explicit polynomial bound for its space complexity, can be constructed efficiently. We will not explicitly verify this claim because it can be immediately seen to be true for the same reasons as those pointed out at the end of Section 13 of [8] when justifying the similar claim for CLA4.

Consider an arbitrary CLA5-provable sentence \( X \). In showing that \( X \) has a polynomial space solution, we proceed by induction on the length of its proof.

Assume \( X \) is an axiom of CLA5. If \( X \) is one of Peano axioms, then it is a true elementary sentence and therefore is won by a machine that makes no moves and consumes no space. And if \( X = \exists y \forall x (y = x') \) (Axiom 8), then it is won by a machine that (for the constant \( x \) chosen by Environment for the variable \( y \)) computes the value \( a \) of \( x+1 \), makes the move \( a \) and retires in a moveless infinite loop that consumes no space.

Next, suppose \( X \) is obtained from premises \( X_1, \ldots, X_n \) by LC. By the induction hypothesis, for each \( i \in \{1, \ldots, n\} \), we already have a solution (HPM) \( N_i \) of \( X_i \) together with an explicit polynomial bound \( \xi_i \) for the space complexity of \( N_i \). Of course, we can think of each such HPM \( N_i \) as an \( n \)-ary GHPM that ignores its inputs. Then, by clause 2 of Theorem 10.1 of [8], we can (efficiently) construct a solution \( M(\langle N_1 \rangle, \ldots, \langle N_n \rangle) \) of \( X \), together with an explicit polynomial bound \( \tau(\xi_1, \ldots, \xi_n) \) for its space complexity.

Finally, suppose \( X \) is (the \( \cap \)-closures of) \( F(x) \), where \( F(x) \) is a polynomially bounded formula, and \( X \) is obtained by CLA5-Induction on \( x \). So, the premises are (the \( \cap \)-closures of) \( F(0) \) and \( F(x) \rightarrow F(x') \). By the induction hypothesis, there are HPMs \( N \) and \( K \) — with explicit polynomial bounds \( \xi \) and \( \zeta \) for their space complexities, respectively — that solve these two premises, respectively. Fix them. We want to construct a solution \( M \) of \( F(x) \).

As we did in Section 13 of [8], we replace \( N, K \) by their “reasonable counterparts” \( N' \) and \( K' \) and the corresponding explicit polynomial bounds \( \xi', \zeta' \) for their space complexities. For simplicity, we further replace the two bounds \( \xi', \zeta' \) by the common bound \( \phi = \xi' + \zeta' \) for the space complexities of both machines \( N' \) and \( K' \). For further simplicity considerations, we will assume that the environment of our purported solution \( M \) of \( F(x) \) never makes illegal moves, for otherwise \( M \) easily detects illegal behavior and, being an automatic winner, retires in an infinite loop that consumes no space. We further assume that the environment of \( M \), just like \( N' \) and \( K' \), plays “reasonably”, that is, never makes unreasonable moves. This will not affect the outcome of the game in \( \bot \)’s (\( M \)’s) favor, as \( \bot \)’s unreasonable moves always result in the corresponding subgame’s being lost by \( \bot \), anyway. From our description of \( M \) it will be clear that, as long as Environment plays legally and “reasonably”, so does \( M \), because all it does is copycatting moves by Environment in the real play and moves by \( N' \) and \( K' \) in a series of imaginary (simulated) plays. For the same reason, the imaginary adversaries of the simulated \( N' \) and \( K' \) will also play legally and “reasonably”. To summarize, all — real or imaginary — machines that we consider, as well as their — real or imaginary — adversaries, play legally and “reasonably”.
To describe $\mathcal{M}$, assume $x, \vec{v}$ are exactly the free variables of $F(x)$ (the case of $F(x)$ having no free occurrences of $x$ is trivial and we exclude it from our considerations), so that, in an expanded form, $F(x)$ can be rewritten as $F(x, \vec{v})$. At the beginning, our $\mathcal{M}$ waits for Environment to choose constants for the free variables of $F(x, \vec{v})$. Assume $k$ is the constant chosen for the variable $x$, and $\vec{c}$ are the constants chosen for $\vec{v}$. Since the case of $k=0$ is straightforward and not worth considering separately, we will additionally assume that $k \geq 1$. From now on, we shall write $F'(x)$ as an abbreviation of $F(x, \vec{c})$. Further, we shall write $N'_0$ 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 constants $\vec{c}$ for the variables $\vec{v}$. So, $N'_0$ wins the constant game $F'(0)$. Similarly, for any $i \geq 1$, we will write $K'_i$ for the machine that works just like $\mathcal{K'}$ does in the scenario where the adversary, at the beginning of the play, has chosen the constants $\vec{c}$ for the variables $\vec{v}$ and the constant $i-1$ for the variable $x$. So, $K'_i$ wins the constant game $F'(i-1) \rightarrow F'(i)$. Similarly, we will write $\mathcal{M}_k$ for the machine that works just like $\mathcal{M}$ after the above event of Environment’s having chosen $k$ and $\vec{c}$ for $x$ and $\vec{v}$, respectively. So, in order to complete our description of $\mathcal{M}$, it would be sufficient to show that $\mathcal{M}_k$ wins $F'(k)$.

Throughout the rest of this description:

- $\ell$ denotes the size of the greatest constant among $k, \vec{c}$.
- $\varnothing$ denotes the depth (number of labmoves in the longest legal run) of $F(0)$.
- $q$ denotes the number of all tape symbols of the machines $\mathcal{N'}$ and $\mathcal{K'}$ (we may safely assume that the two machines have the same sets of tape symbols).
- $s$ denotes the number of all states of the machines $\mathcal{N'}$ and $\mathcal{K'}$ (again, we may safely assume that the two machines have the same sets of states).

In view of our assumption that the machines $\mathcal{N}'_0$, $\mathcal{K}'_1, \ldots, \mathcal{K}'_k$ and $\mathcal{M}_k$ as well as their adversaries play the corresponding games $F'(0)$, $F'(0) \rightarrow F'(1)$, $\ldots$, $F'(k-1) \rightarrow F'(k)$ and $F'(k)$ legally and “reasonably”, as was done in \textbf{[2]}, one can easily write a term $\eta(w)$ with a single variable $w$ such that the sizes of moves ever made by either player in any of the above games never exceed $\eta(l)$. For instance, if $F'(x)$ is

\[
\ell \cup \{ |u| \leq |x| \times |z| \land \eta(|v|) \leq |u| + |x| \rightarrow G \}
\]

where $G$ is elementary, then $\eta(w)$ can be taken to be $w \times w + w + 0^\ell$. Fix this $\eta$. Since both $\mathcal{N'}$ and $\mathcal{K'}$ run in space $\varnothing$, we obviously have:

**The spacecost of any cycle of any play by $\mathcal{N}'_0$ or $\mathcal{K}'_i$ ($1 \leq i \leq k$) does not exceed $\ell(\eta(1))$.** (8)

Let the **symbolwise length** of a position $\Phi$ mean $T+1$, where $T$ is the number of cells that $\Phi$ takes when spelled on the run tape. Taking into account that the sizes of no moves by either player in the plays of $F'(0)$, $F'(0) \rightarrow F'(1)$, $\ldots$, $F'(k-1) \rightarrow F'(k)$ exceed $\eta(l)$ and that at most $2\varnothing$ moves can be (legally) made in those plays, we obviously have:

**The symbolwise length of the position spelled on the run tape of $\mathcal{N}'_0$ or $\mathcal{K}'_i$ (each $i \in \{1, \ldots, k\}$), at any time in any play, does not exceed $2\varnothing \eta(l) + 2\varnothing + 1$.** (9)

(Here “$+ 2\varnothing$” is to account for the labels “\(\uparrow\)” and “\(\downarrow\)” attached to moves.)

Let $\mathcal{L}$ be an abbreviation defined by

\[
\mathcal{L} = s \times (\ell(\eta(1)) \times (2\varnothing \eta(l) + 2\varnothing + 1) \times (q^{\ell(\eta(1))}) \times (q^{2\varnothing \eta(l) + 2\varnothing + 1})�.
\]

We claim that, under our assumption that all parties play legally and “reasonably”, we have:

**Consider any machine $\mathcal{H} \in \{\mathcal{N}'_0, \mathcal{K}'_1, \ldots, \mathcal{K}'_k\}$, and any cycle (step, time) $c$ of any play by $\mathcal{H}$. If the adversary of $\mathcal{H}$ does not move at any time $d$ with $d \geq c$, then $\mathcal{H}$ does not move at any time $d$ with $d \geq c + \mathcal{L}$.** (10)
To prove (10), let us consider any (legal and “reasonable”) play by any machine $H \in \{N_0', K_1', \ldots, K_k'\}$, and answer the following question: How many different configurations are there that may emerge in the play? There are at most $s$ possibilities for the state of such a configuration. These possibilities are accounted for by the 1st of the five factors of $Ł$. Next, in view of (3), there are at most $\phi(n(1))$ possible locations of the work-tape head. This number is accounted for by the 2nd factor of $Ł$. Next, in view of (4), there are at most $2\phi(n(1)) + 2\phi + 1$ possible locations of the run-tape head and this number is accounted for by the 3rd factor of $Ł$. Next, in view of (5), obviously there are at most $q^{\phi(n(1))}$ possible contents of the work tape, and this number is accounted for by the 4th factor of $Ł$. Finally, in view of (6), there are at most $q^{2\phi(n(1)) + 2\phi + 1}$ possible contents of the run tape, and this number is accounted for by the 5th factor of $Ł$. Thus, there are at most $Ł$ possible configurations. Now, consider the scenario where the adversary of $H$ makes no moves beginning from a clock cycle $c$. Assume, for a contradiction, that $H$ makes a move $\alpha$ at some time $d$ with $d > c + Ł$. Since there are fewer that $d - c$ configurations, some configuration should repeat itself between the steps $c$ and $d$. In other words, $H$ is in an infinite loop. Hence, it will make the same move $\alpha$ over and over again, which means that $H$ does not play legally (there are at most $d$ legal moves by $\top$ in the play), contrary to our assumptions. This completes our proof of claim (10).

The idea underlying the work of $M_k$ can be summarized by saying that what $M_k$ does is synchronization — in the sense explained in Section 13 of [8] — between $k + 2$ games, real or imaginary (simulated). Namely:

- It synchronizes the imaginary play of $F'(0)$ by $N_0'$ with the antecedent of the imaginary play of $F'(0) \rightarrow F'(1)$ by $K_1'$.
- For each $i$ with $1 \leq i < k$, it synchronizes the consequent of the imaginary play of $F'(i - 1) \rightarrow F'(i)$ by $K_i'$ with the antecedent of the imaginary play of $F'(i) \rightarrow F'(i + 1)$ by $K_{i+1}'$.
- It synchronizes the consequent of the imaginary play of $F'(k - 1) \rightarrow F'(k)$ by $K_k'$ with the real play of $F'(k)$.

Therefore, since $N_0'$ wins $F'(0)$ and each $K_i'$ $(1 \leq i \leq k)$ wins $F'(i - 1) \rightarrow F'(i)$, $M_k$ wins $F'(k)$ and hence $M$ wins $F(x)$, as desired.

In section 13 of [8], synchronization in the above style was achieved by simulating all imaginary plays in parallel. Our present case does not allow doing the same though, and synchronization should be done in a more careful way. Namely, a parallel simulation of all plays is no longer possible, because there are exponentially (in the size of $k$) many simulations to perform, which would require an exponential amount of space. So, instead, simulations in the present case should be performed in some sequential rather than parallel manner, with subsequent simulations recycling the space used by the previous ones. Below is our attempt to describe the idea of how this is achieved.

$M_k$ starts out by simulating the play of $F'(0)$ by $N_0'$ for $Ł$ steps in the scenario where the adversary does not move. Note that, in view of (10), considering only $Ł$ steps is sufficient in the sense that, if $N_0'$ is ever going to make any moves in this scenario, it will do so (make all of its moves) within $Ł$ steps. Next, observe that this simulation only requires about as much space as $N_0'$ itself does, which, by (5), is polynomial — namely, (asymptotically) it does not exceed $\phi(n(1))$. Let $\hat{\beta}_l^0$ be the (possible empty) sequence of moves made by $N_0'$ in the above simulation. The procedure that we have just described will be referred to as $UP_0$, and the sequence $\langle \hat{\beta}_l^0 \rangle$ considered to be its output on input 0. That is, $UP_0(0) = \langle \hat{\beta}_l^0 \rangle$.

$M_k$ remembers the above $\langle \hat{\beta}_l^0 \rangle$, forgets everything else about the previous simulation (cleans up the space used by the latter to prepare it for reusage, that is), and now simulates the play of $F'(0) \rightarrow F'(1)$ by $K_1'$ for $Ł$ steps in the scenario where the adversary of the latter, during the very first step, made the moves $\hat{\beta}_l^0$ in the antecedent. This way, $M_k$ evens out (matches, synchronizes, balances) the antecedent of the imaginary play of $F'(0) \rightarrow F'(1)$ with the imaginary play of $F'(0)$. As in the previous case, the present simulation only requires a polynomial amount of space. Let $\hat{\beta}_l^1$ be the sequence of moves by $K_1'$ in the consequent of $F'(0) \rightarrow F'(1)$ detected during the above simulation (at this point, we do not care about the moves made in the antecedent). The procedure that we have just described will be referred to as $UP_0(1)$, with the sequence $\langle \hat{\beta}_l^1 \rangle$ considered to be its output on input 1.

---

2Remember that a scanning head of an HPM can never move beyond the leftmost blank cell.

3Here we implicitly rely on the obvious fact that the value of $Ł$ can be computed and stored in polynomial space. Also, as probably can be understood, “$Ł$ steps” means $Ł$ steps of $N_0'$, rather than $Ł$ steps of $M_k$ itself.
\( M_k \) remembers the above \( \langle \beta_1^1 \rangle \), forgets everything else about the previous simulation (including the previously remembered \( \langle \beta_1^0 \rangle \)), and now simulates the play of \( F'(1) \to F'(2) \) by \( K'_1 \) for \( 2 \) steps in the scenario where the adversary of the latter, during the very first step, made the moves \( \beta_1^1 \) in the antecedent. That is, now \( M_k \) even out the antecedent of \( F'(1) \to F'(2) \) with the consequent of \( F'(0) \to F'(1) \). Let \( \beta_1^2 \) be the sequence of moves by \( K'_1 \) in the consequent of \( F'(1) \to F'(2) \) detected during this simulation. The procedure that we have just described will be referred to as \( \text{UP}_0(2) \), with \( \langle \beta_1^2 \rangle \) considered to be its output on input 2.

Continuing in the above way, having consumed only a polynomial amount of space, \( M_k \) finds — after performing \( \text{UP}_0(0(k)) \) — the sequence \( \beta_1^2 \) of moves made in the consequent of \( F'(k-1) \to F'(k) \) by \( K'_k \). This completes the first, “upward” stage of the work of \( M_k \). \( M_k \) makes these very moves \( \beta_1^2 \) in its real play of \( F'(k) \), this way achieving the effect of matching the latter with the consequent of \( F'(k-1) \to F'(k) \).

Now \( M_k \) waits for Environment to move (maybe it has already moved while \( M_k \) was busy with its first stage, in which case there is no need to wait — \( M_k \) just reads the already made moves from its run tape). If Environment never moves, \( M_k \) obviously wins \( F'(k) \), because the synchronization it has maintained so far is final in this case. Suppose now Environment moves (or has already moved), namely, makes the sequence \( \alpha_1 \) of moves. This means that the so-far-maintained balance is no longer perfect, as \( F'(k) \) is no longer synchronized with the consequent of \( F'(k-1) \to F'(k) \). In order to restore balance, it would be sufficient for \( M_k \) to simulate the play of \( K'_k \) again, this time in the scenario where the adversary made the earlier-computed moves \( \beta_1^{k-1} \) in the antecedent and then, after waiting for \( 2 \) steps, the moves \( \alpha_1 \) in the consequent. However, the trouble is that \( M'_k \) has already forgotten \( \beta_1^{k-1} \) (it could not afford to remember all of the previously computed \( \beta_1 \)'s because their quantity is exponential). Not to worry though. \( M_k \) can recompute \( \beta_1^{k-1} \) by running \( \text{UP}_0(k-1) \) all over again. This will take the same (polynomial) amount of space as \( \text{UP}_0(k-1) \) does, plus the (again polynomial) amount of space required to keep \( \alpha_1 \) in memory while running \( \text{UP}_0(k-1) \).

So, now \( M_k \) knows both \( \beta_1^{k-1} \) and \( \alpha_1 \). Using this information, it simulates the play of \( F'(k-1) \to F'(k) \) by \( K'_k \) for \( 2 \) steps in the scenario where, at the very beginning of the play, the adversary made the moves \( \beta_1^{k-1} \) in the antecedent and then, after \( 2 \) steps, the moves \( \alpha_1 \) in the consequent. Let \( \gamma_1^{k-1} \) be the sequence of moves made by \( K'_k \) during the above \( 2 \) steps in the antecedent. The procedure that \( M_k \) has just performed we refer to as \( \text{DOWN}_1(k, \alpha_1) \), with \( \langle \gamma_1^{k-1} \rangle \) considered to be its output on input \((k, \alpha_1)\).

\( M_k \) remembers the above \( \langle \gamma_1^{k-1} \rangle \), and forgets everything else about the previous simulation. Next, it runs \( \text{UP}_0(k-2) \) to recompute the already forgotten \( \beta_1^{k-2} \). Now, \( M_k \) simulates the play of \( F'(k-2) \to F'(k-1) \) by \( K'_k \) for \( 2 \) steps in the scenario where at the very beginning of the play, the adversary made the moves \( \beta_1^{k-2} \) in the antecedent and then, after \( 2 \) steps, the moves \( \alpha_1 \) in the consequent. Let \( \gamma_1^{k-1} \) be the sequence of moves made by \( K'_k \) during the above \( 2 \) steps in the antecedent. The procedure that \( M_k \) has just performed we refer to as \( \text{DOWN}_1(k-1, \alpha_1) \), with \( \langle \gamma_1^{k-1} \rangle \) considered to be its output on input \((k-1, \alpha_1)\).

Continuing in the above way, having consumed only a polynomial amount of space, \( M_k \) finds — after performing \( \text{DOWN}_1(1, \alpha_1) \) — the sequence \( \gamma_1^1 \) of moves made in the antecedent of \( F'(0) \to F'(1) \) by \( K'_1 \). This completes the second, “downward” stage of the work of \( M_k \), and starts the third, “upward” stage.

Namely, \( M_k \) now simulates the play of \( F'(0) \) by \( N'_0 \) for \( 2 \) steps in the scenario where the adversary made its first (and only) sequence \( \gamma_1^1 \) of moves after \( 2 \) steps since the beginning of the play. Let \( \beta_0^1 \) be the sequence of moves made by \( N'_0 \) during the first \( 2 \) steps of this simulation, and \( \beta_0^2 \) be the sequence of moves made by \( N'_0 \) during the subsequent \( 2 \) steps. Notice that the present \( \beta_1^1 \) is the same as the one computed earlier by the procedure \( \text{UP}_0(0) \). The procedure that \( M_k \) has just performed we refer to as \( \text{UP}_1(0, \alpha_1) \), with \( \langle \beta_1^1, \beta_0^2 \rangle \) considered to be its output on input \((0, \alpha_1)\).

\( M_k \) remembers the above \( \langle \beta_1^1, \beta_0^2 \rangle \), and forgets everything else about the previous simulation. Further, it runs \( \text{DOWN}_1(2, \alpha_1) \) to recompute its output \( \langle \gamma_1^2 \rangle \). Now, \( M_k \) simulates the play of \( F'(0) \to F'(1) \) by \( K'_1 \) for \( 32 \) steps in the scenario where the adversary of the latter, during the very first step, made the moves \( \beta_0^1 \) in the antecedent, then, after the subsequent \( 2 \) steps, the moves \( \gamma_1^2 \) in the consequent, and then, after \( 2 \) more steps, the moves \( \beta_0^2 \) in the antecedent. Let \( \beta_1^2 \) be the sequence of moves made by \( K'_1 \) in the consequent during the first \( 2 \) steps of this simulation, and \( \beta_0^2 \) be the sequence of moves made by \( K'_1 \) in the consequent during the subsequent \( 2 \) steps. Notice that the present \( \beta_1^2 \) is the same as the one computed earlier by the procedure \( \text{UP}_0(1) \). The procedure that \( M_k \) has just performed we refer to as \( \text{UP}_1(1, \alpha_1) \), with \( \langle \beta_1^1, \beta_0^2 \rangle \) considered to be its output on input \((1, \alpha_1)\).

Again, \( M_k \) remembers the above \( \langle \beta_1^1, \beta_0^2 \rangle \), and runs \( \text{DOWN}_1(3, \alpha_1) \) to recompute its output \( \langle \gamma_1^2 \rangle \). Then
it simulates the play of $F'(1) \rightarrow F'(2)$ by $\mathcal{K}_2$ for $3\Sigma$ steps in the scenario where the adversary of the latter, during the very first step, made the moves $\vec{\beta}_1$ in the antecedent, then, after the subsequent $\Sigma$ steps, the moves $\vec{\gamma}_1$ in the consequent, and then, after $\Sigma$ more steps, the moves $\vec{\beta}_2$ in the antecedent. Let $\vec{\beta}_1^k$ be the sequence of moves made by $\mathcal{K}_2$ in the consequent during the first $\Sigma$ steps of this simulation, and $\vec{\beta}_2^k$ be the sequence of moves made by $\mathcal{K}_2$ in the consequent during the subsequent $\Sigma$ steps. The procedure that $\mathcal{M}_k$ has just performed we refer to as $\mathrm{UP}_1(2, \vec{\alpha}_1)$, with $\vec{\beta}_1, \vec{\beta}_2$ considered to be its output on input $(2, \vec{\alpha}_1)$.

Continuing in the above way, having consumed only a polynomial amount of space, $\mathcal{M}_k$ finds — after performing $\mathrm{UP}_1(k, \vec{\alpha}_1)$ — the sequence $(\vec{\beta}_1^k, \vec{\beta}_2^k)$ of moves made in the consequent of $F'(k-1) \rightarrow F'(k)$ by $\mathcal{K}_2'$. $\mathcal{M}_k$ has already made the moves $\vec{\beta}_1^k$ in the real play of $F'(k)$, so now, to synchronize $F'(k)$ with the consequent of $F'(k-1) \rightarrow F'(k)$, it makes $\vec{\beta}_2^k$ as its second series of moves in the real play. Now, $\mathcal{M}_k$ waits until it sees a second nonempty series $\vec{\alpha}_2$ of moves by Environment, after which it starts its fourth, “downward” stage whose steps we refer to as $\mathrm{DOWN}_2(k, \vec{\alpha}_1, \vec{\alpha}_2)$, $\mathrm{DOWN}_2(k-1, \vec{\alpha}_1, \vec{\alpha}_2)$, $\ldots$, $\mathrm{DOWN}_2(1, \vec{\alpha}_1, \vec{\alpha}_2)$. Upon its completion, $\mathcal{M}_k$ switches to its fifth, “upward” stage, whose steps we refer to as $\mathrm{UP}_2(0, \vec{\alpha}_1, \vec{\alpha}_2)$, $\mathrm{UP}_2(1, \vec{\alpha}_1, \vec{\alpha}_2)$, $\ldots$, $\mathrm{UP}_2(k, \vec{\alpha}_1, \vec{\alpha}_2)$. And so on.

Here is a more formal definition of the above procedures $\mathrm{UP}_j$ and $\mathrm{DOWN}_j$:

**Procedure** $\mathrm{UP}_j(i, \vec{\alpha}_1, \ldots, \vec{\alpha}_j)$, where $0 \leq j \leq d$, $0 \leq i \leq k$, and each $\vec{\alpha}_e (1 \leq e \leq j)$ is a nonempty sequence of moves:

**Step 1.** Create a record $\vec{v}$ and initialize it to 0.

**Step 2.** Unless $j=0$, run $\mathrm{DOWN}_j(1, \vec{\alpha}_1, \ldots, \vec{\alpha}_j)$ and remember its output $(\vec{\gamma}_1, \ldots, \vec{\gamma}_j)$ for as long as needed.

**Step 3.** Simulate the play of $F'(0)$ by $\mathcal{N}_0'$ for $(j+1)\Sigma$ steps in the scenario where, for each $e$ with $1 \leq e \leq j$, the adversary makes the sequence $\vec{\gamma}_e$ of moves on (the single) cycle #e$\Sigma$ (if $j=0$, this means just simulating $\mathcal{N}_0'$ for $\Sigma$ steps in the scenario where the adversary does not move). Create the record $(\vec{\beta}_1, \ldots, \vec{\beta}_{j+1})$ and, for each $e$ with $1 \leq e \leq j+1$, initialize the value of $\vec{\beta}_e$ to the sequence of moves made by $\mathcal{N}_0'$ during (including) cycles #e$\Sigma$ through #e$\Sigma-1$.

**Step 4.** While $\vec{v} \neq i$, do the following:

**Substep 4.1.** Update the content of the record $\vec{v}$ through incrementing it by 1.

**Substep 4.2.** Unless $j=0$, run $\mathrm{DOWN}_j(j+1, \vec{\alpha}_1, \ldots, \vec{\alpha}_j)$ and remember its output $(\vec{\gamma}_1, \ldots, \vec{\gamma}_j)$ for as long as needed.

**Substep 4.3.** Simulate the play of $F'(\vec{v}-1) \rightarrow F'(\vec{v})$ by $\mathcal{K}_0'$ for $(2j+1)\Sigma$ steps in the scenario where, for each $e$ with $1 \leq e \leq j+1$, the adversary makes the sequence $\vec{\beta}_e$ of moves on cycle #e$\Sigma$ and, for each $e$ with $1 \leq e \leq j$, the adversary makes the sequence $\vec{\gamma}_e$ of moves on cycle #e$\Sigma-1$ in the consequent. Update the record $(\vec{\beta}_1, \ldots, \vec{\beta}_{j+1})$ by changing $\vec{\beta}_1$ to the sequence of moves made by $\mathcal{K}_0'$ in the consequent during (including) cycles #0$\Sigma$ through #2j$\Sigma-1$, and changing each $\vec{\beta}_e$ with $2 \leq e \leq j+1$ to the sequence of moves made by $\mathcal{K}_0'$ in the consequent during (including) cycles #(2e-3)$\Sigma$ through #(2e-1)$\Sigma-1$.

**Step 5.** Return (the content of the record) $(\vec{\beta}_1, \ldots, \vec{\beta}_{j+1})$.

**Procedure** $\mathrm{DOWN}_j(i, \vec{\alpha}_1, \ldots, \vec{\alpha}_j)$, where $1 \leq j \leq d$, $1 \leq i \leq k$, and each $\vec{\alpha}_e (1 \leq e \leq j)$ is a nonempty sequence of moves:

**Step 1.** Create a record $\vec{v}$ and initialize it to $k+1$.

**Step 2.** Create a record $(\vec{\gamma}_1, \ldots, \vec{\gamma}_j)$ and initialize each $\vec{\gamma}_e (1 \leq e \leq j)$ to $\vec{\alpha}_e$.

**Step 3.** While $\vec{v} \neq i$, do the following:

**Substep 3.1.** Update the content of the record $\vec{v}$ through decrementing it by 1.

**Substep 3.2.** Run $\mathrm{UP}_j-1(\vec{v}-1, \vec{\alpha}_1, \ldots, \vec{\alpha}_{j-1})$ and remember its output $(\vec{\beta}_1, \ldots, \vec{\beta}_j)$ for as long as needed.

**Substep 3.3.** Simulate the play of $F'(\vec{v}-1) \rightarrow F'(\vec{v})$ by $\mathcal{K}_0'$ for $2j\Sigma$ steps in the scenario where, for each $e$ with $1 \leq e \leq j$: on cycle #e$\Sigma$-2$\Sigma$, the adversary made the sequence $\vec{\beta}_e$ of moves in the antecedent, and on cycle #(2e-2)$\Sigma$, the adversary made the sequence $\vec{\gamma}_e$ of moves in the consequent. Update the content of the record $(\vec{\gamma}_1, \ldots, \vec{\gamma}_j)$ by changing the value of each $\vec{\gamma}_e (1 \leq e \leq j)$ to the sequence of the moves made by $\mathcal{K}_0'$ in the antecedent during (including) cycles #(2e-2)$\Sigma$ through #2e$\Sigma-1$.

**Step 4.** Return (the content of the record) $(\vec{\gamma}_1, \ldots, \vec{\gamma}_j)$.

Now, we are ready to precisely describe the work of $\mathcal{M}_k$. It creates a record $j$ initialized to 1 and a record $\langle \vec{\alpha}_1, \ldots, \vec{\alpha}_{j-1} \rangle$ initialized to the empty sequence $\langle \rangle$, and then acts according to the following algorithm:

**Step 1.** Compute the value $(\vec{\beta}_1, \ldots, \vec{\beta}_j)$ of $\mathrm{UP}_j-1(k, \vec{\alpha}_1, \ldots, \vec{\alpha}_{j-1})$, and make the moves $\vec{\beta}_j$. 
Step 2. Keep polling the run tape until you see a new (jth, previously unseen) nonempty sequence \( \vec{\alpha}_j \) of moves by Environment\(^4\). If and when such an \( \vec{\alpha}_j \) is found, update the record \( \langle \vec{\alpha}_1, \ldots, \vec{\alpha}_{j-1}, \vec{\alpha}_j \rangle \) to \( \langle \vec{\alpha}_1, \ldots, \vec{\alpha}_{j-1}, \vec{\alpha}_j \rangle \), increment the content of the record \( j \) by 1, and go back to Step 1.

This completes our description of \( M_k \) and hence of \( M \).

It is not hard to see that, as promised, \( M_k \) indeed does a full synchronization between the \( k+2 \) games, real or imaginary, and hence wins. A long but straightforward analysis of the algorithm followed by (\( M_k \) and hence) \( M \), technical details of which are left to the reader in case he or she is not satisfied by our earlier explanations, reveals that a bound for the space complexity of \( M \) can be written as \( \mu(\phi(\ell)) \), where \( \mu(\cdot) \) is a certain polynomial (in fact, linear) function. The function \( \mu(\phi(\ell)) \) is the sought polynomial bound for the space complexity of \( M \).

4 The extensional completeness of CLA5

This section is devoted to proving the completeness part of Theorem 2.4. It means showing that, for any arithmetical problem \( A \) that has a polynomial space solution, there is a theorem of CLA5 which, under the standard interpretation, equals ("expresses") \( A \).

So, let us pick an arbitrary polynomial-space-solvable arithmetical problem \( A \). By definition, \( A \) is an arithmetical problem because, for some sentence \( X \), \( A = X^\dagger \). For the rest of this section, we fix such an \( X \), and fix \( X \) as an HPM that solves \( A \) (and hence \( X^\dagger \)) in polynomial space. Specifically, we assume that \( X \) runs in space \( \chi \), where \( \chi \), which we also fix for the rest of this section, is a single-variable term — and hence can be seen/written as an explicit polynomial function — with \( \chi(x) \geq x \) for all \( x \). We also agree that, throughout this section, "formula" exclusively means a subformula of \( X \), in which some variables may be renamed.

\( X \) may not necessarily be provable in CLA5, and our goal is to construct another sentence \( \overline{X} \) for which, just like for \( X \), we have \( A = \overline{X}^\dagger \) and which, perhaps unlike \( X \), is provable in CLA5.

Remember the sentence \( L \) from Section 14.3 of [8], saying that \( X \) does not win \( X \) in time \( \chi \). Here we redefine that sentence so that now it says the same but about space rather than time. Namely, let \( E_1(\vec{x}) \ldots, E_n(\vec{x}) \) be all subformulas of \( X \), where all free variables of each \( E_i(\vec{x}) \) are among \( \vec{x} \). Then our present \( L \) is the \( \lor \)-disjunction of natural formalizations of the following statements:

1. There is a \( \exists \)-illegal position of \( X \) spelled on the run tape of \( X \) on some clock cycle of some computation branch of \( X \).
2. There is a clock cycle \( c \) in some computation branch of \( X \) whose spacecost (see Section 1.1) exceeds \( \chi(\ell) \), where \( \ell \) is the background of \( c \).
3. There is a (finite) legal run \( \Gamma \) of \( X \) generated by \( X \) and a tuple \( \vec{c} \) of constants (\( \vec{c} \) of the same length as \( \vec{x} \)) such that:
   - \( \langle \Gamma \rangle X = E_1(\vec{c}) \), and we have \(-\|E_1(\vec{c})\| \) (i.e., \( \|E_1(\vec{c})\| \) is false),
   - or \( \ldots \), or
   - \( \langle \Gamma \rangle X = E_n(\vec{c}) \), and we have \(-\|E_n(\vec{c})\| \) (i.e., \( \|E_n(\vec{c})\| \) is false).

Next, remember the overline notation from Section 14.4 of [8]. We adopt that notation without any changes, except that \( L \) in it means our present (rather than the old) \( L \). So, for any formula \( E \) including \( X \), \( \overline{E} \) is the result of replacing in \( E \) every politeral \( L \) by \( L \lor \overline{L} \).

The following two lemmas are proven exactly as the corresponding lemmas in [8]:

**Lemma 4.1** Lemma 14.2 of [8] continues to hold. That is:
For any formula \( E \), including \( X \), we have \( E^\dagger = \overline{E} \).

**Lemma 4.2** Lemma 14.3 of [8] continues to hold. That is:
For any formula \( E \), CLA5 \( \vdash L \rightarrow \overline{E} \).

\(^4\)This will require scanning the run tape from the rightmost previously scanned cell up to the leftmost blank cell.
In view of Lemma 14.4 what now remains to do for the completion of our completeness proof is to show that $\text{CLA}_4 \vdash \overline{X}$.

We encode configurations as in Appendix A of [8]. Also remember the meanings (which we adopt here without any changes) of “legitimate configuration”, “yield” and “deterministic successor” from Section 14.5 of [8]. As in [8], terminologically we often identify configurations with their codes. For instance, we may say “configuration $z$” where what is really meant is “the configuration encoded by $z$”.

Based on reasons similar to those relied upon in the preceding section when justifying [11], we can write a function $\chi'(z)$ (fix it) polynomial in $z$ (and hence exponential in the size $|z|$ of $z$) such that the following lemma holds:

**Lemma 4.3** (PA $\vdash$ :) Suppose $z$ is a legitimate configuration, and $X$ moves in the $i$th (some $i \geq 0$) deterministic successor of $z$. Then, as long as $X$ (indeed) runs in space $\chi$, $i < \chi'(z)$.

Let $E(s)$ be a formula all of whose free variables are among $s$ (but not necessarily vice versa), and $z$ be a variable not among $s$. We will write $E^o(z, s)$ to denote an elementary formula whose free variables are $z$ and those of $E(s)$, and which is a natural arithmetization of the predicate that, for any constants $a, \vec{c}$ in the roles of $z, s$, holds (that is, $E^o(a, \vec{c})$ is true) iff $a$ is a legitimate configuration and its yield is $E(\vec{c})$. Further, we will write $E^o_0(z, \vec{s})$ to denote an elementary formula whose free variables are $z$ and those of $E(s)$, and which is a natural arithmetization of the predicate that, for any constants $a, \vec{c}$ in the roles of $z, \vec{s}$, holds iff $E^o(a, \vec{c}) \land E^o(b, \vec{c})$ is true, where $b$ is the $\chi(a)$th deterministic successor of $a$.

Thus, our present $E^o(a, \vec{c})$ means virtually the same as in Section 14.5 of [8], while our present $E^o_0(z, \vec{s})$ is a modified version of the $E^o_0(z, \vec{s})$ of [8]. Namely, now $b$ is the $\chi'(a)$th (rather than $\chi(a)$th as in [8]) deterministic successor of $a$.

**Lemma 4.4** Lemma 14.4 of [8] continues to hold. That is: Assume $E(s)$ is a non-critical formula all of whose free variables are among $s$. Then

$$\text{PA} \vdash \forall(E^o_0(z, \vec{s}) \rightarrow \|E(s)\|).$$

**Proof.** Assume the conditions of the lemma. Argue in PA. Consider arbitrary (\forall) values of $z$ and $s$, which we continue writing as $z$ and $s$. Suppose, for a contradiction, that $E^o_0(z, \vec{s})$ is true but $\|E(s)\|$ is false. The falsity of $\|E(s)\|$ implies the falsity of $\|E(s)\|$. This is so because the only difference between the two formulas is that, wherever the latter has some politeral $L$, the former has a $\lor$-disjunction containing $L$ as a disjunct.

The truth of $E^o_0(z, \vec{s})$ implies that $X$ reaches the configuration (computation step) $z$ and, in the scenario where Environment does not move, $X$ does not move either for at least $\chi'(z)$ steps afterwards. If $X$ does not move even after $\chi'(z)$ steps, then it has lost the game, because the eventual position hit in the play is $E(s)$ and the elementarization of the latter is false (as observed in [8], every such game is lost). And if $X$ does make a move sometime after $\chi'(z)$ steps, then, in view of Lemma 4.3, $X$ does not run in space $\chi$. Thus, in either case, $X$ does not win $X$ in space $\chi$, that is, $L$ is true. The rest of this proof is identical to what follows claim (23) in the proof of Lemma 14.4 of [8].

**Lemma 4.5** Lemma 14.5 of [8] continues to hold. That is:

Assume $E(s)$ is a critical formula all of whose free variables are among $s$. Then

$$\text{CLA}_5 \vdash \exists E^o_0(z, \vec{s}) \rightarrow \forall\overline{E(s)}. \quad (11)$$

**Proof.** The same as the proof of Lemma 14.5 of [8], taking into account that Lemma 14.3 of [8] on which the latter relies continues to hold in our present case according to Lemma 4.3.

In the subsequent two lemmas, the notational and terminological conventions of Appendix A of [8] are adopted without any changes whatsoever.

**Lemma 4.6** Lemmas A.1 through A.6 of [8] continue to hold (with “CLA5” instead of “CLA4”, of course).\footnote{Asymptotically, $\chi'(z)$ is $O(s^{\chi(|z|)})$, where $s$ is the number of symbols of which configurations are composed (see Section A.1 of [8]).}
Lemma 4.7 Lemma A.7 of [8] holds in our present case in the following stronger form:

CLA5 ⊢ C → A'(z, r) ∪ ∃x B(z, x).

Proof. This proof, as expected, is very close to the proof of Lemma A.7 of [8]. Argue in CLA5. By CLA5-Induction on r, we want to show

\[ C(z) → ∃x (|x| ≤ |z| + |r| ∧ A(z, x, r)) ∪ ∃x (|x| ≤ (|z| + |r|)0 ∧ B(z, x)), \]

from which the target \( C(z) → A'(z, r) ∪ ∃x B(z, x) \) easily follows by LC.

To solve the base \( C(z) → ∃x (|x| ≤ |z| + |0| ∧ A(z, x, 0)) ∪ ∃x (|x| ≤ (|z| + |0|)0 ∧ B(z, x)) \), we figure out whether the state of \( z \) is a move state or not. If yes, we choose the right \( ∪ \)-disjunct; if not, we choose the left \( ∪ \)-disjunct. In either case, we further choose the value of \( z \) for the variable \( x \) and win.

The inductive step is

\[
\begin{align*}
(C(z) → ∃x (|x| ≤ |z| + |r| ∧ A(z, x, |r|)) & \cup ∃x (|x| ≤ (|z| + |r|)0 ∧ B(z, x))) \\
(C(z) → ∃x (|x| ≤ |z| + |r'| ∧ A(z, x, |r'|)) & \cup ∃x (|x| ≤ (|z| + |r'|)0 ∧ B(z, x)))
\end{align*}
\]

(12)

To solve (12), we wait till Environment selects one of the two \( ∪ \)-disjuncts in the antecedent. If the right \( ∪ \)-disjunct is selected, we wait further till a constant \( c \) for \( x \) is selected there. Then we select the right \( ∪ \)-disjunct in the consequent, and choose the same \( c \) for \( x \) in it.

Suppose now the left \( ∪ \)-disjunct is selected in the antecedent of (12). Wait further till a constant \( c \) for \( x \) is selected there. We may assume that \( A(z, c, r) \) is true, or else we win the game. Using Lemma A.6 of [8] (which continues to hold by our Lemma 4.7), we find the deterministic successor \( d \) of the configuration \( c \). With a little thought, one can see that the size of \( d \) cannot exceed the sum of the sizes of \( z \) and \( r' \) more than twice, so that \(|d| ≤ (|z| + |r'|)0\) holds. We figure out whether the state of \( d \) is a move state or not. If not, we select the left \( ∪ \)-disjunct in the consequent of (12), otherwise, select the right disjunct. In either case, we further choose \( d \) for \( x \) and win. ■

Lemma 4.8 Lemma 14.6 of [8] continues to hold. That is:

Assume \( E(\bar{s}) \) is a formula all of whose free variables are among \( \bar{s} \), and \( y \) is a variable not occurring in \( E(\bar{s}) \). Then:

(a) For every \( (∨, y) \)-development \( H_1(y, \bar{s}) \) of \( E(\bar{s}) \), CLA5 proves \( E^\circ(\bar{z}, \bar{s}) → ∃u H^\circ(\bar{u}, y, \bar{s}). \)

(b) Where \( H_1(y, \bar{s}), …, H_n(y, \bar{s}) \) are all of the \( (∨, y) \)-developments of \( E(\bar{s}) \), CLA5 proves

\[ E^\circ(\bar{z}, \bar{s}) → E^\circ(\bar{z}, \bar{s}) \cup ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∪ ∃u ∆

Proof. The proof of clause (a) of Lemma 14.6, given in Section A.2 of [8], only relies on Lemmas A.4 and A.6 of [8], which (by our Lemma 4.6), continue to hold in our present case. So, clause (a) of our lemma is taken care of.

For clause (b), assume its conditions. In CLA5, we can solve (13) as follows. Assume \( E^\circ(\bar{z}, \bar{s}) \), which, of course, implies \( C(z) \). Since \( \chi'(z) \) is a polynomial function, we may assume that it is represented as a legitimate term of the language of PA, so that Fact 12.6 of [8] applies. We compute the value of \( \chi'(z) \), and use that value to specify \( r \) in the resource of Lemma 4.7. As a result, we get the resource \( A'(z, \chi'(z)) ∪ ∃x B(z, x) \). This means that we will either know that \( A'(z, \chi'(z)) \) is true, or find a constant \( a \) for which we will know that \( B(z, a) \) is true.

If \( A'(z, \chi'(z)) \) is true, then so is \( E^\circ(\bar{z}, \bar{s}) \) and, by choosing the latter, we win (13).

Now suppose \( B(z, a) \) is true. Then the desired conclusion can be achieved by literally repeating the corresponding part of the proof given in Section A.3 of [8], as all lemmas relied upon there continue to hold in our present case as well. ■
Lemma 4.9 Lemma 14.7 of [8] continues to hold. That is:
Assume \( E(\hat{s}) \) is a formula all of whose free variables are among \( \hat{s} \). Then \( \text{CLA5} \) proves \( E^\circ(z, \hat{s}) \rightarrow E(\hat{s}) \).

Proof. The proof of Lemma 14.7 of [8] goes through here without any changes, taking into account that Lemmas 14.3, 14.4, 14.5 and 14.6 of [8] on which the latter relies continue to hold in our present case according to Lemmas 14.2, 14.3, 14.5 and 14.8.

Now we are ready to claim the target result of this section in exactly the same way as at the end of Section 14 of [8]. Namely: Let \( a \) be the code of the start configuration of \( \mathcal{X} \), and \( \hat{a} \) a standard variable-free term representing \( a \), such as 0 followed by \( a \)’s. Of course, \( \text{PA} \) and hence \( \text{CLA5} \) proves \( X^\circ(\hat{a}) \). By Fact 12.6 of [8], \( \text{CLA5} \) proves \( \sqcup z(z=\hat{a}) \). By Lemma 4.9 \( \text{CLA5} \) also proves \( \sqcap z(X^\circ(z) \rightarrow \overline{X}) \). These three can be seen to imply \( \overline{X} \) by LC. Thus, \( \text{CLA5} \vdash \overline{X} \), as desired.

5 CLA6, a theory of elementary recursive computability

The language of theory \( \text{CLA6} \) is the same as that of \( \text{CLA5} \), and so are its axioms and the logical rule LC. In addition, just like \( \text{CLA5} \), \( \text{CLA6} \) has a single nonlogical rule, which we call \( \text{CLA6-Induction} \):

\[
\frac{\sqcap(F(0)) \quad \sqcap(F(x) \rightarrow F(x'))}{\sqcap(F(x))}
\]

where \( F(x) \) is any exponentially bounded formula.

Thus, the only difference between \( \text{CLA5} \) and \( \text{CLA6} \) is that, while the induction rule of the former requires the formula \( F(x) \) to be polynomially bounded, the induction rule of the latter has the weaker requirement that \( F(x) \) should be \textit{exponentially bounded}. Below we explain the precise meaning of this term.

For a variable \( x \), by an \textit{exponential sizebound for} \( x \) we shall mean a standard formula of the language of \( \text{PA} \) saying that \( |x| \leq \tau(y_1, \ldots, y_n) \), where \( y_1, \ldots, y_n \) are any variables different from \( x \), and \( \tau(y_1, \ldots, y_n) \) is any \( (0', +', \times') \)-combination of \( y_1, \ldots, y_n \). For instance, \( |x| \leq y+z \) is an exponential sizebound for \( x \), which is a formula of \( \text{PA} \) saying that the size of \( x \) does not exceed the sum of \( y \) and \( z \). Now, we say that a formula \( F \) is \textit{exponentially bounded} iff:

- Whenever \( \sqcap \forall x G(x) \) is a subformula of \( F \), \( G(x) \) has the form \( S(x) \rightarrow H(x) \), where \( S(x) \) is an exponential sizebound for \( x \).
- Whenever \( \sqcup \exists x G(x) \) is a subformula of \( F \), \( G(x) \) has the form \( S(x) \land H(x) \), where \( S(x) \) is an exponential sizebound for \( x \).

Fact 5.1 Every \( \text{CLA5} \)-provable (and hence also every \( \text{CLA4} \)-provable) sentence is provable in \( \text{CLA6} \).

Proof. We only need to show that \( \text{CLA6} \) is closed under \( \text{CLA5-Induction} \). So, assume \( F(x) \) is a polynomially bounded formula, and \( \text{CLA6} \) proves (the \( \sqcap \)-closures of) both of the following two premises of \( \text{CLA5-Induction} \):

\[
F(0); \tag{14}
\]

\[
F(x) \rightarrow F(x'). \tag{15}
\]

Our goal is to show that \( \text{CLA6} \) proves (the \( \sqcap \)-closure of) \( F(x) \), the conclusion of \( \text{CLA5-Induction} \).

For every variable \( z \) and every polynomial sizebound \( S(z) \) for \( z \) that looks like \( |z| \leq \tau(|y_1|, \ldots, |y_n|) \), let \( S'(z) \) denote the exponential sizebound \( |z| \leq \tau(y_1, \ldots, y_n) \) for \( z \). Further, let \( F'(x) \) be the result of simultaneously replacing in \( F(x) \):

- every subformula \( \sqcap z(S(z) \rightarrow G) \) by \( \sqcap z\left(S'(z) \rightarrow (S(z) \rightarrow G)\right) \);
- every subformula \( \sqcup z(S(z) \land G) \) by \( \sqcup z\left(S'(z) \land (S(z) \land G)\right) \).
Note that $F'(x)$ is an exponentially bounded formula. Further, for each of the above sizebounds $S(z)$, $\text{PA}$ obviously proves $\forall z(S(z) \rightarrow S'(z))$. This fact, together with (14) and (15), can be easily seen to imply $F'(0)$ and $F'(x) \rightarrow F'(x')$ by LC. Thus, $\text{CLA6}$ proves both $F'(0)$ and $F'(x) \rightarrow F'(x')$. Since $F'(x)$ is an exponentially bounded formula, $\text{CLA6}$-Induction applies, by which $\text{CLA6}$ proves $F'(x)$. The latter, again in conjunction with $\forall z(S(z) \rightarrow S'(z))$, can be seen to imply $F(x)$ by LC. ■

In what follows, we use $\text{EXP}(x)$ as an (alternative, linear) notation for the function $2^x$.

Remember the concept of an explicit polynomial function $\tau$ from Section 10 of [8]. An explicit elementary recursive function $\tau = \langle \tau_1, \ldots, \tau_f \rangle$ is defined in the same way, with the only difference that now, together with $', +, \times$ and $f_1, \ldots, f_{i-1}$, each functional (graph) $\tau_f$ is allowed to contain $\text{EXP}$ as an additional function letter with its standard interpretation. For instance, $\langle \text{EXP}(x+z), f_1(f_1(x)) \rangle$ (with its two elements written as graphs) is an explicit elementary recursive function. This term represents — and hence we identify it with — the function $2^{2^{x+z}+2^{x+z}}$. When $\tau$ is an explicit elementary recursive function and $M$ is a $\tau$ time (resp. space) machine, we say that $\tau$ is an explicit elementary recursive bound for the time (resp. space) complexity of $M$.

We say that a given HPM $M$ runs in elementary recursive time (resp. space) iff there is an (explicit) elementary recursive function $\tau$ such that $M$ runs in time (resp. space) $\tau$. And we say that a given problem has an elementary recursive solution iff it has a solution that runs in elementary recursive time. The reason why we omitted the word “time” here is that, as it is not hard to see (left as an exercise for the reader), a problem has an elementary recursive time solution if and only if it has an elementary recursive space solution.

**Theorem 5.2** An arithmetical problem has an elementary recursive solution iff it is provable in $\text{CLA6}$.

Furthermore, there is an efficient procedure that takes an arbitrary extended $\text{CLA6}$-proof of an arbitrary sentence $X$ and constructs a solution of $X$ (of $X^1$, that is) together with an explicit elementary recursive bound for its time complexity.

**Proof.** The soundness (“if”) part of this theorem will be proven in Section 5 and the completeness (“only if”) part in Section 4. ■

## 6 The soundness of $\text{CLA6}$

As in Section 3 here we will limit ourselves to proving the first, main part (of the soundness part) of Theorem 5.2, the “furthermore” clause of the theorem can be taken care of in the same way as in the similar proof for $\text{CLA4}$ given in [8].

Consider any $\text{CLA6}$-provable sentence $X$. We proceed by induction on its proof.

Assume $X$ is an axiom of $\text{CLA6}$. As in the cases of $\text{CLA4}$ and $\text{CLA5}$, if $X$ is one of Peano axioms, then it is a true elementary sentence and therefore is won by a machine that makes no moves. And if $X$ is $\forall z \exists y(y=x')$, then it is won by a machine that (for the constant $x$ chosen by Environment for the variable $x$) computes the value $a$ of $x+1$, makes the move $a$ and retires in a moveless infinite loop.

Next, suppose $X$ is obtained from premises $X_1, \ldots, X_n$ by LC. By the induction hypothesis, for each $i \in \{1, \ldots, n\}$, we already have a solution (HPM) $N_i$ of $X_i$ together with an explicit elementary recursive bound $\xi_i$ for the time complexity of $N_i$. We can think of each such HPM $N_i$ as an $n$-ary GHPM that ignores its inputs. Then, in view of clause 2 of Theorem 10.1 of [8], we can (efficiently) construct a solution $M(\langle N_1\rangle, \ldots, \langle N_n\rangle)$ of $X$, together with an explicit elementary recursive bound $\tau(\xi_1, \ldots, \xi_n)$ for its time complexity.

Finally, suppose $X$ is (the $\exists$-closure of) $F(x)$, where $F(x)$ is an exponentially bounded formula, and $X$ is obtained by $\text{CLA6}$-Induction on $x$. So, the premises are (the $\exists$-closures of) $F(0)$ and $F(x) \rightarrow F(x')$. By the induction hypothesis, there are HPMs $N$ and $K$ — with explicit elementary recursive bounds $\xi, \zeta$ for their time complexities, respectively — that solve these two premises, respectively. Fix them. We want to construct a solution $M$ of $F(x)$.

As we did in Section 13 of [8] or in Section 5 of the present paper, we replace $N$ and $K$ by their “reasonable counterparts” $N'$ and $K'$ and the corresponding explicit elementary recursive bounds $\xi', \zeta'$ for their time complexities. Of course, the meaning of “reasonable” is correspondingly redefined now. Namely, it means...
that the size of the constant chosen by a player for a variable bound by a bounded choice quantifier does not violate the conditions imposed on it by the exponential (rather than polynomial as before) sizebound for that variable. As before, we also assume that the environment of $\mathcal{M}$ plays legally and “reasonably”. Then, from our description of $\mathcal{M}$, it will be immediately clear that so do $\mathcal{M}$ and the imaginary adversaries of $\mathcal{N}'$ and $\mathcal{K}'$. Thus, all machines that we consider, as well as their adversaries, play legally and “reasonably”.

To describe $\mathcal{M}$, as in Section 3, assume $x, \vec{v}$ are exactly the free variables of $F(x)$, so that $F(x)$ can be rewritten as $F(x, \vec{v})$. At the beginning, $\mathcal{M}$ waits for Environment to choose constants for the free variables of $F(x, \vec{v})$. Assume $k$ is the constant chosen for the variable $x$, and $\vec{c}$ are the constants chosen for $\vec{v}$. From now on, we shall write $F'(x)$ for $F(x, \vec{c})$. Further, we shall write $N_0'$ for the machine that works just like $N'$ does in the scenario where the adversary, at the beginning of the play, has chosen the constants $\vec{c}$ for the variables $\vec{v}$. So, $N_0'$ wins the game $F'(0)$. Similarly, for any $i \geq 1$, we will write $K_i'$ for the machine that works just like $K'$ does in the scenario where the adversary, at the beginning of the play, has chosen the constants $\vec{c}$ for the variables $\vec{v}$ and the constant $i-1$ for the variable $x$. So, $K_i'$ wins the game $F'(i-1) \rightarrow F'(i)$. Similarly, we will write $M_k$ for the machine that works just like $M$ does after the above event of Environment’s having chosen $k$ and $\vec{c}$ for $x$ and $\vec{v}$, respectively. So, in order to complete our description of $\mathcal{M}$, it would suffice to simply define $M_k$ and say that, after Environment has chosen constants for all free variables of $F(x)$, $\mathcal{M}$ continues playing as $M_k$.

The work of $M_k$ consists in continuously polling its run tape to see if Environment has made any new moves, combined with simulating, in parallel, one play of $F'(0)$ by $N'$ and — for each $i \in \{1, \ldots, k\}$ — one play of $F'(i-1) \rightarrow F'(i)$ by $K_i'$. In this mixture of one real and $k+1$ imaginary plays, $\mathcal{M}$ synchronizes $k+1$ pairs of (sub)games, real or imaginary. Namely:

- It synchronizes the consequent of the imaginary play of $F'(k-1) \rightarrow F'(k)$ by $K_i'$ with the real play of $F'(k)$.
- For each $i \in \{1, \ldots, k-1\}$, it synchronizes the consequent of the imaginary play of $F'(i-1) \rightarrow F'(i)$ by $K_i'$ with the antecedent of the imaginary play of $F'(i) \rightarrow F'(i+1)$ by $K_{i+1}'$.
- It synchronizes the imaginary play of $F'(0)$ by $N_0'$ with the antecedent of the imaginary play of $F'(0) \rightarrow F'(1)$ by $K_1'$.

This completes our description of $M_k$ and hence of $\mathcal{M}$. Remembering our assumption that ($\mathcal{N}', \mathcal{K}'$ and hence) $\mathcal{N}', \mathcal{K}'$ win the corresponding games, it is obvious that $\mathcal{M}_k$ wins $F'(k)$ and hence $\mathcal{M}$ wins $\bigwedge \{F(x)\}$, as desired. It now remains to show that the time complexity of $\mathcal{M}$ is also as desired.

For the rest of this proof, pick and fix an arbitrary play (computation branch) of $\mathcal{M}$, and an arbitrary clock cycle $\epsilon$ on which $\mathcal{M}$ makes a move $\alpha$ in the real play of $F(x)$. Let $h$ and $\ell$ be the timecost and the background of this move, respectively. Let $k, F'(x), N_0', K_1', \ldots, K_k, M_k$ be as in the description of the work of $\mathcal{M}$. Note that $\ell$ is not smaller than the size of the greatest of the constants chosen by Environment for the free variables of $F(x)$. Remembering that all players that we consider play legally and “reasonably”, one can easily write an explicit elementary recursive function $\eta(\ell)$ (fix it) such that we have:

$$\text{The sizes of no moves ever made by } M_k \text{ or the simulated } N_0', K_i' \ (1 \leq i \leq k) \text{ exceed } \eta(\ell). \tag{16}$$

For instance, if $F(x)$ is $\bigwedge u (|u| \leq x \land z \land \bigwedge v (|v| \leq u+x \rightarrow G))$ where $G$ is elementary, then $\eta(\ell)$ can be taken to be $\text{EXP}(\ell) \times \text{EXP}(\ell) + \text{EXP}(\ell) + 0''''$. The polling, simulation and copycat performed by $M_k$ do impose some time overhead, but the latter, as in Section 13 of 8, can be safely ignored. We will further pretend that the polling and the several simulations happen in a truly parallel fashion, in the sense that $M_k$ spends a single clock cycle on tracing a single computation step of all $k+1$ machines simultaneously, as well as on checking out its run tape to see if Environment has made a new move. If so, the rest of our argument is almost literally the same as in Section 13 of 8. Namely:

Let $\beta_1, \ldots, \beta_m$ be the moves by simulated machines that $M_k$ detects by time $\ell$, arranged according to the times $t_1 \leq \ldots \leq t_m$ of their detections (which, by our simplifying assumptions, coincide with the timestamps of the corresponding moves in the corresponding simulated plays). Let $d = \epsilon - h$. Let $j$ be the smallest integer among $1, \ldots, m$ such that $t_j \geq d$. Since each simulated machine runs in time $\phi$, in view of (16) it is clear that $t_j - d$ does not exceed $\phi(\eta(\ell))$. NOR does $t_i - t_{i-1}$ for any $i$ with $j < i \leq m$. Hence $t_m - d \leq (m - j + 1) \times \phi(\eta(\ell))$. Since
m, j ≥ 1, let us be generous and simply say that \( t_m - d \leq m \times (\eta(\ell)) \). But notice that \( \beta_m \) is a move made by \( K_k \) in the consequent of \( F'(k-1) \rightarrow F'(k) \), immediately (by our simplifying assumptions) copied by \( M_k \) in the real play when it made its move \( \alpha \). In other words, \( c = t_m \). And \( c - d = h \). So, \( h \) does not exceed \( m \times (\eta(\ell)) \).

And, by (10), the size of \( a \) does not exceed \( m \times (\eta(\ell)) \), either. But observe that \( k \leq 2^\ell \) and that \( m \) cannot exceed \( k + 1 \) times the depth \( d \) of \( F'(0) \); therefore, \( m \leq d \times (2^\ell + 1) \). Thus, (as long as we pretend that there is no polling/simulation/copycat overhead) neither the timecost nor the size of \( a \) exceed \( d \times (2^\ell + 1) \times (\eta(\ell)) \).

An upper bound for the above function \( d \times (2^\ell + 1) \times (\eta(\ell)) \), even after “correcting” the latter so as to precisely account for the so far suppressed polling/simulation/copycat overhead, can be expressed as an explicit elementary recursive function \( \tau \). This is exactly the sought explicit elementary recursive bound for the time complexity of \( M \).

7 The extensional completeness of CLA6

We treat \( EXP(x) \) as a pseudoterm and, when writing “\( z = EXP(x) \)” within a formula, it is to be understood as an abbreviation of a standard formula of the language of \( PA \) saying that \( z \) equals \( 2^x \).

Fact 7.1 CLA6 ⊨ \( \square z (z = EXP(x)) \).

Proof. Argue in CLA6. By CLA6-Induction on \( x \), we want to prove \( \square z (|z| \leq x' \land z = EXP(x)) \), which immediately implies the target \( \square z (z = EXP(x)) \) by LC.

The basis \( \square z (|z| \leq 0' \land z = EXP(0)) \) is solved by computing the value of \( 0' \) and choosing that value for \( z \), which yields the true \( |0'| \leq 0' \land 0' = EXP(0) \).

To solve the inductive step \( \square z (|z| \leq x' \land z = EXP(x)) \rightarrow \square z (|z| \leq x'' \land z = EXP(x')) \), we wait till Environment chooses a value \( a \) for \( z \) in the antecedent. Then we compute the value of \( a0 \) and choose that value for \( z \) in the consequent, yielding the true \( |a| \leq x' \land a = EXP(x) \rightarrow |a0| \leq x'' \land a0 = EXP(x') \). ■

By an elementary recursive tree-term we mean a term of the language of CL12 (but not necessarily one of the languages of CLA4-CLA6) containing no constants other than \( 0 \) and no function symbols other than \( + \) (unary), \( + \) (binary), \( \times \) (binary), \( EXP \) (unary). Every such term represents an elementary recursive function of the same arity as the number of variables in the term) under the standard meaning of its function symbols and \( 0 \). We treat every \( n \)-ary elementary recursive tree-term \( \tau(x_1, \ldots, x_n) \) (can be simply written as \( \tau \) instead) as a pseudoterm of the language of CLA6 and, when writing “\( z = \tau(x_1, \ldots, x_n) \)”

it is to be understood as an abbreviation of a standard formula of \( PA \) saying that \( z \) equals the value of \( \tau(x_1, \ldots, x_n) \). Such a formula is “standard” in the sense that \( PA \) knows the construction of \( \tau \). That is, for instance, if the root of \( \tau(x) \) has the label \( + \) and the two tree-terms rooted at the children of the root are \( \theta_1(x) \) and \( \theta_2(x) \), then \( PA \models \forall x \tau(x) = \theta_1(x) + \theta_2(x) \).

Every explicit elementary recursive function \( \tau \) can be translated, in a standard way, into an equivalent (“equivalent” in the sense of representing the same function) unary elementary recursive tree-term, which we here shall denote by \( \tau^* \). This allows us to identify \( \tau \) with \( \tau^* \) and treat the former, just like the latter, as a pseudoterm. Namely, when writing “\( z = \tau \)” within a formula of the language of CLA6, it is to be understood as “\( z = \tau^* \).”

Fact 7.2 For any explicit elementary recursive function \( \tau \) (not containing \( z \)), CLA6 ⊨ \( \square z (z = \tau) \).

Proof. As we know, “\( \square z (z = \tau) \)” means “\( \square z (z = \sigma) \)”, where \( \sigma = \tau^* \). We prove CLA6 ⊨ \( \square z (z = \sigma) \) by (meta)induction on the complexity of the elementary recursive tree-term \( \sigma \). The base cases and the cases of \( \sigma \) being \( \theta' \), \( \theta_1 + \theta_2 \) or \( \theta_1 \times \theta_2 \) are handled as in the proof of Lemma 12.6 of [8]. The case of \( \sigma \) being \( EXP(\theta) \) is also similar. Namely, by the induction hypothesis, CLA6 proves \( \square z (z = \theta) \). And, by Fact 7.1 CLA6 also proves \( \square z (z = EXP(x)) \). These two easily imply the desired \( \square z (z = EXP(\theta)) \) by LC. ■

The rest of this section is devoted to a proof of the extensional completeness of CLA6. Our argument here is almost the same as in Section 4, which, in turn, as we remember, mainly consisted in showing that all relevant lemmas employed in the similar completeness proof of CLA4 given in [8] continued to hold in the present case as well.
We pick an arbitrary elementary-recursively-solvable arithmetical problem \( A \) and a sentence \( X \) with \( A = X^\dagger \). For the rest of this section, we fix \( X \), and fix \( \mathcal{X} \) as an HPM that solves \( A \) (and hence \( X^\dagger \)) in elementary recursive time. Specifically, we assume that \( \mathcal{X} \) runs in time \( \chi \), where \( \chi \), which we also fix for the rest of this section, is an explicit elementary recursive function. We also agree that, throughout the rest of this section, “formula” exclusively means a subformula of \( X \), in which some variables may be renamed.

Our goal is to construct a sentence \( X \) for which, just like for \( X \), we have \( A = X^\dagger \) and which, perhaps unlike \( X \), is provable in CLA6.

Again, remember the sentence \( L \) from Section 14.3 of [8], saying that \( X \) does not win \( X \) in time \( \chi \). We adopt this meaning of \( L \) without any changes (only now \( \chi \) is our present \( \chi \) rather than that of [8], of course).

The overline notation introduced in Section 14.4 of [8] also retains its old meaning without any changes. And the same holds for the single-circle and double-circle notations \( E^\circ(z,\vec{s}) \) and \( E^\circ\circ(z,\vec{s}) \) introduced in Section 14.5 of [8]. Since all relevant concepts here are the same as in [8], we have:

**Lemma 7.3** Lemmas 14.2 through 14.5 of [8] continue to hold in our present case (with “CLA6” instead of “CLA4”).

Namely, according to Lemma 14.2 of [8], we have \( X^\dagger = \overline{X}^\dagger \). So, what now remains to do for the completion of our completeness proof is to show that \( \text{CLA6} \vdash \overline{X} \).

In the subsequent two lemmas, the notational and terminological conventions of Appendix A of [8] are adopted without any changes.

**Lemma 7.4** Lemmas A.1 through A.7 of [8] continue to hold (with “CLA6” instead of “CLA4”).

**Proof.** The proofs of those lemmas given in [8] go through without any changes, as the differences between our present context and the context of [8] are fully irrelevant to them.

**Lemma 7.5** Lemma 14.6 of [8] continues to hold. That is:

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

(a) For every \((\bot, y)\)-development \( H_i(y, \vec{s}) \) of \( E(\vec{s}) \), \( \text{CLA6} \) proves \( E^\circ(z,\vec{s}) \rightarrow \sqcup u H^\circ_i(u,y,\vec{s}) \).

(b) Where \( H_1(y,\vec{s}),...,H_n(y,\vec{s}) \) are all of the \((\top, y)\)-developments of \( E(\vec{s}) \), \( \text{CLA6} \) proves

\[
E^\circ(z,\vec{s}) \rightarrow E^\circ(z,\vec{s}) \sqcup \sqcup \sqcup \sqcup \sqcup H^\circ_1(u,y,\vec{s}) \sqcup \ldots \sqcup \sqcup \sqcup H^\circ_n(u,y,\vec{s}) .
\]

**Proof.** The proof of clause (a) of Lemma 14.6, given in Section A.2 of [8], only relies on Lemmas A.4 and A.6 of [8], which (by our Lemma 7.4), continue to hold in the present case. So, clause (a) of our lemma is taken care of.

The proof of clause (b) of Lemma 14.6, given in Section A.3 of [8], only relies on Lemmas A.3, A.5, A.6, A.7 and Fact 12.6 of [8]. By our Lemma 7.4 those lemmas continue to hold in the present case. And Fact 12.6 of [8] we now replace by Fact 7.2 of the present paper. With this adjustment, the rest of this proof is virtually the same as the proof given in Section A.3 of [8]. So, clause (b) of our lemma is also taken care of.

**Lemma 7.6** Lemma 14.7 of [8] continues to hold. That is:

Assume \( E(\vec{s}) \) is a formula all of whose free variables are among \( \vec{s} \). Then \( \text{CLA6} \) proves \( E^\circ(z,\vec{s}) \rightarrow \overline{E(\vec{s})} \).

**Proof.** The proof of Lemma 14.7 of [8] goes through here without any changes, taking into account that Lemmas 14.3, 14.4, 14.5 and 14.6 of [8] on which the latter relies continue to hold in our present case according to Lemmas 7.3 and 7.5.

Now we can claim the target result of this section in exactly the same way as in the last paragraph of Section 14 of [8] or the last paragraph of Section 4 of the present paper.
8 CLA7, a theory of primitive recursive computability

The language of CLA7 is the same as those of CLA5, CLA6, and so are its axioms and the logical rule LC. In addition, just like CLA5 and CLA6, theory CLA7 has a single nonlogical rule, which we call CLA7-Induction:

\[
\begin{align*}
\Box(F(0)) & \quad \Box(F(x) \rightarrow F(x')) \\
\Box(F(x)) & 
\end{align*}
\]

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

Thus, the only difference between CLA7 and CLA5 or CLA6 is that, while the induction rules of the latter require the formula \( F(x) \) to be polynomially or exponentially bounded, the induction rule of the former imposes no restrictions on \( F(x) \) at all.

Fact 8.1 Every CLA6-provable (and hence also every CLA4-provable and every CLA5-provable) sentence is provable in CLA7.

Proof. This is straightforward, as CLA6-Induction is a special case of CLA7-Induction.

Let \( f \) be a function letter of the language of CL12 of indicated (by the number of explicitly shown arguments) arity.

An absolute primitive recursive definition of \( f \) is a CLA12-formula of one of the following forms:

(I) \( \forall x (f(x)=x') \).

(II) \( \forall x_1 \ldots \forall x_n (f(x_1, \ldots, x_n) = 0) \).

(III) \( \forall x_1 \ldots \forall x_n (f(x_1, \ldots, x_n) = x_i) \) (some \( i \in \{1, \ldots, n\} \)).

And a relative primitive recursive definition of \( f \) is a CLA12-formula of one of the following forms:

(IV) \( \forall x_1 \ldots \forall x_n \left( f(x_1, \ldots, x_n) = g(h_1(x_1, \ldots, x_n), \ldots, h_m(x_1, \ldots, x_n)) \right) \).

(\( \forall x_2 \ldots \forall x_n (f(0, x_2, \ldots, x_n) = g(x_2, \ldots, x_n)) \) \( \land \)

(V) \( \forall x_1 \forall x_2 \ldots \forall x_n \left( f(x_1', x_2, \ldots, x_n) = h(x_1, f(x_1, x_2, \ldots, x_n), x_2, \ldots, x_n) \right) \).

We say that (IV) defines \( f \) in terms of \( g, h_1, \ldots, h_m \). Similarly, we say that (V) defines \( f \) in terms of \( g \) and \( h \).

A primitive recursive construction of \( f \) is a sequence \( E_1, \ldots, E_k \) of CLA12-formulas, where each \( E_i \) is a primitive recursive definition of some \( g_i \), all such \( g_i \) are distinct, \( g_k = f \) and, for each \( i \), \( E_i \) is either an absolute primitive recursive definition of \( g_i \), or a relative primitive recursive definition of \( g_i \) in terms of some \( g_j, g_s \) with \( j < i \).

Terminologically, we will usually identify a primitive recursive construction of a function \( f \) with the function \( f \) itself. Further, to keep our terminology uniform, we will be using the words “explicit primitive recursive function” as a synonym of “primitive recursive construction of a unary function”. When \( \tau \) is an explicit primitive recursive function and \( M \) is a \( \tau \) time (resp. space) machine, we say that \( \tau \) is an explicit primitive recursive bound for the time (resp. space) complexity of \( M \).

We say that a given HPM \( M \) runs in primitive recursive time (resp. space) iff there is an explicit primitive recursive function \( \tau \) such that \( M \) runs in time (resp. space) \( \tau \). And we say that a given problem has a primitive recursive solution iff it has a solution that runs in primitive recursive time. The reason why we omitted the word “time” here is that, as in the case of elementary recursiveness, it is not hard to see that a problem has a primitive recursive time solution if and only if it has a primitive recursive space solution.

Theorem 8.2 An arithmetical problem has a primitive recursive solution iff it is provable in CLA7.

Furthermore, there is an efficient procedure that takes an arbitrary extended CLA7-proof of an arbitrary sentence \( X \) and constructs a solution of \( X \) (of \( X' \), that is) together with an explicit primitive recursive bound for its time complexity.

Proof. The soundness (“if”) part of this theorem will be proven in Section \( \| \) and the completeness (“only if”) part in Section \( \| \).
9 The soundness of CLA7

As in Sections 3 and 6 we will limit ourselves to proving the pre-“furthermore” part (of the soundness part) of Theorem 8.2. Consider any CLA7-provable sentence $X$. We proceed by induction on its proof.

The case of $X$ being an axiom is handled in the same way as in the soundness proofs for the previous systems. So is the case of $X$ being obtained by LC.

For the rest of this section, suppose $X$ is (the $\sqcap$-closure of) $F(x)$, and $X$ is obtained by CLA7-Induction on $x$. So, the premises are (the $\sqcap$-closures of) $F(0)$ and $F(x) \rightarrow F(x')$. By the induction hypothesis, there are HPMs $\mathcal{N}$ and $\mathcal{K}$ — with explicit primitive recursive bounds $\xi, \zeta$ for their time complexities, respectively — that solve these two premises, respectively. We replace $\xi$ and $\zeta$ by one common bound $\phi = \xi + \zeta$ for the time complexities of both $\mathcal{N}$ and $\mathcal{K}$.

As was done earlier, we will assume that the adversary of the purported solution $\mathcal{M}$ of $F(x)$ that we are going to construct never makes illegal moves. From our description of $\mathcal{M}$ it will be clear that, as long as Environment plays legally, so does $\mathcal{M}$.

To describe $\mathcal{M}$, as before, assume $x, \vec{v}$ are exactly the free variables of $F(x)$, so that $F(x)$ can be rewritten as $F(x, \vec{v})$. At the beginning, $\mathcal{M}$ waits for Environment to choose constants for the free variables of $F(x, \vec{v})$. Assume $k$, with $k \geq 1$ (the case of $k = 0$ is straightforward), is the constant chosen for the variable $x$, and $\vec{c}$ are the constants chosen for $\vec{v}$. From now on, we shall write $F'(x)$ for $F(x, \vec{c})$. Further, we shall write $\mathcal{N}_0$ 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 constants $\vec{c}$ for the variables $\vec{v}$. So, $\mathcal{N}_0$ wins $F'(0)$. Similarly, for any $i \geq 1$, we will write $\mathcal{K}_i$ for the machine that works just like $\mathcal{K}$ does in the scenario where the adversary, at the beginning of the play, has chosen the constants $\vec{c}$ for the variables $\vec{v}$ and the constant $i-1$ for the variable $x$. So, $\mathcal{K}_i$ wins $F'(i-1) \rightarrow F'(i)$. Similarly, we will write $\mathcal{M}_k$ for the machine that works just like $\mathcal{M}$ does after the above event of Environment’s having chosen $k$ and $\vec{c}$ for $x$ and $\vec{v}$, respectively. So, in order to complete our description of $\mathcal{M}$, it would suffice do simply define $\mathcal{M}_k$ and say that, after Environment has chosen constants for all free variables of $F(x)$, $\mathcal{M}$ continues playing as $\mathcal{M}_k$.

The work of $\mathcal{M}_k$ consists in continuously polling its run tape to see if Environment has made any new moves, combined with simulating, in parallel, one play of $F'(0)$ by $\mathcal{N}_0$ and — for each $i \in \{1, \ldots, k\}$ — one play of $F'(i-1) \rightarrow F'(i)$ by $\mathcal{K}_i$. In this mixture of one real and $k+1$ imaginary plays, $\mathcal{M}$ synchronizes $k+1$ pairs of (sub)games, real or imaginary. Namely:

- It synchronizes the consequent of the imaginary play of $F'(k-1) \rightarrow F'(k)$ by $\mathcal{K}_k$ with the real play of $F'(k)$.
- For each $i \in \{1, \ldots, k-1\}$, it synchronizes the consequent of the imaginary play of $F'(i-1) \rightarrow F'(i)$ by $\mathcal{K}_i$ with the antecedent of the imaginary play of $F'(i) \rightarrow F'(i+1)$ by $\mathcal{K}_{i+1}$.
- It synchronizes the imaginary play of $F'(0)$ by $\mathcal{N}_0$ with the antecedent of the imaginary play of $F'(0) \rightarrow F'(1)$ by $\mathcal{K}_1$.

This completes our description of $\mathcal{M}_k$ and hence of $\mathcal{M}$. Remembering our assumption that $\mathcal{N}, \mathcal{K}$ win the corresponding games, it is obvious that $\mathcal{M}_k$ wins $F'(k)$ and hence $\mathcal{M}$ wins $\sqcap \{F(x)\}$, as desired. It now remains to show that the time complexity of $\mathcal{M}$ is also as desired.

For the rest of this proof, pick and fix an arbitrary play of $\mathcal{M}$, and an arbitrary clock cycle $\epsilon$ on which $\mathcal{M}$ makes a move $\alpha$ in the real play of $F(x)$. Let $h$ and $\ell$ be the timecost and the background of this move, respectively. Let $k, F'(x), \mathcal{N}_0, \mathcal{K}_1, \ldots, \mathcal{K}_k, \mathcal{M}_k$ be as in the description of the work of $\mathcal{M}$.

As done before in similar proofs, we ignore the polling, simulation and copycat overhead, and also pretend that the polling and the several simulations happen in a truly parallel fashion, in the sense that $\mathcal{M}$ spends a single clock cycle on tracing a single computation step of all $k+1$ machines simultaneously, as well as on checking out its run tape to see if Environment has made a new move.

Let $\beta_1, \ldots, \beta_m$ be the moves by simulated machines that $\mathcal{M}_k$ detects by time $\epsilon$, arranged according to the times $t_1 \leq \ldots \leq t_m$ of their detections (which, by our simplifying assumptions, coincide with the timestamps of those moves in the corresponding simulated plays). Let $\mathcal{H}_1, \ldots, \mathcal{H}_m \in \{\mathcal{N}_0, \mathcal{K}_1, \ldots, \mathcal{K}_k\}$ be the machines that made these moves, respectively. Let $d = \epsilon - h$. Let $j$ be the smallest integer among $1, \ldots, m$ such that $t_j \geq d$. Note that the background of the play by $\mathcal{K}_j$ at time $j$ does not exceed $\ell$. And, since $\mathcal{K}_j$ runs in time $\phi$, neither the size of $\beta_j$ nor $t_j - d$ exceed $\phi(\ell)$. For similar reasons, with $\phi(\ell)$ now acting in the role of $\ell$, the work of $\mathcal{K}_j$}
neither the size of $β_{j+1}$ nor $t_{j+1} - t_j$ exceed $ϕ(ϕ(ℓ))$. Therefore, neither the size of $β_j$ nor $t_j$ exceed $2ϕ(ϕ(ℓ))$. Similarly, neither the size of $β_{j+2}$ nor $t_{j+2} - t_j$ exceed $3ϕ(ϕ(ℓ))$). And so on. Thus, neither the size of $β_m$ nor $t_m - d$ exceed $(m - j + 1) × ϕ_ {m-j+1}(ℓ)$ and hence (as $m, j ≥ 1$) $m × ϕ_m(ℓ)$, where $ϕ_m$ means the $m$-fold composition of $ϕ$ with itself. Also note that $m$ cannot exceed $2^ℓ × d$, where $d$ is the depth of $F(x)$. We conclude that neither the size of $β_m$ nor $h$ exceed $2^ℓ × d × ϕ^2 × ϕ(ℓ)$. But notice that $β_m$ is a move made by $K_k$ in the consequent of $F′(k - 1) → F(k)$, immediately (by our simplifying assumptions) copied by $M_k$ in the real play when it made its move $α$. In other words, $x = t_m$. Thus, (as long as we pretend that there is no polling/simulation/copycat overhead) neither the timecost nor the size of $α$ exceed $2^ℓ × d × ϕ^2 × ϕ(ℓ)$.

Obviously an upper bound for the above function $2^ℓ × d × ϕ^2 × ϕ(ℓ)$, even after “correcting” the latter so as to precisely account for the so far suppressed polling/simulation/copycat overhead, can be expressed as an explicit primitive recursive function $τ(ℓ)$. This is exactly the sought explicit primitive recursive bound for the time complexity of $M$.

10 The extensional completeness of CLA7

We treat each $n$-ary primitive recursive construction $τ = τ(x_1, . . . , x_n)$ as a pseudoterm and, when we write “$z = τ(x_1, . . . , x_n)$” (or just “$z = τ$”) within a formula, it is to be understood as an abbreviation of a standard formula of $PA$ saying that $z$ equals the value of $τ(x_1, . . . , x_n)$. Such a formula is “standard” in the sense that $PA$ knows the definition of $τ$. That is, for instance, if $τ(x)$ is defined (in its primitive recursive construction) by $∀x (τ(x) = θ(ϕ(x)))$, then $PA ⊢ ∀x (τ(x) = θ(ϕ(x)))$.

Fact 10.1 For any explicit primitive recursive function $τ$ (not containing $z$), $CLA7 ⊢ ∃z(z = τ)$.

Proof. We generalize the above statement by allowing $τ$ to be a primitive recursive construction of any (not necessarily unary) function, and prove such a generalized statement by metainduction on the complexity of (the construction of) $τ$. This requires considering the five cases I-V from Section 8, depending on which of them applies last in the construction of $τ$.

Case I: $τ$ is a function defined by $∀x (τ(x) = x′)$. This sentence is thus provable in $PA$. By Axiom 8, $CLA7$ also proves $∀x ∃y (y = x′)$. The desired $∃z (z = τ(x))$ is an easy logical consequence of these two.

Case II: $τ$ is a function defined by $∀x_1 . . . ∀x_n (τ(x_1, . . . , x_n) = 0)$. This sentence is thus provable in $PA$. The target $∃z (z = τ(x_1, . . . , x_n))$ is an immediate logical consequence of it.

Case III: $τ$ is a function defined by $∀x_1 . . . ∀x_n (τ(x_1, . . . , x_n) = x_i)$. Similar to the preceding case.

Case IV: $τ$ is a function defined by $∀x_1 . . . ∀x_n (τ(x_1, . . . , x_n) = ϕ(ψ_1(x_1, . . . , x_n), . . . , ψ_m(x_1, . . . , x_n)))$. This sentence is thus provable in $PA$. By the induction hypothesis, $CLA7$ also proves $∃z (z = ϕ(x_1, . . . , x_m))$ and — for each $i ∈ \{1, . . . , m\}$ — $∃z (z = ψ_i(x_1, . . . , x_n))$. These provabilities can be seen to imply the provability of the target $∃z (z = τ(x_1, . . . , x_n))$ by LC.

Case V: $τ$ is a function defined by

\[
∀x_1 . . . ∀x_n (τ(x_1, x_2, . . . , x_n) = ϕ(x_1, τ(x_1, x_2, . . . , x_n))) \land
∀x_1 ∀x_2 . . . ∀x_n (τ(x_1′, x_2, . . . , x_n) = ϕ(τ(x_1, x_1′, x_2, . . . , x_n), x_2, . . . , x_n)),
\]

(17)

so that the above sentence is provable in $PA$. By the induction hypothesis, $CLA7$ also proves both of the following:

\[
∃z (z = θ(x_2, . . . , x_n));
\]

(18)

\[
∃z (z = ϕ(x_0, . . . , x_n)).
\]

(19)

By LC, (17), (18) and (19) can be seen to imply both of the following:

\[
∃z (z = τ(0, x_2, . . . , x_n));
\]

(20)

\[
∃z (z = τ(x_1, x_2, . . . , x_n)) \rightarrow ∃z (z = τ(x_1′, x_2, . . . , x_n)).
\]

(21)
Now, the target \( \forall z(z = \tau(x_1, x_2, \ldots, x_n)) \) follows from (20) and (21) by CLA7-Induction on \( x_1 \).

The rest of our completeness proof for CLA7 is literally the same as the completeness proof for CLA6 found in Section 7, with the only difference that now \( \chi \) is an explicit primitive recursive (rather than elementary recursive) function; also, where Section 7 relied on Fact 7.2 now we rely on Fact 10.1 instead.

11 On the intensional strength of CLA5, CLA6 and CLA7

The following theorem is proven in literally the same way as Theorem 16.1 of [8]:

**Theorem 11.1** Let \( X \) and \( \mathbb{L} \) be as in Section 4 (resp. Section 7, resp. Section 10). Then CLA5 (resp. CLA6, resp. CLA7) proves \( \neg \mathbb{L} \rightarrow X \).

So, whatever was said in Section 16 of [8] about the import of this theorem, extends to our present systems CLA5, CLA6 and CLA7 as well. This includes Theorem 16.2 of [8]. To re-state that theorem, we extend the earlier concept of constructive provability to our present complexity classes. Namely, we say that \( \text{PA constructively proves} \) the polynomial space (resp. elementary recursive, resp. primitive recursive) computability of a sentence \( X \) iff, for some particular HPM \( \mathcal{X} \) and some particular explicit polynomial (resp. elementary recursive, resp. primitive recursive) function \( \chi \), \( \text{PA} \) proves that \( \mathcal{X} \) is a \( \chi \)-space (resp. \( \chi \)-time or \( \chi \)-space) solution of \( X \).

Then the following theorem is proven in literally the same way as Theorem 16.2 of [8]:

**Theorem 11.2** Let \( X \) be any sentence such that \( \text{PA constructively proves} \) the polynomial space (resp. elementary recursive, resp. primitive recursive) computability of \( X \). Then CLA5 (resp. CLA6, resp. CLA7) proves \( X \).

References

[1] S. Buss. *Bounded Arithmetic* (revised version of Ph. D. thesis). Bibliopolis, 1986.

[2] P. Clote and G. Takeuti. *Bounded arithmetic for NC, ALogTIME, L and NL*. Annals of Pure and Applied Logic 56 (1992), pp. 73-117.

[3] G. Japaridze. *The logic of interactive Turing reduction*. Journal of Symbolic Logic 72 (2007), pp. 243-276.

[4] G. Japaridze. *Intuitionistic computability logic*. Acta Cybernetica 18 (2007), pp. 77-113.

[5] G. Japaridze. *The intuitionistic fragment of computability logic at the propositional level*. Annals of Pure and Applied Logic 147 (2007), pp.187-227.

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

[7] G. Japaridze. *A logical basis for constructive systems*. Journal of Logic and Computation (in press). doi: 10.1093/logcom/exr009, 38 pages.

[8] G. Japaridze. *Introduction to clarithmetic I*. Information and Computation 209 (2011), pp. 1312-1354.