A logical basis for constructive systems

Giorgi Japaridze

School of Computer Science and Technology, Shandong University, PRC;
Department of Computing Sciences, Villanova University, USA

Abstract

The work is devoted to *Computability logic* (CoL) — the philosophical/mathematical platform and long-term project for redeveloping classical logic after replacing *truth* by *computability* in its underlying semantics. This article elaborates some basic complexity theory for the CoL framework. Then it proves soundness and completeness for the deductive system **CL12** with respect to the semantics of CoL, including the version of the latter based on polynomial time computability instead of computability-in-principle. **CL12** is a sequent calculus system, where the meaning of a sequent intuitively can be characterized as "the succedent is algorithmically reducible to the antecedent", and where formulas are built from predicate letters, function letters, variables, constants, identity, negation, parallel and choice connectives, and blind and choice quantifiers. A case is made that **CL12** is an adequate logical basis for constructive applied theories, including complexity-oriented ones.

MSC: primary: 03F50; secondary: 03D75; 03D15; 03D20; 68Q10; 68T27; 68T30

Keywords: Computability logic; Interactive computation; Implicit computational complexity; Game semantics; Constructive logics; Efficiency logics

1 Introduction

Computability logic, to which this contribution is devoted and which we shall henceforth refer to as **CoL**, was introduced in [10, 14, 23] as a semantically conceived open-ended framework and long-term research project for redeveloping logic as a formal theory of *computability*. That is as opposed to the more traditional view of logic as a formal theory of *truth*. The expressions of CoL — formulas, sequents, cirquents — stand for interactive computational problems, understood as games played by a machine against its environment, and computability of such problems means existence of a machine that always wins. The main ambition of the overall CoL project is to provide ever more expressive and powerful tools for systematically telling what can be computed and how, just as classical logic is a systematic tool for telling what is true.

Finding new converts is not among the pursuits of the present work. Numerous articles have been published on the subject in recent years ([10]-[26],[28]), and the reader is assumed to have some basic familiarity with the philosophy, motivations and techniques of CoL. If not, he or she may want to take a look at the first 10 sections of [23] for a tutorial-style introduction and survey. Doing so would be helpful even if not technically necessary, as this paper provides all relevant definitions.

The single deductive system dealt with in the present paper is **CL12**. Its formulas are built in the standard way from predicate and function letters, variables, constants, identity $=$, negation $\neg$, parallel connectives $\wedge$, $\vee$, $\rightarrow$, choice connectives $\sqcap$, $\sqcup$, blind quantifiers $\forall, \exists$ and choice quantifiers $\bigsqcap, \bigsqcup$. **CL12** is a sequent calculus system, where every sequent looks like

$$E_1, \ldots, E_n \circ\!\!- F$$

($n \geq 0$; the E_i and F are formulas),[1] semantically understood as an abbreviation of

$$\overset{|}{\circ} E_1 \wedge \ldots \wedge \overset{|}{\circ} E_n \rightarrow F,$$

[1]The unfortunate coincidence — or rather symmetry — between our sequent symbol and Girard's symbol for linear implication is merely graphical. The meaning of the latter is closer to that of our $\rightarrow$.

with $⫰$ being the ordinary *branching recurrence* operator. The system is shown to be sound and complete in the sense that a sequent is **CL12**-provable if and only if it has a uniform ("purely logical") solution, i.e. an algorithmic strategy that wins the game/sequent under any interpretation of its non-logical components such as predicate and function letters. Furthermore, such a strategy can be effectively — in fact, efficiently — extracted from a proof of the sequent.

Logic **CL12** was first introduced in [24], where it was proven to be sound and complete in the above sense, but with $\bullet\!\!\!\not\!-$ instead of our present $\circ\!\!-$. The former is a version of the latter that only allows re-using antecedental resources a finite number of times (otherwise the game is considered lost even if its succedent part is won). Successfully switching from $\bullet\!\!\!\not\!-$ to $\circ\!\!-$ is a significant advance from both the philosophical and technical points of view. As repeatedly argued in the earlier literature on CoL, it is $A \circ\!\!- B$ — rather than the stronger $A \bullet\!\!\!\not\!- B$, $A \succ\!\!- B$, $A \rightarrow B$, etc. — that adequately captures our ultimate, most general intuition of algorithmically reducing B to A. Correspondingly, it is uniform validity of $E_1, \ldots, E_n \circ\!\!- F$ rather than of $E_1, \ldots, E_n \bullet\!\!\!\not\!- F$ that corresponds to our ultimate intuition of "purely logically" reducing the succedent to the antecedent. It can be characterized in other words as "the succedent is a *logical consequence* of the antecedent". It is exactly this intuition that is of paramount importance when dealing with **CL12**-based applied theories such as the version **CLA1** of Peano arithmetic constructed in [24]. Completeness with respect to "logical consequence" guarantees that one can reliably use intuition on games and strategies to prove results in the theory. Without having such a guarantee, one would be forced to resort to point-by-point syntactic derivations, which could make successfully studying and developing **CL12**-based applied theories next to impossible. Thus, while [24] had chosen "the right logic" **CL12** as a basis for its system of arithmetic, this choice was made just by good luck, as no justification for it was provided or found.

The above was about why and how the present paper strengthens the completeness result of [24] for **CL12**: $A \circ\!\!- B$ is generally easier to win than $A \bullet\!\!\!\not\!- B$, and hence the corresponding completeness theorem is harder to prove. But this paper also strengthens — perhaps in an even more important way — the soundness result of [24]. For the first time in CoL's history, it brings computational complexity into the framework of the project. Namely, certain natural concepts of time and space complexities of winning strategies are defined for games. It is shown that **CL12** remains sound (as well as complete, of course) if one considers polynomial time computability instead of computability-in-principle, and that the associated "logical consequence" relation preserves polynomial time computability, as well as Ω-time and Ω-space computabilities for any class Ω of functions containing all polynomial functions and closed under composition. This opens a whole new world of potential applications of CoL in general and **CL12** in particular. One can construct and explore, in a systematic way, not only computability-oriented applied theories such as the above-mentioned arithmetic **CLA1**, but complexity-oriented theories as well, among the best known earlier examples of which is Buss's [5] bounded arithmetic. For instance, a **CL12**-based arithmetic for polynomial time computability will be an extension of classical Peano arithmetic. The single logical rule of inference of it, as well as of any other **CL12**-based theory, would be

From $E_1, \ldots, E_n$ conclude F, as long as **CL12** *proves $E_1, \ldots, E_n \circ\!\!- F$.*

Extra-Peano nonlogical axioms of such a system would be some polynomial time computable formulas/problems, such as, say, $\sqcap x \sqcup y (y = x+1)$. And nonlogical rules of inference, if any, would be rules preserving the property of polynomial time computability. Then every theorem of the system, seen as an arithmetical problem, will be polynomial time computable, with a polynomial time solution of the problem extractable from the proof. The author expects that certain simple and elegant systems in this style can achieve not merely soundness, but completeness as well, in the sense that every arithmetical problem with a polynomial time solution is expressed by some theorem of the system. In the same style, by varying nonlogical extra-Peano axioms and rules, one can construct and study systems for polynomial space computability, elementary recursive computability, primitive recursive computability, provably recursive computability, etc.

A notable advantage of **CL12**-based arithmetics over the other complexity-oriented systems such as the earlier versions of bounded arithmetic and including those based on intuitionistic logic ([6, 27]), would be preserving the full expressiveness and deductive power of classical arithmetic while still being computationally and complexity-theoretically sound and meaningful. Every such system can be seen as a programming language where (efficient) programs can be automatically extracted from proofs, with "programming" thus simply meaning theorem-proving[2] and with the (generally undecidable) problem of whether a program meets

[2]In a more ambitious and, at this point, somewhat fantastic perspective, after developing reasonable theorem-provers for

its specification being fully neutralized. Hence the importance of the just-mentioned advantage of **CL12**-based systems over the other, known systems with similar aspirations — which typically happen to be inherently weak theories — is obvious: the stronger a system, the better the chances that a proof/program will be found for a declarative, non-preprocessed, ad hoc specification. Among the virtues of CoL is that it allows us to achieve constructive heights without throwing out the baby (such as classical logic or Peano arithmetic) with the bath water.

The main purpose of the present publication is to provide a logical basis and reusable point of departure for developing complexity-oriented applied theories in the above style — the new line of research where the author predicts significant and fruitful activities in the near future.

2 Remembering constant games and some operations on them

Even though the reader is expected to have some prior familiarity with CoL, for the sake of safety and convenience of references, here we reproduce the basic relevant definitions. It should also be noted that certain old concepts — such as that of a non-constant game — have been substantially generalized in this paper. On the other hand, certain other concepts, such as that of an HPM, have been simplified at the expense of (here unnecessary) generality. The definitions of the basic operations on games given in the present section are different from — yet equivalent to — the definitions of the same operations found elsewhere.

Computational problems are understood as games between two players: $\top$ (Machine) and $\bot$ (Environment). A **move** means any finite string over the standard keyboard alphabet. A **labeled move** (**labmove**) is a move prefixed with $\top$ or $\bot$, with such a prefix (**label**) indicating which player has made the move. A **run** is a (finite or infinite) sequence of labmoves, and a **position** is a finite run.

We will be exclusively using the letters Φ, Γ, Δ for runs, and α, β for moves. The letter $\wp$ will always be a variable for players, and

$$\overline{\wp}$$

will mean "$\wp$'s adversary" ("the other player"). Runs will be often delimited by "$\langle$" and "$\rangle$", with $\langle\rangle$ thus denoting the **empty run**. The meaning of an expression such as $\langle \Phi, \wp\alpha, \Gamma\rangle$ must be clear: this is the result of appending to the position $\langle\Phi\rangle$ the labmove $\langle\wp\alpha\rangle$ and then the run $\langle\Gamma\rangle$.

The following is a formal definition of constant games, combined with some less formal conventions regarding the usage of certain terminology.

Definition 2.1 A **constant game** is a pair $A = (\mathbf{Lr}^A, \mathbf{Wn}^A)$, where:

1. $\mathbf{Lr}^A$ is a set of runs satisfying the condition that a (finite or infinite) run is in $\mathbf{Lr}^A$ iff all of its nonempty finite initial segments are in $\mathbf{Lr}^A$ (notice that this implies $\langle\rangle \in \mathbf{Lr}^A$). The elements of $\mathbf{Lr}^A$ are said to be **legal runs** of A, and all other runs are said to be **illegal**. We say that α is a **legal move** for $\wp$ in a position Φ of A iff $\langle \Phi, \wp\alpha\rangle \in \mathbf{Lr}^A$; otherwise α is **illegal**. When the last move of the shortest illegal initial segment of Γ is $\wp$-labeled, we say that Γ is a **$\wp$-illegal** run of A.

2. $\mathbf{Wn}^A$ is a function that sends every run Γ to one of the players $\top$ or $\bot$, satisfying the condition that if Γ is a $\wp$-illegal run of A, then $\mathbf{Wn}^A\langle\Gamma\rangle = \overline{\wp}$. When $\mathbf{Wn}^A\langle\Gamma\rangle = \wp$, we say that Γ is a **$\wp$-won** (or **won by** $\wp$) run of A; otherwise Γ is **lost** by $\wp$. Thus, an illegal run is always lost by the player who has made the first illegal move in it.

A constant game A is said to be **elementary** iff $\mathbf{Lr}^A = \{\langle\rangle\}$, i.e., A does not have any nonempty legal runs. There are exactly two elementary constant games: $\top$ with $\mathbf{Wn}^{\top}\langle\rangle = \top$, and $\bot$ with $\mathbf{Wn}^{\bot}\langle\rangle = \bot$. Standard true sentences, such as "snow is white" or "0=0", are understood as the game $\top$, and false sentences, such as "snow is black" or "0=1", as the game $\bot$. Correspondingly, the two games $\top$ and $\bot$ will be referred to as **propositions**.

Let us remember the operation of *prefixation*. It takes two arguments: a constant game A and a position Φ that must be a legal position of A (otherwise the operation is undefined), and returns the game $\langle\Phi\rangle A$.

efficiency-oriented arithmetics, "programming" would simply mean stating the goal/specification — i.e., writing a formula that represents the computational problem whose efficient solution is sought for systematic usage in the future. The compiler's job would be finding a proof (the hard part) and translating it into a solution (the easy part). The process of compiling could thus take long but, once compiled, the program would run fast ever after.

Intuitively, $\langle\Phi\rangle A$ is the game playing which means playing A starting (continuing) from position Φ. That is, $\langle\Phi\rangle A$ is the game to which A **evolves** (will be "**brought down**") after the moves of Φ have been made. Here is a definition:

Definition 2.2 Let A be a constant game and Φ a legal position of A. The game $\langle\Phi\rangle A$ is defined by:

- $\mathbf{Lr}^{\langle\Phi\rangle A} = \{\Gamma \mid \langle\Phi,\Gamma\rangle \in \mathbf{Lr}^{A}\}$;
- $\mathbf{Wn}^{\langle\Phi\rangle A}\langle\Gamma\rangle = \mathbf{Wn}^{A}\langle\Phi,\Gamma\rangle$.

Convention 2.3 A terminological convention important to remember is that we often identify a legal position Φ of a game A with the game $\langle\Phi\rangle A$. So, for instance, we may say that the move 1 by $\bot$ brings the game $B_0 \sqcap B_1$ down to the position B_1. Strictly speaking, B_1 is not a position but a game, and what *is* a position is $\langle\bot 1\rangle$, which we here identified with the game $B_1 = \langle\bot 1\rangle(B_0 \sqcap B_1)$.

Note that, in order to define the **Lr** component of a constant game A, it would suffice to specify what the **initial legal (lab)moves** — i.e., the elements of $\{\wp\alpha \mid \langle\wp\alpha\rangle \in \mathbf{Lr}^{A}\}$ — are, and to what game the game A is brought down after such an initial legal labmove $\wp\alpha$ is made (this can be seen to hold even in recursive definitions of game operations as in clauses 1 and 3 of Definition 2.4 below). Then, the set of legal runs of A will be uniquely defined. Similarly, note that defining the **Wn** component for only legal runs of A would be sufficient, for then it uniquely extends to all runs. With these observations in mind, we can (re)define the operations $\neg$, $\wedge$, $\vee$, $\sqcap$, $\sqcup$ in a new fashion as follows:

Definition 2.4 Let A, B, $A_0, A_1, \ldots$ be constant games, and n a positive integer.

1. $\neg A$ (**negation**) is defined by:
   - **(i)** $\langle\wp\alpha\rangle \in \mathbf{Lr}^{\neg A}$ iff $\langle\overline{\wp}\alpha\rangle \in \mathbf{Lr}^{A}$. Such an initial legal labmove $\wp\alpha$ brings the game down to $\neg\langle\overline{\wp}\alpha\rangle A$.
   - **(ii)** Whenever Γ is a legal run of $\neg A$, $\mathbf{Wn}^{\neg A}\langle\Gamma\rangle = \top$ iff $\mathbf{Wn}^{A}\langle\overline{\Gamma}\rangle = \bot$. Here $\overline{\Gamma}$ means Γ with each label $\wp$ changed to $\overline{\wp}$.

2. $A_0 \sqcap \ldots \sqcap A_n$ (**choice conjunction**) is defined by:
   - **(i)** $\langle\wp\alpha\rangle \in \mathbf{Lr}^{A_0 \sqcap \ldots \sqcap A_n}$ iff $\wp = \bot$ and $\alpha = i \in \{0,\ldots,n\}$.[3] Such an initial legal labmove $\bot i$ brings the game down to A_i.
   - **(ii)** Whenever Γ is a legal run of $A_0 \sqcap \ldots \sqcap A_n$, $\mathbf{Wn}^{A_0 \sqcap \ldots \sqcap A_n}\langle\Gamma\rangle = \bot$ iff Γ looks like $\langle\bot i, \Delta\rangle$ ($i \in \{0,\ldots,n\}$) and $\mathbf{Wn}^{A_i}\langle\Delta\rangle = \bot$.

3. $A_0 \wedge \ldots \wedge A_n$ (**parallel conjunction**) is defined by:
   - **(i)** $\langle\wp\alpha\rangle \in \mathbf{Lr}^{A_0 \wedge \ldots \wedge A_n}$ iff $\alpha = i.\beta$, where $i \in \{0,\ldots,n\}$ and $\langle\wp\beta\rangle \in \mathbf{Lr}^{A_i}$. Such an initial legal labmove $\wp i.\beta$ brings the game down to
   $$A_0 \wedge \ldots \wedge A_{i-1} \wedge \langle\wp\beta\rangle A_i \wedge A_{i+1} \wedge \ldots \wedge A_n.$$
   - **(ii)** Whenever Γ is a legal run of $A_0 \wedge \ldots \wedge A_n$, $\mathbf{Wn}^{A_0 \wedge \ldots \wedge A_n}\langle\Gamma\rangle = \top$ iff, for each $i \in \{0,\ldots,n\}$, $\mathbf{Wn}^{A_i}\langle\Gamma^{i.}\rangle = \top$. Here $\Gamma^{i.}$ means the result of removing, from Γ, all labmoves except those that look like $\wp i.\alpha$, and then further changing each such (remaining) $\wp i.\alpha$ to $\wp\alpha$.[4]

4. $A_0 \sqcup \ldots \sqcup A_n$ (**choice disjunction**) and $A_0 \vee \ldots \vee A_n$ (**parallel disjunction**) are defined exactly as $A_0 \sqcap \ldots \sqcap A_n$ and $A_0 \wedge \ldots \wedge A_n$, respectively, only with "$\top$" and "$\bot$" interchanged.

5. The infinite $\sqcap$-conjunction $A_0 \sqcap A_1 \sqcap \ldots$ is defined exactly as $A_0 \sqcap \ldots \sqcap A_n$, only with "$i \in \{0,1,\ldots\}$" instead of "$i \in \{0,\ldots,n\}$". Similarly for the infinite versions of $\sqcup$, $\wedge$ and $\vee$.

6. $A \rightarrow B$ (**strict reduction**) is treated as an abbreviation of $(\neg A) \vee B$.

[3]Here the number i is identified with its binary representation. The same applies to the other clauses of this definition.

[4]Intuitively, $\Gamma^{i.}$ is the run played in the A_i component. The present condition thus means that $\top$ wins a $\wedge$-conjunction of games iff it wins in each conjunct.

We also agree that, when $k = 1$, $A_1 \sqcap \ldots \sqcap A_k$ simply means A_1, and so do $A_1 \sqcup \ldots \sqcup A_k$, $A_1 \wedge \ldots \wedge A_k$ and $A_1 \vee \ldots \vee A_k$. We further agree that, when the set $\{A_1, \ldots, A_k\}$ is empty ($k = 0$, that is), both $A_1 \sqcap \ldots \sqcap A_k$ and $A_1 \wedge \ldots \wedge A_k$ mean $\top$, while both $A_1 \sqcup \ldots \sqcup A_k$ and $A_1 \vee \ldots \vee A_k$ mean $\bot$.

Example 2.5 The game $(0{=}0 \sqcap 0{=}1) \to (10{=}11 \sqcap 10{=}10)$, i.e. $\neg(0{=}0 \sqcap 0{=}1) \vee (10{=}11 \sqcap 10{=}10)$, has thirteen legal runs, which are:

1 $\langle\rangle$. It is won by $\top$, because $\top$ is the winner in the right $\vee$-disjunct (consequent).

2 $\langle \top 0.0 \rangle$. (The labmove of) this run brings the game down to $\neg 0{=}0 \vee (10{=}11 \sqcap 10{=}10)$, and $\top$ is the winner for the same reason as in the previous case.

3 $\langle \top 0.1 \rangle$. It brings the game down to $\neg 0{=}1 \vee (10{=}11 \sqcap 10{=}10)$, and $\top$ is the winner because it wins in both $\vee$-disjuncts.

4 $\langle \bot 1.0 \rangle$. It brings the game down to $\neg(0{=}0 \sqcap 0{=}1) \vee 10{=}11$. $\top$ loses as it loses in both $\vee$-disjuncts.

5 $\langle \bot 1.1 \rangle$. It brings the game down to $\neg(0{=}0 \sqcap 0{=}1) \vee 10{=}10$. $\top$ wins as it wins in the right $\vee$-disjunct.

6-7 $\langle \top 0.0, \bot 1.0 \rangle$ and $\langle \bot 1.0, \top 0.0 \rangle$. Both bring the game down to the false $\neg 0{=}0 \vee 10{=}11$, and both are lost by $\top$.

8-9 $\langle \top 0.1, \bot 1.0 \rangle$ and $\langle \bot 1.0, \top 0.1 \rangle$. Both bring the game down to the true $\neg 0{=}1 \vee 10{=}11$, which makes $\top$ the winner.

10-11 $\langle \top 0.0, \bot 1.1 \rangle$ and $\langle \bot 1.1, \top 0.0 \rangle$. Both bring the game down to the true $\neg 0{=}0 \vee 10{=}10$, so $\top$ wins.

12-13 $\langle \top 0.1, \bot 1.1 \rangle$ and $\langle \bot 1.1, \top 0.1 \rangle$. Both bring the game down to the true $\neg 0{=}1 \vee 10{=}10$, so $\top$ wins.

Later we will be using some relaxed informal jargon already established in CoL for describing runs and strategies, referring to moves by their intuitive meanings or their effects on the game. For instance, the initial labmove $\top 0.0$ in a play of the game $p \sqcap q \to r$ we can characterize as "$\top$ made the move 0.0". Remembering the meaning of the prefix "0." of this move, we may as well say that "$\top$ made the move 0 in the antecedent". Further remembering the effect of such a move on the antecedent, we may just as well say "$\top$ chose p (or the left $\sqcap$-conjunct) in the antecedent". We may also say something like "$\top$ (made the move that) brought the game down to $p \to r$", or "$\top$ (made the move that) brought the antecedent down to p".

To (re)define the operation $\circlearrowleft$ in the style of Definition 2.4, we need some preliminaries. What we call a **tree of games** is a structure defined inductively as an element of the smallest set satisfying the following conditions:

- Every constant game A is a tree of games. The one-element sequence $\langle A \rangle$ is said to be the **yield** of such a tree, and the **address** of A in this tree is the empty bit string.
- Whenever $\mathcal{A}$ is a tree of games with yield $\langle A_1, \ldots, A_m \rangle$ and $\mathcal{B}$ is a tree of games with yield $\langle B_1, \ldots, B_n \rangle$, the pair $\mathcal{A} \circ \mathcal{B}$ is a tree of games with yield $\langle A_1, \ldots, A_m, B_1, \ldots, B_n \rangle$. The **address** of each A_i in this tree is $0w$, where w is the address of A_i in $\mathcal{A}$. Similarly, the address of each B_i is $1w$, where w is the address of B_i in $\mathcal{B}$.

Example: Where A, B, C, D are constant games, $(A \circ B) \circ (C \circ (A \circ D))$ is a tree of games with yield $\langle A, B, C, A, D \rangle$. The address of the first A of the yield, to which we may as well refer as the first **leaf** of the tree, is 00; the address of the second leaf B is 01; the address of the third leaf C is 10; the address of the fourth leaf A is 110; and the address of the fifth leaf D is 111.

Note that $\circ$ is not an operation on games, but just a symbol used instead of the more common comma to separate the two parts of a pair. And a tree of games itself is not a game, but a collection of games arranged into a certain structure, just as a sequence of games is not a game but a collection of games arranged as a list.

For bit strings u and w, we will write $u \preceq w$ to indicate that u is a (not necessarily proper) **prefix** (initial segment) of w.

Definition 2.6 Let $A_1, \ldots, A_n$ ($n \geq 1$) be constant games, and $\mathcal{T}$ be a tree of games with yield $\langle A_1, \ldots, A_n \rangle$. Let $w_1, \ldots, w_n$ be the addresses of $A_1, \ldots, A_n$ in $\mathcal{T}$, respectively. The game $⫰\mathcal{T}$ (the **branching recurrence** of $\mathcal{T}$) is defined by:

(i) $\langle \wp\alpha \rangle \in \mathbf{Lr}^{⫰\mathcal{T}}$ iff one of the following conditions is satisfied:

1. $\wp\alpha = \wp u.\beta$, where $u \preceq w_i$ for at least one $i \in \{1, \ldots, n\}$ and, for each i with $u \preceq w_i$, $\langle \wp\beta \rangle \in \mathbf{Lr}^{A_i}$. We call such a move a **nonreplicative** (lab)move. It brings the game down to $⫰\mathcal{T}'$, where $\mathcal{T}'$ is the result of replacing A_i by $\langle \wp\beta \rangle A_i$ in $\mathcal{T}$ for each i with $u \preceq w_i$. If here u is w_i (rather than a proper prefix of such) for one of $i \in \{1, \ldots, n\}$, we say that the move $\wp u.\beta$ is **focused**. Otherwise it is **unfocused**.
2. $\wp\alpha = \bot w_i{:}$, where $i \in \{1, \ldots, n\}$. We call such a move a **replicative** (lab)move. It brings the game down to $⫰\mathcal{T}'$, where $\mathcal{T}'$ is the result of replacing A_i by $(A_i \circ A_i)$ in $\mathcal{T}$.

(ii) Whenever Γ is a legal run of $⫰\mathcal{T}$, $\mathbf{Wn}^{⫰\mathcal{T}}\langle \Gamma \rangle = \top$ iff, for each $i \in \{1, \ldots, n\}$ and every infinite bit string v with $w_i \preceq v$, we have $\mathbf{Wn}^{A_i}\langle \Gamma^{\preceq v} \rangle = \top$. Here $\Gamma^{\preceq v}$ means the result of deleting, from Γ, all labmoves except those that look like $\wp u.\alpha$ for some bit string u with $u \preceq v$, and then further changing each such (remaining) labmove $\wp u.\alpha$ to $\wp\alpha$.[5]

Example 2.7 Let

$$G \;=\; p \sqcup (q \sqcap (r \sqcap (s \sqcup t))),$$

where p, q, r, s, t are constant elementary games. And let

$$\Gamma \;=\; \langle \bot{:},\ \top.1,\ \bot 0.0,\ \bot 1.1,\ \bot 1{:},\ \bot 10.0,\ \bot 11.1,\ \top 11.0 \rangle.$$

Then Γ is a legal run of $⫰G$. Below we trace, step by step, the effects of its moves on $⫰G$.

The 1st (lab)move $\bot{:}$ means that $\bot$ replicates the (only) leaf of the tree, with the address of that leaf being the empty bit string. This move brings the game down to — in the sense that $\langle \bot{:} \rangle ⫰G$ is — the following game:

$$⫰\Big(\big(p \sqcup (q \sqcap (r \sqcap (s \sqcup t)))\big) \circ \big(p \sqcup (q \sqcap (r \sqcap (s \sqcup t)))\big)\Big).$$

The 2nd move $\top.1$ means choosing the second $\sqcup$-disjunct $q \sqcap (r \sqcap (s \sqcup t))$ in *both* leaves of the tree. This is so because the addresses of those leaves are 0 and 1, and the empty bit string — seen between "$\top$" and ".1" in $\top.1$ — is an initial segment of both addresses. The effect of this unfocused move is the same as the effect of the two consecutive focused moves $\top 0.1$ and $\top 1.1$ (in whatever order) would be, but $\top$ might have its reasons for having made an unfocused move. Among such reasons could be that $\top$ did not notice $\bot$'s initial move (or the latter arrived late over the asynchronous network) and thought that the position was still G, in which case making the moves $\top 0.1$ and $\top 1.1$ would be simply illegal. Note also that the ultimate effect of the move $\top.1$ on the game would remain the same as it is now even if this move was made before the replicative move $\bot{:}$. It is CoL's striving to achieve this sort of flexibility and asynchronous-communication-friendliness that has determined our seemingly "strange" choice of trees rather than sequences as the underlying structures for $⫰$-games. Any attempt to deal with sequences instead of trees would encounter the problem of violating what CoL calls the *static* (speed-independent) property of games.[6]

Anyway, the position resulting from the second move of Γ is

$$⫰\Big(\big(q \sqcap (r \sqcap (s \sqcup t))\big) \circ \big(q \sqcap (r \sqcap (s \sqcup t))\big)\Big).$$

The effect of the 3rd move $\bot 0.0$ is choosing the left $\sqcap$-conjunct q in the left (0-addressed) leaf of the tree, which results in

$$⫰\Big(q \circ \big(q \sqcap (r \sqcap (s \sqcup t))\big)\Big).$$

[5]Intuitively, $\Gamma^{\preceq v}$ is the run played in one of the multiple "copies" of A_i that have been generated in the play, with v acting as a (perhaps longer than necessary yet meaningful) "address" of that copy.

[6]While not technically necessary for the purposes of this paper, here we still reproduce a definition of the static property for constant games. For either player $\wp$, let us say that a run Υ is a **$\wp$-delay** of a run Γ iff (1) for both players $\wp' \in \{\top, \bot\}$, the subsequence of $\wp'$-labeled moves of Υ is the same as that of Γ, and (2) for any $n, k \geq 1$, if the nth $\wp$-labeled move is made later than (is to the right of) the kth $\neg\wp$-labeled move in Γ, then so is it in Υ. Next, let us say that a run is **$\wp$-legal** iff it is not $\wp$-illegal. Now, we say that a constant game A is **static** iff, whenever a run Υ is a $\wp$-delay of a run Γ, we have: (i) if Γ is a $\wp$-legal run of A, then so is Υ, and (ii) if Γ is a $\wp$-won run of A, then so is Υ.

Similarly, the 4th move $\bot 1.1$ chooses the right $\sqcap$-conjunct in the right leaf of the tree, resulting in

$$\circ\!\!\!\mid\Big(q \circ \big(r \sqcap (s \sqcup t)\big)\Big).$$

The 5th move $\bot 1{:}$ replicates the right leaf, bringing the game down to

$$\circ\!\!\!\mid\Big(q \circ \big((r \sqcap (s \sqcup t)) \circ (r \sqcap (s \sqcup t))\big)\Big).$$

The 6th move $\bot 10.0$ chooses the left $\sqcap$-conjunct in the second (00-addressed) leaf, and, similarly, the 7th move $\bot 11.1$ chooses the right $\sqcap$-conjunct in the third (11-addressed) leaf. These two moves bring the game down to

$$\circ\!\!\!\mid\Big(q \circ \big(r \circ (s \sqcup t)\big)\Big).$$

The last, 8th move $\top 11.0$ chooses the left $\sqcup$-disjunct of the third leaf, and the final position is

$$\circ\!\!\!\mid\big(q \circ (r \circ s)\big).$$

According to clause (ii) of Definition 2.6, Γ is a $\top$-won run of $\circ\!\!\!\mid G$ iff, for any infinite bit string v, $\Gamma^{\preceq v}$ is a $\top$-won run of G. Observe that for any infinite — or, "sufficiently long" finite — bit string v, $\Gamma^{\preceq v}$ is either $\langle \top 1, \bot 0\rangle$ (if $v = 0\ldots$) or $\langle \top 1, \bot 1, \bot 0\rangle$ (if $v = 10\ldots$) or $\langle \top 1, \bot 1, \bot 1, \top 0\rangle$ (if $v = 11\ldots$). We also have $\langle \top 1, \bot 0\rangle G = q$, $\langle \top 1, \bot 1, \bot 0\rangle G = r$ and $\langle \top 1, \bot 1, \bot 1, \top 0\rangle G = s$. So it is no accident that we see q, r, s at the leaves in the final position. Correspondingly, the game is won iff each one of these three propositions is true.

The cases where $\bot$ makes infinitely many replications in a run Γ of a game $\circ\!\!\!\mid H$ and hence the "eventual tree" is infinite are similar, with the only difference that the "addresses" of the "leaves" of such a "tree", corresponding to different plays of H, may be infinite bit strings. But, again, the overall game $\circ\!\!\!\mid H$ will be won by $\top$ iff all of those plays — all $\Gamma^{\preceq v}$ where v is an infinite bit string, that is — are $\top$-won plays of H.

Definition 2.8 Let B, $A_1, \ldots, A_n$ ($n \geq 0$) be constant games. We define $A_1, \ldots, A_n \circ\!\!- B$ — let us call it the **ultimate reduction** of B to $A_1, \ldots, A_n$ — as the game $\circ\!\!\!\mid A_1 \wedge \ldots \wedge \circ\!\!\!\mid A_n \rightarrow B$.

3 Generalized universes and non-constant games

Constant games can be seen as generalized propositions: while propositions in classical logic are just elements of $\{\top, \bot\}$, constant games are functions from runs to $\{\top, \bot\}$. Our concept of a game generalizes that of a constant game in the same sense as the classical concept of a predicate generalizes that of a proposition.

We fix some infinite set of expressions called **variables**, and use the letters x, y, z, s, r, t as metavariables for them. We also fix another infinite set (disjoint from the previous one) of expressions called **constants**:

$$\{0, 1, 10, 11, 100, 101, 110, 111, 1000, \ldots\}.$$

These are thus **binary numerals** — the strings matching the regular expression $0 \cup 1(0 \cup 1)^*$. We will be typically identifying such strings — by some rather innocent abuse of concepts — with the natural numbers represented by them in the standard binary notation, and vice versa. We will be mostly using a, b, c, d as metavariables for constants.

A **universe** (of discourse) is a pair $(U, ^U)$, where U is a nonempty set, and ^{U}, called the **naming function** of the universe, is a function that sends each constant c to an element c^U of U. The intuitive meaning of $c^U = s$ is that c is a **name** of s. Both terminologically and notationally, we will typically identify each universe $(U, ^U)$ with its first component and, instead of "$(U, ^U)$", write simply "U", keeping in mind that each such "universe" U comes with a fixed associated function ^{U}.

A universe $U = (U, ^U)$ is said to be **ideal** iff U coincides with the above-fixed set of constants, and ^{U} is the identity function on that set. All earlier papers on CoL dealt only with ideal universes. This was for simplicity considerations, yielding no loss of generality as so far no results have relied on the assumption that the underlying universes were ideal. Our present treatment, however, for both technical and philosophical reasons, *does* call for the above-defined, more general, concept of a universe — a universe where some objects may have unique names, some objects have many names, and some objects have no names at all.[7] Note that real-world universes are typically not ideal: not all people living or staying in the United States have Social Security numbers; most stars and planets of the Galaxy have no names at all, while some have several names (Morning Star = Evening Star = Venus); etc. A natural example of a non-ideal universe from the world of mathematics would be the set of real numbers, only some of whose elements have names, such as 5, 1/3, $\sqrt{2}$ or π. Generally, no uncountable universe would be ideal for the simple reason that there can only be countably many names. This is so because names, by their very nature and purpose, have to be finite objects. Observe also that many properties of common interest such as computability or decidability, are usually sensitive to how objects are named. For instance, strictly speaking, computing a function $f(x)$ means the ability to tell, after seeing a (the) *name* of an arbitrary object $\mathfrak{a}$, to produce a (the) *name* of the object $\mathfrak{b}$ with $\mathfrak{b} = f(\mathfrak{a})$. Similarly, an algorithm that decides a predicate $p(x)$ on a set S, strictly speaking, takes not *elements* of S — which may be abstract objects such as numbers or graphs — but rather *names* of those elements (such as binary numerals or codes). It is not hard to come up with a nonstandard naming of natural numbers through binary numerals where the predicate "x is even" is undecidable. On the other hand, for any undecidable arithmetical predicate $p(x)$, one can come up with a naming function such that $p(x)$ becomes decidable — for instance, one that assigns even-length names to all $\mathfrak{a}$ with $p(\mathfrak{a})$ and assigns odd-length names to all $\mathfrak{a}$ with $\neg p(\mathfrak{a})$. Classical logic exclusively deals with objects of a universe without a need for also considering names for them, as it is not concerned with decidability or computability. CoL, on the other hand, with its computational semantics, inherently calls for being more careful about differentiating between objects and their names, and hence for explicitly considering universes in the form $(U,^U)$ rather than just U as classical logic does.

By a **valuation** on a universe U, or a U-valuation, we mean a mapping e that sends each variable x to an element $e(x)$ of U. For technical convenience, we extend every such mapping to all constants as well, by stipulating that, for any constant c, $e(c) = c^U$. When a universe U is fixed, irrelevant or clear from the context, we may omit references to it and simply say "valuation". In these terms, a classical predicate p can be understood as a function that sends each valuation e to a proposition, i.e., to a constant predicate. Similarly, what we call a game sends valuations to constant games:

Definition 3.1 Let U be a universe. A **game on** U is a function A from U-valuations to constant games. We write $e[A]$ (rather than $A(e)$) to denote the constant game returned by A on valuation e. Such a constant game $e[A]$ is said to be an **instance** of A. For readability, we usually write $\mathbf{Lr}^A_e$ and $\mathbf{Wn}^A_e$ instead of $\mathbf{Lr}^{e[A]}$ and $\mathbf{Wn}^{e[A]}$.

Just as this is the case with propositions versus predicates, constant games in the sense of Definition 2.1 will be thought of as special, constant cases of games in the sense of Definition 3.1. In particular, each constant game A' is the game A such that, for every valuation e, $e[A] = A'$. From now on we will no longer distinguish between such A and A', so that, if A is a constant game, it is its own instance, with $A = e[A]$ for every e.

Where n is a natural number, we say that a game A is n**-ary** iff there is are n variables such that, for any two valuations e_1 and e_2 that agree on all those variables, we have $e_1[A] = e_2[A]$. Generally, a game that is n-ary for some n, is said to be **finitary**. Our paper is going to exclusively deal with finitary games and, for this reason, we agree that, from now on, when we say "game", we always mean "finitary game".

For a variable x and valuations e_1, e_2, we write $e_1 \equiv_x e_2$ to mean that the two valuations have the same universe and agree on all variables other than x.

We say that a game A **depends** on a variable x iff there are two valuations e_1, e_2 with $e_1 \equiv_x e_2$ such that $e_1[A] \neq e_2[A]$. An n-ary game thus depends on at most n variables. And constant games are nothing but 0-ary games, i.e., games that do not depend on any variables.

We say that a (not necessarily constant) game A is **elementary** iff so are all of its instances $e[A]$.

Just as constant games are generalized propositions, games can be treated as generalized predicates. Namely, we will view each predicate p of whatever arity as the same-arity elementary game such that, for every valuation e, $\mathbf{Wn}^p_e\langle\rangle = \top$ iff p is true at e. And vice versa: every elementary game p will be viewed as the same-arity predicate which is true at a given valuation e iff $\mathbf{Wn}^p_e\langle\rangle = \top$. Thus, for us, "predicate"

[7]Further generalizations are possible if and when a need arises. Namely, one may depart from our present assumption that the set of constants is infinite and/or fixed, as long as there is a fixed constant — say, 0 — that belongs to every possible set of constants ever considered. No results of this or any earlier papers on CoL would be in any way affected by doing so.

and "elementary game" are synonyms. Accordingly, any standard terminological or notational conventions familiar from the literature for predicates also apply to them to them viewed as elementary games.

Just as the Boolean operations straightforwardly extend from propositions to all predicates, our operations $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\sqcap$, $\sqcup$, $\overset{|}{\circ}$, $\circ\!-$ extend from constant games to all games. This is done by simply stipulating that $e[\ldots]$ commutes with all of those operations: $\neg A$ is the game such that, for every valuation e, $e[\neg A] = \neg e[A]$; $A \sqcap B$ is the game such that, for every e, $e[A \sqcap B] = e[A] \sqcap e[B]$; etc. So does the operation of prefixation: provided that Φ is a legal position of every instance of A, $\langle\Phi\rangle A$ is understood as the unique game such that, for every e, $e[\langle\Phi\rangle A] = \langle\Phi\rangle e[A]$.

Definition 3.2 Let A be a game on a universe U, $x_1, \ldots, x_n$ be pairwise distinct variables, and $\mathfrak{t}_1, \ldots, \mathfrak{t}_n$ be constants and/or variables. The result of **substituting** $x_1, \ldots, x_n$ **by** $\mathfrak{t}_1, \ldots, \mathfrak{t}_n$ **in** A, denoted $A(x_1/\mathfrak{t}_1, \ldots, x_n/\mathfrak{t}_n)$, is defined by stipulating that, for every valuation e, $e[A(x_1/\mathfrak{t}_1, \ldots, x_n/\mathfrak{t}_n)] = e'[A]$, where e' is the valuation that sends each x_i to $e(\mathfrak{t}_i)$ and agrees with e on all other variables.

Following the standard readability-improving practice established in the literature for predicates, we will often fix pairwise distinct variables $x_1, \ldots, x_n$ for a game A and write A as $A(x_1, \ldots, x_n)$. Representing A in this form sets a context in which we can write $A(\mathfrak{t}_1, \ldots, \mathfrak{t}_n)$ to mean the same as the more clumsy expression $A(x_1/\mathfrak{t}_1, \ldots, x_n/\mathfrak{t}_n)$.

Definition 3.3 Let $A(x)$ be a game on a given universe. On the same universe, $\sqcap xA(x)$ (**choice universal quantification**) and $\sqcup xA(x)$ (**choice existential quantification**) are defined as the following two games, respectively:

$$A(0) \sqcap A(1) \sqcap A(10) \sqcap A(11) \sqcap A(100) \sqcap \ldots;$$
$$A(0) \sqcup A(1) \sqcup A(10) \sqcup A(11) \sqcup A(100) \sqcup \ldots.$$

Thus, every initial legal move of $\sqcap xA(x)$ or $\sqcup xA(x)$ is a constant $c \in \{0, 1, 10, 11, 100, \ldots\}$, which in our informal language we may refer to as "the constant chosen (by the corresponding player) for x".

We say that a game A is **unistructural** iff, for any two valuations e_1 and e_2, we have $\mathbf{Lr}^A_{e_1} = \mathbf{Lr}^A_{e_2}$. Of course, all constant or elementary games are unistructural. It can also be easily seen that all our game operations preserve the unistructural property of games. For the purposes of the present paper, considering only unistructural games would be sufficient.

We define the remaining operations $\forall$ and $\exists$ only for unistructural games:

Definition 3.4 Below $A(x)$ is an arbitrary unistructural game on a universe U. On the same universe:

1. The game $\forall xA(x)$ (**blind universal quantification**) is defined by stipulating that, for every U-valuation e, player $\wp$ and move α, we have:

   (i) $\langle\wp\alpha\rangle \in \mathbf{Lr}^{\forall xA(x)}_e$ iff $\langle\wp\alpha\rangle \in \mathbf{Lr}^{A(x)}_e$. Such an initial legal labmove $\wp\alpha$ brings the game $e[\forall xA(x)]$ down to $e[\forall x\langle\wp\alpha\rangle A(x)]$.

   (ii) Whenever Γ is a legal run of $e[\forall xA(x)]$, $\mathbf{Wn}^{\forall xA(x)}_e\langle\Gamma\rangle = \top$ iff, for every valuation g with $g \equiv_x e$, $\mathbf{Wn}^{A(x)}_g\langle\Gamma\rangle = \top$.

2. The game $\exists xA(x)$ (**blind existential quantification**) is defined in exactly the same way, only with $\top$ and $\bot$ interchanged.

Example 3.5 Consider the ideal universe $U = \{0, 1, 10, 11, 100, \ldots\}$. Let G be the following game on U, with the predicates *Even* and *Odd* having their expected meanings:

$$\forall y\Big(\mathit{Even}(y) \sqcup \mathit{Odd}(y) \rightarrow \sqcap x\big(\mathit{Even}(x{+}y) \sqcup \mathit{Odd}(x{+}y)\big)\Big).$$

Then the sequence $\langle\bot 1.11,\ \bot 0.0,\ \top 1.1\rangle$ is a legal run of G, the effects of the moves of which are shown below:

$$\begin{array}{ll}
G: & \forall y\Big(\mathit{Even}(y) \sqcup \mathit{Odd}(y) \rightarrow \sqcap x\big(\mathit{Even}(x{+}y) \sqcup \mathit{Odd}(x{+}y)\big)\Big) \\
\langle\bot 1.11\rangle G: & \forall y\big(\mathit{Even}(y) \sqcup \mathit{Odd}(y) \rightarrow \mathit{Even}(11{+}y) \sqcup \mathit{Odd}(11{+}y)\big) \\
\langle\bot 1.11, \bot 0.0\rangle G: & \forall y\big(\mathit{Even}(y) \rightarrow \mathit{Even}(11{+}y) \sqcup \mathit{Odd}(11{+}y)\big) \\
\langle\bot 1.11, \bot 0.0, \top 1.1\rangle G: & \forall y\big(\mathit{Even}(y) \rightarrow \mathit{Odd}(11{+}y)\big)
\end{array}$$

The play hits (ends as) the true proposition $\forall y\big(\mathit{Even}(y) \rightarrow \mathit{Odd}(11{+}y)\big)$ and hence is won by $\top$.

Example 3.6 The sequence

$$\langle \top 0.1.:,\ \bot 1.10,\ \top 0.1.0.10,\ \top 0.1.0.10,\ \bot 0.1.0.100,\ \top 0.1.1.100,\ \top 0.1.1.10,\ \bot 0.1.1.1000,\ \top 1.1000\rangle$$

is a legal run of the game

$$\forall x\bigl(x^3=(x\times x)\times x\bigr),\ \sqcap x\sqcap y\sqcup z(z=x\times y)\ \circ\!\!- \ \sqcap x\sqcup y(y=x^3). \tag{1}$$

Below we see how the game evolves according to the scenario of this run:

$$\begin{array}{ll}
 & \forall x\bigl(x^3=(x\times x)\times x\bigr),\ \sqcap x\sqcap y\sqcup z(z=x\times y)\ \circ\!\!-\ \sqcap x\sqcup y(y=x^3)\\
\top 0.1.:\text{ yields} & \forall x\bigl(x^3=(x\times x)\times x\bigr),\ \sqcap x\sqcap y\sqcup z(z=x\times y)\circ\sqcap x\sqcap y\sqcup z(z=x\times y)\ \circ\!\!-\ \sqcap x\sqcup y(y=x^3)\\
\bot 1.10\text{ yields} & \forall x\bigl(x^3=(x\times x)\times x\bigr),\ \sqcap x\sqcap y\sqcup z(z=x\times y)\circ\sqcap x\sqcap y\sqcup z(z=x\times y)\ \circ\!\!-\ \sqcup y(y=10^3)\\
\top 0.1.0.10\text{ yields} & \forall x\bigl(x^3=(x\times x)\times x\bigr),\ \sqcap y\sqcup z(z=10\times y)\circ\sqcap x\sqcap y\sqcup z(z=x\times y)\ \circ\!\!-\ \sqcup y(y=10^3)\\
\top 0.1.0.10\text{ yields} & \forall x\bigl(x^3=(x\times x)\times x\bigr),\ \sqcup z(z=10\times 10)\circ\sqcap x\sqcap y\sqcup z(z=x\times y)\ \circ\!\!-\ \sqcup y(y=10^3)\\
\bot 0.1.0.100\text{ yields} & \forall x\bigl(x^3=(x\times x)\times x\bigr),\ (100=10\times 10)\circ\sqcap x\sqcap y\sqcup z(z=x\times y)\ \circ\!\!-\ \sqcup y(y=10^3)\\
\top 0.1.1.100\text{ yields} & \forall x\bigl(x^3=(x\times x)\times x\bigr),\ (100=10\times 10)\circ\sqcap y\sqcup z(z=100\times y)\ \circ\!\!-\ \sqcup y(y=10^3)\\
\top 0.1.1.10\text{ yields} & \forall x\bigl(x^3=(x\times x)\times x\bigr),\ (100=10\times 10)\circ\sqcup z(z=100\times 10)\ \circ\!\!-\ \sqcup y(y=10^3)\\
\bot 0.1.1.1000\text{ yields} & \forall x\bigl(x^3=(x\times x)\times x\bigr),\ (100=10\times 10)\circ(1000=100\times 10)\ \circ\!\!-\ \sqcup y(y=10^3)\\
\top 1.1000\text{ yields} & \forall x\bigl(x^3=(x\times x)\times x\bigr),\ (100=10\times 10)\circ(1000=100\times 10)\ \circ\!\!-\ 1000=10^3
\end{array}$$

The play hits a true proposition and hence is won by $\top$. Note that here, unlike the case in the previous example, $\top$ is the winner no matter what universe we consider and what the meanings of the expressions $x\times y$ and x^3 are. In fact, $\top$ has a "purely logical" winning strategy in this game, in the sense that the strategy is successful regardless of whether things have their standard arithmetic meanings or some other meanings. This follows from the promised soundness of **CL12** and the fact — illustrated later in Example 7.2 — that (1) is provable in **CL12**.

4 Interactive machines revisited

In traditional game-semantical approaches, including Blass's [3, 4] approach which is the closest precursor of ours, player's strategies are understood as *functions* — typically as functions from interaction histories (positions) to moves, or sometimes ([1]) as functions that only look at the latest move of the history. This *strategies-as-functions* approach, however, is generally inapplicable in the context of CoL, whose relaxed semantics, in striving to get rid of "bureaucratic pollutants" and only deal with the remaining true essence of games, does not impose any regulations on which player can or should move in a given situation. Here, in many cases, either player may have (legal) moves, and then it is unclear whether the next move should be the one prescribed by $\top$'s strategy function or the one prescribed by the strategy function of $\bot$. For a game semantics whose ambition is to provide a comprehensive, natural and direct tool for modeling interaction, the strategies-as-functions approach would be less than adequate, even if technically possible. This is so for the simple reason that the strategies that real computers follow are not functions. If the strategy of your personal computer was a function from the history of interaction with you, then its performance would keep noticeably worsening due to the need to read the continuously lengthening — and, in fact, practically infinite — interaction history every time before responding. Fully ignoring that history and looking only at your latest keystroke in the spirit of [1] is also certainly not what your computer does, either. The advantages of our approach thus become especially appreciable when one tries to bring complexity theory into interactive computation: hardly (m)any really meaningful and interesting complexity-theoretic concepts can be defined for games (particularly, games that may last long) with the strategies-as-functions approach.

In CoL, ($\top$'s effective) strategies are defined in terms of interactive machines, where computation is one continuous process interspersed with — and influenced by — multiple "input" (environment's moves) and "output" (machine's moves) events. Of several, seemingly rather different yet equivalent, machine models of interactive computation studied in CoL, this paper only employs the most basic, **HPM** ("Hard-Play Machine") model.

Remember that an HPM is a Turing machine with the additional capability of making moves. The adversary can also move at any time, with such moves being the only nondeterministic events from the machine's perspective. Along with the ordinary read/write **work tape**, the machine has an additional, read-only tape called the **run tape**.[8] The latter, serving as a dynamic input, at any time spells the "current position" of the play. Its role is to make the run fully visible to the machine. In these terms, an algorithmic solution ($\top$'s winning strategy) for a given constant game A is understood as an HPM $\mathcal{M}$ such that, no matter how the environment acts during its interaction with $\mathcal{M}$ (what moves it makes and when), the run incrementally spelled on the run tape is a $\top$-won run of A. As for $\bot$'s strategies, there is no need to define them: all possible behaviors by $\bot$ are accounted for by the different possible nondeterministic updates of the run tape of an HPM.

In the above outline, we described HPMs in a relaxed fashion, without being specific about details such as, say, how, exactly, moves are made by the machine, how many moves either player can make at once, what happens if both players attempt to move "simultaneously", etc. All reasonable design choices yield the same class of winnable games as long as we consider the natural subclass of games called **static**. Such games are obtained by imposing a certain simple formal condition on games (see the footnote on page 6, or Section 5 of [23]), which is not really necessary to reproduce here as nothing in this paper relies on it. We shall only point out that, intuitively, static games are interactive tasks where the relative speeds of the players are irrelevant, as it never hurts a player to postpone making moves. In other words, static games are games that are contests of intellect rather than contests of speed. And one of the theses that CoL philosophically relies on is that static games present an adequate formal counterpart of our intuitive concept of "pure", speed-independent interactive computational problems. Correspondingly, CoL restricts its attention (more specifically, possible interpretations of the atoms of its formal language) to static games. All elementary games turn out to be trivially static, and the class of static games turns out to be closed under all game operations studied in CoL. More specifically, all games expressible in the language of the later-defined logic **CL12** are static (as well as finitary and unistructural). Such games are not necessarily constant but, due to being finitary, can and will be thought of to be constant by identifying them with their $\sqcap$-closures. Correspondingly, in this paper, we use the term "**computational problem**", or simply "**problem**", as a synonym of "constant static game".

While design choices are unimportant and "negotiable", we still want to agree on some technical details for clarity. Just like an ordinary Turing machine, an HPM has a finite set of **states**, one of which has the special status of being the **start state**. There are no accept, reject, or halt states, but there are specially designated states called **move states**. Either tape of the machine has a beginning but no end and is divided into infinitely many **cells**, arranged in the left-to-right order. At any time, each cell contains one symbol from a certain fixed finite set of **tape symbols**. The **blank** symbol, as well as $\top$ and $\bot$, are among the tape symbols. We also assume that these three symbols are not among the symbols that any (legal or illegal) move can ever contain. Either tape has its own **scanning head**, at any given time looking (located) at one of the cells of the tape. A transition from one **computation step** ("**clock cycle**", **"time"**) to another happens according to the fixed **transition function** of the machine. The latter, depending on the current state, and the symbols seen by the two heads on the corresponding tapes, deterministically prescribes the next state, the tape symbol by which the old symbol should be overwritten in the current cell (the cell currently scanned by the head) of the work tape, and, for each head, the direction — one cell left or one cell right — in which the head should move. A constraint here is that the blank symbol, $\top$ or $\bot$ can never be written by the machine on the work tape. An attempt to move left when the head of a given tape is looking at the first (leftmost) cell results in staying put. So does an attempt to move right when the head is looking at the blank symbol.

When the machine starts working, it is in its start state, both scanning heads are looking at the leftmost cells of the corresponding tapes, and (all cells of) both tapes are blank (i.e., contain the blank symbol). Whenever the machine enters a move state, the string α spelled by (the contents of) its work tape cells, starting from the first cell and ending with the cell immediately left to the work-tape scanning head, will be automatically appended — at the beginning of the next clock cycle — to the contents of the run tape in the $\top$-prefixed form $\top\alpha$. And, on every transition, whether the machine is in a move state or not, any finite sequence $\bot\beta_1, \ldots, \bot\beta_m$ of $\bot$-labeled moves may be nondeterministically appended to the content of the run tape. If the above two events happen on the same clock cycle, then the moves will be appended to the contents of the run tape in the following order: $\top\alpha\bot\beta_1 \ldots \bot\beta_m$.

With each labmove that emerges on the run tape we associate its **timestamp**, which is the number of the clock cycle immediately preceding the cycle on which the move first emerged on the run tape. Intuitively,

[8]The EPMs from the earlier literature on CoL also had a third tape called the *valuation tape*. The latter, however, becomes redundant in our present treatment due to the fact that we are exclusively interested in constant games or finitary games identified with their (constant) $\sqcap$-closures.

the timestamp indicates on which cycle the move was *made* rather than *appeared* on the run tape: a move made during cycle $\#i$ appears on the run tape on cycle $\#i+1$ rather than $\#i$. Also, we agree that the count of clock cycles starts from 0, meaning that the very first clock cycle is cycle #0 rather than #1.

A **configuration** is a full description of (the "current") contents of the work and run tapes, the locations of the two scanning heads, and the state of the machine. A **computation branch** is an infinite sequence $C_0, C_1, C_2, \ldots$ of configurations, where C_0 is the initial configuration (as explained earlier), and every C_{i+1} is a configuration that could have legally followed (again, in the sense explained earlier) C_i. For a computation branch B, the **run spelled by** B is the run Γ incrementally spelled on the run tape in the corresponding scenario of interaction. We say that such a Γ is **a run generated by** the machine.

We say that a given HPM $\mathcal{M}$ **wins** (**computes**, **solves**) a given constant game A — and write $\mathcal{M} \models A$ — iff every run Γ generated by $\mathcal{M}$ is a $\top$-won run of A. We say that A is **computable** iff there is an HPM $\mathcal{M}$ with $\mathcal{M} \models A$; such an HPM is said to be an (algorithmic) **solution**, or **winning strategy**, for A.

5 Towards interactive complexity

At present, the theory of interactive computation is far from being well developed, and even more so is the corresponding complexity theory. The studies of interactive computation in the context of complexity, while having going on since long ago, have been relatively scattered and ad hoc: more often than not, interaction has been used for better understanding certain complexity issues for traditional, non-interactive problems rather than being treated as an object of systematic studies in its own rights (examples would be alternating computation [7], or interactive proof systems and Arthur-Merlin games [8, 2]). As if complexity theory was not "complex" enough already, taking it to the interactive level would most certainly generate a by an order of magnitude greater diversity of species from the complexity zoo. The present work is the first modest attempt to bring complexity issues into CoL. Here we introduce one, perhaps the simplest, way of measuring the complexity of (our non-functional) strategies out of the huge and interesting potential variety of complexity measures meaningful and useful in the interactive context.

The **size** of a move α means the length of α as a string. In the context of a given computation branch of a given HPM $\mathcal{M}$, by the **background** of a clock cycle c we mean the greatest of the sizes of Environment's moves made by (before) time c, or 0 if there are no such moves. If $\mathcal{M}$ makes a move on cycle c, then the background of that move[9] means the background of c. Next, whenever $\mathcal{M}$ makes a move on cycle c, by the **timecost** of that move we mean $c-d$, where d is the greatest cycle with $d < c$ on which a move was made by either player, or is 0 if there is no such cycle.

Throughout this paper, an n-ary ($n \geq 0$) **arithmetical function** means a total function from n-tuples of natural numbers to natural numbers. "Unary" is a synonym of "1-ary".

Definition 5.1 Let h be an unary arithmetical function, and $\mathcal{M}$ an HPM.

1. We say that $\mathcal{M}$ **runs in time** h, or that $\mathcal{M}$ is an h **time machine**, or that h is a **bound** for the time complexity of $\mathcal{M}$, iff, in every play (computation branch), for any clock cycle c on which $\mathcal{M}$ makes a move, neither the timecost nor the size of that move exceeds $h(\ell)$, where ℓ is the background of c.

2. We say that $\mathcal{M}$ **runs in space** h, or that $\mathcal{M}$ is an h **space machine**, or that h is a **bound** for the space complexity of $\mathcal{M}$, iff, in every play (computation branch), for any clock cycle c, the number of cells ever visited by the work-tape head of $\mathcal{M}$ by time c does not exceed $h(\ell)$, where ℓ is the background of c.

Our time complexity concept can be seen to be in the spirit of what is usually called *response time*. The latter generally does not and should not depend on the length of the preceding interaction history. On the other hand, it is not and should not merely be a function of the adversary's last move, either. A similar characterization applies to our concept of space complexity. Both complexity measures are equally meaningful whether it be in the context of "short-lasting" games (such as the ones represented by the formulas of the later-defined logic **CL12**) or the context of games that may have "very long" and even infinitely long legal runs.

Let A be a constant game, h an unary arithmetical function, and $\mathcal{M}$ an HPM. We say that $\mathcal{M}$ **wins** (**computes**, **solves**) A **in time** h, or that $\mathcal{M}$ **is an** h **time solution for** A, iff $\mathcal{M}$ is an h time machine

[9]As easily understood, here and in similar contexts, "move" means a move not as a *string*, but as an *event*, namely, the event of $\mathcal{M}$ making a move at time c.

with $\mathcal{M} \models A$. We say that A is **computable** (**winnable**, **solvable**) **in time** h iff it has an h time solution. Similarly for "**space**" instead of "time".

When we say **polynomial time**, it is to be understood as "time h for some polynomial function h". Similarly for **polynomial space**.

6 The language of logic CL12 and its semantics

Logic **CL12** will be axiomatically constructed in Section 7. The present section is merely devoted to its *language*. The building blocks of the formulas of the latter are:

- **Nonlogical predicate letters**, for which we use p, q as metavariables. With each predicate letter is associated a fixed nonnegative integer called its **arity**. We assume that, for any n, there are infinitely many n-ary predicate letters.
- **Function letters**, for which we use f, g as metavariables. Again, each function letter comes with a fixed **arity**, and we assume that, for any n, there are infinitely many n-ary function letters.
- The binary **logical predicate letter** $=$.
- Infinitely many **variables** and **constants**. These are the same as the ones fixed in Section 3.

Terms, for which we use τ, ψ, ξ as metavariables, are built from variables, constants and function letters in the standard way. An **atomic formula** is $p(\tau_1, \ldots, \tau_n)$, where p is an n-ary predicate letter and the τ_i are terms. When p is 0-ary, we write p instead of $p()$. Also, we write $\tau_1 = \tau_2$ instead of $=(\tau_1, \tau_2)$, and $\tau_1 \neq \tau_2$ instead of $\neg(\tau_1 = \tau_2)$. **Formulas** are built from atomic formulas, propositional connectives $\top, \bot$ (0-ary), $\neg$ (1-ary), $\wedge$, $\vee$, $\sqcap$, $\sqcup$ (2-ary), variables and quantifiers $\forall, \exists, \bigsqcap, \bigsqcup$ in the standard way, with the exception that, officially, $\neg$ is only allowed to be applied to atomic formulas. The definitions of *free* and *bound* occurrences of variables are standard (with $\bigsqcap, \bigsqcup$ acting as quantifiers along with $\forall, \exists$). A formula with no free occurrences of variables is said to be **closed**.

Note that, terminologically, $\top$ and $\bot$ do not count as atoms. For us, atoms are formulas containing no logical operators. The formulas $\top$ and $\bot$ do not qualify because they *are* (0-ary) logical operators themselves.

$\neg E$, where E is not atomic, will be understood as a standard abbreviation: $\neg\top = \bot$, $\neg\neg E = E$, $\neg(A \wedge B) = \neg A \vee \neg B$, $\neg \bigsqcap x E = \bigsqcup x \neg E$, etc. And $E \rightarrow F$ will be understood as an abbreviation of $\neg E \vee F$.

Parentheses will often be omitted — as we just did — if there is no danger of ambiguity. When omitting parentheses, we assume that $\neg$ and the quantifiers have the highest precedence, and $\rightarrow$ has the lowest precedence. An expression $E_1 \wedge \ldots \wedge E_n$, where $n \geq 2$, is to be understood as $E_1 \wedge (E_2 \wedge (\ldots \wedge (E_{n-1} \wedge E_n) \ldots))$. Sometimes we can write this expression for an unspecified $n \geq 0$ (rather than $n \geq 2$). Such a formula, in the case of $n = 1$, should be understood as simply E_1. Similarly for $\vee$, $\sqcap$, $\sqcup$. As for the case of $n = 0$, $\wedge$ and $\sqcap$ should be understood as $\top$ while $\vee$ and $\sqcup$ as $\bot$.

Sometimes a formula F will be represented as $F(s_1, \ldots, s_n)$, where the s_i are variables. When doing so, we do not necessarily mean that each s_i has a free occurrence in F, or that every variable occurring free in F is among $s_1, \ldots, s_n$. However, it *will* always be assumed (usually only implicitly) that the s_i are pairwise distinct, and have no bound occurrences in F. In the context set by the above representation, $F(\tau_1, \ldots, \tau_n)$ will mean the result of replacing, in F, each occurrence of each s_i by term τ_i. When writing $F(\tau_1, \ldots, \tau_n)$, it will always be assumed (again, usually only implicitly) that the terms $\tau_1, \ldots, \tau_n$ contain no variables that have bound occurrences in F, so that there are no unpleasant collisions of variables when doing replacements.

Similar — well established in the literature — notational conventions apply to terms.

A **sequent** is an expression $E_1, \ldots, E_n \circ\!\!- F$, where $E_1, \ldots, E_n$ ($n \geq 0$) and F are formulas. Here $E_1, \ldots, E_n$ is said to be the **antecedent** of the sequent, and F said to be the **succedent**.

By a **free** (resp. **bound**) **variable** of a sequent we shall mean a variable that has a free (resp. bound) occurrence in one of the formulas of the sequent. For safety and simplicity, throughout the rest of this paper we assume that the sets of all free and bound variables of any sequent that we ever consider — unless strictly implied otherwise by the context — are disjoint. This restriction, of course, does not yield any loss of expressive power as variables can always be renamed so as to satisfy this condition.

An **interpretation**[10] is a pair $(U,^*)$, where $U = (U,^U)$ is a universe and * is a function that sends:

- every n-ary function letter f to a function $f^* : U^n \to U$;
- every nonlogical n-ary predicate letter p to an n-ary predicate (elementary game) $p^*(s_1, \ldots, s_n)$ on U which does not depend on any variables other than $s_1, \ldots, s_n$.

The above uniquely extends to a mapping that sends each term τ to a function τ^*, and each formula or sequent S to a game S^*, by stipulating that:

1. $c^* = c^U$ (any constant c).
2. $s^* = s$ (any variable s).
3. Where f is an n-ary function letter and $\tau_1, \ldots, \tau_n$ are terms, $\big(f(\tau_1, \ldots, \tau_n)\big)^* = f^*(\tau_1^*, \ldots, \tau_n^*)$.
4. Where τ_1 and τ_2 are terms, $(\tau_1 = \tau_2)^*$ is $\tau_1^* = \tau_2^*$.
5. Where p is an n-ary nonlogical predicate letter and $\tau_1, \ldots, \tau_n$ are terms, $\big(p(\tau_1, \ldots, \tau_n)\big)^* = p^*(\tau_1^*, \ldots, \tau_n^*)$.
6. * commutes with all logical operators, seeing them as the corresponding game operations: $\bot^* = \bot$, $(E_1 \wedge \ldots \wedge E_n)^* = E_1^* \wedge \ldots \wedge E_n^*$, $(\sqcap x E)^* = \sqcap x(E^*)$, etc.
7. Similarly, * sees the sequent symbol $\circ\!\!-$ as the same-name game operation, that is, $(E_1, \ldots, E_n \circ\!\!- F)^* = E_1^*, \ldots, E_n^* \circ\!\!- F^*$.

While an interpretation is a pair $(U,^*)$, terminologically and notationally we will usually identify it with its second component and write * instead of $(U,^*)$, keeping in mind that every such "interpretation" * comes with a fixed universe U, said to be the **universe of** *. When O is a function letter, a predicate letter, a constant or a formula, and $O^* = W$, we say that * **interprets** O as W. We can also refer to such a W as "O **under interpretation** *".

When a given formula is represented as $F(x_1, \ldots, x_n)$, we will typically write $F^*(x_1, \ldots, x_n)$ instead of $\big(F(x_1, \ldots, x_n)\big)^*$. A similar practice will be used for terms as well.

We agree that, for a sequent or formula S, an interpretation * and an HPM $\mathcal{M}$, whenever we say that $\mathcal{M}$ is a **solution** of S^* or write $\mathcal{M} \models S^*$, we mean that $\mathcal{M}$ is a solution of the (constant) game $\sqcap x_1 \ldots \sqcap x_n(S^*)$, where $x_1, \ldots, x_n$ are exactly the free variables of S, listed according to their lexicographic order. We call the above game the **$\sqcap$-closure** of S^*, and denote it by $\sqcap S^*$.

Note that, for any given sequent or formula S, the **Lr** component of the game $\sqcap S^*$ does not depend on the interpretation *. Hence we can safely say "legal run of $\sqcap S$" — or even just "legal run of S" — without indicating an interpretation applied to the sequent.

We say that an HPM $\mathcal{M}$ is a **uniform solution**, or a **logical solution**, of a sequent X iff, for any interpretation *, $\mathcal{M} \models X^*$.

Intuitively, a logical solution is (indeed) a "purely logical" solution. "Logical" in the sense that it does not depend on the universe and the meanings of the nonlogical symbols (predicate and function letters) — does not depend on a (the) interpretation *, that is. It is exactly these kinds of solutions that we are interested in when seeing CoL as a logical basis for applied theories or knowledge base systems. As a universal-utility tool, CoL (or a CoL-based compiler) would have no knowledge of the meanings of those nonlogical symbols (the meanings that will be changing from application to application and from theory to theory), other than what is explicitly given by the target formula and the axioms or the knowledge base of the system.

7 Logic CL12

The purpose of the deductive system **CL12** that we construct in this section is to axiomatize the set of sequents with logical solutions. Our formulation of the system relies on the terminology and notation explained below.

[10]The concept of an interpretation in CoL is usually more general than the present one. Interpretations in our present sense are called **perfect**. But here we omit the word "perfect" as we do not consider any nonperfect interpretations, anyway.

1. A **surface occurrence** of a subformula is an occurrence that is not in the scope of any choice operators ($\sqcap$, $\sqcup$, $\bigsqcap$ and/or $\bigsqcup$).

2. A formula not containing choice operators — i.e., a formula of the language of classical first order logic — is said to be **elementary**.

3. A sequent is **elementary** iff all of its formulas are so.

4. The **elementarization**
$$\|F\|$$
of a formula F is the result of replacing in F all $\sqcup$- and $\bigsqcup$-subformulas by $\bot$, and all $\sqcap$- and $\bigsqcap$-subformulas by $\top$. Note that $\|F\|$ is (indeed) an elementary formula.

5. The **elementarization** $\|G_1, \ldots, G_n \circ\!\!- F\|$ of a sequent $G_1, \ldots, G_n \circ\!\!- F$ is the elementary formula
$$\|G_1\| \wedge \ldots \wedge \|G_n\| \rightarrow \|F\|.$$

6. A sequent is said to be **stable** iff its elementarization is classically valid; otherwise it is **unstable**. By "classical validity", in view of Gödel's completeness theorem, we mean provability in classical first-order calculus with constants, function letters and $=$, where $=$ is treated as the logical *identity* predicate (so that, say, $x{=}x$, $x{=}y \rightarrow (E(x) \rightarrow E(y))$, etc. are provable).

7. We will be using the notation
$$F[E]$$
to mean a formula F together with some (single) fixed surface occurrence of a subformula E. Using this notation sets a context, in which $F[H]$ will mean the result of replacing in $F[E]$ the (fixed) occurrence of E by H. Note that here we are talking about some *occurrence* of E. Only that occurrence gets replaced when moving from $F[E]$ to $F[H]$, even if the formula also had some other occurrences of E.

8. By a **rule** (of inference) in this section we mean a binary relation $\mathbb{Y}\mathcal{R}X$, where $\mathbb{Y} = \langle Y_1, \ldots, Y_n \rangle$ is a finite sequence of sequents and X is a sequent. Instances of such a relation are schematically written as
$$\frac{Y_1, \ldots, Y_n}{X},$$
where $Y_1, \ldots, Y_n$ are called the **premises**, and X is called the **conclusion**. Whenever $\mathbb{Y}\mathcal{R}X$ holds, we say that X **follows** from $\mathbb{Y}$ by $\mathcal{R}$.

9. Expressions such as $\vec{G}, \vec{K}, \ldots$ will usually stand for finite sequences of formulas. The standard meaning of an expression such as $\vec{G}, F, \vec{K}$ should also be clear.

THE RULES OF CL12

CL12 has the six rules listed below, with the following additional conditions/explanations:

1. In $\sqcup$-Choose and $\sqcap$-Choose, $i \in \{0, 1\}$.

2. In $\bigsqcup$-Choose and $\bigsqcap$-Choose, $\mathfrak{t}$ is either a constant or a variable with no bound occurrences in the premise, and $H(\mathfrak{t})$ is the result of replacing by $\mathfrak{t}$ all free occurrences of x in $H(x)$ (rather than vice versa).

$\sqcup$-Choose

$$\frac{\vec{G} \ \circ\!\!- \ F[H_i]}{\vec{G} \ \circ\!\!- \ F[H_0 \sqcup H_1]}$$

$\sqcap$-Choose

$$\frac{\vec{G},\ E[H_i],\ \vec{K} \ \circ\!\!- \ F}{\vec{G},\ E[H_0 \sqcap H_1],\ \vec{K} \ \circ\!\!- \ F}$$

⊔-Choose

$$\frac{\vec{G} \circ\!\!- F[H(\mathfrak{t})]}{\vec{G} \circ\!\!- F[\sqcup x H(x)]}$$

⊓-Choose

$$\frac{\vec{G},\ E[H(\mathfrak{t})],\ \vec{K} \circ\!\!- F}{\vec{G},\ E[\sqcap x H(x)],\ \vec{K} \circ\!\!- F}$$

Replicate

$$\frac{\vec{G}, E, \vec{K}, E \circ\!\!- F}{\vec{G}, E, \vec{K} \circ\!\!- F}$$

Wait

$$\frac{Y_1, \ldots, Y_n}{X}$$ $(n \geq 0)$, where all of the following five conditions are satisfied:

1. $\sqcap$**-Condition:** Whenever X has the form $\vec{G} \circ\!\!- F[H_0 \sqcap H_1]$, both of the sequents $\vec{G} \circ\!\!- F[H_0]$ and $\vec{G} \circ\!\!- F[H_1]$ are among $Y_1, \ldots, Y_n$.

2. $\sqcup$**-Condition:** Whenever X has the form $\vec{G}, E[H_0 \sqcup H_1], \vec{K} \circ\!\!- F$, both of the sequents $\vec{G}, E[H_0], \vec{K} \circ\!\!- F$ and $\vec{G}, E[H_1], \vec{K} \circ\!\!- F$ are among $Y_1, \ldots, Y_n$.

3. ⊓**-Condition:** Whenever X has the form $\vec{G} \circ\!\!- F[\sqcap x H(x)]$, for some variable y not occurring in X, the sequent $\vec{G} \circ\!\!- F[H(y)]$ is among $Y_1, \ldots, Y_n$. Here and below, $H(y)$ is the result of replacing by y all free occurrences of x in $H(x)$ (rather than vice versa).

4. ⊔**-Condition:** Whenever X has the form $\vec{G}, E[\sqcup x H(x)], \vec{K} \circ\!\!- F$, for some variable y not occurring in X, the sequent $\vec{G}, E[H(y)], \vec{K} \circ\!\!- F$ is among $Y_1, \ldots, Y_n$.

5. **Stability condition:** X is stable.

As will be seen in Section 8, each rule — seen bottom-up — encodes an action that a winning strategy should take in a corresponding situation, and the name of each rule is suggestive of that action. For instance, Wait (indeed) prescribes the strategy to wait till the adversary moves. This explains why we have called "Replicate" the rule which otherwise is nothing but what is commonly known as Contraction.

A **CL12-proof** of a sequent X is a sequence $X_1, \ldots, X_n$ of sequents, with $X_n = X$, such that, each X_i follows by one of the rules of **CL12** from some (possibly empty in the case of Wait, and certainly empty in the case of $i = 1$) set $\mathcal{P}$ of premises such that $\mathcal{P} \subseteq \{X_1, \ldots, X_{i-1}\}$. When a **CL12**-proof of X exists, we say that X is **provable** in **CL12**, and write **CL12** $\vdash X$.

A **CL12-proof** of a formula F will be understood as a **CL12**-proof of the empty-antecedent sequent $\circ\!\!- F$. Accordingly, **CL12** $\vdash F$ means **CL12** $\vdash \circ\!\!- F$.

Fact 7.1 **CL12** *is a conservative extension of classical logic. That is, an elementary sequent* $E_1, \ldots, E_n \circ\!\!- F$ *is provable in* **CL12** *iff the formula* $E_1 \wedge \ldots \wedge E_n \rightarrow F$ *is valid in the classical sense.*

Proof. Assume $E_1, \ldots, E_n, F$ are elementary formulas. If $E_1 \wedge \ldots \wedge E_n \rightarrow F$ is classically valid, then $E_1, \ldots, E_n \circ\!\!- F$ follows from the empty set of premises by Wait. And if $E_1 \wedge \ldots \wedge E_n \rightarrow F$ is not classically valid, then $E_1, \ldots, E_n \circ\!\!- F$ cannot be the conclusion of any of the rules of **CL12** except Replicate. However, applying (bottom-up) Replicate does not take us any closer to finding a proof of the sequent, as the premise still remains an unstable elementary sequent. ■

CL12 can also be seen to be a conservative extension of the earlier known logic **CL3** studied in [15].[11] The latter is nothing but the empty-antecedent fragment of **CL12** without function letters and identity.

Example 7.2 In this example, $\times$ is a binary function letter and ^{3} is a unary function letter. We write $x \times y$ and x^3 instead of $\times(x, y)$ and $^3(x)$, respectively. The following sequence of sequents is a **CL12**-proof of the sequent (1) from Example 3.6. It may be worth observing that the strategy used by $\top$ in that example, in a sense, "follows" our present proof step-by-step in the bottom-up direction. And this is no accident: as we are going to see in the course of proving the soundness of **CL12**, every **CL12**-proof rather directly encodes a winning strategy.

1. $\forall x\bigl(x^3=(x\times x)\times x\bigr),\ t=s\times s,\ r=t\times s \ \circ\!\!-\ r=s^3$ Wait: (no premises)
2. $\forall x\bigl(x^3=(x\times x)\times x\bigr),\ t=s\times s,\ r=t\times s \ \circ\!\!-\ \sqcup y(y=s^3)$ $\sqcup$-Choose: 1
3. $\forall x\bigl(x^3=(x\times x)\times x\bigr),\ t=s\times s,\ \sqcup z(z=t\times s) \ \circ\!\!-\ \sqcup y(y=s^3)$ Wait: 2
4. $\forall x\bigl(x^3=(x\times x)\times x\bigr),\ t=s\times s,\ \sqcap y\sqcup z(z=t\times y) \ \circ\!\!-\ \sqcup y(y=s^3)$ $\sqcap$-Choose: 3
5. $\forall x\bigl(x^3=(x\times x)\times x\bigr),\ t=s\times s,\ \sqcap x\sqcap y\sqcup z(z=x\times y) \ \circ\!\!-\ \sqcup y(y=s^3)$ $\sqcap$-Choose: 4
6. $\forall x\bigl(x^3=(x\times x)\times x\bigr),\ \sqcup z(z=s\times s),\ \sqcap x\sqcap y\sqcup z(z=x\times y) \ \circ\!\!-\ \sqcup y(y=s^3)$ Wait: 5
7. $\forall x\bigl(x^3=(x\times x)\times x\bigr),\ \sqcap y\sqcup z(z=s\times y),\ \sqcap x\sqcap y\sqcup z(z=x\times y) \ \circ\!\!-\ \sqcup y(y=s^3)$ $\sqcap$-Choose: 6
8. $\forall x\bigl(x^3=(x\times x)\times x\bigr),\ \sqcap x\sqcap y\sqcup z(z=x\times y),\ \sqcap x\sqcap y\sqcup z(z=x\times y) \ \circ\!\!-\ \sqcup y(y=s^3)$ $\sqcap$-Choose: 7
9. $\forall x\bigl(x^3=(x\times x)\times x\bigr),\ \sqcap x\sqcap y\sqcup z(z=x\times y) \ \circ\!\!-\ \sqcup y(y=s^3)$ Replicate: 8
10. $\forall x\bigl(x^3=(x\times x)\times x\bigr),\ \sqcap x\sqcap y\sqcup z(z=x\times y) \ \circ\!\!-\ \sqcap x\sqcup y(y=x^3)$ Wait: 9

Example 7.3 The formula $\forall x p(x) \rightarrow \sqcap x p(x)$ is provable in **CL12**. It follows from $\forall x p(x) \rightarrow p(y)$ by Wait. The latter, in turn, follows by Wait from the empty set of premises.

On the other hand, the formula $\sqcap x p(x) \rightarrow \forall x p(x)$, i.e. $\sqcup x \neg p(x) \vee \forall x p(x)$, in not provable. Indeed, its elementarization is $\perp \vee \forall x p(x)$, which is not classically valid. Hence $\sqcup x \neg p(x) \vee \forall x p(x)$ cannot be derived by Wait. Replicate can also be dismissed for obvious reasons. This leaves us with $\sqcup$-Choose. But if $\sqcup x \neg p(x) \vee \forall x p(x)$ is derived by $\sqcup$-Choose, then the premise should be $\neg p(\mathfrak{t}) \vee \forall x p(x)$ for some variable or constant $\mathfrak{t}$. The latter, however, is a classically non-valid elementary formula and hence, by Fact 7.1, is not provable.

Example 7.4 The formula $\sqcap x \sqcup y\bigl(p(x) \rightarrow p(y)\bigr)$ is provable in **CL12** as follows:

1. $p(s) \rightarrow p(s)$ Wait:
2. $\sqcup y\bigl(p(s) \rightarrow p(y)\bigr)$ $\sqcup$-Choose: 1
3. $\sqcap x \sqcup y\bigl(p(x) \rightarrow p(y)\bigr)$ Wait: 2

On the other hand, the formula $\sqcup y \sqcap x\bigl(p(x) \rightarrow p(y)\bigr)$ can be seen to be unprovable, even though its classical counterpart $\exists y \forall x\bigl(p(x) \rightarrow p(y)\bigr)$ is a classically valid elementary formula and hence provable in **CL12**.

Example 7.5 While the formula $\forall x \exists y\bigl(y=f(x)\bigr)$ is classically valid and hence provable in **CL12**, its constructive counterpart $\sqcap x \sqcup y\bigl(y=f(x)\bigr)$ can be easily seen to be unprovable. This is no surprise. In view of the expected soundness of **CL12**, provability of $\sqcap x \sqcup y\bigl(y=f(x)\bigr)$ would imply that every function f is computable, which, of course, is not the case.

Exercise 7.6 To see the resource-consciousness of **CL12**, show that it does not prove $p \sqcap q \rightarrow (p \sqcap q) \wedge (p \sqcap q)$, even though this formula has the form $F \rightarrow F \wedge F$ of a classical tautology. Then show that, in contrast, **CL12** proves the *sequent* $p \sqcap q \ \circ\!\!-\ (p \sqcap q) \wedge (p \sqcap q)$ because, unlike the antecedent of a $\rightarrow$-combination, the antecedent of a $\circ\!\!-$-combination is reusable (trough Replicate).

[11] Essentially the same logic, called **L**, was in fact known as early as in [9].

Exercise 7.7 Show that **CL12** $\vdash \sqcup x \sqcap y\, p(x,y) \circ\!\!- \sqcup x\big(\sqcap y\, p(x,y) \wedge \sqcap y\, p(x,y)\big)$. Then observe that, on the other hand, **CL12** does not prove any of the formulas

$$\begin{array}{rcl}
\sqcup x \sqcap y\, p(x,y) & \to & \sqcup x\big(\sqcap y\, p(x,y) \wedge \sqcap y\, p(x,y)\big);\\
\sqcup x \sqcap y\, p(x,y) \;\wedge\; \sqcup x \sqcap y\, p(x,y) & \to & \sqcup x\big(\sqcap y\, p(x,y) \wedge \sqcap y\, p(x,y)\big);\\
\sqcup x \sqcap y\, p(x,y) \;\wedge\; \sqcup x \sqcap y\, p(x,y) \;\wedge\; \sqcup x \sqcap y\, p(x,y) & \to & \sqcup x\big(\sqcap y\, p(x,y) \wedge \sqcap y\, p(x,y)\big);\\
 & \ldots &
\end{array}$$

Intuitively, this contrast is due to the fact that, even though both $\overset{|}{\circ} A$ and $\curlywedge A = A \wedge A \wedge A \wedge \ldots$ are resources allowing to reuse A any number of times, the "branching" form of reusage offered by $\overset{|}{\circ} A$ is substantially stronger than the "parallel" form of reusage offered by $\curlywedge A$. $\overset{|}{\circ} \sqcup x \sqcap y\, p(x,y) \to \sqcup x\big(\sqcap y\, p(x,y) \wedge \sqcap y\, p(x,y)\big)$ is a valid principle of CoL while $\curlywedge \sqcup x \sqcap y\, p(x,y) \to \sqcup x\big(\sqcap y\, p(x,y) \wedge \sqcap y\, p(x,y)\big)$ is not.

8 The soundness of CL12

We say that a logical solution $\mathcal{M}$ of a sequent X is **well-behaved** iff the following conditions are satisfied:

1. There is an integer d such that, in every play, $\mathcal{M}$ makes at most d replicative moves in the antecedent of X.
2. Every non-replicative move that $\mathcal{M}$ makes in the antecedent of X is focused.
3. Every time when $\mathcal{M}$ chooses a constant c for a variable x in some $\sqcup xG$ or $\sqcap xG$ component of X, c is either 0, or a constant that occurs in X, or a constant already chosen by Environment for some variable y in some $\sqcap yG$ or $\sqcup yG$ component of $\sqcap X$.

The terms of the language of **CL12**, identified with their parse trees, are tree-style structures, so let us call them **tree-terms**. A more general and economical way to represent terms, however, is to allow merging some or all identical-content nodes in such trees, thus turning them into (directed, acyclic, rooted, edge-ordered multi-) graphs. Let us call these (unofficial) sorts of terms **graph-terms**. The idea of representing linguistic objects in the form of graphs rather than trees is central in the approach called *cirquent calculus* ([13, 20]), and has already proven its worth. We find that idea particularly useful in our present, complexity-sensitive context. Figure 1 illustrates two terms representing the same polynomial function y^8, with the term on the right being a tree-term and the term on the left being a graph-term. As this example suggests, graph-terms are generally exponentially smaller than the corresponding tree-terms, which explains our preference for the former as a standard way of writing (in our metalanguage) polynomial terms. Figure 1 also makes it unnecessary to formally define graph-terms, as their meaning must be perfectly clear after looking at this single example.

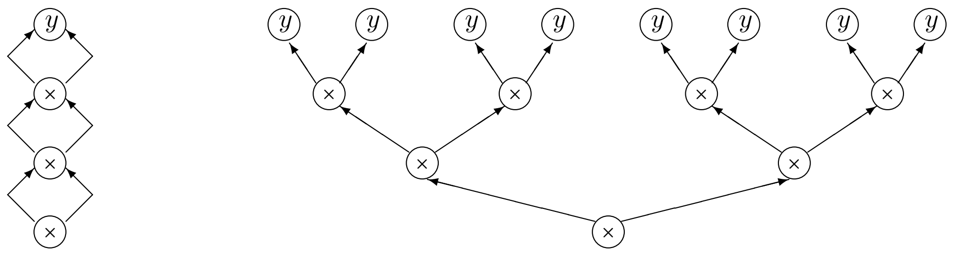

Figure 1: A graph-term and the corresponding tree-term

By an **explicit polynomial function** τ we shall mean a graph-term not containing (at its leaves) any constants other than 0, and not containing (at its internal nodes) any function letters other than $'$ (unary), $+$ (binary) and $\times$ (binary). The total number k of the variables $y_1,\ldots,y_k$ occurring in (at the leaves of) τ is said to be the **arity** of τ. Terminologically and notationally we shall usually identify such a term τ with the k-ary arithmetical function represented by it under the standard arithmetical interpretation (x' means

$x+1$). So, for instance, either term of Figure 1 is a unary explicit polynomial function, representing — and identified with — the function y^8.

When τ is a unary explicit polynomial function and $\mathcal{M}$ is a τ time (resp. space) machine, we say that τ is an **explicit polynomial bound** for the time (resp. space) complexity of $\mathcal{M}$.

Another auxiliary concept that we are going to rely on in this section and later is that of a **generalized HPM** (**GHPM**). For a natural number n, an n-ary GHPM is defined in the same way as an HPM, with the difference that the former takes n natural numbers as inputs (say, provided on a separate, read-only *input tape*); such inputs are present at the very beginning of the work of the machine and remain unchanged throughout it. An ordinary HPM is thus nothing but a 0-ary GHPM. When $\mathcal{M}$ is an n-ary GHPM and $c_1, \ldots, c_n$ are natural numbers, $\mathcal{M}(c_1, \ldots, c_n)$ denotes the HPM that works just like $\mathcal{M}$ in the scenario where the latter has received $c_1, \ldots, c_n$ as inputs. We will assume that some reasonable encoding (through natural numbers) of GHPMs is fixed. When $\mathcal{M}$ is a GHPM, $\ulcorner\mathcal{M}\urcorner$ denotes its **code**.

Theorem 8.1 *Every sequent provable in* **CL12** *has a well-behaved polynomial time and polynomial space*[12] *logical solution. Furthermore, such a solution, together with an explicit polynomial bound for both its time and space complexities, can be efficiently constructed*[13] *from a proof of the sequent.*

The rest of this section is exclusively devoted to a proof of the above theorem. For pedagogical reasons, we first prove the main part of the theorem, without the "furthermore" clause, which will be taken care only at the end of the section. Our proof of the "pre-furthermore" part proceeds by induction on the length of (the number of sequents involved in) a **CL12**-proof of a sequent X and, as such, is nothing but a combination of six cases, corresponding to the six rules of **CL12** by which the final sequent X could have been derived from its premises (if any).

In each case, our efforts will be focused on showing how to construct an HPM $\mathcal{M}$ — a logical solution of the conclusion — from an arbitrary instance

$$\frac{Y_1, \ldots, Y_n}{X}$$

of the rule and arbitrary HPMs $\mathcal{N}_1, \ldots, \mathcal{N}_n$ — well-behaved polynomial time and polynomial space solutions of the premises that exist according to the induction hypothesis. Typically it will be immediately clear from our description of $\mathcal{M}$ that it is well-behaved and runs in polynomial time and space, and that its work in no way does depend on an interpretation * applied to the sequents involved, so that the solution is logical. Also, our implicit assumption will be that $\mathcal{M}$'s adversary never makes illegal moves, or else $\mathcal{M}$ easily detects the illegal behavior and retires[14] with a decisive victory.

Since an interpretation * is typically irrelevant in such proofs, we will usually omit it and write simply S where, strictly speaking, S^* is meant. That is, we identify formulas or sequents with the games into which they turn once an interpretation is applied to them. Accordingly, in contexts where S^* has to be understood as $\sqcap S^*$ anyway (e.g., when talking about computability of S^*), we may omit "$\sqcap$" and write S instead of $\sqcap S$.

8.1 $\sqcup$-Choose

$$\frac{\vec{G} \circ\!\!- F[H_i]}{\vec{G} \circ\!\!- F[H_0 \sqcup H_1]}$$

Assume (induction hypothesis) that ξ is a unary explicit polynomial function and $\mathcal{N}$ is a well-behaved ξ time and ξ space logical solution of the premise $\vec{G} \circ\!\!- F[H_i]$. We want to (show how to) construct a well-behaved logical solution $\mathcal{M}$ of the conclusion $\vec{G} \circ\!\!- F[H_0 \sqcup H_1]$, together with an explicit polynomial bound τ for its time and space complexities.

[12]Note that, unlike ordinary Turing machines, not every polynomial time HPM runs in polynomial space. In this paper we leave unaddressed the natural question about whether every (interactive) computational problem with a polynomial time solution also has a polynomial space solution.

[13]Here and later in similar meta-contexts, by "efficiently" we mean "in polynomial time". Also, "can be efficiently constructed" precisely means that there is an efficient (polynomial time) procedure that does the construction for an arbitrary proof of an arbitrary sequent.

[14]Technically, "retiring" can be understood as going into an infinite loop that makes no moves and consumes no space.

For the beginning, let us consider the case when the conclusion (and hence also the premise) is closed, i.e., has no free occurrences of variables. The idea here is very simple: $\sqcup$-Choose most directly encodes an action that $\mathcal{M}$ should perform in order to successfully solve the conclusion. Namely, $\mathcal{M}$ should choose H_i and then continue playing as $\mathcal{N}$. $\mathcal{M}$ wins because the above initial move brings the conclusion down to the premise, and $\mathcal{N}$ wins the latter. And $\mathcal{M}$ automatically inherits the well-behavedness of $\mathcal{N}$. $\mathcal{M}$ is only "slightly" slower[15] than $\mathcal{N}$, and it would be no problem to indicate an explicit polynomial function τ (depending on ξ) such that $\mathcal{M}$ runs in time τ. The same applies to space, so that we may assume that the above τ is also a polynomial bound for the space complexity of $\mathcal{M}$. Also note that our construction does not depend on an interpretation * applied to the sequents under question, so that $\mathcal{M}$ is a logical solution of the conclusion.

It now remains to consider the case when the conclusion is not closed. This is pretty similar to the previous case. The only difference in the work of $\mathcal{M}$ will be that now, before making the move that brings the conclusion down to the premise, $\mathcal{M}$ waits till Environment chooses constants for all free variables of $\vec{G} \circ\!- F[H_0 \sqcup H_1]$. Then, after choosing H_i, it continues playing as $\mathcal{N}$ would play in the scenario where, at the very beginning of the play, the adversary of the latter chose the same constants for the free variables of $\vec{G} \circ\!- F[H_i]$ as $\mathcal{M}$'s environment did.

8.2 $\sqcap$-Choose

This case is similar to the previous one.

8.3 $\sqcup$-Choose

$$\frac{\vec{G} \ \circ\!- \ F[H(\mathfrak{t})]}{\vec{G} \ \circ\!- \ F[\sqcup x H(x)]}$$

Taking into account that the choice existential quantifier is nothing but a "long" choice disjunction, this case is rather similar to the case of $\sqcup$-Choose. As in that case, assume $\mathcal{N}$ is a well-behaved logical solution of the premise and ξ is an explicit polynomial bound for its time and space complexities. We want to construct a well-behaved polynomial time and polynomial space logical solution $\mathcal{M}$ of the conclusion.

First, consider the case of $\mathfrak{t}$ being a constant. We let $\mathcal{M}$ be a machine that works as follows. At the beginning, $\mathcal{M}$ waits till Environment specifies some constants for all free variables of $\vec{G} \circ\!- F[\sqcup x H(x)]$. For readability, we continue referring to the resulting game as $\vec{G} \circ\!- F[\sqcup x H(x)]$, even though, strictly speaking, it is $e[\vec{G} \circ\!- F[\sqcup x H(x)]]$, where e is a valuation that agrees with the choices that Environment just made for the free variables of the sequent. Now $\mathcal{M}$ makes the move that brings $\vec{G} \circ\!- F[\sqcup x H(x)]$ down to $\vec{G} \circ\!- F[H(\mathfrak{t})]$. For instance, if $\vec{G} \circ\!- F[\sqcup x H(x)]$ is $\vec{G} \circ\!- E \wedge (K \vee \sqcup x H(x))$ and thus $\vec{G} \circ\!- F[H(\mathfrak{t})]$ is $\vec{G} \circ\!- E \wedge (K \vee H(\mathfrak{t}))$, then 1.1.1.$\mathfrak{t}$ is such a move. After this move, $\mathcal{M}$ "turns itself into $\mathcal{N}$" in the same fashion as in the proof of the case of $\sqcup$-Choose. And, again, $\mathcal{M}$ is guaranteed to be a τ time and τ space logical solution of the conclusion, where τ is an explicit polynomial function that can be easily constructed from ξ.

The case of $\mathfrak{t}$ being a variable that is among the free variables of the conclusion can be handled in a similar way, with the only difference that now, when making the move that brings the conclusion down to the premise, $\mathcal{M}$ chooses, for x, the constant chosen by Environment for $\mathfrak{t}$.

The remaining case is that of $\mathfrak{t}$ being a variable that is not among the free variables of the conclusion. In this case, when making the move that brings the conclusion down to the premise, $\mathcal{M}$ (arbitrarily) chooses the constant 0 for x; in addition, when "turning itself into $\mathcal{N}$", $\mathcal{M}$ follows the scenario where $\mathcal{N}$'s adversary chose the same constant 0 for $\mathfrak{t}$.

In any of the above cases, it is clear that $\mathcal{M}$ is a logical solution of the conclusion, and that $\mathcal{M}$ inherits the well-behaved property of $\mathcal{N}$.

8.4 $\sqcap$-Choose

This case is similar to the previous one.

[15]Namely, it is simply the *same* in the asymptotic sense.

8.5 Replicate

$$\frac{\vec{G}, E, \vec{K}, E \circ\!\!- F}{\vec{G}, E, \vec{K} \circ\!\!- F}$$

Remembering that we agreed to see no distinction between sequents and the games they represent, and disabbreviating $\vec{G}$,$\vec{K}$, $\circ\!\!-$, the premise and the conclusion of this rule can be rewritten as the following two games, respectively:

$$\overset{|}{\circ}G_1 \wedge \ldots \wedge \overset{|}{\circ}G_m \;\wedge\; \overset{|}{\circ}E \;\wedge\; \overset{|}{\circ}K_1 \wedge \ldots \wedge \overset{|}{\circ}K_n \;\wedge\; \overset{|}{\circ}E \;\;\rightarrow\;\; F \tag{2}$$

$$\overset{|}{\circ}G_1 \wedge \ldots \wedge \overset{|}{\circ}G_m \;\wedge\; \overset{|}{\circ}E \;\wedge\; \overset{|}{\circ}K_1 \wedge \ldots \wedge \overset{|}{\circ}K_n \;\;\rightarrow\;\; F \tag{3}$$

Assume $\mathcal{N}$ is a well-behaved logical solution of the premise, and ξ is an explicit polynomial bound for its time and space complexities. We let a solution $\mathcal{M}$ of the conclusion be a machine that works as follows.

After Environment chooses some constants for all free variables of (3), $\mathcal{M}$ makes a replicative move in the $\overset{|}{\circ}E$ component of the latter, bringing the game down to

$$\overset{|}{\circ}G_1 \wedge \ldots \wedge \overset{|}{\circ}G_m \;\wedge\; \overset{|}{\circ}(E \circ E) \;\wedge\; \overset{|}{\circ}K_1 \wedge \ldots \wedge \overset{|}{\circ}K_n \;\;\rightarrow\;\; F \tag{4}$$

(more precisely, it will be not (4) but $e[(4)]$, where e is a valuation that agrees with Environment's choices for the free variables of the sequent. As we did earlier, however, notationally we ignore this difference).

Now we need to observe that (4) is "essentially the same as" (2), so that $\mathcal{M}$ can continue playing "essentially as" $\mathcal{N}$ would play in the scenario where the adversary of $\mathcal{N}$ chose the same constants for free variables as the adversary of $\mathcal{M}$ just did. All that $\mathcal{M}$ needs to do to account for the minor technical differences between (4) and (2) is to make a very simple "reinterpretation" of moves. Namely:

- Any move made within any of the $\overset{|}{\circ}G_i$ or $\overset{|}{\circ}K_i$ components of (4) $\mathcal{M}$ sees exactly as $\mathcal{N}$ would see the same move in the same component of (2), and vice versa.
- Any replicative or focused nonreplicative move made within the left (resp. right) leaf of the $\overset{|}{\circ}(E \circ E)$ component of (4) $\mathcal{M}$ sees as $\mathcal{N}$ would see the same move as if it was made in the (single) leaf of the first (resp. second) $\overset{|}{\circ}E$ component of (2), and vice versa.
- Any unfocused nonreplicative move made by Environment in the $\overset{|}{\circ}(E \circ E)$ component of (4) $\mathcal{M}$ sees as $\mathcal{N}$ would see the same move made twice (but on the same clock cycle) by Environment: once in the leaf of the first $\overset{|}{\circ}E$ component of (2), and once in the leaf of the second $\overset{|}{\circ}E$ component of (2).

With a little thought, it can be seen that $\mathcal{M}$ wins because so does $\mathcal{N}$. Further, neither making the initial replicative move nor "reinterpreting" moves in the above fashion is expensive in terms of time or space, so that, based on ξ, it would be no problem to specify an explicit polynomial bound τ for the time and space complexities of $\mathcal{M}$. And, as always, $\mathcal{M}$ obviously inherits the well-behavedness of $\mathcal{N}$.

8.6 Wait

$$\frac{Y_1, \ldots, Y_n}{X}$$

(where $n \geq 0$ and the $\sqcap$-, $\sqcup$-, $\bigsqcap$-, $\bigsqcup$- and Stability conditions are satisfied).

We shall rely on the following lemma. It can be verified by a straightforward induction on the complexity of Z, which we omit. Remember that $\langle\rangle$ stands for the empty run.

Lemma 8.2 *For any sequent Z, valuation e and interpretation *, $\mathbf{Wn}_e^{Z^*}\langle\rangle = \mathbf{Wn}_e^{\|Z\|^*}\langle\rangle$.*

Assume $\mathcal{N}_1, \ldots, \mathcal{N}_n$ are well-behaved logical solutions of $Y_1, \ldots, Y_n$, respectively, and $\xi_1, \ldots, \xi_n$ are polynomial bounds for the time and space complexities of the corresponding machines. We let $\mathcal{M}$, a logical solution of X, be a machine that works as follows.

At the beginning, as always, $\mathcal{M}$ waits till Environment chooses some constants for all free variables of the conclusion. Let e be a (the) valuation that agrees with the choices just made by Environment (in the previous cases, we have suppressed the e parameter, but now we prefer to deal with it explicitly). So, the conclusion is now brought down to $e[X]$. After this event, $\mathcal{M}$ continues waiting until Environment makes one more move. If such a move is never made, then the run of (the $\sqcap$-closure of) X generated in the play can be simply seen as the empty run of $e[X]$. Due to the Stability condition, $\|X\|$ is classically valid, meaning that $\mathbf{Wn}_e^{\|X\|}\langle\rangle = \top$. But then, in view of Lemma 8.2, $\mathbf{Wn}_e^{X}\langle\rangle = \top$. This makes $\mathcal{M}$ the winner.

Suppose now Environment makes a move α. With a little thought, one can see that any (legal) move α by Environment brings the game $e[X]$ down to $g[Y_i]$ for a certain valuation g and one of the premises Y_i of the rule. For example, if X is $\circ\!\!-(E \sqcap F) \vee \sqcap x G(x)$, then a legal move α by Environment should be either 1.0.0 or 1.0.1 or 1.1.c for some constant c. In the case $\alpha = 1.0.0$, the above-mentioned premise Y_i will be $\circ\!\!- E \vee \sqcap x G(x)$, and g will be the same as e. In the case $\alpha = 1.0.1$, Y_i will be $\circ\!\!- F \vee \sqcap x G(x)$, and g, again, will be the same as e. Finally, in the case $\alpha = 1.1.c$, Y_i will be $\circ\!\!-(E \sqcap F) \vee G(y)$ for a variable y not occurring in X, and g will be the valuation that sends y to the object named by c and agrees with e on all other variables, so that $g[\circ\!\!-(E \sqcap F) \vee G(y)]$ is $e[\circ\!\!-(E \sqcap F) \vee G(c)]$, with the latter being the game to which $e[X]$ is brought down by the labmove $\bot 1.1.c$.

After the above event, $\mathcal{M}$ does the usual trick of turning itself into — and continuing playing as — $\mathcal{N}_i$, with the only difference that, if $g \neq e$, the behavior of $\mathcal{N}_i$ should be followed for the scenario where the adversary of the latter, at the very beginning of the play, chose constants for the free variables of Y_i in accordance with g rather than e.

As in the preceding proofs, it can be seen that $\mathcal{M}$ is a well-behaved logical solution of X. Keeping in mind that $\mathcal{M}$ is not billed for the time during which it waits for Environment to move, it is clear that, asymptotically, its time complexity does not exceed (the generously taken) $\xi_1 + \ldots + \xi_n$. Unlike time, however, $\mathcal{M}$ *will* be billed for any extra space consumed while waiting. But, fortunately, it does not use any space during that period, as it simply keeps reading the leftmost blank cell of its run tape to see if Environment has made a move. So, an explicit polynomial bound τ for both the time and space complexities of $\mathcal{M}$ can be easily obtained from $\xi_1, \ldots, \xi_n$.

8.7 Taking care of the "furthermore" clause of Theorem 8.1.

Thus, we have shown how to construct, from a proof of X, an HPM $\mathcal{M}$ and an explicit polynomial function τ such that $\mathcal{M}$ solves X in time and space τ. Obviously our construction is effective. It remains to see that it also is — or, at least, can be made — efficient. Of course, at every step of our construction (for each sequent of the proof, that is), in Sections 8.1-8.6, the solution $\mathcal{M}$ of the step and its time/space complexity bound τ is obtained efficiently from previously constructed $\mathcal{M}$s and τs. This, however, does not guarantee that the entire construction will be efficient as well. For instance, if the proof has n steps and the size of each $\mathcal{M}$ that we construct for each step is twice the size of the previously constructed HPMs, then the size of the eventual HPM will exceed 2^n and thus the construction will not be efficient, even if each of the n steps of it is so.

A trick that we can use to avoid an exponential growth of the sizes of the machines that we construct and thus achieve the efficiency of the entire construction is to deal with GHPMs instead of HPMs. Namely, assume the proof of X is the sequence $X_1, \ldots, X_n$ of sequents, with $X = X_n$. Let $\mathcal{M}_1, \ldots, \mathcal{M}_n$ be the HPMs constructed as we constructed $\mathcal{M}$s earlier in Sections 8.1-8.6 for the corresponding steps/sequents. Remember that each such $\mathcal{M}_i$ was defined in terms of $\mathcal{M}_{j_1}, \ldots, \mathcal{M}_{j_k}$ for some $j_1, \ldots, j_k < i$. For simplicity and uniformity, we may just as well say that each $\mathcal{M}_i$ was defined in terms of all $\mathcal{M}_1, \ldots, \mathcal{M}_n$, with those $\mathcal{M}_j$s that were not among $\mathcal{M}_{j_1}, \ldots, \mathcal{M}_{j_k}$ simply ignored in the description of the work of $\mathcal{M}_i$. Now, for each such $\mathcal{M}_i$, let $\mathcal{M}'_i$ be the n-ary GHPM whose description is obtained from that of $\mathcal{M}_i$ by replacing each reference to (any previously constructed) $\mathcal{M}_j$ by "$\mathcal{M}'_j(\ulcorner \mathcal{M}'_1 \urcorner, \ldots, \ulcorner \mathcal{M}'_n \urcorner)$ where, for each $e \in \{1, \ldots, n\}$, $\mathcal{M}'_e$ is the machine encoded by the eth input".[16] As it is easy to see by induction on i, $\mathcal{M}_i$ and $\mathcal{M}'_i(\ulcorner \mathcal{M}'_1 \urcorner, \ldots, \ulcorner \mathcal{M}'_n \urcorner)$ are essentially the same, in the sense that our earlier analysis of the play and time/space complexity of the former applies to the latter just as well. So, $\mathcal{M}'_n(\ulcorner \mathcal{M}'_1 \urcorner, \ldots, \ulcorner \mathcal{M}'_n \urcorner)$ wins X_n, i.e. X. At the same time, note that the size of each GHPM $\mathcal{M}'_i$ is independent of the sizes of the other (previously constructed) GHPMs.

[16]For simplicity, here we assume that every number is a code of some n-ary GHPM; alternatively, $\mathcal{M}'_i$ can be defined so that it does nothing if any of its relevant inputs is not the code of some n-ary GHPM.

Based on this fact, with some analysis, one can see that then the HPM $\mathcal{M}'_n(\ulcorner\mathcal{M}'_1\urcorner,\ldots,\ulcorner\mathcal{M}'_n\urcorner)$ is indeed constructed efficiently.

As for the explicit polynomial bounds $\tau_1,\ldots,\tau_n$ for the time and space complexities of the n HPMs $\mathcal{M}'_1(\ulcorner\mathcal{M}'_1\urcorner,\ldots,\ulcorner\mathcal{M}'_n\urcorner)$, $\ldots$, $\mathcal{M}'_n(\ulcorner\mathcal{M}'_1\urcorner,\ldots,\ulcorner\mathcal{M}'_n\urcorner)$, their sizes can be easily seen to be polynomial in the size of the proof. That is because, for each $i \in \{1,\ldots,n\}$, the size of τ_i only increases the sizes of the earlier constructed τ_js by adding (rather than multiplying by) a certain polynomial quantity.[17] Thus, the explicit bound τ_n for the time and space complexities of the eventual HPM $\mathcal{M}'_n(\ulcorner\mathcal{M}_1\urcorner,\ldots,\ulcorner\mathcal{M}_n\urcorner)$ is indeed constructed efficiently.

9 The completeness of CL12

Theorem 9.1 *Every sequent with a logical solution is provable in* **CL12***.*

Proof. Assume X is a sequent not provable in **CL12**. Our goal is to show that X has no logical solution (let alone a polynomial time, polynomial space and/or well-behaved logical solution).

Here we describe a *counterstrategy*, i.e., Environment's strategy, against which any particular HPM (in the usual role of $\top$) loses X^* for an appropriately selected interpretation *. In precise terms, as a mathematical object, our counterstrategy — let us call it $\mathcal{C}$ — is a (not necessarily effective) function that prescribes, for each possible content of the run tape that may arise during the process of playing the game, a (possibly empty) sequence of moves that Environment should make during the corresponding clock cycle. In what follows, whenever we say that $\mathcal{C}$ wins or loses, we mean that so does $\bot$ when it acts according to such prescriptions. $\mathcal{C}$ and $\bot$ will be used interchangeably, that is.

By a *variables-to-constants mapping* — or **vc-mapping** for short — for a sequent Y we shall mean a function whose domain is some finite set of variables that contains all (but not necessarily only) the free variables of Y and whose range is some set of constants not occurring in Y, such that to any two (graphically) different variables are assigned (graphically) different constants. When e is a vc-mapping for Y, by $e[Y]$ we shall mean the result of replacing in Y each free occurrence of every variable with the constant assigned to that variable by e.

At the beginning of the play, $\mathcal{C}$ chooses different constants for (all) different free variables of X, also making sure that none of these constants are among the ones that occur in X. Let g be the corresponding vc-mapping for X. This initial series of moves brings X (under whatever interpretation) down to the constant game $g[X]$ (under the same interpretation).

The way $\mathcal{C}$ works after that can be defined recursively. At any time, $\mathcal{C}$ deals with a pair (Y,e), where Y is a **CL12**-unprovable sequent and e is a vc-mapping for Y, such that $e[Y]$ is the game to which the initial $\sqcap X$ has been brought down "by now". The initial value of Y is X, and the initial value of e is the above vc-mapping g. How $\mathcal{C}$ acts on (Y,e) depends on whether Y is stable or not.

CASE 1: Y is stable. Then there should be a **CL12**-unprovable sequent Z satisfying one of the following conditions, for otherwise Y would be derivable by Wait. $\mathcal{C}$ selects one such Z (say, lexicographically the smallest one), and acts according to the corresponding prescription as given below.

Subcase 1.1: Y has the form $\vec{E}\circ\!\!-F[G_0\sqcap G_1]$, and Z is $\vec{E}\circ\!\!-F[G_i]$ ($i=0$ or $i=1$). In this case, $\mathcal{C}$ makes the move that brings Y down to Z (more precisely, $e[Y]$ down to $e[Z]$), and calls itself on (Z,e).

Subcase 1.2: Y has the form $\vec{E},F[G_0\sqcap G_1],\vec{K}\circ\!\!-H$, and Z is $\vec{E},F[G_i],\vec{K}\circ\!\!-H$. This subcase is similar to the previous one.

Subcase 1.3: Y has the form $\vec{E}\circ\!\!-F[\sqcap xG(x)]$, and Z is $\vec{E}\circ\!\!-F[G(y)]$, where y is a variable not occurring in Y. In this case, $\mathcal{C}$ makes a move that brings Y down to $\vec{E}\circ\!\!-F[G(c)]$ for some (say, the smallest) constant c such that c is different from any constant occurring in $e[Y]$. After this move, $\mathcal{C}$ calls itself on (Z,e'), where e' is the vc-mapping for Z that sends y to c and agrees with e on all other variables.

Subcase 1.4: Y has the form $\vec{E},F[\sqcap xG(x)],\vec{K}\circ\!\!-H$, and Z is $\vec{E},F[G(y)],\vec{K}\circ\!\!-H$, where y is a variable not occurring in Y. This subcase is similar to the previous one.

[17]The fact that we represent complexity bounds as graph-terms rather than tree-terms is relevant here. It would however be irrelevant if we had defined **CL12**-proofs as trees rather than sequences of sequents. The reason why we have opted for linear rather than tree-like proofs is that, at the expense of recycling/reusing intermediate steps, linear proofs can be exponentially smaller than tree-like proofs.

$\mathcal{C}$ repeats the above until (the continuously updated) Y becomes unstable. This results is some finite series of moves made by $\mathcal{C}$. We assume that all these moves are made during a single clock cycle (remember that there are no restrictions in the HPM model on how many moves Environment can make during a single cycle).

CASE 2: Y is unstable. $\mathcal{C}$ does not make any moves, but rather waits until its adversary makes a move.

Subcase 2.1: The adversary never makes a move. Then the run of $e[Y]$ that is generated is empty. As Y is unstable, $\|Y\|$ and hence $\|e[Y]\|$ is not classically valid. That is, $\|e[Y]\|$ is false in some classical model. But classical models are nothing but our interpretations restricted to elementary formulas. So, $\|e[Y]\|$ is false under some interpretation *. This, in view of Lemma 8.2, implies that $\mathbf{Wn}^{(e[Y])^*}\langle\rangle = \bot$ and hence $\mathcal{C}$ is the winner.

Subcase 2.2: The adversary makes a move α. We may assume that such a move is legal, or else $\mathcal{C}$ immediately wins. There are two further subcases to consider here:

Subsubcase 2.2.1: α is a move in the succedent, or a nonreplicative move in one of the components of the antecedent, of Y. With a little thought, it can be seen that then α brings $e[Y]$ down to $e'[Z]$, where Z is a sequent from which Y follows by one of the four Choose rules, and e' is a certain vc-mapping for Z. In this case, $\mathcal{C}$ calls itself on (Z, e').

Subsubcase 2.2.2: α is a replicative move in one of the components of the antecedent of Y. Namely, assume Y (after disabbreviating $\circ\!\!-$) is the game (3) of Subsection 8.5, and the replicative move is made in its $⫰E$ component. This brings $e[Y]$ down to $e[(4)]$. The latter, however, is "essentially the same as" $e[Z]$, where Z abbreviates the game (2). So, $\mathcal{C}$ can pretend that $e[Y]$ has been brought down to $e[Z]$, and call itself on (Z, e). The exact meaning of "pretend" here is that, after calling itself on (Z, e), $\mathcal{C}$ modifies its behavior — by "reinterpreting" moves — in the same style as machine $\mathcal{M}$ modified $\mathcal{N}$'s behavior in Subsection 8.5.

This completes our description of the work of $\mathcal{C}$.

Assume a situation corresponding to Subsubcase 2.2.2 occurs only finitely many times. Note that all other cases, except Subcase 2.1, strictly decrease the complexity of Y. So, the play finally stabilizes in a situation corresponding to Subcase 2.1 and, as was seen when discussing that subcase, $\mathcal{C}$ wins.

Now, assume a situation corresponding to Subsubcase 2.2.2 occurs infinitely many times, that is, $\mathcal{C}$'s adversary makes infinitely many replications in the antecedent. And, for a contradiction, assume that

$$\mathcal{C} \textit{ loses the play of } \bigsqcap X^* \textit{ on every interpretation } ^*. \tag{5}$$

Let F be the (constant/closed) game/formula to which the succedent of the original $g[X]$ is eventually brought down. Similarly, let $\mathcal{A}$ be the set of all (closed) formulas to which various copies of various formulas of the antecedent of $g[X]$ are eventually brought down. With a little thought and with Lemma 8.2 in mind, it can be seen that (5) implies the following:

$$\textit{The set } \{\|E\| \mid E \in \mathcal{A}\} \cup \{\|\neg F\|\} \textit{ is unsatisfiable (in the classical sense).} \tag{6}$$

By the compactness theorem for classical logic, (6) implies that, for some *finite* subset $\mathcal{A}'$ of $\mathcal{A}$, we have:

$$\textit{The set } \{\|E\| \mid E \in \mathcal{A}'\} \cup \{\|\neg F\|\} \textit{ is unsatisfiable (in the classical sense).} \tag{7}$$

Consider a step t in the work of $\mathcal{C}$ such that, beginning from t and at every subsequent step, the antecedent of (the then current) $e[Y]$ contains all formulas of $\mathcal{A}'$. It follows easily from (7) that, beginning from t, (the continuously updated) Y remains stable. This means that $\mathcal{C}$ deals only with CASE 1. But, after making a certain finite number of moves as prescribed by CASE 1, Y is brought down to a stable sequent that contains no surface occurrences of $\sqcap, \bigsqcap$ in the succedent and no surface occurrences of $\sqcup, \bigsqcup$ in the antecedent. Every such sequent follows from the empty set of premises by Wait, which is a contradiction because, as we know, the sequent Y at any step of the work of $\mathcal{C}$ remains **CL12**-unprovable. ■

Remark 9.2 While CoL takes no interest in nonalgorithmic "solutions" of problems, it would still be a pity to let one fact go officially unnoticed. Virtually nothing in our proof of Theorem 9.1 relies on the fact that an HPM whose non-existence is proven there follows an algorithmic strategy. So, Theorem 9.1 can be strengthened by saying that, if **CL12** does not prove a sequent X, then X does not even have a nonalgorithmic logical solution. Precisely defining the meaning of a "nonalgorithmic", or rather "not-necessarily-algorithmic" logical solution, is not hard. The most straightforward way to do so would be to simply take our present definition of a logical solution but generalize its underlying model of computation by allowing HPMs to have oracles — in the standard sense — for whatever functions.

As an aside, among the virtues of CoL is that it eliminates the need for many ad hoc inventions such as the just-mentioned oracles. Namely, observe that a problem A is computable by an HPM with an oracle for a function $f(x)$ if and only if the problem $\sqcap x \sqcup y\big(y = f(x)\big) \circ\!\!- A$ is computable in the ordinary sense (i.e., computable by an ordinary HPM without any oracles). So, a CoL-literate person, regardless of his or her aspirations, would never really have to speak in terms of oracles or nonalgorithmic strategies. This explains why '*CoL takes no interest in nonalgorithmic "solutions" of problems*'.

10 Logical consequence

The following theorem is an immediate corollary of Theorems 8.1 and 9.1.

Theorem 10.1 *For any sequent X, the following conditions are equivalent:*

(i) **CL12** $\vdash X$.

(ii) *X has a logical solution.*

(iii) *X has a well-behaved logical solution which runs in polynomial time and space.*

Definition 10.2 Let $E_1, \ldots, E_n$ $(n \geq 0)$ and F be any formulas. We say that F is a **logical consequence** of $E_1, \ldots, E_n$ iff the sequent $E_1, \ldots, E_n \circ\!\!- F$, in the role of X, satisfies any of the (equivalent) conditions (i)-(iii) of Theorem 10.1.

As noted in Section 1, the following rule, which we (also) call **Logical Consequence**, will be the only logical rule of inference in **CL12**-based applied systems:

From $E_1, \ldots, E_n$ conclude F as long as F is a logical consequence of $E_1, \ldots, E_n$.

Remember from the earlier essays on CoL that, philosophically speaking, computational *resources* are symmetric to computational problems: what is a problem for one player to solve is a resource that the other player can use. Namely, having a problem A as a computational resource intuitively means having the ability to successfully solve/win A. For instance, as a resource, $\sqcap x \sqcup y(y = x^2)$ means the ability to tell the square of any number.

According to the following thesis, logical consequence lives up to its name. A justification for it, as well as an outline of its significance, was provided in Section 1:

Thesis 10.3 Assume $E_1, \ldots, E_n, F$ are formulas such that there is a *-independent (whatever interpretation *) intuitive description and justification of a winning strategy for F^*, which relies on the availability and "recyclability" — in the strongest sense possible — of $E_1^*, \ldots, E_n^*$ as computational resources. Then F is a logical consequence of $E_1, \ldots, E_n$.

Example 10.4 Imagine a **CL12**-based applied formal theory, in which we have already proven two facts: $\forall x\big(x^3 = (x \times x) \times x\big)$ (the meaning of "cube" in terms of multiplication) and $\sqcap x \sqcap y \sqcup z(z = x \times y)$ (the computability of multiplication), and now we want to derive $\sqcap x \sqcup y(y = x^3)$ (the computability of "cube"). This is how we can reason to justify $\sqcap x \sqcup y(y = x^3)$:

> *Consider any s (selected by Environment for x in $\sqcap x \sqcup y(y = x^3)$). We need to find s^3. Using the resource $\sqcap x \sqcap y \sqcup z(z = x \times y)$, we first find the value t of $s \times s$, and then the value r of $t \times s$. According to $\forall x(x^3 = (x \times x) \times x)$, such an r is the sought s^3.*

Thesis 10.3 promises that the above intuitive argument will be translatable into a **CL12**-proof of

$$\forall x\big(x^3 = (x \times x) \times x\big),\ \sqcap x \sqcap y \sqcup z(z = x \times y)\ \circ\!\!-\ \sqcap x \sqcup y(y = x^3)$$

(and hence the succedent will be derivable in the theory by Logical Consequence as the formulas of the antecedent are already proven). Such a proof indeed exists — see Example 7.2.

Remember the concept of a k-ary explicit polynomial function τ defined in Section 8. Here we generalize it to the concept of a (k, n)-ary **explicit polynomial functional** by allowing the term τ to contain, on top of variables and $0, ', +, \times$, additional n ($n \geq 0$) unary function letters $f_1, \ldots, f_n$, semantically treated as placeholders for unary arithmetical functions. Replacing $f_1, \ldots, f_n$ by names $g_1, \ldots, g_n$ of some particular unary arithmetical functions turns τ into the corresponding k-ary arithmetical function, which we shall denote by $\tau(g_1, \ldots, g_n)$. For instance, the term of Figure 2 is a $(1, 2)$-ary explicit polynomial functional. Let us denote it by τ. Then, if g means "square" and h means "cube", $\tau(g, h)$ is the unary arithmetical function $(y^2 + y^3)^3$.

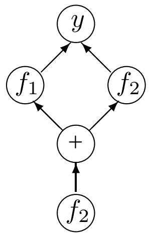

Figure 2: An explicit polynomial functional expressing $f_2\big(f_1(y) + f_2(y)\big)$

The following theorem is a central result of this section. The form in which its "furthermore" clause is stated may seem a little strange or arbitrary (for instance, why does it deal with GHPMs rather than HPMs? And why do those GHPMs take the codes of each other as inputs?). However, the author foresees that it is exactly the present form of the theorem that will be of use when developing **CL12**-based applied theories in the future, namely, in showing that winning strategies can be not only effectively but also efficiently extracted from proofs in such theories. Remembering the technique that we employed in Section 8.7 may provide some insights into the reasons for such an expectation.

Theorem 10.5 *If a formula F is a logical consequence of formulas $E_1, \ldots, E_n$ and * is an interpretation such that each E_i^* ($1 \leq i \leq n$) is computable, then F^* is computable. Furthermore:*

1. *There is an efficient procedure that takes an arbitrary* **CL12***-proof of an arbitrary sequent $E_1, \ldots, E_n \circ\!\!- F$ and constructs a n-ary GHPM $\mathcal{M}$, together with a $(1, n)$-ary explicit polynomial functional τ, such that, for any interpretation *, any n-ary GHPMs $\mathcal{N}_1, \ldots, \mathcal{N}_n$ and any unary arithmetical functions $g_1, \ldots, g_n$, if each $\mathcal{N}_i(\ulcorner \mathcal{N}_1 \urcorner, \ldots, \ulcorner \mathcal{N}_n \urcorner)$ is a g_i time solution of E_i^*, then $\mathcal{M}(\ulcorner \mathcal{N}_1 \urcorner, \ldots, \ulcorner \mathcal{N}_n \urcorner)$ is a $\tau(g_1, \ldots, g_n)$ time solution of F^*.*

2. *The same holds for "space" instead of "time".*

Proof. Consider an arbitrary sequent $E_1, \ldots, E_n \circ\!\!- F$ together with a **CL12**-proof of it. By Theorem 8.1, there a well-behaved logical solution $\mathcal{K}$ of the sequent which runs in time and space ξ for some explicit polynomial function ξ, and these $\mathcal{K}$ and ξ — fix them — can be efficiently found. Consider an arbitrary interpretation * (which, as done before, we shall notationally suppress), arbitrary n-ary GHPMs $\mathcal{N}_1, \ldots, \mathcal{N}_n$ and assume that each $\mathcal{N}_i(\ulcorner \mathcal{N}_1 \urcorner, \ldots, \ulcorner \mathcal{N}_n \urcorner)$ wins E_i in time (resp. space) g_i under that interpretation. Below we describe an n-ary GHPM $\mathcal{M}$ such that $\mathcal{M}(\ulcorner \mathcal{N}_1 \urcorner, \ldots, \ulcorner \mathcal{N}_n \urcorner)$ wins F under the same interpretation. It is important to note that our *construction* of $\mathcal{M}$ does not depend on *, $\mathcal{N}_1, \ldots, \mathcal{N}_n$, $g_1, \ldots, g_n$ and hence on the assumptions that we have just made about them; only our *claim* that $\mathcal{M}(\ulcorner \mathcal{N}_1 \urcorner, \ldots, \ulcorner \mathcal{N}_n \urcorner)$ wins F, and our further claims about its time and space complexities, do.

To describe the above GHPM $\mathcal{M}$ means to describe the work of the HPM $\mathcal{M}(c_1, \ldots, c_n)$ for arbitrary numbers $c_1, \ldots, c_n$. Furthermore, we may assume that these numbers are the codes of (the earlier-mentioned, arbitrary) n-ary GHPMs $\mathcal{N}_1, \ldots, \mathcal{N}_n$, because, if this is not the case (i.e., if some c_i is not the code of some n-ary GHPM $\mathcal{N}_i$), how $\mathcal{M}(c_1, \ldots, c_n)$ works is irrelevant, so we may let $\mathcal{M}$ simply do nothing when its inputs do not have the expected forms. Thus, in what follows, we need to describe the work of the HPM $\mathcal{M}(\ulcorner \mathcal{N}_1 \urcorner, \ldots, \ulcorner \mathcal{N}_n \urcorner)$.

As always, we let our machine $\mathcal{M}(\ulcorner \mathcal{N}_1 \urcorner, \ldots, \ulcorner \mathcal{N}_n \urcorner)$, at the beginning of the play, wait till Environment selects a constant for each free variable of F. Let us fix e as the vc-mapping (see Section 9) whose domain is the set of all free variables of $E_1,\ldots,E_n \circ\!\!- F$ such that e sends every free variable of F to the constant just chosen by Environment for it, and (arbitrarily) sends all other variables to 0.

We describe the work of $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ afterwards at a high level. A more detailed description and analysis would be neither feasible (since it would be prohibitively long and technical) nor necessary.

To understand the idea, let us first consider the simple case where $\mathcal{K}$ never makes any replicative moves in the antecedent of $E_1,\ldots,E_n \circ\!\!- F$. The main part of the work of $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ consists in continuously polling its run tape to see if Environment has made any new moves, combined with simulating, in parallel, a play of $E_1,\ldots,E_n \circ\!\!- F$ by the machine $\mathcal{K}$ and — for each $i\in\{1,\ldots,n\}$ — a play of E_i by the machine $\mathcal{N}_i(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$. In this simulation, $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ "imagines" that, at the beginning of the play, for each free variable x of the corresponding formula or sequent, the adversary of each machine has chosen the constant $e(x)$, where e is the earlier fixed vc-mapping. After the above initial moves by the real and imaginary adversaries, each of the $n{+}2$ games G that we consider here will be brought down to $e[G]$ but, for readability and because e is fixed, we shall usually omit e and write simply G instead of $e[G]$.

Since we here assume that $\mathcal{K}$ never makes any replications in the antecedent of $E_1,\ldots,E_n \circ\!\!- F$, playing this game essentially means simply playing

$$E_1 \wedge \ldots \wedge E_n \rightarrow F. \tag{8}$$

We may assume that, in the real play of F, Environment does not make illegal moves, for then $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ immediately detects this and retires, being the winner. We can also safely assume that the simulated machines do not make illegal moves of the corresponding games, or else our assumptions about their winning those games would be wrong.[18]

If so, what $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ does in the above mixture of the real and simulated plays is that it applies copycat between $n{+}1$ pairs of (sub)games, real or imaginary. Namely, it mimics, in (the real play of) F, $\mathcal{K}$'s moves made in the consequent of (the imaginary play of) (8), and vice versa: uses Environment's moves made in the real play of F as $\mathcal{K}$'s (imaginary) adversary's moves in the consequent of (8). Further, for each $i\in\{1,\ldots,n\}$, $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ uses the moves made by $\mathcal{N}_i(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ in E_i as $\mathcal{K}$'s adversary's moves in the E_i component of (8), and vice versa: uses the moves made by $\mathcal{K}$ in that component as $\mathcal{N}_i(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$'s adversary's moves in E_i.

Therefore, the final positions hit by the $n{+}2$ imaginary and real plays

$$E_1,\ \ldots,\ E_n,\ \ E_1\wedge\ldots\wedge E_n \rightarrow F \ \text{ and } \ F$$

will retain the above forms, i.e., will be

$$E'_1,\ \ldots,\ E'_n,\ \ E'_1\wedge\ldots\wedge E'_n \rightarrow F' \ \text{ and } \ F'$$

for some $E'_1,\ldots,E'_n,F'$. Our assumption that the machines $\mathcal{N}_1(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner),\ldots,\mathcal{N}_n(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ and $\mathcal{K}$ win the games $E_1,\ldots,E_n$ and $F_1\wedge\ldots\wedge F_n\rightarrow F$ implies that each $G\in\{E'_1,\ \ldots,\ E'_n,\ E'_1\wedge\ldots\wedge E'_n\rightarrow F'\}$ is $\top$-won, in the sense that $\mathbf{Wn}^G\langle\rangle=\top$. It is then obvious that so should be F'. Thus, the (real) play of F brings it down to the $\top$-won F', meaning that $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ wins F. Note that $\mathcal{M}$ can be constructed efficiently, as promised in the theorem.

The next thing to clarify is why a bound for the running time (resp. space) of $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ can be expressed as $\tau(g_1,\ldots,g_n)$, where τ is an n-ary explicit polynomial functional. Let $\phi(x)$ be an abbreviation of $g_1(x)+\ldots+g_n(x)+\xi(x)$. Thus, each of the $n{+}1$ simulated machines that we consider runs in — the very generously selected for the sake of simplicity — time (resp. space) ϕ. Note that, because ϕ contains ξ, we have $\phi(x)\geq x$. Next, let us fix $\mathfrak{b}$ as twice the sum of the maximum lengths of legal runs of $E_1,\ldots,E_n$ and F. We may assume that, in any case, $\mathfrak{b}\geq n{+}1$.

The simulation and copycat performed by $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ do impose some time and space overhead. But the latter is only polynomial and, in our subsequent analysis, can be safely ignored. That is, for the sake of simplicity, we are going to pretend that the space that $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ consumes does not exceed the sum of the spaces consumed by the simulated machines, that $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ copies moves in its copycat routine instantaneously, and that the times that $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ ever spends "thinking" about what move to make are the times during which it is waiting for simulated machines to make one or

[18]Since we need to construct $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ no matter whether those assumptions are true or not, we can let $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ simply retire as soon as it detects some illegal behavior.

several moves. Furthermore, we will pretend that simulation happens in a truly parallel fashion, in the sense that $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ spends a single clock cycle on tracing a single computation step of all machines simultaneously. Also, a move α made in F, when "copied" in the consequent of (8), will become $1.\alpha$; however, we shall ignore this minor difference and pretend that the size of α is the same as that of $1.\alpha$. Similarly for moves made in $E_1,\ldots,E_n$ and "copied" in the antecedent of (8).

We start with space complexity. Consider an arbitrary play (computation branch) of $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$, and an arbitrary clock cycle c. Let ℓ be the background of c. In the simulations of $\mathcal{K}$ and $\mathcal{N}_1(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$, $\ldots$, $\mathcal{N}_n(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$, every move made by the imaginary adversary of one of these machines is a copy of either a move made by Environment in the real play, or a move made by one of the machines $\mathcal{K}$, $\mathcal{N}_1(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$, $\ldots$, $\mathcal{N}_n(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ during simulation. Let $\beta_1,\ldots,\beta_m$ be the moves by simulated machines that $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ detects by time c, arranged according to the times of their detections.[19] Let $\mathcal{H}_1,\ldots,\mathcal{H}_m\in\{\mathcal{K},\mathcal{N}_1(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner),\ldots,\mathcal{N}_n(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)\}$ be the machines that made these moves, respectively. The size of β_1 cannot exceed $\phi(\ell)$. That is because, by the time when $\mathcal{H}_1$ made the move β_1, all (if any) moves by $\mathcal{H}_1$'s imaginary adversary were copies of moves made by Environment in the real play rather than moves made by some other simulated machines, and hence the background of β_1 in the simulated play of $\mathcal{H}_1$ did not exceed ℓ. And this means that the ϕ space machine $\mathcal{H}_1$ would not have enough space to construct β_1 on its work tape before making this move if the size of the latter was greater than $\phi(\ell)$. For similar reasons, with $\phi(\ell)$ now acting in the role of ℓ, the size of β_2 cannot exceed $\phi(\phi(\ell))$. Similarly, the size of β_3 cannot exceed $\phi(\phi(\phi(\ell)))$, etc. Also notice that at most $\mathfrak{b}$ moves can be made altogether in the mixture of the real and the imaginary plays, so that $m\leq\mathfrak{b}$. Thus, the size of no move made in this mixture by time c exceeds $\phi^{\mathfrak{b}}(\ell)$ ($\phi^{\mathfrak{b}}$ means the $\mathfrak{b}$-fold composition of ϕ with itself). Therefore, as all simulated machines run in space ϕ, the space consumed by each of the $n+1$ simulations by time c does not exceed $\phi^{\mathfrak{b}+1}(\ell)$. Hence the total space that $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ has used by time c does not exceed $(n+1)\times\phi^{\mathfrak{b}+1}(\ell)$. Let us be generous and, remembering that $\mathfrak{b}\geq n+1$, write the latter as $\mathfrak{b}\times\phi^{\mathfrak{b}+1}(\ell)$ instead, to eliminate explicit dependence on n (this is helpful for our later purposes). Of course, $\mathfrak{b}\times\phi^{\mathfrak{b}+1}(\ell)$, even after accounting for various overheads that we have suppressed in our simplified bookkeeping, can be written as $\tau(g_1,\ldots,g_n)$, where τ is an n-ary explicit polynomial functional. This is exactly the sought bound for the space complexity of $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$. Obviously τ can be constructed efficiently.

Now we look at time complexity. Consider an arbitrary play of $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$, and an arbitrary clock cycle c on which $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ makes a move α. Let ℓ, $\beta_1,\ldots,\beta_m$, $\mathcal{H}_1,\ldots,\mathcal{H}_m$ be as in the previous case. For reasons similar to those employed there, we find that, for each $i\in\{1,\ldots,m\}$, the size of β_i does not exceed $\Re$, where $\Re=\phi^{\mathfrak{b}}(\ell)$. Let $t_1,\ldots,t_k$ be the times at which the above moves $\beta_1,\ldots,\beta_m$ were detected by $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ (that is, made by the corresponding machines). Further, let k be the timecost of α, and let $d=c-k$. Let j be the smallest integer among $1,\ldots,m$ such that $t_j\geq d$. Since $\mathcal{H}_j$ runs in time ϕ, it is clear that t_j-d does not exceed $\phi(\Re)$. Nor does $t_{i+1}-t_i$ for any $i\in\{j,\ldots,m\}$. Hence $t_m-d\leq(m-j+1)\times\phi(\Re)$. But notice that β_m is a move made by $\mathcal{K}$ in the consequent of (8), immediately (by our simplifying assumptions) copied by $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ in the real play when it made its move α. In other words, $c=t_m$. And $c-d=k$. So, k — the timecost of α — does not exceed $(m-j+1)\times\phi(\Re)$ and hence $\mathfrak{b}\times\phi(\Re)$. Nor does the size of α. Now $\mathfrak{b}\times\phi(\Re)$, even after accounting for various overheads that we have suppressed in our simplified bookkeeping, can be written as $\tau(g_1,\ldots,g_n)$, where τ is an n-ary explicit polynomial functional, which can be constructed efficiently. This is exactly the sought bound for the time complexity of $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$.

Whatever we have said so far was about the simple case when $\mathcal{K}$ makes no replicative moves in the antecedent of $E_1,\ldots,E_n\circ\!\!-F$. How different is the general case, where $\mathcal{K}$ can make replications? Not very different. The overall work of $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ remains the same, with the only difference that, every time $\mathcal{K}$ replicates one of E_i (more precisely, to whatever a given copy of E_i has evolved by that time), $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ splits the corresponding simulation of $\mathcal{N}_i(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ into two identical copies, with the same past but possibly diverging futures. This increases the number of simulated plays and the corresponding number of to-be-synchronized (by the copycat routine) pairs of games by one, but otherwise $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ continues working as in the earlier described scenario. $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ is guaranteed to win for the same reasons as before. Furthermore, the time and space complexity analysis

[19] According to our simplified view, $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ detects such a move at the same time (clock cycle) as the time at which the move is made in the simulated play by the corresponding machine. So, in case two or more of the moves $\beta_1,\ldots,\beta_m$ are made simultaneously by the corresponding machines, there can be more than one arrangement of these moves "according to their detection times"; which one is chosen, however, is irrelevant.

that we provided earlier still remains valid. The point is that, as we remember, $\mathcal{K}$ is well-behaved, so that, even if it makes replicative moves, it does so only a certain bounded number of times. Thus, the parameter $\mathfrak{b}$ on which we relied earlier still remains constant (the new $\mathfrak{b}$ is only by a constant factor greater than the old one), and thus so do the the overall number of simulations performed by $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ and the depth of compositions of ϕ within the τ term. ■

The following fact is an immediate corollary of Theorem 10.5:

Corollary 10.6
1. Whenever a formula F is a logical consequence of formulas $E_1,\ldots,E_n$ and the latter have polynomial time solutions under a given interpretation *, so does the former. Such a solution, together with an explicit polynomial bound for its time complexity, can be efficiently constructed from a **CL12***-proof of $E_1,\ldots,E_n \circ\!\!- F$, solutions for $E_1^*,\ldots,E_n^*$, and explicit polynomial bounds for their time complexities.*
2. The same holds for "space" instead of "time".

But the import of Theorem 10.5 extends far beyond the above corollary. The theorem implies that, for any class Ω of functions that contains all polynomial functions and is closed under composition, the rule of Logical Consequence preserves Ω-time and Ω-space computabilities. This means that **CL12** is an adequate logical basis for a wide class of complexity-oriented or complexity-sensitive applied systems. Among those, other than systems for polynomial time and polynomial space computabilities, are many other naturally emerging theories worth studying, such as those for elementary recursive (in the sense of Kalmar) computability,[20] primitive recursive computability, (**PA**-) provably recursive computability, general recursive computability, etc. **CL12** — more precisely, the associated rule of Logical Consequence — is adequate because, on one hand, by Theorem 10.5, it is sound for all such systems, and, on the other hand, by Theorem 9.1 and/or Thesis 10.3 (feel free to also throw Remark 9.2 into the mix), it is as strong as a logical rule of inference could possibly be.

11 Some admissible rules of CL12

We say that a given rule is **admissible** in **CL12** iff, for every instance of the rule, whenever all premises are provable, so is the conclusion.

Before closing this paper, we want to identify a few admissible rules of **CL12** for possible future use and reference. Among such rules are:

Exchange	**Weakening**	**Cut**
$\dfrac{\vec{E},H,G,\vec{K}\circ\!\!- F}{\vec{E},G,H,\vec{K}\circ\!\!- F}$	$\dfrac{\vec{E}\circ\!\!- F}{\vec{E},\vec{K}\circ\!\!- F}$	$\dfrac{\vec{E}\circ\!\!- F \qquad \vec{K},F\circ\!\!- G}{\vec{E},\vec{K}\circ\!\!- G}$

Fact 11.1 *Exchange, Weakening and Cut are admissible in* **CL12**.

Proof. Section 6 of [24] proves that these rules preserve a certain concept of validity. Section 8 of the same paper also proves that **CL12** is sound and complete with respect to that concept of validity. So, the rules are admissible in **CL12**. Of course, the admissibility of Exchange and Weakening can as well be seen directly, using a straightforward syntactic argument. ■

Note that, in view of the admissibility of Exchange and Weakening (and the presence of Replicate), an equivalent formulation of **CL12** would be one that sees the antecedent of a sequent as a set rather than sequence or even a multiset of formulas. Such a formulation would only have five rules, with the rule of Replicate being trivial and hence redundant there.

[20]It can be seen that elementary recursive time is equivalent to elementary recursive space, so we omit the specification "time" or "space" here. The same applies to primitive recursive computability, provably recursive computability and general recursive computability.

Among the six rules of **CL12**, the expensive one is Wait as, at times, it may require too many premises. Using the following four rules instead of Wait can very significantly shorten proofs. The notation $F^{\vee}[G]$ (resp. $F^{\wedge}[G]$) employed in our formulation of those rules means the same as our earlier (Section 7) agreed-on $F[G]$, with the additional restriction that the fixed surface occurrence of G is not in the scope of $\wedge$ (resp. $\vee$). Also, y is a variable not occurring in the conclusion.

$\sqcap$-Introduction

$$\frac{\vec{E} \circ\!\!- F^{\vee}[G_0] \qquad\qquad \vec{E} \circ\!\!- F^{\vee}[G_1]}{\vec{E} \circ\!\!- F^{\vee}[G_0 \sqcap G_1]}$$

$\sqcup$-Introduction

$$\frac{\vec{E}, F^{\wedge}[G_0], \vec{K} \circ\!\!- H \qquad\qquad \vec{E}, F^{\wedge}[G_1], \vec{K} \circ\!\!- H}{\vec{E}, F^{\wedge}[G_0 \sqcup G_1], \vec{K} \circ\!\!- H}$$

$\bigsqcap$-Introduction

$$\frac{\vec{E} \circ\!\!- F^{\vee}[G(y)]}{\vec{E} \circ\!\!- F^{\vee}[\bigsqcap x G(x)]}$$

$\bigsqcup$-Introduction

$$\frac{\vec{E}, F^{\wedge}[G(y)], \vec{K} \circ\!\!- H}{\vec{E}, F^{\wedge}[\bigsqcup x G(x)], \vec{K} \circ\!\!- H}$$

Fact 11.2 *$\sqcap$-Introduction, $\sqcup$-Introduction, $\bigsqcap$-Introduction and $\bigsqcup$-Introduction are admissible in* **CL12**.

Proof. Easy induction, details of which we omit. ■

References

[1] S. Abramsky and R. Jagadeesan. *Games and full completeness for multiplicative linear logic.* **Journal of Symbolic Logic** 59 (1994), pp. 543-574.

[2] L. Babai and M. Shlomo. *Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity classes.* **Journal of Computer System Sciences** 36 (1988), pp. 254-276.

[3] A. Blass. *Degrees of indeterminacy of games.* **Fundamenta Mathematicae** 77 (1972), pp. 151-166.

[4] A. Blass. *A game semantics for linear logic.* **Annals of Pure and Applied Logic** 56 (1992), pp. 183-220.

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

[6] S. Buss. *The polynomial hierarchy and intuitionistic bounded arithmetic.* **Lecture Notes in Computer Science** 223 (1986), pp. 77-103.

[7] A.Chandra, D. Kozen and L. Stockmeyer. *Alternation.* **Journal of the ACM** 28 (1981), pp. 114–133.

[8] S. Goldwasser, S. Micali and C. Rackoff. *The knowledge complexity of interactive proof systems.* **SIAM Journal on Computing** 18 (1989), pp. 186-208.

[9] G. Japaridze. *The logic of tasks.* **Annals of Pure and Applied Logic** 117 (2002), pp. 263-295.

[10] G. Japaridze. *Introduction to computability logic.* **Annals of Pure and Applied Logic** 123 (2003), pp. 1-99.

[11] G. Japaridze. *Propositional computability logic I.* **ACM Transactions on Computational Logic** 7 (2006), pp. 302-330.

[12] G. Japaridze. *Propositional computability logic II.* **ACM Transactions on Computational Logic** 7 (2006), pp. 331-362.

[13] G. Japaridze. *Introduction to cirquent calculus and abstract resource semantics.* **Journal of Logic and Computation** 16 (2006), pp. 489-532.

[14] G. Japaridze. *Computability logic: a formal theory of interaction.* In: **Interactive Computation: The New Paradigm**. D. Goldin, S. Smolka and P. Wegner, eds. Springer 2006, pp. 183-223.

[15] G. Japaridze. *From truth to computability I.* **Theoretical Computer Science** 357 (2006), pp. 100-135.

[16] G. Japaridze. *From truth to computability II.* **Theoretical Computer Science** 379 (2007), pp. 20-52.

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

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

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

[20] G. Japaridze. *Cirquent calculus deepened.* **Journal of Logic and Computation** 18 (2008), pp. 983-1028.

[21] G. Japaridze. *Sequential operators in computability logic.* **Information and Computation** 206 (2008), pp. 1443-1475.

[22] G. Japaridze. *Many concepts and two logics of algorithmic reduction.* **Studia Logica** 91 (2009), pp. 1-24.

[23] G. Japaridze. *In the beginning was game semantics.* In: **Games: Unifying Logic, Language, and Philosophy**. O. Majer, A.-V. Pietarinen and T. Tulenheimo, eds. Springer 2009, pp. 249-350.

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

[25] G. Japaridze. *Toggling operators in computability logic.* **Theoretical Computer Science** 412 (2011), pp. 971-1004.

[26] I. Mezhirov and N. Vereshchagin. *On abstract resource semantics and computability logic.* **Journal of Computer and System Sciences** 76 (2010), pp. 356-372.

[27] H. Schwichtenberg. *An arithmetic for polynomial-time computation.* **Theoretical Computer Science** 357 (2006), pp. 202-214.

[28] W. Xu and S. Liu. *Knowledge representation and reasoning based on computability logic.* **Journal of Jilin University** 47 (2009), pp. 1230-1236.