Acta Cybernetica **18** (2007) 77–113.

Intuitionistic computability logic*

Giorgi Japaridze†

Abstract

Computability logic (CL) is a systematic formal theory of computational tasks and resources, which, in a sense, can be seen as a semantics-based alternative to (the syntactically introduced) linear logic. With its expressive and flexible language, where formulas represent computational problems and "truth" is understood as algorithmic solvability, CL potentially offers a comprehensive logical basis for constructive applied theories and computing systems inherently requiring constructive and computationally meaningful underlying logics. Among the best known constructivistic logics is Heyting's intuitionistic calculus **INT**, whose language can be seen as a special fragment of that of CL. The constructivistic philosophy of **INT**, however, just like the resource philosophy of linear logic, has never really found an intuitively convincing and mathematically strict semantical justification. CL has good claims to provide such a justification and hence a materialization of Kolmogorov's known thesis "**INT** = logic of problems". The present paper contains a soundness proof for **INT** with respect to the CL semantics.

Keywords: computability logic, interactive computation, game semantics, linear logic, intuitionistic logic

1 Introduction

Computability logic (CL), introduced recently in [7], is a formal theory of computability in the same sense as classical logic is a formal theory of truth. It understands formulas as (interactive) computational problems, and their "truth" as algorithmic solvability. Computational problems, in turn, are defined as games played by a machine against the environment, with algorithmic solvability meaning existence of a machine that always wins the game.

Intuitionistic computability logic is not a modification or version of CL. The latter takes pride in its universal applicability, stability and "immunity to possible future revisions and tampering" ([7], p. 12). Rather, what we refer to as *intuitionistic computability logic* is just a — relatively modest — fragment of CL, obtained

*This material is based upon work supported by the National Science Foundation under Grant No. 0208816

†Computing Sciences Department, Villanova University, 800 Lancaster Avenue, Villanova, PA 19085, USA E-mail: giorgi.japaridze@villanova.edu

by mechanically restricting its formalism to a special sublanguage. It was conjectured in [7] that the (set of the valid formulas of the) resulting fragment of CL is described by Heyting's *intuitionistic calculus* **INT**. The present paper is devoted to a verification of the soundness part of that conjecture.

Bringing **INT** and CL together could signify a step forward not only in logic but also in theoretical computer science. **INT** has been attracting the attention of computer scientists since long ago. And not only due to the beautiful phenomenon within the 'formulas-as-types' approach known as the Curry-Howard isomorphism. **INT** appears to be an appealing alternative to classical logic within the more traditional approaches as well. This is due to the general constructive features of its deductive machinery, and Kolmogorov's [14] well-known yet so far rather abstract thesis according to which intuitionistic logic is (or should be) a logic of problems. The latter inspired many attempts to find a "problem semantics" for the language of intuitionistic logic [5, 13, 16], none of which, however, has fully succeeded in justifying **INT** as a logic of problems. Finding a semantical justification for **INT** was also among the main motivations for Lorenzen [15], who pioneered game-semantical approaches in logic. After a couple of decades of trial and error, the goal of obtaining soundness and completeness of **INT** with respect to Lorenzen's game semantics was achieved [3]. The value of such an achievement is, however, dubious, as it came as a result of carefully tuning the semantics and adjusting it to the goal at the cost of sacrificing some natural intuitions that a game semantics could potentially offer.[1] After all, some sort of a specially designed technical semantics can be found for virtually every formal system, but the whole question is how natural and usable such a semantics is in its own right. In contrast, the CL semantics was elaborated without any target deductive construction in mind, following the motto "*Axiomatizations should serve meaningful semantics rather than vice versa*". Only retroactively was it observed that the semantics of CL yields logics similar to or identical with some known axiomatically introduced constructivistic logics such as linear logic or **INT**. Discussions given in [7, 8, 10, 11] demonstrate how naturally the semantics of CL emerges and how much utility it offers, with potential application areas ranging from the pure theory of (interactive) computation to knowledgebase systems, systems for planning and action, and constructive applied theories. As this semantics has well-justified claims to be a semantics of computational problems, the results of the present article speak strongly in favor of Kolmogorov's thesis, with a promise of a full materialization of the thesis in case a completeness proof of **INT** is also found.

The main utility of the present result is in the possibility to base applied theories or knowledgebase systems on **INT**. Nonlogical axioms — or the knowledge base — of such a system would be any collection of (formulas expressing) problems whose

[1] Using Blass's [2] words, 'Supplementary rules governing repeated attacks and defenses were devised by Lorenzen so that the formulas for which P [proponent] has a winning strategy are exactly the intuitionistically provable ones'. Quoting [6], 'Lorenzen's approach describes logical validity exclusively in terms of rules without appealing to any kind of truth values for atoms, and this makes the semantics somewhat vicious ... as it looks like just a "pure" syntax rather than a semantics'.

algorithmic solutions are known. Then, our soundness theorem for **INT** — which comes in a strong form called uniform-constructive soundness — guarantees that every theorem T of the theory also has an algorithmic solution and, furthermore, such a solution can be effectively constructed from a proof of T. This makes **INT** a problem-solving tool: finding a solution for a given problem reduces to finding a proof of that problem in the theory.

It is not an ambition of the present paper to motivationally (re)introduce and (re)justify computability logic and its intuitionistic fragment in particular. This job has been done in [7] and once again — in a more compact way — in [10]. An assumption is that the reader is familiar with at least the motivational/philosophical parts of either paper and this is why (s)he decided to read the present article. While helpful in fully understanding the import of the present results, from the purely technical point of view such a familiarity, however, is not necessary, as this paper provides all necessary definitions. Even if so, [7] and/or [10] could still help a less advanced reader in getting a better hold of the basic technical concepts. Those papers are written in a semitutorial style, containing ample examples, explanations and illustrations, with [10] even including exercises.

2 A brief informal overview of some basic concepts

As noted, formulas of CL represent interactive computational problems. Such problems are understood as games between two players: $\top$, called **machine**, and $\bot$, called **environment**. $\top$ is a mechanical device with a fully determined, algorithmic behavior. On the other hand, there are no restrictions on the behavior of $\bot$. A problem/game is considered (algorithmically) solvable/winnable iff there is a machine that wins the game no matter how the environment acts.

Logical operators are understood as operations on games/problems. One of the important groups of such operations, called **choice operations**, consists of $\sqcap, \sqcup, \bigsqcap, \bigsqcup$, in our present approach corresponding to the intuitionistic operators of conjunction, disjunction, universal quantifier and existential quantifier, respectively. $A_1 \sqcap \ldots \sqcap A_n$ is a game where the first legal move ("choice"), which should be one of the elements of $\{1, \ldots, n\}$, is by the environment. After such a move/choice i is made, the play continues and the winner is determined according to the rules of A_i; if a choice is never made, $\bot$ loses. $A_1 \sqcup \ldots \sqcup A_n$ is defined in a symmetric way with the roles of $\bot$ and $\top$ interchanged: here it is $\top$ who makes an initial choice and who loses if such a choice is not made. With the universe of discourse being $\{1, 2, 3, \ldots\}$, the meanings of the "big brothers" $\bigsqcap$ and $\bigsqcup$ of $\sqcap$ and $\sqcup$ can now be explained by $\bigsqcap x A(x) = A(1) \sqcap A(2) \sqcap A(3) \sqcap \ldots$ and $\bigsqcup x A(x) = A(1) \sqcup A(2) \sqcup A(3) \sqcup \ldots$.

The remaining two operators of intuitionistic logic are the binary $\circ\!\!-$ ("intuitionistic implication") and the 0-ary \$ ("intuitionistic absurd"), with the intuitionistic negation of F simply understood as an abbreviation for $F \circ\!\!- \$$. The intuitive meanings of $\circ\!\!-$ and \$ are "reduction" (in the weakest possible sense) and "a problem of universal strength", respectively. In what precise sense is \$ a universal-strength problem will be seen in Section 6. As for $\circ\!\!-$, its meaning can

be better explained in terms of some other, more basic, operations of CL that have no official intuitionistic counterparts.

One group of such operations comprises **negation** $\neg$ and the so called **parallel operations** $\wedge, \vee, \rightarrow$. Applying $\neg$ to a game A interchanges the roles of the two players: $\top$'s moves and wins become $\bot$'s moves and wins, and vice versa. Say, if *Chess* is the game of chess from the point of view of the white player, then $\neg$*Chess* is the same game as seen by the black player. Playing $A_1 \wedge \ldots \wedge A_n$ (resp. $A_1 \vee \ldots \vee A_n$) means playing the n games in parallel where, in order to win, $\top$ needs to win in all (resp. at least one) of the components A_i. Back to our chess example, the two-board game *Chess* $\vee \neg$*Chess* can be easily won by just mimicking in *Chess* the moves made by the adversary in $\neg$*Chess* and vice versa. On the other hand, winning *Chess*$\sqcup\neg$*Chess* is not easy at all: here $\top$ needs to choose between *Chess* and $\neg$*Chess* (i.e. between playing white or black), and then win the chosen one-board game. Technically, a move α in the kth $\wedge$-conjunct or $\vee$-disjunct is made by prefixing α with 'k.'. For example, in (the initial position of) $(A \sqcup B) \vee (C \sqcap D)$, the move '2.1' is legal for $\bot$, meaning choosing the first $\sqcap$-conjunct in the second $\vee$-disjunct of the game. If such a move is made, the game will continue as $(A \sqcup B) \vee C$. One of the distinguishing features of CL games from the more traditional concepts of games ([1, 2, 3, 6, 15]) is the absence of *procedural rules* — rules strictly regulating which of the players can or should move in any given situation. E.g., in the above game $(A \sqcup B) \vee (C \sqcap D)$, $\top$ also has legal moves — the moves '1.1' and '1.2'. In such cases CL allows either player to move, depending on who wants or can act faster.[2] As argued in [7] (Section 3), only this "free" approach makes it possible to adequately capture certain natural intuitions such as truly parallel/concurrent computations.

The operation $\rightarrow$ is defined by $A \rightarrow B = (\neg A) \vee B$. Intuitively, this is the problem of *reducing* B to A: solving $A \rightarrow B$ means solving B having A as an external *computational resource*. Resources are symmetric to problems: what is a problem to solve for one player is a resource that the other player can use, and vice versa. Since A is negated in $(\neg A) \vee B$ and negation means switching the roles, A appears as a resource rather than problem for $\top$ in $A \rightarrow B$. To get a feel of $\rightarrow$ as a problem reduction operation, the following — already "classical" in CL — example may help. Let, for any m, n, *Accepts*(m, n) mean the game where none of the players has legal moves, and which is automatically won by $\top$ if Turing machine m accepts input n, and otherwise automatically lost. This sort of zero-length games are called **elementary** in CL, which understands every classical proposition/predicate as an elementary game and vice versa, with "true"="won by $\top$" and "false"="lost by $\top$". Note that then $\sqcap x \sqcap y\big(\mathit{Accepts}(x, y) \sqcup \neg \mathit{Accepts}(x, y)\big)$ expresses the acceptance problem as a decision problem: in order to win, the machine should be able to tell whether x accepts y or not (i.e., choose the true disjunct) for any particular values for x and y selected by the environment. This problem is undecidable, which obviously means that there is no machine that (always) wins

[2] This is true for the case when the underlying model of computation is HPM (see Section 5), but seemingly not so when it is EPM — the model employed in the present paper. It should be remembered, however, that EPM is viewed as a secondary model in CL, admitted only due to the fact that it has been proven ([7]) to be equivalent to the basic HPM model.

the game $\bigsqcap x \bigsqcap y\big(\mathit{Accepts}(x,y) \sqcup \neg \mathit{Accepts}(x,y)\big)$. However, the acceptance problem is known to be algorithmically reducible to the halting problem. The latter can be expressed by $\bigsqcap x \bigsqcap y\big(\mathit{Halts}(x,y) \sqcup \neg \mathit{Halts}(x,y)\big)$, with the obvious meaning of the elementary game/predicate $\mathit{Halts}(x,y)$. This reducibility translates into our terms as existence of a machine that wins

$$\bigsqcap x \bigsqcap y\big(\mathit{Halts}(x,y) \sqcup \neg \mathit{Halts}(x,y)\big) \rightarrow \bigsqcap x \bigsqcap y\big(\mathit{Accepts}(x,y) \sqcup \neg \mathit{Accepts}(x,y)\big). \quad (1)$$

Such a machine indeed exists. A successful strategy for it is as follows. At the beginning, $\top$ waits till $\bot$ specifies some values m and n for x and y in the consequent, i.e. makes the moves '$2.m$' and '$2.n$'. Such moves, bringing the consequent down to $\mathit{Accepts}(m,n) \sqcup \neg \mathit{Accepts}(m,n)$, can be seen as asking the question "does machine m accept input n?". To this question $\top$ replies by the counterquestion "does m halt on n?", i.e. makes the moves '$1.m$ and '$1.n$', bringing the antecedent down to $\mathit{Halts}(m,n) \sqcup \neg \mathit{Halts}(m,n)$. The environment has to correctly answer this counterquestion, or else it loses. If it answers "no" (i.e. makes the move '1.2' and thus further brings the antecedent down to $\neg \mathit{Halts}(m,n)$), $\top$ also answers "no" to the original question in the consequent (i.e. makes the move '2.2'), with the overall game having evolved to the true and hence $\top$-won proposition/elementary game $\neg \mathit{Halts}(m,n) \rightarrow \neg \mathit{Accepts}(m,n)$. Otherwise, if the environment's answer is "yes" (move '1.1'), $\top$ simulates Turing machine m on input n until it halts, and then makes the move '2.1' or '2.2' depending whether the simulation accepted or rejected.

Various sorts of reduction have been defined and studied in an ad hoc manner in the literature. A strong case can be made in favor of the thesis that the reduction captured by our $\rightarrow$ is the most basic one, with all other reasonable concepts of reduction being definable in terms of $\rightarrow$. Most natural of those concepts is the one captured by the earlier-mentioned operation of "intuitionistic implication" $\circ\!\!-$, with $A \circ\!\!- B$ defined in terms of $\rightarrow$ and (yet another natural operation) $\mathbin{⫰}$ by $A \circ\!\!- B = (\mathbin{⫰} A) \rightarrow B$. What makes $\circ\!\!-$ so natural is that it captures our intuition of reducing one problem to another in the weakest possible sense. The well-established concept of Turing reduction has the same claim. But the latter is only defined for non-interactive, two-step (question/answer, or input/output) problems, such as the above halting or acceptance problems. When restricted to this sort of problems, as one might expect, $\circ\!\!-$ indeed turns out to be equivalent to Turing reduction. The former, however, is more general than the latter as it is applicable to all problems regardless their forms and degrees of interactivity. Turing reducibility of a problem B to a problem A is defined as the possibility to algorithmically solve B having an oracle for A. Back to (1), the role of $\bot$ in the antecedent is in fact that of an oracle for the halting problem. Notice, however, that the usage of the oracle is limited there as it only can be employed once: after querying regarding whether m halts of n, the machine would not be able to repeat the same query with different parameters m' and n', for that would require two "copies" of $\bigsqcap x \bigsqcap y\big(\mathit{Halts}(x,y) \sqcup \neg \mathit{Halts}(x,y)\big)$ rather than one. On the other hand, Turing reduction to A and, similarly, our $A \circ\!\!- \ldots$, allow unlimited and recurring usage of A, which the resource-conscious CL understands as $\rightarrow$-reduction not to A

but to the stronger problem expressed by $⍚A$, called the **branching recurrence** of A.[3] Two more recurrence operations have been introduced within the framework of CL ([10]): *parallel recurrence* $⅄$ and *sequential recurrence* $⅂$. Common to all of these operations is that, when applied to a resource A, they turn it into a resource that allows to reuse A an unbounded number of times. The difference is in how "reusage" is exactly understood. Imagine a computer that has a program successfully playing *Chess*. The resource that such a computer provides is obviously something stronger than just *Chess*, for it allows to play *Chess* as many times as the user wishes, while *Chess*, as such, only assumes one play. The simplest operating system would allow to start a session of *Chess*, then — after finishing or abandoning and destroying it — start a new play again, and so on. The game that such a system plays — i.e. the resource that it supports/provides — is $⅂$*Chess*, which assumes an unbounded number of plays of *Chess* in a sequential fashion. However, a more advanced operating system would not require to destroy the old session(s) before starting a new one; rather, it would allow to run as many parallel sessions as the user needs. This is what is captured by $⅄$*Chess*, meaning nothing but the infinite conjunction $\mathit{Chess} \wedge \mathit{Chess} \wedge \ldots$. As a resource, $⅄$*Chess* is obviously stronger than $⅂$*Chess* as it gives the user more flexibility. But $⅄$ is still not the strongest form of reusage. A really good operating system would not only allow the user to start new sessions of *Chess* without destroying old ones; it would also make it possible to branch/replicate each particular session, i.e. create any number of "copies" of any already reached position of the multiple parallel plays of *Chess*, thus giving the user the possibility to try different continuations from the same position. After analyzing the formal definition of $⍚$ given in Section 3 — or, better, the explanations provided in Section 13 of [7] — the reader will see that $⍚$*Chess* is exactly what accounts for this sort of a situation. $⅄$*Chess* can then be thought of as a restricted version of $⍚$*Chess* where only the initial position can be replicated. A well-justified claim can be made that $⍚A$ captures our strongest possible intuition of "recycling"/"reusing" A. This automatically translates into another claim, according to which $A \circ\!\!- B$, i.e. $⍚A \rightarrow B$, captures our weakest possible — and hence most natural — intuition of reducing B to A.

As one may expect, the three concepts of recurrence validate different principles. For example, one can show that the left $⊔$- or $⨅$-introduction rules of **INT**, which are sound with $A \circ\!\!- B$ understood as $⍚A \rightarrow B$, would fail if $A \circ\!\!- B$ was understood as $⅄A \rightarrow B$ or $⅂A \rightarrow B$. A naive person familiar with linear logic and seeing philosophy-level connections between our recurrence operations and Girard's [4] *storage* operator !, might ask which of the three recurrence operations "corresponds" to !. In the absence of a clear resource semantics for linear logic, perhaps such a question would not be quite meaningful though. Closest to our present approach is that of [1], where Blass proved soundness for the propositional fragment of **INT** with respect to his semantics, reintroduced 20 years later [2] in the new context of linear logic.

[3] The term "branching recurrence" and the symbols $⍚$ and $\circ\!\!-$ were established in [10]. The earlier paper [7] uses "branching conjunction", ! and $\Rightarrow$ instead. In the present paper, $\Rightarrow$ has a different meaning — that of a separator of the two parts of a sequent.

To appreciate the difference between $\rightarrow$ and $\circ\!\!-$, let us remember the Kolmogorov complexity problem. It can be expressed by $\sqcap u \sqcup z K(z,u)$, where $K(z,u)$ is the predicate "z is the size of the smallest (code of a) Turing machine that returns u on input 1". Just like the acceptance problem, the Kolmogorov complexity problem has no algorithmic solution but is algorithmically reducible to the halting problem. However, such a reduction can be shown to essentially require recurring usage of the resource $\sqcap x \sqcap y\big(\mathit{Halts}(x,y) \sqcup \neg \mathit{Halts}(x,y)\big)$. That is, while the following game is winnable by a machine, it is not so with $\rightarrow$ instead of $\circ\!\!-$:

$$\sqcap x \sqcap y\big(\mathit{Halts}(x,y) \sqcup \neg \mathit{Halts}(x,y)\big) \circ\!\!- \sqcap u \sqcup z K(z,u). \tag{2}$$

Here is $\top$'s strategy for (2) in relaxed terms: $\top$ waits till $\bot$ selects a value m for u in the consequent, thus asking $\top$ the question "what is the Kolmogorov complexity of m?". After this, starting from $i = 1$, $\top$ does the following: it creates a new copy of the (original) antecedent, and makes the two moves in it specifying x and y as i and 1, respectively, thus asking the counterquestion "does machine i halt on input 1?". If $\bot$ responds by choosing $\neg\mathit{Halts}(i,1)$ ("no"), $\top$ increments i by one and repeats the step; otherwise, if $\bot$ responds by $\mathit{Halts}(i,1)$ ("yes"), $\top$ simulates machine i on input 1 until it halts; if it sees that machine i returned m, it makes the move in the consequent specifying z as $|i|$ (here $|i|$ means the size of i, i.e., $|i| = \log_2 i$), thus saying that $|i|$ is the Kolmogorov complexity of m; otherwise, it increments i by one and repeats the step.

3 Constant games

Now we are getting down to formal definitions of the concepts informally explained in the previous section. Our ultimate concept of games will be defined in the next section in terms of the simpler and more basic class of games called constant games. To define this class, we need some technical terms and conventions. Let us agree that by a **move** we mean any finite string over the standard keyboard alphabet. One of the non-numeric and non-punctuation symbols of the alphabet, denoted ♠, is designated as a special-status move, intuitively meaning a move that is always illegal to make. A **labeled move** (**labmove**) is a move prefixed with $\top$ or $\bot$, with its prefix (**label**) indicating which player has made the move. A **run** is a (finite or infinite) sequence of labeled moves, and a **position** is a finite run.

Convention 1. We will be exclusively using the letters $\Gamma, \Theta, \Phi, \Psi, \Upsilon$ for runs, $\wp$ for players, α, β, γ for moves, and λ for labmoves. Runs will be often delimited with "$\langle$" 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 position $\langle\Phi\rangle$ the labmove $\langle\wp\alpha\rangle$ and then the run $\langle\Gamma\rangle$. $\neg\Gamma$ (not to confuse this $\neg$ with the same-shape game operation of negation) will mean the result of simultaneously replacing every label $\top$ in every labmove of Γ by $\bot$ and vice versa. Another important notational convention is that, for a string/move α, Γ^{α} means the result of removing from Γ all labmoves except those of the form $\wp\alpha\beta$, and then deleting the prefix 'α' in the remaining moves, i.e. replacing each such $\wp\alpha\beta$ by $\wp\beta$.

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

Definition 2. *A* **constant game** *is a pair* $A = (\mathbf{Lr}^A, \mathbf{Wn}^A)$*, where:*

1. $\mathbf{Lr}^A$ *is a set of runs not containing (whatever-labeled) move* ♠*, satisfying the condition that a (finite or infinite) run is in* $\mathbf{Lr}^A$ *iff all of its nonempty finite — not necessarily proper — 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* $\mathbf{Wn}^A\langle\Gamma\rangle \neq \wp$*. 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.*

Definition 3. *Let* A*,* B*,* $A_1, A_2, \ldots$ *be constant games, and* $n \in \{2, 3, \ldots\}$*.*

1. $\neg A$ *is defined by:* $\Gamma \in \mathbf{Lr}^{\neg A}$ *iff* $\neg\Gamma \in \mathbf{Lr}^A$*;* $\mathbf{Wn}^{\neg A}\langle\Gamma\rangle = \top$ *iff* $\mathbf{Wn}^A\langle\neg\Gamma\rangle = \bot$*.*
2. $A_1 \sqcap \ldots \sqcap A_n$ *is defined by:* $\Gamma \in \mathbf{Lr}^{A_1\sqcap\ldots\sqcap A_n}$ *iff* $\Gamma = \langle\rangle$ *or* $\Gamma = \langle\bot i, \Theta\rangle$*, where* $i \in \{1, \ldots, n\}$ *and* $\Theta \in \mathbf{Lr}^{A_i}$*;* $\mathbf{Wn}^{A_1\sqcap\ldots\sqcap A_n}\langle\Gamma\rangle = \bot$ *iff* $\Gamma = \langle\bot i, \Theta\rangle$*, where* $i \in \{1, \ldots, n\}$ *and* $\mathbf{Wn}^{A_i}\langle\Theta\rangle = \bot$*.*
3. $A_1 \wedge \ldots \wedge A_n$ *is defined by:* $\Gamma \in \mathbf{Lr}^{A_1\wedge\ldots\wedge A_n}$ *iff every move of* Γ *starts with 'i.' for one of the* $i \in \{1, \ldots, n\}$ *and, for each such* i*,* $\Gamma^{i.} \in \mathbf{Lr}^{A_i}$*; whenever* $\Gamma \in \mathbf{Lr}^{A_1\wedge\ldots\wedge A_n}$*,* $\mathbf{Wn}^{A_1\wedge\ldots\wedge A_n}\langle\Gamma\rangle = \top$ *iff, for each* $i \in \{1, \ldots, n\}$*,* $\mathbf{Wn}^{A_i}\langle\Gamma^{i.}\rangle = \top$*.*
4. $A_1 \sqcup \ldots \sqcup A_n$ *and* $A_1 \vee \ldots \vee A_n$ *are defined exactly as* $A_1 \sqcap \ldots \sqcap A_n$ *and* $A_1 \wedge \ldots \wedge A_n$*, respectively, only with "*$\top$*" and "*$\bot$*" interchanged. And* $A \rightarrow B$ *is defined as* $(\neg A) \vee B$*.*
5. *The infinite* $\sqcap$*-conjunction* $A_1 \sqcap A_2 \sqcap \ldots$ *is defined exactly as* $A_1 \sqcap \ldots \sqcap A_n$*, only with "*$i \in \{1, 2, \ldots\}$*" instead of "*$i \in \{1, \ldots, n\}$*". Similarly for the infinite versions of* $\sqcup$*,* $\wedge$*,* $\vee$*.*
6. *In addition to the earlier-established meanings, the symbols* $\top$ *and* $\bot$ *also denote two special — simplest — games, defined by* $\mathbf{Lr}^\top = \mathbf{Lr}^\bot = \{\langle\rangle\}$*,* $\mathbf{Wn}^\top\langle\rangle = \top$ *and* $\mathbf{Wn}^\bot\langle\rangle = \bot$*.*

An important operation not explicitly mentioned in Section 2 is what is called *prefixation*. This operation 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 $\langle\Phi\rangle A$. 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. We have already used this intuition when explaining the meaning of choice operations in Section 2: we said that after $\bot$ makes an initial move $i \in \{1, \ldots, n\}$, the game

$A_1 \sqcap \ldots \sqcap A_n$ continues as A_i. What this meant was nothing but that $\langle \bot i \rangle (A_1 \sqcap \ldots \sqcap A_n) = A_i$. Similarly, $\langle \top i \rangle (A_1 \sqcup \ldots \sqcup A_n) = A_i$. Here is the definition of prefixation:

Definition 4. *Let A be a constant game and Φ a legal position of A. The game $\langle \Phi \rangle A$ is defined by:* $\mathbf{Lr}^{\langle \Phi \rangle A} = \{\Gamma \mid \langle \Phi, \Gamma \rangle \in \mathbf{Lr}^A\}$; $\mathbf{Wn}^{\langle \Phi \rangle A} \langle \Gamma \rangle = \mathbf{Wn}^A \langle \Phi, \Gamma \rangle$.

The operation $\luebang$ is somewhat more complex and its definition relies on certain additional conventions. We will be referring to (possibly infinite) strings of 0s and 1s as **bit strings**, using the letters w, u as metavariables for them. The expression wu, meaningful only when w is finite, will stand for the concatenation of strings w and u. We write $w \preceq u$ to mean that w is a (not necessarily proper) initial segment of u. The letter ϵ will exclusively stand for the **empty bit string**.

Convention 5. By a **tree** we mean a nonempty set T of bit strings, called **branches** of the tree, such that, for every w, u, we have: (a) if $w \in T$ and $u \preceq w$, then $u \in T$; (b) $w0 \in T$ iff $w1 \in T$; (c) if w is infinite and every finite u with $u \preceq w$ is in T, then $w \in T$. Note that T is finite iff every branch of it is finite. A **complete branch** of T is a branch w such that no bit string u with $w \preceq u \neq w$ is in T. Finite branches are called **nodes**, and complete finite branches called **leaves**.

Definition 6. *We define the notion of a* **prelegal position***, together with the function* Tree *that associates a finite tree* $\mathrm{Tree}\langle \Phi \rangle$ *with each prelegal position Φ, by the following induction:*

1. $\langle\rangle$ *is a prelegal position, and* $\mathrm{Tree}\langle\rangle = \{\epsilon\}$.
2. $\langle \Phi, \lambda \rangle$ *is a prelegal position iff Φ is so and one of the following two conditions is satisfied:*
   a) $\lambda = \bot w$*: for some leaf w of* $\mathrm{Tree}\langle \Phi \rangle$*. We call this sort of a labmove λ* **replicative***. In this case* $\mathrm{Tree}\langle \Phi, \lambda \rangle = \mathrm{Tree}\langle \Phi \rangle \cup \{w0, w1\}$.
   b) λ *is $\bot w.\alpha$ or $\top w.\alpha$ for some node w of* $\mathrm{Tree}\langle \Phi \rangle$ *and move α. We call this sort of a labmove λ* **nonreplicative***. In this case* $\mathrm{Tree}\langle \Phi, \lambda \rangle = \mathrm{Tree}\langle \Phi \rangle$.

The terms "replicative" and "nonreplicative" also extend from labmoves to moves. When a run Γ is infinite, it is considered prelegal iff all of its finite initial segments are so. For such a Γ, the value of $\mathrm{Tree}\langle \Gamma \rangle$ *is the smallest tree such that, for every finite initial segment Φ of Γ,* $\mathrm{Tree}\langle \Phi \rangle \subseteq \mathrm{Tree}\langle \Gamma \rangle$.

Convention 7. Let u be a bit string and Γ any run. Then $\Gamma^{\preceq u}$ will stand for the result of first removing from Γ all labmoves except those that look like $\wp w.\alpha$ for some bit string w with $w \preceq u$, and then deleting this sort of prefixes 'w.' in the remaining labmoves, i.e. replacing each remaining labmove $\wp w.\alpha$ (where w is a bit string) by $\wp\alpha$. Example: If $u = 101000\ldots$ and $\Gamma = \langle \top\epsilon.\alpha_1, \bot{:}, \bot 1.\alpha_2, \top 0.\alpha_3, \bot 1{:}, \top 10.\alpha_4 \rangle$, then $\Gamma^{\preceq u} = \langle \top\alpha_1, \bot\alpha_2, \top\alpha_4 \rangle$.

Definition 8. *Let A be a constant game. The game $\luebang A$ is defined by:*

1. *A position* Φ *is in* $\mathbf{Lr}^{\mathbin{\mathring{\downarrow}}A}$ *iff* Φ *is prelegal and, for every leaf* w *of* $\mathrm{Tree}\langle\Phi\rangle$, $\Phi^{\preceq w} \in \mathbf{Lr}^{A}$.
2. *As long as* $\Gamma \in \mathbf{Lr}^{\mathbin{\mathring{\downarrow}}A}$, $\mathbf{Wn}^{\mathbin{\mathring{\downarrow}}A}\langle\Gamma\rangle = \top$ *iff* $\mathbf{Wn}^{A}\langle\Gamma^{\preceq u}\rangle = \top$ *for every infinite bit string* u.[4]

Next, we officially reiterate the earlier-given definition of $\circ\!\!-$ *by stipulating that* $A \circ\!\!- B =_{def} \mathbin{\mathring{\downarrow}}A \rightarrow B$.

Remark 9. Intuitively, a legal run Γ of $\mathbin{\mathring{\downarrow}}A$ can be thought of as a multiset Z of parallel legal runs of A. Specifically, $Z = \{\Gamma^{\preceq u} \mid u \textit{ is a complete branch of } \textit{Tree}\langle\Gamma\rangle\}$, with complete branches of $\textit{Tree}\langle\Gamma\rangle$ thus acting as names for — or "representing" — elements of Z. In order for $\top$ to win, every run from Z should be a $\top$-won run of A. The runs from Z typically share some common initial segments and, put together, can be seen as forming a tree of labmoves, with $\textit{Tree}\langle\Gamma\rangle$ — that we call the *underlying tree-structure* of Z — in a sense presenting the shape of that tree. The meaning of a replicative move w: — making which is an exclusive privilege of $\bot$ — is creating in (the evolving) Z two copies of position $\Gamma^{\preceq w}$ out of one. And the meaning of a nonreplicative move $w.\alpha$ is making move α in all positions $\Gamma^{\preceq u}$ of (the evolving) Z with $w \preceq u$. This is a brutally brief explanation, of course. The reader may find it very helpful to see Section 13 of [7] for detailed explanations and illustrations of the intuitions associated with our $\mathbin{\mathring{\downarrow}}$-related formal concepts.[5]

4 Not-necessarily-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 $\{\top, \bot\}$. As we know, however, propositions only offer a very limited expressive power, and classical logic needs to consider the more general concept of predicates, with propositions being nothing but special — constant — cases of predicates. The situation in CL is similar. Our concept of (simply) game generalizes that of constant game in the same sense as the classical concept of predicate generalizes that of proposition.

Let us fix two infinite sets of expressions: the set $\{v_1, v_2, \ldots\}$ of **variables** and the set $\{1, 2, \ldots\}$ of **constants**. Without loss of generality here we assume that the above collection of constants is exactly the universe of discourse — i.e. the set over which the variables range — in all cases that we consider. By a **valuation** we mean a function e that sends each variable x to a constant $e(x)$. In these terms, a classical predicate P can be understood as a function that sends each valuation e to a proposition, i.e. constant predicate. Similarly, what we call a game sends valuations to constant games:

[4] For reasons pointed out on page 39 of [7], the phrase "for every infinite bit string u" here can be equivalently replaced by "for every complete branch u of $\textit{Tree}\langle\Gamma\rangle$". Similarly, in clause 1, "every leaf w of $\textit{Tree}\langle\Phi\rangle$" can be replaced by "every infinite bit string w".

[5] A couple of potentially misleading typos have been found in Section 13 of [7]. The current erratum note is maintained at http://www.csc.villanova.edu/~japaridz/CL/erratum.pdf

Definition 10. *A* **game** *is a function A from valuations to constant games. We write $e[A]$ (rather than $A(e)$) to denote the constant game returned by A for valuation e. Such a constant game $e[A]$ is said to be an* **instance** *of A. For readability, we often write $\mathbf{Lr}_e^A$ and $\mathbf{Wn}_e^A$ 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 will be thought of as special, constant cases of games in the sense of Definition 10. 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.

We say that a game A **depends on** a variable x iff there are two valuations e_1, e_2 that agree on all variables except x such that $e_1[A] \neq e_2[A]$. Constant games thus do not depend on any variables.

Just as the Boolean operations straightforwardly extend from propositions to all predicates, our operations $\neg, \wedge, \vee, \rightarrow, \sqcap, \sqcup, \mathbin{\downarrow\!\!\circ}, \circ\!\!-$ 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 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.

To generalize the standard operation of substitution of variables to games, let us agree that by a **term** we mean either a variable or a constant; the domain of each valuation e is extended to all terms by stipulating that, for any constant c, $e(c) = c$.

Definition 11. *Let A be a game, $x_1, \ldots, x_n$ pairwise distinct variables, and $t_1, \ldots, t_n$ any (not necessarily distinct) terms. The result of* **substituting** $x_1, \ldots, x_n$ **by** $t_1, \ldots, t_n$ **in** A*, denoted $A(x_1/t_1, \ldots, x_n/t_n)$, is defined by stipulating that, for every valuation e, $e[A(x_1/t_1, \ldots, x_n/t_n)] = e'[A]$, where e' is the valuation for which we have $e'(x_1) = e(t_1)$, $\ldots$, $e'(x_n) = e(t_n)$ and, for every variable $y \notin \{x_1, \ldots, x_n\}$, $e'(y) = e(y)$.*

Intuitively $A(x_1/t_1, \ldots, x_n/t_n)$ is A with $x_1, \ldots, x_n$ remapped to $t_1, \ldots, t_n$, respectively. Following the standard readability-improving practice established in the literature for predicates, we will often fix a tuple $(x_1, \ldots, x_n)$ of pairwise distinct variables for a game A and write A as $A(x_1, \ldots, x_n)$. It should be noted that when doing so, by no means do we imply that $x_1, \ldots, x_n$ are of all of (or only) the variables on which A depends. Representing A in the form $A(x_1, \ldots, x_n)$ sets a context in which we can write $A(t_1, \ldots, t_n)$ to mean the same as the more clumsy expression $A(x_1/t_1, \ldots, x_n/t_n)$.

In the above terms, we now officially reiterate the earlier-given definitions of the two main quantifier-style operations $\bigsqcap$ and $\bigsqcup$:

$$\bigsqcap x A(x) =_{def} A(1) \sqcap A(2) \sqcap A(3) \sqcap \ldots$$

and

$$\bigsqcup x A(x) =_{def} A(1) \sqcup A(2) \sqcup A(3) \sqcup \ldots.$$

5 Computational problems and their algorithmic solvability

Our games are obviously general enough to model anything that one would call a (two-agent, two-outcome) interactive problem. However, they are a bit too general. There are games where the chances of a player to succeed essentially depend on the relative speed at which its adversary acts. A simple example would be a game where both players have a legal move in the initial position, and which is won by the player who moves first. CL does not want to consider this sort of games meaningful computational problems. Definition 4.2 of [7] imposes a simple condition on games and calls games satisfying that condition **static**. We are not reproducing that definition here as it is not relevant for our purposes. It should be however mentioned that, according to one of the theses on which the philosophy of CL relies, the concept of static games is an adequate formal counterpart of our intuitive concept of "pure", speed-independent interactive computational problems. All meaningful and reasonable examples of games — including all elementary games — are static, and the class of static games is closed under all of the game operations that we have seen (Theorem 14.1 of [7]). Let us agree that from now on the term "**computational problem**", or simply "**problem**", is a synonym of "static game".

Now it is time to explain what *computability* of such problems means. The definitions given in this section are semiformal. The omitted technical details are rather standard or irrelevant and can be easily restored by anyone familiar with Turing machines. If necessary, the corresponding detailed definitions can be found in Part II of [7].

[7] defines two models of interactive computation, called the *hard-play machine* (*HPM*) and the *easy-play machine* (*EPM*). Both are sorts of Turing machines with the capability of making moves, and have three tapes: the ordinary read/write *work tape*, and the read-only *valuation* and *run* tapes. The valuation tape contains a full description of some valuation e (say, by listing the values of e at $v_1, v_2, \ldots$), and its content remains fixed throughout the work of the machine. As for the run tape, it serves as a dynamic input, at any time spelling the current position, i.e. the sequence of the (lab)moves made by the two players so far. So, every time one of the players makes a move, that move — with the corresponding label — is automatically appended to the content of this tape. In the HPM model, there is no restriction on the frequency of environment's moves. In the EPM model, on the other hand, the machine has full control over the speed of its adversary: the environment can (but is not obligated to) make a (one single) move only when the machine explicitly allows it to do so — the event that we call **granting permission**. The only "fairness" requirement that such a machine is expected to satisfy is that it should grant permission every once in a while; how long that "while" lasts, however, is totally up to the machine. The HPM and EPM models seem to be two extremes, yet, according to Theorem 17.2 of [7], they yield the same class of winnable static games. The present paper will only deal with the EPM model, so let us take a little closer look at it.

Let $\mathcal{M}$ be an EPM. A *configuration* of $\mathcal{M}$ is defined in the standard way: this is a full description of the ("current") state of the machine, the locations of its three scanning heads and the contents of its tapes, with the exception that, in order to make finite descriptions of configurations possible, we do not formally include a description of the unchanging (and possibly essentially infinite) content of the valuation tape as a part of configuration, but rather account for it in our definition of computation branch as will be seen below. The *initial configuration* is the configuration where $\mathcal{M}$ is in its start state and the work and run tapes are empty. A configuration C' is said to be an *e-successor* of configuration C in $\mathcal{M}$ if, when valuation e is spelled on the valuation tape, C' can legally follow C in the standard — standard for multitape Turing machines — sense, based on the transition function (which we assume to be deterministic) of the machine and accounting for the possibility of nondeterministic updates — depending on what move $\bot$ makes or whether it makes a move at all — of the content of the run tape when $\mathcal{M}$ grants permission. Technically granting permission happens by entering one of the specially designated states called "permission states". An **e-computation branch** of $\mathcal{M}$ is a sequence of configurations of $\mathcal{M}$ where the first configuration is the initial configuration and every other configuration is an e-successor of the previous one. Thus, the set of all e-computation branches captures all possible scenarios (on valuation e) corresponding to different behaviors by $\bot$. Such a branch is said to be **fair** iff permission is granted infinitely many times in it. Each e-computation branch B of $\mathcal{M}$ incrementally spells — in the obvious sense — a run Γ on the run tape, which we call the **run spelled by** B. Then, for a game A we write $\mathcal{M} \models_e A$ to mean that, whenever Γ is the run spelled by some e-computation branch B of $\mathcal{M}$ and Γ is not $\bot$-illegal, then branch B is fair and $\mathbf{Wn}_e^A\langle\Gamma\rangle = \top$. We write $\mathcal{M} \models A$ and say that $\mathcal{M}$ **computes** (**solves, wins**) A iff $\mathcal{M} \models_e A$ for every valuation e. Finally, we write $\models A$ and say that A is **computable** iff there is an EPM $\mathcal{M}$ with $\mathcal{M} \models A$.

6 The language of INT and the extended language

As mentioned, the language of intuitionistic logic can be seen as a fragment of that of CL. The main building blocks of the language of **INT** are infinitely many **problem letters**, or **letters** for short, for which we use $P, Q, R, S, \ldots$ as metavariables. They are what in classical logic are called 'predicate letters', and what CL calls 'general letters'. With each letter is associated a nonnegative integer called its *arity*. \$ is one of the letters, with arity 0. We refer to it as the **logical letter**, and call all other letters **nonlogical**. The language also contains infinitely many *variables* and *constants* — exactly the ones fixed in Section 4. "*Term*" also has the same meaning as before. An **atom** is $P(t_1, \ldots, t_n)$, where P is an n-ary letter and the t_i are terms. Such an atom is said to be *P-based*. If here each term t_i is a constant, we say that $P(t_1, \ldots, t_n)$ is **grounded**. A P-based atom is n-ary, logical, nonlogical etc. iff P is so. When P is 0-ary, we write P instead of $P()$. **INT-Formulas** are the elements of the smallest class of expressions such that all

atoms are **INT**-formulas and, if $F, F_1, \ldots, F_n$ $(n \geq 2)$ are **INT**-formulas and x is a variable, then the following expressions are also **INT**-formulas: $(F_1) \circ\!\!- (F_2)$, $(F_1) \sqcap \ldots \sqcap (F_n)$, $(F_1) \sqcup \ldots \sqcup (F_n)$, $\bigsqcap x(F)$, $\bigsqcup x(F)$. Officially there is no negation in the language of **INT**. Rather, the intuitionistic negation of F is understood as $F \circ\!\!- \$$. In this paper we also employ a more expressive formalism that we call the **extended language**. The latter has the additional connectives $\top, \bot, \neg, \wedge, \vee, \rightarrow, \mathbin{!}\!\circ$ on top of those of the language of **INT**, extending the above formation rules by adding the clause that $\top$, $\bot$, $\neg F$, $(F_1) \wedge \ldots \wedge (F_n)$, $(F_1) \vee \ldots \vee (F_n)$, $(F_1) \rightarrow (F_2)$ and $\mathbin{!}\!\circ(F)$ are formulas as long as F, $F_1, \ldots, F_n$ are so. $\top$ and $\bot$ count as logical atoms. Henceforth by (simply) "**formula**", unless otherwise specified, we mean a formula of the extended language. Parentheses will often be omitted in formulas when this causes no ambiguity. With $\bigsqcap$ and $\bigsqcup$ being **quantifiers**, the definitions of **free** and **bound** occurrences of variables are standard.

In concordance with a similar notational convention for games on which we agreed in Section 4, sometimes a formula F will be represented as $F(x_1, \ldots, x_n)$, where the x_i are pairwise distinct variables. When doing so, we do not necessarily mean that each such x_i has a free occurrence in F, or that every variable occurring free in F is among $x_1, \ldots, x_n$. In the context set by the above representation, $F(t_1, \ldots, t_n)$ will mean the result of replacing, in F, each free occurrence of each x_i $(1 \leq i \leq n)$ by term t_i. In case each t_i is a variable y_i, it may be not clear whether $F(x_1, \ldots, x_n)$ or $F(y_1, \ldots, y_n)$ was originally meant to represent F in a given context. Our disambiguating convention is that the context is set by the expression that was used earlier. That is, when we first mention $F(x_1, \ldots, x_n)$ and only after that use the expression $F(y_1, \ldots, y_n)$, the latter should be understood as the result of replacing variables in the former rather than vice versa.

Let x be a variable, t a term and $F(x)$ a formula. t is said to be **free for** x **in** $F(x)$ iff none of the free occurrences of x in $F(x)$ is in the scope of $\bigsqcap t$ or $\bigsqcup t$. Of course, when t is a constant, this condition is always satisfied.

An **interpretation** is a function * that sends each n-ary letter P to a static game ${}^*P = P^*(x_1, \ldots, x_n)$, where the x_i are pairwise distinct variables. This function induces a unique mapping that sends each formula F to a game F^* (in which case we say that * **interprets** F as F^* and that F^* is the **interpretation of** F under *) by stipulating that:

1. Where P is an n-ary letter with ${}^*P = P^*(x_1, \ldots, x_n)$ and $t_1, \ldots, t_n$ are terms, $(P(t_1, \ldots, t_n))^* = P^*(t_1, \ldots, t_n)$.
2. * commutes with all operators: $\top^* = \top$, $(F \circ\!\!- G)^* = F^* \circ\!\!- G^*$, $(F_1 \wedge \ldots \wedge F_n)^* = F_1^* \wedge \ldots \wedge F_n^*$, $(\bigsqcap x F)^* = \bigsqcap x(F^*)$, etc.

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 $\big(F(x_1, \ldots, x_n)\big)^*$.

For a formula F, we say that an interpretation * is **admissible for** F, or simply F**-admissible**, iff the following conditions are satisfied:

1. For every n-ary letter P occurring in F, where ${}^*P = P^*(x_1, \ldots, x_n)$, the game $P^*(x_1, \ldots, x_n)$ does not depend on any variables that are not among $x_1, \ldots, x_n$ but occur (whether free or bound) in F.

2. $\$^* = B \sqcap F_1^* \sqcap F_2^* \sqcap \ldots$, where B is an arbitrary problem and $F_1, F_2, \ldots$ is the lexicographic list of all grounded nonlogical atoms of the language.

Speaking philosophically, an admissible interpretation * interprets \$ as a "strongest problem": the interpretation of every grounded atom and hence — according to Lemma 27 — of every formula is reducible to $\$^*$, and reducible in a certain uniform, interpretation-independent way. Viewing $\$^*$ as a resource, it can be seen as a universal resource that allows its owner to solve any problem. Our choice of the dollar notation here is no accident: money is an illustrative example of an all-powerful resource in the world where everything can be bought. "Strongest", "universal" or "all-powerful" do not necessarily mean "impossible". So, the intuitionistic negation $F \circ\!\!- \$$ of F here does not have the traditional "F^* is absurd" meaning. Rather, it means that F^* (too) is of universal strength. Turing completeness, NP-completeness and similar concepts are good examples of "being of universal strength". $\$^*$ is what [7] calls a *standard universal problem* of the universe $\langle F_1^*, F_2^*, \ldots\rangle$. Briefly, a *universal problem* of a universe (sequence) $\langle A_1, A_2, \ldots\rangle$ of problems is a problem U such that $\models U \rightarrow A_1 \sqcap A_2 \sqcap \ldots$ and hence $\models U \circ\!\!- A_1 \sqcap A_2 \sqcap \ldots$, intuitively meaning a problem to which each A_i is reducible. For every B, the problem $U = B \sqcap A_1 \sqcap A_2 \ldots$ satisfies this condition, and universal problems of this particular form are called *standard*. Every universal problem U of a given universe can be shown to be equivalent to a standard universal problem U' of the same universe, in the sense that $\models U \circ\!\!- U'$ and $\models U' \circ\!\!- U$. And all of the operators of **INT** can be shown to preserve equivalence. Hence, restricting universal problems to standard ones does not result in any loss of generality: a universal problem can always be safely assumed to be standard. See section 23 of [7] for an extended discussion of the philosophy and intuitions associated with universal problems. Here we only note that interpreting \$ as a universal problem rather than (as one might expect) as $\bot$ yields more generality, for $\bot$ is nothing but a special, extreme case of a universal problem. *Our soundness theorem for* **INT***, of course, continues to hold with* $\bot$ *instead of* \$.

Let F be a formula. We write $\Vdash F$ and say that F is **valid** iff $\models F^*$ for every F-admissible interpretation *. For an EPM $\mathcal{E}$, we write $\mathcal{E}\Vdash F$ and say that $\mathcal{E}$ is a **uniform solution** for F iff $\mathcal{E} \models F^*$ for every F-admissible interpretation *. Finally, we write $\Vdash F$ and say that F is **uniformly valid** iff there is a uniform solution for F. Note that uniform validity automatically implies validity but not vice versa. Yet, these two concepts have been conjectured to be extensionally equivalent (Conjecture 26.2 of [7]).

7 The Gentzen-style axiomatization of INT

A **sequent** is a pair $\underline{G} \Rightarrow K$, where K is an **INT**-formula and $\underline{G}$ is a (possibly empty) finite sequence of **INT**-formulas. In what follows, E, F, K will be used as metavariables for formulas, and $\underline{G}, \underline{H}$ as metavariables for sequences of formulas. We think of sequents as formulas, identifying $\Rightarrow K$ with K, $F \Rightarrow K$ with $\circ\!\!- F \rightarrow K$

(i.e. $F \circ\!\!- K$), and $E_1, \ldots, E_n \Rightarrow K$ ($n \geq 2$) with $\downarrow\!\circ E_1 \wedge \ldots \wedge \downarrow\!\circ E_n \rightarrow K$.[6] This allows us to automatically extend the concepts such as validity, uniform validity, etc. from formulas to sequents. A formula K is considered **provable** in **INT** iff the sequent $\Rightarrow K$ is so.

Deductively, logic **INT** is given by the following 15 rules. This axiomatization is known (or can be easily shown) to be equivalent to other "standard" formulations, including Hilbert-style axiomatizations for **INT** and/or versions where a primitive symbol for negation is present while \$ is absent, or where $\sqcap$ and $\sqcup$ are strictly binary, or where variables are the only terms of the language.[7]

Below $\underline{G}, \underline{H}$ are any (possibly empty) sequences of **INT**-formulas; $n \geq 2$; $1 \leq i \leq n$; x is any variable; E, F, K, F_1, ..., F_n, K_1, ..., K_n, $F(x)$, $K(x)$ are any **INT**-formulas; y is any variable not occurring (whether free or bound) in the conclusion of the rule; in Left $\bigsqcap$ (resp. Right $\bigsqcup$), t is any term free for x in $F(x)$ (resp. in $K(x)$). $\mathcal{P} \longmapsto C$ means "from premise(s) $\mathcal{P}$ conclude C". When there are multiple premises in $\mathcal{P}$, they are separated with a semicolon.

Identity		$\longmapsto$	$K \Rightarrow K$
Domination		$\longmapsto$	$\$ \Rightarrow K$
Exchange	$\underline{G}, E, F, \underline{H} \Rightarrow K$	$\longmapsto$	$\underline{G}, F, E, \underline{H} \Rightarrow K$
Weakening	$\underline{G} \Rightarrow K$	$\longmapsto$	$\underline{G}, F \Rightarrow K$
Contraction	$\underline{G}, F, F \Rightarrow K$	$\longmapsto$	$\underline{G}, F \Rightarrow K$
Right $\circ\!\!-$	$\underline{G}, F \Rightarrow K$	$\longmapsto$	$\underline{G} \Rightarrow F \circ\!\!- K$
Left $\circ\!\!-$	$\underline{G}, F \Rightarrow K_1;\ \underline{H} \Rightarrow K_2$	$\longmapsto$	$\underline{G}, \underline{H}, K_2 \circ\!\!- F \Rightarrow K_1$
Right $\sqcap$	$\underline{G} \Rightarrow K_1;\ \ldots;\ \underline{G} \Rightarrow K_n$	$\longmapsto$	$\underline{G} \Rightarrow K_1 \sqcap \ldots \sqcap K_n$
Left $\sqcap$	$\underline{G}, F_i \Rightarrow K$	$\longmapsto$	$\underline{G}, F_1 \sqcap \ldots \sqcap F_n \Rightarrow K$
Right $\sqcup$	$\underline{G} \Rightarrow K_i$	$\longmapsto$	$\underline{G} \Rightarrow K_1 \sqcup \ldots \sqcup K_n$
Left $\sqcup$	$\underline{G}, F_1 \Rightarrow K;\ \ldots;\ \underline{G}, F_n \Rightarrow K$	$\longmapsto$	$\underline{G}, F_1 \sqcup \ldots \sqcup F_n \Rightarrow K$
Right $\bigsqcap$	$\underline{G} \Rightarrow K(y)$	$\longmapsto$	$\underline{G} \Rightarrow \bigsqcap x K(x)$
Left $\bigsqcap$	$\underline{G}, F(t) \Rightarrow K$	$\longmapsto$	$\underline{G}, \bigsqcap x F(x) \Rightarrow K$
Right $\bigsqcup$	$\underline{G} \Rightarrow K(t)$	$\longmapsto$	$\underline{G} \Rightarrow \bigsqcup x K(x)$
Left $\bigsqcup$	$\underline{G}, F(y) \Rightarrow K$	$\longmapsto$	$\underline{G}, \bigsqcup x F(x) \Rightarrow K$

[6] In order to fully remain within the language of **INT**, we could understand $E_1, \ldots, E_n \Rightarrow K$ as $E_1 \circ\!\!- (E_2 \circ\!\!- \ldots \circ\!\!- (E_n \circ\!\!- K) \ldots)$, which can be shown to be equivalent to our present understanding. We, however, prefer to read $E_1, \ldots, E_n \Rightarrow K$ as $\downarrow\!\circ E_1 \wedge \ldots \wedge \downarrow\!\circ E_n \rightarrow K$ as it seems more convenient to work with.

[7] That we allow constants is only for technical convenience. This does not really yield a stronger language, as constants behave like free variables and can be thought of as such.

Theorem 12. (Soundness:) *If* **INT** $\vdash S$*, then* $\Vdash S$ *(any sequent* S*). Furthermore,* **(uniform-constructive soundness:)** *there is an effective procedure that takes any* **INT***-proof of any sequent* S *and constructs a uniform solution for* S*.*

Proof. See Section 12. □

8 CL2-derived validity lemmas

In our proof of Theorem 12 we will need a number of lemmas concerning uniform validity of certain formulas. Some such validity proofs will be given directly in Section 10. But some proofs come "for free", based on the soundness theorem for logic **CL2** proven in [12]. **CL2** is a propositional logic whose logical atoms are $\top$ and $\bot$ (but not \$) and whose connectives are $\neg, \wedge, \vee, \rightarrow, \sqcap, \sqcup$. It has two sorts of nonlogical atoms, called *elementary* and *general*. General atoms are nothing but 0-ary atoms of our extended language; elementary atoms, however, are something not present in the extended language. We refer to the formulas of the language of **CL2** as **CL2-formulas**. In this paper, the **CL2**-formulas that do not contain elementary atoms — including $\top$ and $\bot$ that count as such — are said to be **general-base**. Thus, every general-base formula is a formula of our extended language, and its validity or uniform validity means the same as in Section 6.[8]

Understanding $F \rightarrow G$ as an abbreviation for $\neg F \vee G$, a **positive occurrence** in a **CL2**-formula is an occurrence that is in the scope of an even number of occurrences of $\neg$; otherwise the occurrence is **negative**. A **surface occurrence** is an occurrence that is not in the scope of $\sqcap$ and/or $\sqcup$. The **elementarization** of a **CL2**-formula F is the result of replacing in F every surface occurrence of every subformula of the form $G_1 \sqcap \ldots \sqcap G_n$ (resp. $G_1 \sqcup \ldots \sqcup G_n$) by $\top$ (resp. $\bot$), and every positive (resp. negative) surface occurrence of every general atom by $\bot$ (resp. $\top$). A **CL2**-formula F is said to be **stable** iff its elementarization is a tautology of classical logic. With these conventions, **CL2** is axiomatized by the following three rules:

(a) $\vec{H} \longmapsto F$, where F is stable and $\vec{H}$ is the smallest set of formulas such that, whenever F has a positive (resp. negative) surface occurrence of a subformula $G_1 \sqcap \ldots \sqcap G_n$ (resp. $G_1 \sqcup \ldots \sqcup G_n$), for each $i \in \{1, \ldots, n\}$, $\vec{H}$ contains the result of replacing that occurrence in F by G_i.

(b) $H \longmapsto F$, where H is the result of replacing in F a negative (resp. positive) surface occurrence of a subformula $G_1 \sqcap \ldots \sqcap G_n$ (resp. $G_1 \sqcup \ldots \sqcup G_n$) by G_i for some $i \in \{1, \ldots, n\}$.

(c) $H \longmapsto F$, where H is the result of replacing in F two — one positive and one negative — surface occurrences of some general atom by a nonlogical elementary atom that does not occur in F.

[8] These concepts extend to the full language of **CL2** as well, with interpretations required to send elementary atoms to elementary games (i.e. predicates in the classical sense, understood in CL as games that have no nonemty legal runs).

In this section $p, q, r, s, t, u, w \ldots$ (possibly with indices) will exlusively stand for nonlogical elementary atoms, and P, Q, R, S, T, U, W (possibly with indices) stand for general atoms. All of these atoms are implicitely assumed to be pairwise distinct in each context.

Convention 13. In Section 7 we started using the notation $\underline{G}$ for sequences of formulas. Later we agreed to identify sequences of formulas with $\wedge$-conjunctions of those formulas. So, from now on, an underlined expression such as $\underline{G}$ will mean an arbitrary formula $G_1 \wedge \ldots \wedge G_n$ for some $n \geq 0$. Such an expression will always occur in a bigger context such as $\underline{G} \wedge F$ or $\underline{G} \rightarrow F$; our convention is that, when $n = 0$, $\underline{G} \wedge F$ and $\underline{G} \rightarrow F$ simply mean F.

As we agreed that $p, q, \ldots$ stand for elementary atoms and $P, Q, \ldots$ for general atoms, $\underline{p}, \underline{q}, \ldots$ will correspondingly mean $\wedge$-conjunctions of elementary atoms, and $\underline{P}, \underline{Q}, \ldots$ mean $\wedge$-conjunctions of general atoms.

We will also be underlining complex expressions such as $F \rightarrow G$, $\bigsqcap x F(x)$ or $\downarrow\!\circ F$. $\underline{F \rightarrow G}$ should be understood as $(F_1 \rightarrow G_1) \wedge \ldots \wedge (F_n \rightarrow G_n)$, $\underline{\bigsqcap x F(x)}$ as $\bigsqcap x F_1(x) \wedge \ldots \wedge \bigsqcap x F_n(x)$ (note that only the F_i vary but not x), $\underline{\downarrow\!\circ F}$ as $\downarrow\!\circ F_1 \wedge \ldots \downarrow\!\circ F_n$, $\downarrow\!\circ\underline{\downarrow\!\circ F}$ as $\downarrow\!\circ(\downarrow\!\circ F_1 \wedge \ldots \wedge \downarrow\!\circ F_n)$, $\underline{\downarrow\!\circ F} \rightarrow \underline{F} \wedge G$ as $\downarrow\!\circ F_1 \wedge \ldots \downarrow\!\circ F_n \rightarrow F_1 \wedge \ldots \wedge F_n \wedge G$, etc.

The axiomatization of **CL2** is rather unusual, but it is easy to get a syntactic feel of it once we do a couple of exercises.

Example 14. The following is a **CL2**-proof of $(P \rightarrow Q) \wedge (Q \rightarrow T) \rightarrow (P \rightarrow T)$:

1. $(p \rightarrow q) \wedge (q \rightarrow t) \rightarrow (p \rightarrow t)$ (from $\{\}$ by Rule **(a)**).
2. $(P \rightarrow q) \wedge (q \rightarrow t) \rightarrow (P \rightarrow t)$ (from 1 by Rule **(c)**).
3. $(P \rightarrow Q) \wedge (Q \rightarrow t) \rightarrow (P \rightarrow t)$ (from 2 by Rule **(c)**).
4. $(P \rightarrow Q) \wedge (Q \rightarrow T) \rightarrow (P \rightarrow T)$ (from 3 by Rule **(c)**).

Example 15. Let $n \geq 2$, and let m be the length (number of conjuncts) of both $\underline{R}$ and $\underline{r}$.

a) For $i \in \{1, \ldots, n\}$, the formula of Lemma 17(j) is provable in **CL2**. It follows from $(\underline{R} \rightarrow S_i) \rightarrow (\underline{R} \rightarrow S_i)$ by Rule **(b)**; the latter follows from $(\underline{R} \rightarrow s_i) \rightarrow (\underline{R} \rightarrow s_i)$ by Rule **(c)**; the latter, in turn, can be derived from $(\underline{r} \rightarrow s_i) \rightarrow (\underline{r} \rightarrow s_i)$ applying Rule **(c)** m times. Finally, $(\underline{r} \rightarrow s_i) \rightarrow (\underline{r} \rightarrow s_i)$ is its own elementarization and is a classical tautology, so it follows from the empty set of premises by Rule **(a)**.

b) The formula of Lemma 17(h) is also provable in **CL2**. It is derivable by Rule **(a)** from the set of n premises, easch premise being $(\underline{R} \rightarrow S_1) \wedge \ldots \wedge (\underline{R} \rightarrow S_n) \rightarrow (\underline{R} \rightarrow S_i)$ for some $i \in \{1, \ldots, n\}$. The latter is derivable by Rule **(c)** from $(\underline{R} \rightarrow S_1) \wedge \ldots \wedge (\underline{R} \rightarrow s_i) \wedge \ldots \wedge (\underline{R} \rightarrow S_n) \rightarrow (\underline{R} \rightarrow s_i)$. The latter, in turn, can be derived from $(\underline{R} \rightarrow S_1) \wedge \ldots \wedge (\underline{r} \rightarrow s_i) \wedge \ldots \wedge (\underline{R} \rightarrow S_n) \rightarrow (\underline{r} \rightarrow s_i)$ applying Rule **(c)** m times. Finally, the latter follows from the empty set of premises by Rule **(a)**.

Obviously **CL2** is decidable. This logic has been proven sound and complete in [12]. We only need the soundness part of that theorem restricted to general-base formulas. It sounds as follows:

Lemma 16. *Any general-base* **CL2***-provable formula is valid. Furthermore, there is an effective procedure that takes any* **CL2***-proof of any such formula F and constructs a uniform solution for F.*

A *substitution* is a function f that sends every general atom P of the language of **CL2** to a formula $f(P)$ of the extended language. This mapping extends to all general-base **CL2**-formulas by stipulating that f commutes with all operators: $f(G_1 \to G_2) = f(G_1) \to f(G_2)$, $f(G_1 \sqcap \ldots \sqcap G_k) = f(G_1) \sqcap \ldots \sqcap f(G_k)$, etc. We say that a formula G is a **substitutional instance** of a general-base **CL2**-formula F iff $G = f(F)$ for some substitution f. Thus, "G is a substitutional instance of F" means that G has the same form as F.

In the following lemma, we assume $n \geq 2$ (clauses (h),(i),(j)), and $1 \leq i \leq n$ (clause (j)). Notice that, for the exception of clause (g), the expressions given below are schemata of formulas rather than formulas, for the lengths of their underlined expressions — as well as i and n — may vary.

Lemma 17. *All substitutional instances of all formulas given by the following schemata are uniformly valid. Furthermore, there is an effective procedure that takes any particular formula matching a given scheme and constructs an EPM that is a uniform solution for all substitutional instances of that formula.*

a) $(\underline{R} \wedge P \wedge Q \wedge \underline{S} \to T) \to (\underline{R} \wedge Q \wedge P \wedge \underline{S} \to T)$;
b) $(\underline{R} \to T) \to (\underline{R} \wedge P \to T)$;
c) $\underline{(R \to S)} \to (\underline{W} \wedge \underline{R} \wedge \underline{U} \to \underline{W} \wedge \underline{S} \wedge \underline{U})$;
d) $\big(\underline{R} \wedge P \to Q\big) \to \big(\underline{R} \to (P \to Q)\big)$;
e) $\big(P \to (Q \to T)\big) \wedge (\underline{R} \to Q) \to \big(P \to (\underline{R} \to T)\big)$;
f) $\big(P \to (\underline{R} \to Q)\big) \wedge (\underline{S} \wedge Q \to T) \to (\underline{S} \wedge \underline{R} \wedge P \to T)$;
g) $(P \to Q) \wedge (Q \to T) \to (P \to T)$;
h) $(\underline{R} \to S_1) \wedge \ldots \wedge (\underline{R} \to S_n) \to (\underline{R} \to S_1 \sqcap \ldots \sqcap S_n)$;
i) $(\underline{R} \wedge S_1 \to T) \wedge \ldots \wedge (\underline{R} \wedge S_n \to T) \to \big((\underline{R} \wedge (S_1 \sqcup \ldots \sqcup S_n) \to T\big)$;
j) $(\underline{R} \to S_i) \to (\underline{R} \to S_1 \sqcup \ldots \sqcup S_n)$.

Proof. In order to prove this lemma, it would be sufficient to show that all formulas given by the above schemata are provable in **CL2**. Indeed, if we succeed in doing so, then an effective procedure whose existence is claimed in the lemma could be designed to work as follows: First, the procedure finds a **CL2**-proof of a given formula F. Then, based on that proof and using the procedure whose existence is stated in Lemma 16, it finds a uniform solution $\mathcal{E}$ for that formula. It is not hard to see that the same $\mathcal{E}$ will automatically be a uniform solution for every substitutional instance of F as well.

The **CL2**-provability of the formulas given by clauses (g), (h) and (j) has been verified in Examples 14 and 15. A similar verification for the other clauses is left as an easy syntactic exercise for the reader. □

9 Closure lemmas

In this section we let n range over natural numbers (including 0), x over any variables, F, E, G (possibly with subscripts) over any formulas, and $\mathcal{E}$, $\mathcal{C}$, $\mathcal{D}$ (possibly with subscripts) over any EPMs. Unless otherwise specified, these metavariables are assumed to be universally quantified in a context. The expression $\{EPMs\}$ stands for the set of all EPMs.

Lemma 18. *If $\Vdash F$, then $\Vdash \mathbin{⫰} F$. Furthermore, there is an effective function $h : \{EPMs\} \longrightarrow \{EPMs\}$ such that, for any $\mathcal{E}$ and F, if $\mathcal{E} \Vdash F$, then $h(\mathcal{E}) \Vdash \mathbin{⫰} F$.*

Proof. According to Proposition 21.2 of [7], there is an effective function $h : \{\text{EPMs}\} \longrightarrow \{\text{EPMs}\}$ such that, for any EPM $\mathcal{E}$ and any static game A, if $\mathcal{E} \models A$, then $h(\mathcal{E}) \models \mathbin{⫰} A$. We now claim that if $\mathcal{E} \Vdash F$, then $h(\mathcal{E}) \Vdash \mathbin{⫰} F$. Indeed, assume $\mathcal{E} \Vdash F$. Consider any $\mathbin{⫰} F$-admissible interpretation *. Of course, the same interpretation is also F-admissible. Hence, $\mathcal{E} \Vdash F$ implies $\mathcal{E} \models F^*$. But then, by the known behavior of h, we have $h(\mathcal{E}) \models \mathbin{⫰} F^*$. Since * was arbitrary, we conclude that $h(\mathcal{E}) \Vdash \mathbin{⫰} F$. □

Lemma 19. *If $\Vdash F$, then $\Vdash \bigsqcap x F$. Furthermore, there is an effective function $h : \{EPMs\} \longrightarrow \{EPMs\}$ such that, for any $\mathcal{E}$, x and F, if $\mathcal{E} \Vdash F$, then $h(\mathcal{E}) \Vdash \bigsqcap x F$.*

Proof. Similar to the previous lemma, only based on Proposition 21.1 of [7] instead of 21.2. □

Lemma 20. *If $\Vdash F$ and $\Vdash F \rightarrow E$, then $\Vdash E$. Furthermore, there is an effective function $h : \{EPMs\} \times \{EPMs\} \longrightarrow \{EPMs\}$ such that, for any $\mathcal{E}$, $\mathcal{C}$, F and E, if $\mathcal{E} \Vdash F$ and $\mathcal{C} \Vdash F \rightarrow E$, then $h(\mathcal{E}, \mathcal{C}) \Vdash E$.*

Proof. According to Proposition 21.3 of [7], there is an effective function g that takes any HPM $\mathcal{H}$ and EPM $\mathcal{E}$ and returns an EPM $\mathcal{C}$ such that, for any static games A,B and any valuation e, if $\mathcal{H} \models_e A$ and $\mathcal{E} \models_e A \rightarrow B$, then $\mathcal{C} \models_e B$. Theorem 17.2 of [7], which establishes equivalence between EPMs and HPMs in a constructive sense, allows us to assume that $\mathcal{H}$ is an EPM rather than HPM here. More precisely, we can routinely convert g into an (again effective) function $h : \{\text{EPMs}\} \times \{\text{EPMs}\} \longrightarrow \{\text{EPMs}\}$ such that, for any static games A,B, valuation e and EPMs $\mathcal{E}$ and $\mathcal{C}$,

$$\text{if } \mathcal{E} \models_e A \text{ and } \mathcal{C} \models_e A \rightarrow B, \text{ then } h(\mathcal{E}, \mathcal{C}) \models_e B. \tag{3}$$

We claim that the above function h behaves as our lemma states. Assume $\mathcal{E} \Vdash F$ and $\mathcal{C} \Vdash F \rightarrow E$, and consider an arbitrary valuation e and an arbitrary E-admissible interpretation *. Our goal is to show that $h(\mathcal{E}, \mathcal{C}) \models_e E^*$, which obviously means the same as

$$h(\mathcal{E}, \mathcal{C}) \models_e e[E^*]. \tag{4}$$

We define the new interpretation † by stipulating that, for every n-ary letter P with $P^* = P^*(x_1, \ldots, x_n)$, $P^\dagger$ is the game $P^\dagger(x_1, \ldots, x_n)$ such that, for any tuple $c_1, \ldots, c_n$ of constants, $P^\dagger(c_1, \ldots, c_n) = e[P^*(c_1, \ldots, c_n)]$. Unlike $P^*(x_1, \ldots, x_n)$

that may depend on some "hidden" variables (those that are not among $x_1, \ldots, x_n$), obviously $P^\dagger(x_1, \ldots, x_n)$ does not depend on any variables other that $x_1, \ldots, x_n$. This makes † admissible for any formula, including F and $F \to E$. Then our assumptions $\mathcal{E}\Vdash F$ and $\mathcal{C}\Vdash F \to E$ imply $\mathcal{E} \models_e F^\dagger$ and $\mathcal{C} \models_e F^\dagger \to E^\dagger$. Consequently, by (3), $h(\mathcal{E},\mathcal{C}) \models_e E^\dagger$, i.e. $h(\mathcal{E},\mathcal{C}) \models_e e[E^\dagger]$. Now, with some thought, we can see that $e[E^\dagger] = e[E^*]$. Hence (4) is true. □

Lemma 21. (Modus ponens) *If $\Vdash F_1$, ..., $\Vdash F_n$ and $\Vdash F_1 \wedge \ldots \wedge F_n \to E$, then $\Vdash E$. Furthermore, there is an effective function $h : \{EPMs\}^{n+1} \longrightarrow \{EPMs\}$ such that, for any EPMs $\mathcal{E}_1$, ..., $\mathcal{E}_n$, $\mathcal{C}$ and any formulas $F_1, \ldots, F_n, E$, if $\mathcal{E}_1\Vdash F_1$, ..., $\mathcal{E}_n\Vdash F_n$ and $\mathcal{C}\Vdash F_1 \wedge \ldots \wedge F_n \to E$, then $h(\mathcal{E}_1, \ldots, \mathcal{E}_n, \mathcal{C})\Vdash E$. Such a function, in turn, can be effectively constructed for each particular n.*

Proof. In case $n = 0$, h is simply the identity function $h(\mathcal{C}) = \mathcal{C}$. In case $n = 1$, h is the function whose existence is stated in Lemma 20. Below we will construct h for case $n = 2$. It should be clear how to generalize that construction to any greater n.

Assume $\mathcal{E}_1\Vdash F_1$, $\mathcal{E}_2\Vdash F_2$ and $\mathcal{C}\Vdash F_1 \wedge F_2 \to E$. By Lemma 17(d), $(F_1 \wedge F_2 \to E) \to (F_1 \to (F_2 \to E))$ has a uniform solution. Lemma 20 allows us to combine that solution with $\mathcal{C}$ and find a uniform solution $\mathcal{D}_1$ for $F_1 \to (F_2 \to E)$. Now applying Lemma 20 to $\mathcal{E}_1$ and $\mathcal{D}_1$, we find a uniform solution $\mathcal{D}_2$ for $F_2 \to E$. Finally, applying the same lemma to $\mathcal{E}_2$ and $\mathcal{D}_2$, we find a uniform solution $\mathcal{D}$ for E. Note that $\mathcal{D}$ does not depend on F_1, F_2, E, and that we constructed $\mathcal{D}$ in an effective way from the arbitrary $\mathcal{E}_1$, $\mathcal{E}_2$ and $\mathcal{C}$. Formalizing this construction yields function h whose existence is claimed by the lemma. □

Lemma 22. (Transitivity) *If $\Vdash F \to E$ and $\Vdash E \to G$, then $\Vdash F \to G$. Furthermore, there is an effective function $h : \{EPMs\} \times \{EPMs\} \longrightarrow \{EPMs\}$ such that, for any $\mathcal{E}_1$, $\mathcal{E}_2$, F, E and G, if $\mathcal{E}_1\Vdash F \to E$ and $\mathcal{E}_2\Vdash E \to G$, then $h(\mathcal{E}_1, \mathcal{E}_2)\Vdash F \to G$.*

Proof. Assume $\mathcal{E}_1\Vdash F \to E$ and $\mathcal{E}_2\Vdash E \to G$. By Lemma 17(g), we also have $\mathcal{C}\Vdash (F \to E) \wedge (E \to G) \to (F \to G)$ for some (fixed) $\mathcal{C}$. Lemma 21 allows us to combine the three uniform solutions and construct a uniform solution $\mathcal{D}$ for $F \to G$. □

10 More validity lemmas

As pointed out in Remark 16.4 of [7], when trying to show that a given EPM wins a given game, it is always safe to assume that the runs that the machine generates are never $\bot$-illegal, i.e. that the environment never makes an illegal move, for if it does, the machine automatically wins. This assumption, that we call the **clean environment assumption**, will always be implicitly present in our winnability proofs.

We will often employ a uniform solution for $P \to P$ called the **copy-cat strategy** ($\mathcal{CCS}$). This strategy, that we already saw in Section 2, consists in mimicking, in the antecedent, the moves made by the environment in the consequent, and vice

versa. More formally, the algorithm that $\mathcal{CCS}$ follows is an infinite loop, on every iteration of which $\mathcal{CCS}$ keeps granting permission until the environment makes a move $1.\alpha$ (resp. $2.\alpha$), to which the machine responds by the move $2.\alpha$ (resp. $1.\alpha$). As shown in the proof of Proposition 22.1 of [7], this strategy guarantees success in every game of the form $A \vee \neg A$ and hence $A \rightarrow A$. An important detail is that $\mathcal{CCS}$ never looks at the past history of the game, i.e. the movement of its scanning head on the run tape is exclusively left-to-right. This guarantees that, even if the original game was something else and it only evolved to $A \rightarrow A$ later as a result of making a series of moves, switching to the $\mathcal{CCS}$ after the game has been brought down to $A \rightarrow A$ guarantees success no matter what happened in the past.

Thgroughout this section, F, G, E, K (possibly with indices and attached tuples of variables) range over formulas, x and y over variables, t over terms, n over nonnegative integers, w over bit strings, and α, γ over moves. These (meta)variables are assumed to be universally quantified in a context unless otherwise specified. In accordance with our earlier convention, ϵ means the empty string, so that, say, '$1.\epsilon.\alpha$' is the same as '$1..\alpha$'.

Lemma 23. *$\Vdash \mathbin{!\!\circ} F \rightarrow F$. Furthermore, there is an EPM $\mathcal{E}$ such that, for any F, $\mathcal{E} \Vdash \mathbin{!\!\circ} F \rightarrow F$.*

Proof. The idea of a uniform solution $\mathcal{E}$ for $\mathbin{!\!\circ} F \rightarrow F$ is very simple: just act as $\mathcal{CCS}$, never making any replicative moves in the antecedent and pretending that the latter is F rather than (the stronger) $\mathbin{!\!\circ} F$. The following formal description of the interactive algorithm that $\mathcal{E}$ follows is virtually the same as that of $\mathcal{CCS}$, with the only difference that the move prefix '1.' is replaced by '$1.\epsilon.$' everywhere.

Procedure LOOP: Keep granting permission until the environment makes a move $1.\epsilon.\alpha$ or $2.\alpha$; in the former case make the move $2.\alpha$, and in the latter case make the move $1.\epsilon.\alpha$; then repeat LOOP.

Fix an arbitrary valuation e, interpretation * and e-computation branch B of $\mathcal{E}$. Let Θ be the run spelled by B. From the description of LOOP it is clear that permission will be granted infinitely many times in B, so this branch is fair. Hence, in order to show that $\mathcal{E}$ wins the game, it would suffice to show that $\mathbf{Wn}_e^{\mathbin{!\!\circ} F^* \rightarrow F^*} \langle \Theta \rangle = \top$.

Let Θ_i denote the initial segment of Θ consisting of the (lab)moves made by the players by the beginning of the ith iteration of LOOP in B (if such an iteration exists). By induction on i, based on the clean environment assumption and applying a routine analysis of the the behavior of LOOP and the definitions of the relevant game operations, one can easily find that

a) $\Theta_i \in \mathbf{Lr}_e^{\mathbin{!\!\circ} F^* \rightarrow F^*}$;
b) $\Theta_i^{1.\epsilon.} = \neg \Theta_i^{2.}$;
c) *All of the moves in $\Theta_i^{1.}$ have the prefix '$\epsilon.$'.*

If LOOP is iterated infinitely many times, then the above obviously extends from Θ_i to Θ, because every initial segment of Θ is an initial segment of some Θ_i. And if LOOP is iterated only a finite number m of times, then $\Theta = \Theta_m$. This

is so because the environment cannot make a move $1.\epsilon.\alpha$ or $2.\alpha$ during the mth iteration (otherwise there would be a next iteration), and any other move would violate the clean environment assumption; and, as long as the environment does not move during a given iteration, neither does the machine. Thus, no matter whether LOOP is iterated a finite or infinite number of times, we have:

$$\begin{array}{ll} \text{a) } \Theta \in \mathbf{Lr}_e^{\mathbin{!}\circ F^*\to F^*}; & \\ \text{b) } \Theta^{1.\epsilon.} = \neg\Theta^{2.}\ ; & (5) \\ \text{c) } \textit{All of the moves in } \Theta^{1.} \textit{ have the prefix `}\epsilon.\textit{'.} & \end{array}$$

Since $\Theta \in \mathbf{Lr}_e^{\mathbin{!}\circ F^*\to F^*}$, in order to show that $\mathbf{Wn}_e^{\mathbin{!}\circ F^*\to F^*}\langle\Theta\rangle = \top$, by the definition of $\to$, it would suffice to show that either $\mathbf{Wn}_e^{F^*}\langle\Theta^{2.}\rangle = \top$ or $\mathbf{Wn}_e^{\neg\mathbin{!}\circ F^*}\langle\Theta^{1.}\rangle = \top$. So, assume $\mathbf{Wn}_e^{F^*}\langle\Theta^{2.}\rangle \neq \top$, i.e. $\mathbf{Wn}_e^{F^*}\langle\Theta^{2.}\rangle = \bot$, i.e. $\mathbf{Wn}_e^{\neg F^*}\langle\neg\Theta^{2.}\rangle = \top$. Then, by clause (b) of (5), $\mathbf{Wn}_e^{\neg F^*}\langle\Theta^{1.\epsilon.}\rangle = \top$, i.e. $\mathbf{Wn}_e^{F^*}\langle\neg\Theta^{1.\epsilon.}\rangle = \bot$, i.e. $\mathbf{Wn}_e^{F^*}\langle(\neg\Theta^{1.})^{\epsilon.}\rangle = \bot$. Pick any infinite bit string w. In view of clause (c) of (5), we obviously have $(\neg\Theta^{1.})^{\epsilon.} = (\neg\Theta^{1.})^{\preceq w}$. Hence $\mathbf{Wn}_e^{F^*}\langle(\neg\Theta^{1.})^{\preceq w}\rangle = \bot$. But this, by the definition of $\mathbin{!}\circ$, implies $\mathbf{Wn}_e^{\mathbin{!}\circ F^*}\langle\neg\Theta^{1.}\rangle = \bot$. The latter, in turn, can be rewritten as the desired $\mathbf{Wn}_e^{\neg\mathbin{!}\circ F^*}\langle\Theta^{1.}\rangle = \top$.

Thus, we have shown that $\mathcal{E}$ wins $\mathbin{!}\circ F^* \to F^*$. Since * was arbitrary and the work of $\mathcal{E}$ did not depend on it, we conclude that $\mathcal{E}\Vdash \mathbin{!}\circ F \to F$. □

In the subsequent constructions found in this section, * will always mean an arbitrary but fixed interpretation admissible for the formula whose uniform validity we are trying to prove. Next, e will always mean an arbitrary but fixed valuation — specifically, the valuation spelled on the valuation tape of the machine under question. For readability, we will usually omit the e parameter when it is irrelevant. Also, having already seen one example, in the remaining uniform validity proofs we will typically limit ourselves to just constructing interactive algorithms, leaving the (usually routine) verification of their correctness to the reader. An exception will be the proof of Lemma 34 where, due to the special complexity of the case, correctness verification will be done even more rigorously than we did this in the proof of Lemma 23.

Lemma 24. $\Vdash \mathbin{!}\circ(F \to G) \to (\mathbin{!}\circ F \to \mathbin{!}\circ G)$. *Moreover, there is an EPM* $\mathcal{E}$ *such that, for every* F *and* G, $\mathcal{E}\Vdash \mathbin{!}\circ(F \to G) \to (\mathbin{!}\circ F \to \mathbin{!}\circ G)$.

Proof. A relaxed description of a uniform solution $\mathcal{E}$ for $\mathbin{!}\circ(F \to G) \to (\mathbin{!}\circ F \to \mathbin{!}\circ G)$ is as follows. In $\mathbin{!}\circ(F^* \to G^*)$ and $\mathbin{!}\circ F^*$ the machine is making exactly the same replicative moves (moves of the form w:) as the environment is making in $\mathbin{!}\circ G^*$. This ensures that the tree-structures of the three $\mathbin{!}\circ$-components of the game are identical, and now all the machine needs for a success is to win the game $(F^* \to G^*) \to (F^* \to G^*)$ within each branch of those trees. This can be easily achieved by applying copy-cat methods to the two occurrences of F and the two occurrences of G.

In precise terms, the strategy that $\mathcal{E}$ follows is described by the following interactive algorithm.

Procedure LOOP: Keep granting permission until the adversary makes a move γ. Then:
If $\gamma = 2.2.w$:, then make the moves $1.w$: and $2.1.w$:, and repeat LOOP;
If $\gamma = 2.2.w.\alpha$ (resp. $\gamma = 1.w.2.\alpha$), then make the move $1.w.2.\alpha$ (resp. $2.2.w.\alpha$), and repeat LOOP;
If $\gamma = 2.1.w.\alpha$ (resp. $\gamma = 1.w.1.\alpha$), then make the move $1.w.1.\alpha$ (resp. $2.1.w.\alpha$), and repeat LOOP. □

Lemma 25. $\Vdash \mathbin{⫰}\!\circ F_1 \wedge \ldots \wedge \mathbin{⫰}\!\circ F_n \rightarrow \mathbin{⫰}\!\circ(F_1 \wedge \ldots \wedge F_n)$. *Furthermore, there is an effective procedure that takes any particular value of n and constructs an EPM $\mathcal{E}$ such that, for any $F_1, \ldots, F_n$, $\mathcal{E} \Vdash \mathbin{⫰}\!\circ F_1 \wedge \ldots \wedge \mathbin{⫰}\!\circ F_n \rightarrow \mathbin{⫰}\!\circ(F_1 \wedge \ldots \wedge F_n)$.*

Proof. The idea of a uniform solution here is rather similar to the one in the proof of Lemma 24. Here is the algorithm:

Procedure LOOP: Keep granting permission until the adversary makes a move γ. Then:
If $\gamma = 2.w$:, then make the n moves $1.1.w:, \ldots, 1.n.w$:, and repeat LOOP;
If $\gamma = 2.w.i.\alpha$ (resp. $\gamma = 1.i.w.\alpha$) where $1 \leq i \leq n$, then make the move $1.i.w.\alpha$ (resp. $2.w.i.\alpha$), and repeat LOOP. □

Lemma 26. $\Vdash \mathbin{⫰}\!\circ F \rightarrow \circ F \wedge \mathbin{⫰}\!\circ F$. *Furthermore, there is an EPM $\mathcal{E}$ such that, for any F, $\mathcal{E} \Vdash \mathbin{⫰}\!\circ F \rightarrow \circ F \wedge \mathbin{⫰}\!\circ F$.*

Proof. The idea of a winning strategy here is to first replicate the antecedent turning it into something "essentially the same"[9] as $\mathbin{⫰}\!\circ F^* \wedge \mathbin{⫰}\!\circ F^*$, and then switch to a strategy that is "essentially the same as" the ordinary copy-cat strategy. Precisely, here is how $\mathcal{E}$ works: it makes the move $1.\epsilon$: (replicating the antecedent), after which it follows the following algorithm:

Procedure LOOP: Keep granting permission until the adversary makes a move γ. Then:
If $\gamma = 1.0\alpha$ (resp. $\gamma = 2.1.\alpha$), then make the move $2.1.\alpha$ (resp. 1.0α), and repeat LOOP;
If $\gamma = 1.1\alpha$ (resp. $\gamma = 2.2.\alpha$), then make the move $2.2.\alpha$ (resp. 1.1α), and repeat LOOP;
If $\gamma = 1.\epsilon.\alpha$, then make the moves $2.1.\epsilon.\alpha$ and $2.2.\epsilon.\alpha$, and repeat LOOP. □

Remember from Section 4 that, when t is a constant, $e(t) = t$.

Lemma 27. *For any* **INT***-formula K, $\Vdash \mathbin{⫰}\!\circ\$ \rightarrow K$. Furthermore, there is an effective procedure that takes any* **INT***-formula K and constructs a uniform solution for $\mathbin{⫰}\!\circ\$ \rightarrow K$.*

[9] Using the notation $\circ$ introduced in Section 13 of [7], in precise terms this "something" is $\mathbin{⫰}\!(F^* \circ F^*)$.

Proof. We construct an EPM $\mathcal{E}$ and verify that it is a uniform solution for $⫰\$ \to K$; both the construction and the verification will be done by induction on the complexity of K. The goal in each case is to show that $\mathcal{E}$ generates a $\top$-won run of $e[(⫰\$ \to K)^*] = ⫰e[\$^*] \to e[K^*]$ which, according to our convention, we may write with "e" omitted.

Case 1: $K = \$$. This case is taken care of by Lemma 23.

Case 2: K is a k-ary nonlogical atom $P(t_1, \ldots, t_k)$. Let $c_1, \ldots, c_k$ be the constants $e(t_1), \ldots, e(t_k)$, respectively. Evidently in this case $e[K^*] = e[P^*(c_1, \ldots, c_k)]$, so, the game for which $\mathcal{E}$ needs to generate a winning run is $⫰\$^* \to P^*(c_1, \ldots, c_k)$. Assume $(P(c_1, \ldots, c_k))^*$ is conjunct $\#m$ of (the infinite $\sqcap$-conjunction) $\$^*$. We define $\mathcal{E}$ to be the EPM that acts as follows. At the beginning, if necessary (i.e. unless all t_i are constants), it reads the valuation tape to find $c_1, \ldots, c_k$. Then, using this information, it finds the above number m and makes the move '$1.\epsilon.m$', which can be seen[10] to bring the game down to $⫰P^*(c_1, \ldots, c_k) \to P^*(c_1, \ldots, c_k)$. After this move, $\mathcal{E}$ switches to the strategy whose existence is stated in Lemma 23.

Case 3: $K = ⫰E \to F$. By Lemma 17(b), there is a uniform solution for $(⫰\$ \to F) \to (⫰\$ \wedge ⫰E \to F)$. Lemmas 17(d) and 22 allow us to convert the latter into a uniform solution $\mathcal{D}$ for $(⫰\$ \to F) \to \big(⫰\$ \to (⫰E \to F)\big)$. By the induction hypothesis, there is also a uniform solution $\mathcal{C}$ for $⫰\$ \to F$. Applying Lemma 21 to $\mathcal{C}$ and $\mathcal{D}$, we find a uniform solution $\mathcal{E}$ for $⫰\$ \to (⫰E \to F)$.

Case 4: $K = E_1 \sqcup \ldots \sqcup E_n$. By the induction hypothesis, we know how to construct an EPM $\mathcal{C}_1$ with $\mathcal{C}_1 ⊩ ⫰\$ \to E_1$. Now we define $\mathcal{E}$ to be the EPM that first makes the move 2.1, and then plays the rest of the game as $\mathcal{C}_1$ would play. $\mathcal{E}$ can be seen to be successful because its initial move 2.1 brings $(⫰\$ \to K)^*$ down to $(⫰\$ \to E_1)^*$.

Case 5: $K = E_1 \sqcap \ldots \sqcap E_n$. By the induction hypothesis, for each i with $1 \le i \le n$ we have an EPM $\mathcal{C}_i$ with $\mathcal{C}_i ⊩ ⫰\$ \to E_i$. We define $\mathcal{E}$ to be the EPM that acts as follows. At the beginning, $\mathcal{E}$ keeps granting permission until the adversary makes a move. The clean environment assumption guarantees that this move should be $2.i$ for some $1 \le i \le n$. It brings $(⫰\$ \to E_1 \sqcap \ldots \sqcap E_n)^*$ down to $(⫰\$ \to E_i)^*$. If and after such a move $2.i$ is made, $\mathcal{E}$ continues the rest of the play as $\mathcal{C}_i$.

Case 6: $K = \bigsqcup x E(x)$. By the induction hypothesis, there is an EPM $\mathcal{C}_1$ with $\mathcal{C}_1 ⊩ ⫰\$ \to E(1)$. Now we define $\mathcal{E}$ to be the EPM that first makes the move 2.1, and then plays the rest of the game as $\mathcal{C}_1$. $\mathcal{E}$ can be seen to be successful because its initial move 2.1 brings $\big(⫰\$ \to \bigsqcup x E(x)\big)^*$ down to $\big(⫰\$ \to E(1)\big)^*$.

Case 7: $K = \bigsqcap x E(x)$. By the induction hypothesis, for each constant c there is (and can be effectively found) an EPM $\mathcal{C}_c$ with $\mathcal{C}_c ⊩ ⫰\$ \to E(c)$. Now we define $\mathcal{E}$ to be the EPM that acts as follows. At the beginning, $\mathcal{E}$ keeps granting permission until the adversary makes a move. By the clean environment assumption, such a move should be $2.c$ for some constant c. This move can be seen to bring $\big((⫰\$ \to \bigsqcap x E(x))\big)^*$ down to $\big(⫰\$ \to E(c)\big)^*$. If and after the above move $2.c$ is made, $\mathcal{E}$ plays the rest of the game as $\mathcal{C}_c$. □

[10] See Proposition 13.8 of [7].

Lemma 28. *Assume $n \geq 2$, $1 \leq i \leq n$, and t is a term free for x in $G(x)$. Then the following uniform validities hold. Furthermore, in each case there is an effective procedure that takes any particular values of n, i, t and constructs an EPM which is a uniform solution for the corresponding formula no matter what the values of $F_1, \ldots, F_n$ and/or $G(x)$ (as long as t is free for x in $G(x)$) are.*

a) $\Vdash \mathbin{⫰}\!\circ(F_1 \sqcap \ldots \sqcap F_n) \rightarrow \mathbin{⫰}\!\circ F_i$;
b) $\Vdash \mathbin{⫰}\!\circ \bigsqcap x G(x) \rightarrow \mathbin{⫰}\!\circ G(t)$;
c) $\Vdash \mathbin{⫰}\!\circ(F_1 \sqcup \ldots \sqcup F_n) \rightarrow \mathbin{⫰}\!\circ F_1 \sqcup \ldots \sqcup \mathbin{⫰}\!\circ F_n$;
d) $\Vdash \mathbin{⫰}\!\circ \bigsqcup x G(x) \rightarrow \bigsqcup x \mathbin{⫰}\!\circ G(x)$.

Proof. Below come winning strategies for each case.
Clause (a): Make the move '$1.\epsilon.i$'. This brings the game down to $\mathbin{⫰}\!\circ F_i^* \rightarrow \mathbin{⫰}\!\circ F_i^*$. Then switch to $\mathcal{CCS}$.
Clause (b): Let $c = e(t)$. Read c from the valuation tape if necessary, i.e., if t is a variable (otherwise, simply $c = t$). Then make the move '$1.\epsilon.c$'. This brings the game down to $\mathbin{⫰}\!\circ G^*(c) \rightarrow \mathbin{⫰}\!\circ G^*(c)$. Now, switch to $\mathcal{CCS}$.
Clause (c): Keep granting permission until the adversary makes a move '$1.\epsilon.j$' ($1 \leq j \leq n$), bringing the game down to $\mathbin{⫰}\!\circ F_j^* \rightarrow \mathbin{⫰}\!\circ F_1^* \sqcup \ldots \sqcup \mathbin{⫰}\!\circ F_n^*$. If and after such a move is made (and if not, a win is automatically guaranteed), make the move '$2.j$', which brings the game down to $\mathbin{⫰}\!\circ F_j^* \rightarrow \mathbin{⫰}\!\circ F_j^*$. Finally, switch to $\mathcal{CCS}$.
Clause (d): Keep granting permission until the adversary makes the move '$1.\epsilon.c$' for some constant c. This brings the game down to $\mathbin{⫰}\!\circ G^*(c) \rightarrow \bigsqcup x \mathbin{⫰}\!\circ G^*(x)$. Now make the move '$2.c$', which brings the game down to $\mathbin{⫰}\!\circ G^*(c) \rightarrow \mathbin{⫰}\!\circ G^*(c)$. Finally, switch to $\mathcal{CCS}$. □

Lemma 29. $\Vdash \bigsqcap x\big(F(x) \rightarrow G(x)\big) \rightarrow \big(\bigsqcap x F(x) \rightarrow \bigsqcap x G(x)\big)$. *Furthermore, there is an EPM $\mathcal{E}$ such that, for any $F(x)$ and $G(x)$, $\mathcal{E} \Vdash \bigsqcap x\big(F(x) \rightarrow G(x)\big) \rightarrow \big(\bigsqcap x F(x) \rightarrow \bigsqcap x G(x)\big)$.*

Proof. Strategy: Wait till the environment makes the move '$2.2.c$' for some constant c. This brings the $\bigsqcap x G^*(x)$ component down to $G^*(c)$ and hence the entire game to $\bigsqcap x\big(F^*(x) \rightarrow G^*(x)\big) \rightarrow \big(\bigsqcap x F^*(x) \rightarrow G^*(c)\big)$. Then make the same move c in the antecedent and in $\bigsqcap x F^*(x)$, i.e. make the moves '$1.c$' and '$2.1.c$'. The game will be brought down to $\big(F^*(c) \rightarrow G^*(c)\big) \rightarrow \big(F^*(c) \rightarrow G^*(c)\big)$. Finally, switch to $\mathcal{CCS}$. □

Lemma 30. $\Vdash \bigsqcap x\big(F_1(x) \wedge \ldots \wedge F_n(x) \wedge E(x) \rightarrow G(x)\big) \rightarrow \big(\bigsqcap x F_1(x) \wedge \ldots \wedge \bigsqcap x F_n(x) \wedge \bigsqcup x E(x) \rightarrow \bigsqcup x G(x)\big)$. *Furthermore, there is an effective procedure that takes any particular value of n and constructs an EPM $\mathcal{E}$ such that, for any $F_1(x)$, …, $F_n(x)$, $E(x)$, $G(x)$, $\mathcal{E} \Vdash \bigsqcap x\big(F_1(x) \wedge \ldots \wedge F_n(x) \wedge E(x) \rightarrow G(x)\big) \rightarrow \big(\bigsqcap x F_1(x) \wedge \ldots \wedge \bigsqcap x F_n(x) \wedge \bigsqcup x E(x) \rightarrow \bigsqcup x G(x)\big)$.*

Proof. Strategy for n: Wait till the environment makes a move c in the $\bigsqcup x E^*(x)$ component. Then make the same move c in $\bigsqcup x G^*(x)$, $\bigsqcap x\big(F_1^*(x) \wedge \ldots \wedge F_n^*(x) \wedge E^*(x) \rightarrow G^*(x)\big)$ and each of the $\bigsqcap x F_i^*(x)$ components. After this series of moves the game will have evolved to $\big(F_1^*(c) \wedge \ldots \wedge F_n^*(c) \wedge E^*(c) \rightarrow G^*(c)\big) \rightarrow \big(F_1^*(c) \wedge \ldots \wedge F_n^*(c) \wedge E^*(c) \rightarrow G^*(c)\big)$. Now switch to $\mathcal{CCS}$. □

Lemma 31. *Assume t is free for x in $F(x)$. Then $\vdash F(t) \to \bigsqcup xF(x)$. Furthermore, there is an effective function that takes any t and constructs an EPM $\mathcal{E}$ such that, for any $F(x)$, whenever t is free for x in $F(x)$, $\mathcal{E}\vdash F(t) \to \bigsqcup xF(x)$.*

Proof. Strategy: Let $c = e(t)$. Read c from the valuation tape if necessary. Then make the move '2.c', bringing the game down to $F^*(c) \to F^*(c)$. Then switch to $\mathcal{CCS}$. □

Lemma 32. *Assume F does not contain x. Then $\vdash F \to \bigsqcap xF$. Furthermore, there is an EPM $\mathcal{E}$ such that, for any F and x, as long as F does not contain x, $\mathcal{E}\vdash F \to \bigsqcap xF$.*

Proof. In this case we prefer to explicitly write the usually suppressed parameter e. Consider an arbitrary F not containing x, and an arbitrary interpretation * admissible for $F \to \bigsqcap xF$. The formula $F \to \bigsqcap xF$ contains x yet F does not. From the definition of admissibility and with a little thought we can see that F^* does not depend on x. In turn, this means — as can be seen with some thought — that the move c by the environment (whatever constant c) in $e[\bigsqcap xF^*]$ brings this game down to $e[F^*]$. With this observation in mind, the following strategy can be seen to be successful: Wait till the environment makes the move '2.c' for some constant c. Then read the sequence '1.α_1', ..., '1.α_n' of (legal) moves possibly made by the environment before it made the move '2.c', and make the n moves '2.α_1', ..., '2.α_n'. It can be seen that now the original game $e[F^*] \to e[\bigsqcap xF^*]$ will have been brought down to $\langle\Phi\rangle e[F^*] \to \langle\Phi\rangle e[F^*]$, where $\Phi = \langle \top\alpha_1, \ldots, \top\alpha_n\rangle$. So, switching to $\mathcal{CCS}$ at this point guarantees success. □

Lemma 33. *Assume $F(x)$ does not contain y. Then $\vdash \bigsqcap yF(y) \to \bigsqcap xF(x)$ and $\vdash \bigsqcup xF(x) \to \bigsqcup yF(y)$. In fact, $\mathcal{CCS}\vdash \bigsqcap yF(y) \to \bigsqcap xF(x)$ and $\mathcal{CCS}\vdash \bigsqcup xF(x) \to \bigsqcup yF(y)$.*

Proof. Assuming that $F(x)$ does not contain y and analyzing the relevant definitions, it is not hard to see that, for any interpretation * admissible for $\bigsqcap yF(y) \to \bigsqcap xF(x)$ and/or $\bigsqcup xF(x) \to \bigsqcup yF(y)$, we simply have $\big(\bigsqcap yF(y)\big)^* = \big(\bigsqcap xF(x)\big)^*$ and $\big(\bigsqcup yF(y)\big)^* = \big(\bigsqcup xF(x)\big)^*$. So, in both cases we deal with a game of the form $A \to A$, for which the ordinary copy-cat strategy is successful. □

Our proof of the following lemma is fairly long, for which reason it is given separately in Section 11:

Lemma 34. *$\vdash {\downarrow}{\circ}F \to {\circ}{\downarrow}{\downarrow}F$. Furthermore, there is an EPM $\mathcal{E}$ such that, for any F, $\mathcal{E}\vdash {\downarrow}{\circ}F \to {\circ}{\downarrow}{\downarrow}F$.*

11 Proof of Lemma 34

Roughly speaking, the uniform solution $\mathcal{E}$ for ${\downarrow}{\circ}F \to {\circ}{\downarrow}{\downarrow}F$ that we are going to construct essentially uses a copy-cat strategy between the antecedent and the consequent. However, this strategy cannot be applied directly in the form of our kind

old friend $\mathcal{CCS}$. The problem is that the underlying tree-structure (see Remark 9) of the multiset of (legal) runs of F^* that is being generated in the antecedent should be a simple tree T, while in the consequent it is in fact what can be called a tree T'' of trees. The trick that $\mathcal{E}$ uses is that it sees each edge of T in one of two — blue or yellow — colors. This allows $\mathcal{E}$ to associate with each branch y of a branch x of T'' a single branch z of T, and vice versa. Specifically, with x, y, z being (encoded by) bit strings, z is such that the subsequence of its blue-colored bits (=edges) coincides with x, and the subsequence of its yellow-colored bits coincides with y. By appropriately "translating" and "copying" in the antecedent the replicative moves made by the environment in the consequent, such an isomorphism between the branches of T and the branches of branches of T'' can be successfully maintained throughout the play. With this one-to-one correspondence in mind, every time the environment makes a (nonreplicative) move in the position(s) of F^* represented by a leaf or a set of leaves of T, the machine repeats the same move in the position(s) represented by the corresponding leaf-of-leaf or leaves-of-leaves of T'', and vice versa. The effect achieved by this strategy is that the multisets of all runs of F^* in the antecedent and in the consequent of $\mathop{\downarrow}\limits_{\circ} F^* \rightarrow \mathop{\downarrow}\limits_{\circ}\mathop{\downarrow}\limits_{\circ} F^*$ are identical, which, of course, guarantees a win for $\mathcal{E}$.

Let us now define things more precisely. A **colored bit** b is a pair (c, d), where c, called the **content** of b, is in $\{0, 1\}$, and d, called the **color** of b, is in $\{$*blue,yellow*$\}$. We will be using the notation $\overline{c}$ ("blue c") for the colored bit whose content is c and color is *blue*, and $\underline{c}$ ("yellow c") for the bit whose content is c and color is *yellow*.

A **colored bit string** is a finite or infinite sequence of colored bits. Consider a colored bit string v. The **content** of v is the result of "ignoring the colors" in v, i.e. replacing every bit of v by the content of that bit. The **blue content** of v is the content of the string that results from deleting in v all but blue bits. **Yellow content** is defined similarly. We use $\overline{v}$, $\overline{v}$ and $\underline{v}$ to denote the content, blue content and yellow content of v, respectively. Example: if $v = \overline{1}\underline{00}\overline{0}\underline{1}$, we have $\underline{v} = 10001$, $\overline{v} = 10$ and $\underline{v} = 001$. As in the case of ordinary bit strings, ϵ stands for the empty colored bit string. And, for colored bit strings w and u, $w \preceq u$ again means that w is a (not necessarily proper) initial segment of u.

Definition 35. *A* **colored tree** *is a set T of colored bit strings, called its* **branches***, such that the following conditions are satisfied:*

a) The set $\{\overline{v} \mid v \in T\}$ — that we denote by $\overline{T}$ — is a tree in the sense of Convention 5.

b) For any $w, u \in T$, if $\overline{w} = \overline{u}$, then $w = u$.

c) For no $v \in T$ do we have $\{v\overline{0}, v\underline{1}\} \subseteq T$ or $\{v\underline{0}, v\overline{1}\} \subseteq T$.

A branch v of T is said to be a **leaf** *iff $\overline{v}$ is a leaf of $\overline{T}$.*

When represented in the style of Figure 1 of [7] (page 36), a colored tree will look like an ordinary tree, with the only difference that now every edge will have one of the colors *blue* or *yellow*. Also, by condition (c), both of the outgoing edges ("sibling" edges) of any non-leaf node will have the same color.

Lemma 36. *Assume T is a colored tree, and w, u are branches of T with $\overline{w} \preceq \overline{u}$ and $\underline{w} \preceq \underline{u}$. Then $w \preceq u$.*

Proof. Assume T is a colored tree, $w, u \in T$, and $w \not\preceq u$. We want to show that then $\overline{w} \not\preceq \overline{u}$ or $\underline{w} \not\preceq \underline{u}$. Let v be the longest common initial segment of w and u, so we have $w = vw'$ and $u = vu'$ for some w', u' such that w' is nonempty and w' and u' do not have a nonempty common initial segment. Assume the first bit of w' is $\overline{0}$ (the cases when it is $\overline{1}$, $\underline{0}$ or $\underline{1}$, of course, will be similar). If u' is empty, then w obviously contains more blue bits than u does, and we are done. Assume now u' is nonempty, in particular, b is the first bit of u'. Since w' and u' do not have a nonempty common initial segment, b should be different from $\overline{0}$. By condition (b) of Definition 35, the content of b cannot be 0 (for otherwise we would have $v\overline{0} = vb$ and hence $b = \overline{0}$). Consequently, b is either $\overline{1}$ or $\underline{1}$. The case $b = \underline{1}$ is ruled out by condition (c) of Definition 35. Thus, $b = \overline{1}$. But the blue content of $v\overline{0}$ is $\overline{v}0$ while the blue content of $v\overline{1}$ is $\overline{v}1$. Taking into account the obvious fact that the former is an initial segment of $\overline{w}$ and the latter is an initial segment of $\overline{u}$, we find $\overline{w} \not\preceq \overline{u}$. □

Now comes a description of our EPM $\mathcal{E}$. At the beginning, this machine creates a record T of the type 'finite colored tree', and initializes it to $\{\epsilon\}$. After that, $\mathcal{E}$ follows the following procedure:

Procedure LOOP: Keep granting permission until the adversary makes a move γ. If γ satisfies the conditions of one of the following four cases, act as the corresponding case prescribes. Otherwise go to an infinite loop in a permission state.
Case (i): $\gamma = 2.w$: for some bit string w. Let $v_1, \ldots, v_k$ be[11] all of the leaves v of T with $w = \overline{v}$. Then make the moves $1.\overline{v}_1:, \ldots, 1.\overline{v}_k:$, update T to $T \cup \{v_1\overline{0}, v_1\overline{1}, \ldots, v_k\overline{0}, v_k\overline{1}\}$, and repeat LOOP.
Case (ii): $\gamma = 2.w.u$: for some bit strings w, u. Let $v_1, \ldots, v_k$ be all of the leaves v of T such that $w \preceq \overline{v}$ and $u = \underline{v}$. Then make the moves $1.\overline{v}_1:, \ldots, 1.\overline{v}_k:$, update T to $T \cup \{v_1\underline{0}, v_1\underline{1}, \ldots, v_k\underline{0}, v_k\underline{1}\}$, and repeat LOOP.
Case (iii): $\gamma = 2.w.u.\alpha$ for some bits strings w, u and move α. Let $v_1, \ldots, v_k$ be all of the leaves v of T such that $w \preceq \overline{v}$ and $u \preceq \underline{v}$. Then make the moves $1.\overline{v}_1.\alpha, \ldots, 1.\overline{v}_k.\alpha$, and repeat LOOP.
Case (iv): $\gamma = 1.w.\alpha$ for some bit string w and move α. Let $v_1, \ldots, v_k$ be all of the leaves v of T with $w \preceq \underline{v}$. Then make the moves $2.\overline{v}_1.\underline{v}_1.\alpha, \ldots, 2.\overline{v}_k.\underline{v}_k.\alpha$, and repeat LOOP.

Fix any interpretation *, valuation e and e-computation branch B of $\mathcal{E}$. Let Θ be the run spelled by B. From the above description it is immediately clear that B is a fair. Hence, in order to show that $\mathcal{E}$ wins, it would be sufficient to show that $\mathbf{Wn}_e^{\mathfrak{b}F^* \to \mathfrak{b}\mathfrak{b}F^*}\langle\Theta\rangle = \top$. Notice that the work of $\mathcal{E}$ does not depend on e. And, as e is fixed, we can safely and unambiguously omit this parameter (as we often did in the previous section) in expressions such as $e[A]$, $\mathbf{Lr}_e^A$ or $\mathbf{Wn}_e^A$ and just write

[11] In each of the four cases we assume that the list $v_1, \ldots, v_k$ is arranged lexicographically.

or say A, $\mathbf{Lr}^A$ or $\mathbf{Wn}^A$. Of course, $\mathcal{E}$ is interpretation-blind, so as long as it wins $\mathbin{⫰}F^* \rightarrow \mathbin{⫰}\mathbin{⫰}F^*$, it is a uniform solution for $\mathbin{⫰}F \rightarrow \mathbin{⫰}\mathbin{⫰}F$.

Let $N = \{1, \ldots, m\}$ if LOOP is iterated the finite number m of times in B, and $N = \{1, 2, 3, \ldots\}$ otherwise. For $i \in N$, we let T_i denote the value of record T at the beginning of the ith iteration of LOOP; Θ_i will mean the initial segment of Θ consisting of the moves made by the beginning of the ith iteration of LOOP. Finally, Φ_i will stand for $\neg\Theta_i^{1\cdot}$ and Ψ_i for $\Theta_i^{2\cdot}$.

From the description of LOOP it is immediately obvious that, for each $i \in N$, T_i is a finite colored tree, and that $T_1 \subseteq T_2 \subseteq \ldots \subseteq T_i$. In our subsequent reasoning we will implicitly rely on this fact.

Lemma 37. *For every i with $i \in N$, we have:*

a) Φ_i *is prelegal and* $\mathrm{Tree}\langle\Phi_i\rangle = \underline{\overline{T_i}}$.

b) Ψ_i *is prelegal.*

c) For every leaf x of $\mathrm{Tree}\langle\Psi_i\rangle$, $\Psi_i^{\preceq x}$ *is prelegal.*

d) For every leaf z of T_i, $\overline{z}$ is a leaf of $\mathrm{Tree}\langle\Psi_i\rangle$ *and $\underline{z}$ is a leaf of* $\mathrm{Tree}\langle\Psi_i^{\preceq\overline{z}}\rangle$.

e) For every leaf x of $\mathrm{Tree}\langle\Psi_i\rangle$ *and every leaf y of* $\mathrm{Tree}\langle\Psi_i^{\preceq x}\rangle$, *there is a leaf z of T_i such that $x = \overline{z}$ and $y = \underline{z}$. By Lemma 36, such a z is unique.*

f) For every leaf z of T_i, $\Phi_i^{\preceq\overline{z}} = (\Psi_i^{\preceq\overline{z}})^{\preceq\underline{z}}$.

g) Θ_i is a legal position of $\mathbin{⫰}F^* \rightarrow \mathbin{⫰}\mathbin{⫰}F^*$; hence, $\Phi_i \in \mathbf{Lr}^{⫰F^*}$ and $\Psi_i \in \mathbf{Lr}^{⫰⫰F^*}$.

Proof. We proceed by induction on i. The basis case with $i = 1$ is rather straightforward for each clause of the lemma and we do not discuss it. For the induction step, assume $i + 1 \in N$, and the seven clauses of the lemma are true for i.
Clause (a): By the induction hypothesis, Φ_i is prelegal and $\mathit{Tree}\langle\Phi_i\rangle = \underline{\overline{T}_i}$. Assume first that the ith iteration of LOOP deals with Case (i), so that $\Phi_{i+1} = \langle\Phi_i, \bot\underline{\overline{v}}_1{:}, \ldots, \bot\underline{\overline{v}}_k{:}\rangle$. Each of $\underline{\overline{v}}_1, \ldots, \underline{\overline{v}}_k$ is a leaf of $\underline{\overline{T}}_i$, i.e. a leaf of $\mathit{Tree}\langle\Phi_i\rangle$. This guarantees that Φ_{i+1} is prelegal. Also, by the definition of function *Tree*, we have $\mathit{Tree}\langle\Phi_{i+1}\rangle = \mathit{Tree}\langle\Phi_i\rangle \cup \{\underline{\overline{v}}_1 0, \underline{\overline{v}}_1 1, \ldots, \underline{\overline{v}}_k 0, \underline{\overline{v}}_k 1\}$. But the latter is nothing but $\underline{\overline{T}}_{i+1}$ as can be seen from the description of how Case (i) updates T_i to T_{i+1}. A similar argument applies when the ith iteration of LOOP deals with Case (ii). Assume now the ith iteration of LOOP deals with Case (iii). Note that the moves made in the antecedent of $\mathbin{⫰}F^* \rightarrow \mathbin{⫰}\mathbin{⫰}F^*$ (the moves that bring Φ_i to Φ_{i+1}) are nonreplicative — specifically, look like $\underline{\overline{v}}.\alpha$ where $\underline{\overline{v}} \in \underline{\overline{T}}_i = \mathit{Tree}\langle\Phi_i\rangle$. Such moves yield prelegal positions and do not change the value of *Tree*, so that $\mathit{Tree}\langle\Phi_i\rangle = \mathit{Tree}\langle\Phi_{i+1}\rangle$. It remains to note that T is not updated in this subcase, so that we also have $\underline{\overline{T}}_{i+1} = \underline{\overline{T}}_i$. Hence $\mathit{Tree}\langle\Phi_{i+1}\rangle = \underline{\overline{T}}_{i+1}$. Finally, suppose the ith iteration of LOOP deals with Case (iv). It is the environment who moves in the antecedent of $\mathbin{⫰}F^* \rightarrow \mathbin{⫰}\mathbin{⫰}F^*$, and does so before the machine makes any moves. Then the clean environment assumption — in conjunction with the induction hypothesis — implies that such a move cannot bring Φ_i to an illegal position of $\mathbin{⫰}F^*$ and hence cannot bring it to a non-prelegal position. So, Φ_{i+1} is prelegal. As for $\mathit{Tree}\langle\Phi_{i+1}\rangle = \underline{\overline{T}}_{i+1}$, it holds for the same reason as in the previous case.

Clause (b): If the ith iteration of LOOP deals with Case (i), (ii) or (iii), it is the environment who moves in the consequent of $\mathbin{⫰}F^* \to \mathbin{⫰}\mathbin{⫰}F^*$, and the clean environment assumption guarantees that Ψ_{i+1} is prelegal. Assume now that the ith iteration of LOOP deals with Case (iv), so that $\Psi_{i+1} = \langle \Psi_i, \top\overline{v}_1.\underline{v}_1.\alpha, \ldots, \top\overline{v}_k.\underline{v}_k.\alpha\rangle$. By the induction hypothesis for clause (d), each $\overline{v}_j$ $(1 \le j \le k)$ is a leaf of $\mathit{Tree}\langle\Psi_i\rangle$, so adding the moves $\top\overline{v}_1.\underline{v}_1.\alpha, \ldots, \top\overline{v}_k.\underline{v}_k$ does not bring Ψ_i to a non-prelegal position, nor does it modify $\mathit{Tree}\langle\Psi_i\rangle$ because the moves are nonreplicative. Hence Ψ_{i+1} is prelegal.

Clause (c): Just as in the previous clause, when the ith iteration of LOOP deals with Case (i), (ii) or (iii), the desired conclusion follows from the clean environment assumption. Assume now that the ith iteration of LOOP deals with Case (iv). Consider any leaf x of $\mathit{Tree}\langle\Psi_{i+1}\rangle$. As noted when discussing Case (iv) in the proof of Clause (b), $\mathit{Tree}\langle\Psi_i\rangle = \mathit{Tree}\langle\Psi_{i+1}\rangle$, so x is also a leaf of $\mathit{Tree}\langle\Psi_i\rangle$. Therefore, if $\Psi_{i+1}^{\preceq x} = \Psi_i^{\preceq x}$, the conclusion that $\Psi_{i+1}^{\preceq x}$ is prelegal follows from the induction hypothesis. Suppose now $\Psi_{i+1}^{\preceq x} \ne \Psi_i^{\preceq x}$. Note that then $\Psi_{i+1}^{\preceq x}$ looks like $\langle \Psi_i^{\preceq x}, \top y_1.\alpha, \ldots, \top y_m.\alpha\rangle$, where for each y_j $(1 \le j \le m)$ we have $\overline{z} = x$ and $\underline{z} = y_j$ for some leaf z of T_i. By the induction hypothesis for clause (d), each such y_j is a leaf of $\mathit{Tree}\langle\Psi_i^{\preceq x}\rangle$. By the induction hypothesis for the present clause, $\Psi_i^{\preceq x}$ is prelegal. Adding to such a position the nonreplicative moves $\top y_1.\alpha, \ldots, \top y_m.\alpha$ — where the y_j are leaves of $\mathit{Tree}\langle\Psi_i^{\preceq x}\rangle$ — cannot bring it to a non-prelegal position. Thus, $\Psi_{i+1}^{\preceq x}$ remains prelegal.

Clauses (d) and (e): If the ith iteration of LOOP deals with Cases (iii) or (iv), T_i is not modified, and no moves of the form x: or $x.y$: (where x, y are bit strings) are made in the consequent of $\mathbin{⫰}F^* \to \mathbin{⫰}\mathbin{⫰}F^*$, so $\mathit{Tree}\langle\Psi_i\rangle$ and $\mathit{Tree}\langle\Psi_i^{\preceq x}\rangle$ (any leaf x of $\mathit{Tree}\langle\Psi_i\rangle$) are not affected, either. Hence Clauses (d) and (e) for $i+1$ are automatically inherited from the induction hypothesis for these clauses. This inheritance also takes place — even if no longer "automatically" — when the ith iteration of LOOP deals with Case (i) or (ii). This can be verified by a routine analysis of how Cases (i) and (ii) modify T_i and the other relevant parameters. Details are left to the reader.

Clause (f): Consider any leaf z of T_{i+1}. When the ith iteration of LOOP deals with Case (i) or (ii), no moves of the form $x.\alpha$ are made in the antecedent of $\mathbin{⫰}F^* \to \mathbin{⫰}\mathbin{⫰}F^*$, and no moves of the form $x.y.\alpha$ are made in the consequent (any bit strings x, y). Based on this, it is easy to see that for all bit strings x, y — including the case $x = \overline{z}$ and $y = \underline{z}$ — we have $\Phi_{i+1}^{\preceq x} = \Phi_i^{\preceq x}$ and $(\Psi_{i+1}^{\preceq x})^{\preceq y} = (\Psi_i^{\preceq x})^{\preceq y}$. Hence clause (f) for $i+1$ is inherited from the same clause for i. Now suppose the ith iteration of LOOP deals with Case (iii). Then $T_{i+1} = T_i$ and hence z is also a leaf of T_i. From the description of Case (iii) one can easily see that if $w \not\preceq \overline{z}$ or $u \not\preceq \underline{z}$, we have $\Phi_{i+1}^{\preceq\overline{z}} = \Phi_i^{\preceq\overline{z}}$ and $(\Psi_{i+1}^{\preceq\overline{z}})^{\preceq\underline{z}} = (\Psi_i^{\preceq\overline{z}})^{\preceq\underline{z}}$, so the equation $\Phi_{i+1}^{\preceq\overline{z}} = (\Psi_{i+1}^{\preceq\overline{z}})^{\preceq\underline{z}}$ is true by the induction hypothesis; and if $w \preceq \overline{z}$ and $u \preceq \underline{z}$, then $\Phi_{i+1}^{\preceq\overline{z}} = \langle \Phi_i^{\preceq\overline{z}}, \bot\alpha\rangle$ and $(\Psi_{i+1}^{\preceq\overline{z}})^{\preceq\underline{z}} = \langle (\Psi_i^{\preceq\overline{z}})^{\preceq\underline{z}}, \bot\alpha\rangle$. But, by the induction hypothesis, $\Phi_i^{\preceq\overline{z}} = (\Psi_i^{\preceq\overline{z}})^{\preceq\underline{z}}$. Hence $\Phi_{i+1}^{\preceq\overline{z}} = (\Psi_{i+1}^{\preceq\overline{z}})^{\preceq\underline{z}}$. A similar argument applies when the ith iteration of LOOP deals with Case (iv).

Clause (g): Note that all of the moves made in any of Cases (i)-(iv) of LOOP have the prefix '1.' or '2.', i.e. are made either in the antecedent or the consequent of $\luk F^* \to \luk\luk F^*$. Hence, in order to show that Θ_{i+1} is a legal position of $\luk F^* \to \luk\luk F^*$, it would suffice to verify that $\Phi_{i+1} \in \mathbf{Lr}^{\luk F^*}$ and $\Psi_{i+1} \in \mathbf{Lr}^{\luk\luk F^*}$.

Suppose the ith iteration of LOOP deals with Case (i) or (ii). The clean environment assumption guarantees that $\Psi_{i+1} \in \mathbf{Lr}^{\luk\luk F^*}$. In the antecedent of $\luk F^* \to \luk\luk F^*$ only replicative moves are made. Replicative moves can yield an illegal position (Φ_{i+1} in our case) of a $\luk$-game only if they yield a non-prelegal position. But, by clause (a), Φ_{i+1} is prelegal. Hence it is a legal position of $\luk F^*$.

Suppose now the ith iteration of LOOP deals with Case (iii). Again, that $\Psi_{i+1} \in \mathbf{Lr}^{\luk\luk F^*}$ is guaranteed by the clean environment assumption. So, we only need to verify that $\Phi_{i+1} \in \mathbf{Lr}^{\luk F^*}$. By clause (a), this position is prelegal. So, it remains to see that, for any leaf y of $\mathit{Tree}\langle\Phi_{i+1}\rangle$, $\Phi_{i+1}^{\preceq z} \in \mathbf{Lr}^{\luk F^*}$. Pick an arbitrary leaf y of $\mathit{Tree}\langle\Phi_{i+1}\rangle$ — i.e., by clause (a), of $\overline{T}_{i+1}$. Let z be the leaf of T_{i+1} with $y = \overline{z}$. We already know that $\Psi_{i+1} \in \mathbf{Lr}^{\luk\luk F^*}$. By clause (d), we also know that $\overline{z}$ is a leaf of $\mathit{Tree}\langle\Psi_{i+1}\rangle$. Consequently, $\Psi_{i+1}^{\preceq\overline{z}} \in \mathbf{Lr}^{\luk F^*}$. Again by clause (d), $\underline{z}$ is a leaf of $\mathit{Tree}\langle\Psi_{i+1}^{\preceq\overline{z}}\rangle$. Hence, $(\Psi_{i+1}^{\preceq\overline{z}})^{\preceq\underline{z}}$ should be a legal position of F^*. But, by clause (f), $\Phi_{i+1}^{\preceq\overline{z}} = (\Psi_{i+1}^{\preceq\overline{z}})^{\preceq\underline{z}}$. Thus, $\Phi_{i+1}^{\preceq y} \in \mathbf{Lr}^{F^*}$.

Finally, suppose the ith iteration of LOOP deals with Case (iv). By the clean environment assumption, $\Phi_{i+1} \in \mathbf{Lr}^{\luk F^*}$. Now consider Ψ_{i+1}. This position is prelegal by clause (b). So, in order for Ψ_{i+1} to be a legal position of $\luk\luk F^*$, for every leaf x of $\mathit{Tree}\langle\Psi_{i+1}\rangle$ we should have $\Psi_{i+1}^{\preceq x} \in \mathbf{Lr}^{\luk F^*}$. Consider an arbitrary such leaf x. By clause (c), $\Psi_{i+1}^{\preceq x}$ is prelegal. Hence, a sufficient condition for $\Psi_{i+1}^{\preceq x} \in \mathbf{Lr}^{\luk F^*}$ is that, for every leaf y of $\mathit{Tree}\langle\Psi_{i+1}^{\preceq x}\rangle$, $(\Psi_{i+1}^{\preceq x})^{\preceq y} \in \mathbf{Lr}^{F^*}$. So, let y be an arbitrary such leaf. By clause (e), there is a leaf z of T_{i+1} such that $\overline{z} = x$ and $\underline{z} = y$. Therefore, by clause (f), $\Phi_{i+1}^{\preceq\overline{z}} = (\Psi_{i+1}^{\preceq x})^{\preceq y}$. But we know that $\Phi_{i+1} \in \mathbf{Lr}^{\luk F^*}$ and hence (with clause (a) in mind) $\Phi_{i+1}^{\preceq\overline{z}} \in \mathbf{Lr}^{F^*}$. Consequently, $(\Psi_{i+1}^{\preceq x})^{\preceq y} \in \mathbf{Lr}^{F^*}$. □

Lemma 38. *For every finite initial segment Υ of Θ, there is $i \in N$ such that Υ is a (not necessarily proper) initial segment of Θ_i and hence of every Θ_j with $i \leq j \in N$.*

Proof. The statement of the lemma is straightforward when there are infinitely many iterations of LOOP, for each iteration adds a nonzero number of new moves to the run and hence there are arbitrarily long Θ_is, each of them being an initial segment of Θ. Suppose now LOOP is iterated a finite number m of times. It would be (necessary and) sufficient to verify that in this case $\Theta = \Theta_m$, i.e. no moves are made during the last iteration of LOOP. But this is indeed so. From the description of LOOP we see that the machine does not make any moves during a given iteration unless the environment makes a move γ first. So, assume $\bot$ makes move γ during the mth iteration of LOOP. By the clean environment assumption, we should have $\langle\Theta_m, \bot\gamma\rangle \in \mathbf{Lr}^{\luk F^* \to \luk\luk F^*}$. It is easy to see that such a γ would have to satisfy the conditions of one of the Cases (i)-(iv) of LOOP. But then there would

be an $(m+1)$th iteration of LOOP, contradicting out assumption that there are only m iterations. □

Let us use Φ and Ψ to denote $\neg\Theta^{1\cdot}$ and $\Theta^{2\cdot}$, respectively. Of course, the statement of Lemma 38 is true for Φ and Ψ (instead of Θ) as well. Taking into account that, by definition, a given run is legal if all of its finite initial segments are legal, the following fact is an immediate corollary of Lemmas 38 and 37(g):

$$\Theta \in \mathbf{Lr}^{\mathsf{!}\circ F^*\to\mathsf{!}\circ\mathsf{!}\circ F^*}.\ \textit{Hence,}\ \Phi \in \mathbf{Lr}^{\mathsf{!}\circ F^*}\ \textit{and}\ \Psi \in \mathbf{Lr}^{\mathsf{!}\circ\mathsf{!}\circ F^*}. \tag{6}$$

To complete our proof of Lemma 34, we need to show that $\mathbf{Wn}^{\mathsf{!}\circ F^*\to\mathsf{!}\circ\mathsf{!}\circ F^*}\langle\Theta\rangle = \top$. With (6) in mind, if $\mathbf{Wn}^{\mathsf{!}\circ\mathsf{!}\circ F^*}\langle\Psi\rangle = \top$, we are done. Assume now $\mathbf{Wn}^{\mathsf{!}\circ\mathsf{!}\circ F^*}\langle\Psi\rangle = \bot$. Then, by the definition of $\mathsf{!}\circ$, there is an infinite bit string x such that $\Psi^{\preceq x}$ is a legal but lost (by $\top$) run of $\mathsf{!}\circ F^*$. This means that, for some infinite bit string y,

$$\mathbf{Wn}^{F^*}\langle(\Psi^{\preceq x})^{\preceq y}\rangle = \bot. \tag{7}$$

Fix these x and y. For each $i \in N$, let x_i denote the (obviously unique) leaf of $\mathit{Tree}\langle\Psi_i\rangle$ such that $x_i \preceq x$; and let y_i denote the (again unique) leaf of $\mathit{Tree}\langle\Psi_{\overline{i}}^{\preceq x_i}\rangle$ such that $y_i \preceq y$. Next, let z_i denote the leaf of T_i with $\overline{z}_i = x_i$ and $\underline{z}_i = y_i$. According to Lemma 37(e), such a z_i exists and is unique.

Consider any i with $i+1 \in N$. Clearly $x_i \preceq x_{i+1}$ and $y_i \preceq y_{i+1}$. By our choice of the z_j, we then have $\overline{z}_i \preceq \overline{z}_{i+1}$ and $\underline{z}_i \preceq \underline{z}_{i+1}$. Hence, by Lemma 36, $z_i \preceq z_{i+1}$. Let us fix an infinite bit string z such that for every $i \in N$, $\overline{z}_i \preceq z$. Based on the just-made observation that we always have $z_i \preceq z_{i+1}$, such a z exists.

In view of Lemma 38, Lemma 37(f) easily allows us to find that $\Phi^{\preceq z} = (\Psi^{\preceq x})^{\preceq y}$. Therefore, by (7), $\mathbf{Wn}^{F^*}\langle\Phi^{\preceq z}\rangle = \bot$. By the definition of $\mathsf{!}\circ$, this means that $\mathbf{Wn}^{\mathsf{!}\circ F^*}\langle\Phi\rangle = \bot$. Hence, by the definition of $\to$ and with (6) in mind, $\mathbf{Wn}^{\mathsf{!}\circ F^*\to\mathsf{!}\circ\mathsf{!}\circ F^*}\langle\Theta\rangle = \top$. Done. □

12 Proof of Theorem 12

Now we are ready to prove our main Theorem 12. Consider an arbitrary sequent S with $\mathbf{INT} \vdash S$. By induction on the **INT**-derivation of S, we are going to show that S has a uniform solution $\mathcal{E}$. This is sufficient to conclude that **INT** is 'uniformly sound'. The theorem also claims 'constructive soundness', i.e. that such an $\mathcal{E}$ can be effectively built from a given **INT**-derivation of S. This claim of the theorem will be automatically taken care of by the fact that our proof of the existence of $\mathcal{E}$ is constructive: the uniform-validity and closure lemmas on which we rely provide a way for actually constructing a corresponding uniform solution. With this remark in mind and for the considerations of readability, in what follows we only talk about uniform validity without explicitly mentioning uniform solutions for the corresponding formulas/sequents and without explicitly showing how to construct such solutions. Also, we no longer use $\Rightarrow$ or $\circ\!\!-$, seeing each sequent $\underline{F} \Rightarrow K$ as the formula $\underline{\mathsf{!}\circ F} \to K$ and each subformula $E_1 \circ\!\!- E_2$ of such a formula as $\mathsf{!}\circ E_1 \to E_2$.

This is perfectly legitimate because, by definition, $(\underline{F} \Rightarrow K)^* = (\underline{\circ}F \rightarrow K)^*$ and $(E_1 \circ\!\!- E_2)^* = (\circ E_1 \rightarrow E_2)^*$.

There are 15 cases to consider, corresponding to the 15 possible rules that might have been used at the last step of an **INT**-derivation of S, with S being the conclusion of the rule. In each non-axiom case below, "induction hypothesis" means the assumption that the premise(s) of the corresponding rule is (are) uniformly valid. The goal in each case is to show that the conclusion of the rule is also uniformly valid. "Modus ponens" should be understood as Lemma 21, and "transitivity" as Lemma 22.

Identity: Immediately from Lemma 23.

Domination: Immediately from Lemma 27.

Exchange: By the induction hypothesis, $\Vdash \underline{\circ}G \wedge \circ E \wedge \circ F \wedge \underline{\circ}H \rightarrow K$. And, by Lemma 17(a), $\Vdash (\underline{\circ}G \wedge \circ E \wedge \circ F \wedge \underline{\circ}H \rightarrow K) \rightarrow (\underline{\circ}G \wedge \circ F \wedge \circ E \wedge \underline{\circ}H \rightarrow K)$. Applying modus ponens yields $\Vdash \underline{\circ}G \wedge \circ F \wedge \circ E \wedge \underline{\circ}H \rightarrow K$.

Weakening: Similar to the previous case, using Lemma 17(b) instead of 17(a).

Contraction: By Lemma 17(c) (with empty $\underline{U}$), $\Vdash (\circ F \rightarrow \circ F \wedge \circ F) \rightarrow (\underline{\circ}G \wedge \circ F \rightarrow \underline{\circ}G \wedge \circ F \wedge \circ F)$. And, by Lemma 26, $\Vdash \circ F \rightarrow \circ F \wedge \circ F$. Hence, by modus ponens, $\Vdash \underline{\circ}G \wedge \circ F \rightarrow \underline{\circ}G \wedge \circ F \wedge \circ F$. But, by the induction hypothesis, $\Vdash \underline{\circ}G \wedge \circ F \wedge \circ F \rightarrow K$. Hence, by transitivity, $\Vdash \underline{\circ}G \wedge \circ F \rightarrow K$.

Right $\circ\!\!-$: From Lemma 17(d), $\Vdash \big(\underline{\circ}G \wedge \circ F \rightarrow K\big) \rightarrow \big(\underline{\circ}G \rightarrow (\circ F \rightarrow K)\big)$. And, by the induction hypothesis, $\Vdash \underline{\circ}G \wedge \circ F \rightarrow K$. Applying modus ponens, we get $\Vdash \underline{\circ}G \rightarrow (\circ F \rightarrow K)$.

Left $\circ\!\!-$: By the induction hypothesis,

$$\Vdash \underline{\circ}G \wedge \circ F \rightarrow K_1; \tag{8}$$

$$\Vdash \underline{\circ}H \rightarrow K_2. \tag{9}$$

Our goal is to show that

$$\Vdash \underline{\circ}G \wedge \underline{\circ}H \wedge \circ(\circ K_2 \rightarrow F) \rightarrow K_1. \tag{10}$$

By Lemma 18, (9) implies $\Vdash \circ(\underline{\circ}H \rightarrow K_2)$. Also, by Lemma 24, $\Vdash \circ(\underline{\circ}H \rightarrow K_2) \rightarrow (\circ\underline{\circ}H \rightarrow \circ K_2)$. Applying modus ponens, we get $\Vdash \circ\underline{\circ}H \rightarrow \circ K_2$. Again using Lemma 18, we find $\Vdash \circ(\circ\underline{\circ}H \rightarrow \circ K_2)$, which, (again) by Lemma 24 and modus ponens, implies

$$\Vdash \circ\circ\underline{\circ}H \rightarrow \circ\circ K_2. \tag{11}$$

Combining Lemmas 17(c) (with empty $\underline{W}, \underline{U}$) and 34, by modus ponens, we find $\Vdash \underline{\circ}H \rightarrow \underline{\circ}\circ H$. Next, by lemma 25, $\Vdash \circ\underline{\circ}H \rightarrow \circ\underline{\circ}H$. Hence, by transitivity, $\Vdash \underline{\circ}H \rightarrow \circ\underline{\circ}H$. At the same time, by Lemma 34, $\Vdash \circ\underline{\circ}H \rightarrow \circ\circ\underline{\circ}H$. Again by

transitivity, $\Vdash \underline{\luebbert H} \to \luebbert\luebbert\underline{\luebbert H}$. This, together with (11), by transitivity, yields

$$\Vdash \underline{\luebbert H} \to \luebbert\luebbert K_2. \tag{12}$$

Next, by Lemma 24,

$$\Vdash \luebbert(\luebbert K_2 \to F) \to (\luebbert\luebbert K_2 \to \luebbert F). \tag{13}$$

From Lemma 17(e), $\Vdash \big(\luebbert(\luebbert K_2 \to F) \to (\luebbert\luebbert K_2 \to \luebbert F)\big) \wedge (\underline{\luebbert H} \to \luebbert\luebbert K_2) \to \big(\luebbert(\luebbert K_2 \to F) \to (\underline{\luebbert H} \to \luebbert F)\big)$. This, together with (13) and (12), by modus ponens, yields

$$\Vdash \luebbert(\luebbert K_2 \to F) \to (\underline{\luebbert H} \to \luebbert F). \tag{14}$$

By Lemma 17(f),

$$\begin{array}{l}\Vdash \big(\luebbert(\luebbert K_2 \to F) \to (\underline{\luebbert H} \to \luebbert F)\big) \wedge \big(\luebbert\underline{G} \wedge \luebbert F \to K_1\big)\\ \to \big(\underline{\luebbert G} \wedge \underline{\luebbert H} \wedge \luebbert(\luebbert K_2 \to F) \to K_1\big).\end{array} \tag{15}$$

From (14), (8) and (15), by modus ponens, we obtain the desired (10).

Right $\sqcap$: By the induction hypothesis, $\Vdash \underline{\luebbert G} \to K_1, \ldots, \Vdash \underline{\luebbert G} \to K_n$. And, from Lemma 17(h), $\Vdash (\underline{\luebbert G} \to K_1) \wedge \ldots \wedge (\underline{\luebbert G} \to K_n) \to (\underline{\luebbert G} \to K_1 \sqcap \ldots \sqcap K_n)$. Modus ponens yields $\Vdash \underline{\luebbert G} \to K_1 \sqcap \ldots \sqcap K_n$.

Left $\sqcap$: By Lemma 28(a), $\Vdash \luebbert(F_1 \sqcap \ldots \sqcap F_n) \to \luebbert F_i$; and, by Lemma 17(c), $\Vdash \big(\luebbert(F_1 \sqcap \ldots \sqcap F_n) \to \luebbert F_i\big) \to \big(\underline{\luebbert G} \wedge \luebbert(F_1 \sqcap \ldots \sqcap F_n) \to \underline{\luebbert G} \wedge \luebbert F_i\big)$. Modus ponens yields $\underline{\luebbert G} \wedge \luebbert(F_1 \sqcap \ldots \sqcap F_n) \to \underline{\luebbert G} \wedge \luebbert F_i$. But, by the induction hypothesis, $\Vdash \underline{\luebbert G} \wedge \luebbert F_i \to K$. So, by transitivity, $\Vdash \underline{\luebbert G} \wedge \luebbert(F_1 \sqcap \ldots \sqcap F_n) \to K$.

Right $\sqcup$: By the induction hypothesis, $\Vdash \underline{\luebbert G} \to K_i$. According to Lemma 17(j), $\Vdash (\underline{\luebbert G} \to K_i) \to (\underline{\luebbert G} \to K_1 \sqcup \ldots \sqcup K_n)$. Therefore, by modus ponens, $\Vdash \underline{\luebbert G} \to K_1 \sqcup \ldots \sqcup K_n$.

Left $\sqcup$: By the induction hypothesis, $\Vdash \underline{\luebbert G} \wedge \luebbert F_1 \to K, \ldots, \Vdash \underline{\luebbert G} \wedge \luebbert F_n \to K$. And, by Lemma 17(i), $\Vdash (\underline{\luebbert G} \wedge \luebbert F_1 \to K) \wedge \ldots \wedge (\underline{\luebbert G} \wedge \luebbert F_n \to K) \to \big(\underline{\luebbert G} \wedge (\luebbert F_1 \sqcup \ldots \sqcup \luebbert F_n) \to K\big)$. Hence, by modus ponens,

$$\Vdash \underline{\luebbert G} \wedge (\luebbert F_1 \sqcup \ldots \sqcup \luebbert F_n) \to K. \tag{16}$$

Next, by Lemma 17(c), $\Vdash \big(\luebbert(F_1 \sqcup \ldots \sqcup F_n) \to \luebbert F_1 \sqcup \ldots \sqcup \luebbert F_n\big) \to \big(\underline{\luebbert G} \wedge \luebbert(F_1 \sqcup \ldots \sqcup F_n) \to \underline{\luebbert G} \wedge (\luebbert F_1 \sqcup \ldots \sqcup \luebbert F_n)\big)$. But, by Lemma 28(c), $\Vdash \luebbert(F_1 \sqcup \ldots \sqcup F_n) \to \luebbert F_1 \sqcup \ldots \sqcup \luebbert F_n$. Modus ponens yields $\Vdash \underline{\luebbert G} \wedge \luebbert(F_1 \sqcup \ldots \sqcup F_n) \to \underline{\luebbert G} \wedge (\luebbert F_1 \sqcup \ldots \sqcup \luebbert F_n)$. From here and (16), by transitivity, $\Vdash \underline{\luebbert G} \wedge \luebbert(F_1 \sqcup \ldots \sqcup F_n) \to K$.

Right $\bigsqcap$: First, consider the case when $\underline{\luebbert G}$ is nonempty. By the induction hypothesis, $\Vdash \underline{\luebbert G} \to K(y)$. Therefore, by Lemma 19, $\Vdash \bigsqcap y\big(\underline{\luebbert G} \to K(y)\big)$ and, by Lemma 29 and modus ponens, $\Vdash \bigsqcap y \underline{\luebbert G} \to \bigsqcap y K(y)$. At the same time, by Lemma

32, $\vDash \underline{\downarrow\circ G} \rightarrow \bigsqcap y \underline{\downarrow\circ G}$. By transitivity, we then get $\vDash \underline{\downarrow\circ G} \rightarrow \bigsqcap y K(y)$. But, by Lemma 33, $\vDash \bigsqcap y K(y) \rightarrow \bigsqcap x K(x)$. Transitivity yields $\vDash \underline{\downarrow\circ G} \rightarrow \bigsqcap x K(x)$. The case when $\underline{\downarrow\circ G}$ is empty is simpler, for then $\vDash \underline{\downarrow\circ G} \rightarrow \bigsqcap x K(x)$, i.e. $\vDash \bigsqcap x K(x)$, can be obtained directly from the induction hypothesis by Lemmas 19, 33 and modus ponens.

Left $\bigsqcap$: Similar to Left $\sqcap$, only using Lemma 28(b) instead of 28(a).

Right $\bigsqcup$: By the induction hypothesis, $\vDash \underline{\downarrow\circ G} \rightarrow K(t)$. And, by Lemma 31, $\vDash K(t) \rightarrow \bigsqcup x K(x)$. Transitivity yields $\vDash \underline{\downarrow\circ G} \rightarrow \bigsqcup x K(x)$.

Left $\bigsqcup$: By the induction hypothesis, $\vDash \underline{\downarrow\circ G} \wedge \circ F(y) \rightarrow K$. This, by Lemma 19, implies $\vDash \bigsqcap y\big(\underline{\downarrow\circ G} \wedge \circ F(y) \rightarrow K\big)$. From here, by Lemma 30 and modus ponens, we get

$$\vDash \underline{\bigsqcap y \downarrow\circ G} \wedge \bigsqcup y \downarrow\circ F(y) \rightarrow \bigsqcup y K. \tag{17}$$

By Lemma 17(c), $\vDash (\underline{\downarrow\circ G \rightarrow \bigsqcap y \downarrow\circ G}) \rightarrow \big(\underline{\downarrow\circ G} \wedge \bigsqcup y \downarrow\circ F(y) \rightarrow \underline{\bigsqcap y \downarrow\circ G} \wedge \bigsqcup y \downarrow\circ F(y)\big)$. This, together with Lemma 32, by modus ponens, implies $\underline{\downarrow\circ G} \wedge \bigsqcup y \downarrow\circ F(y) \rightarrow \underline{\bigsqcap y \downarrow\circ G} \wedge \bigsqcup y \downarrow\circ F(y)$. From here and (17), by transitivity, $\vDash \underline{\downarrow\circ G} \wedge \bigsqcup y \downarrow\circ F(y) \rightarrow K$. But, by Lemmas 33, 17(c) and modus ponens, $\vDash \underline{\downarrow\circ G} \wedge \bigsqcup x \downarrow\circ F(x) \rightarrow \underline{\downarrow\circ G} \wedge \bigsqcup y \downarrow\circ F(y)$. Hence, by transitivity,

$$\vDash \underline{\downarrow\circ G} \wedge \bigsqcup x \downarrow\circ F(x) \rightarrow K. \tag{18}$$

Next, by Lemma 17(c), $\vDash \big(\downarrow\circ \bigsqcup x F(x) \rightarrow \bigsqcup x \downarrow\circ F(x)\big) \rightarrow \big(\underline{\downarrow\circ G} \wedge \downarrow\circ \bigsqcup x F(x) \rightarrow \underline{\downarrow\circ G} \wedge \bigsqcup x \downarrow\circ F(x)\big)$. But, by Lemma 28(d), $\vDash \downarrow\circ \bigsqcup x F(x) \rightarrow \bigsqcup x \downarrow\circ F(x)$. Modus ponens yields $\underline{\downarrow\circ G} \wedge \downarrow\circ \bigsqcup x F(x) \rightarrow \underline{\downarrow\circ G} \wedge \bigsqcup x \downarrow\circ F(x)$. From here and (18), by transitivity, $\vDash \underline{\downarrow\circ G} \wedge \downarrow\circ \bigsqcup x F(x) \rightarrow K$. □

References

[1] Blass, A. Degrees of indeterminacy of games. *Fundamenta Mathematicae*, 77:151–166, 1972.

[2] Blass, A. A game semantics for linear logic. *Annals of Pure and Applied Logic*, 56:183–220, 1992.

[3] Felscher, W. Dialogues, strategies, and intuitionistic provability. *Annals of Pure and Applied Logic*, 28: 217–254, 1885.

[4] J.Y. Girard, J. Y. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[5] Gödel, K. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. *Dialectica*, 12:280–287, 1958.

[6] Japaridze, G. A constructive game semantics for the language of linear logic. *Annals of Pure and Applied Logic*, 85:87–156, 1997.

[7] Japaridze, G. Introduction to computability logic. *Annals of Pure and Applied Logic*, 123:1–99, 2003.

[8] Japaridze, G. From truth to computability I. *Theoretical Computer Science*, 357:100–135, 2006.

[9] Japaridze, G. From truth to computability II. *Theoretical Computer Science*, 376:20–52, 2007.

[10] Japaridze, G. Computability logic: a formal theory of interaction. In Goldin, Dina, Smolka, Scott and Wegner, Peter, editors, *Interactive Computation: The New Paradigm*, pages 183–223. Springer, 2006.

[11] Japaridze, G. Propositional computability logic I. *ACM Transactions on Computational Logic*, 7:302–330, 2006.

[12] Japaridze, G. Propositional computability logic II. *ACM Transactions on Computational Logic*, 7:331–362, 2006.

[13] Kleene, S. C. *Introduction to Metamathematics*. D. van Nostrand Company, New York, Toronto, 1952.

[14] Kolmogorov, A. N. Zur Deutung der intuitionistischen Logik. *Mathematische Zeitschrift*, 35:58–65, 1932.

[15] Lorenzen, P. Ein dialogisches Konstruktivitätskriterium. In *Infinitistic Methods* (Proc. Symp. Foundations of Mathematics), pages 193–200, Warsaw, 1961. PWN.

[16] Medvedev, Y. Interpretation of logical formulas by means of finite problems and its relation to the realizability theory. *Soviet Mathematics Doklady*, 4:180-183, 1963.