arXiv:1007.1324v2 [cs.LO] 9 Jul 2010

Separating the basic logics of the basic recurrences

Giorgi Japaridze*

Abstract

This paper shows that, even at the most basic level (namely, in combination with only $\neg, \wedge, \vee$), the parallel, countable branching and uncountable branching recurrences of computability logic validate different principles.

MSC: primary: 03B47; secondary: 03B70; 68Q10; 68T27; 68T15.

Keywords: Computability logic; Game semantics; Recurrence operators

Contents

1 Introduction **1**

2 Recurrence operations: a quick review **2**

3 Parallel recurrence validates both production principles **4**

4 Branching recurrences do not validate short production **5**

5 Countable branching recurrence validates long production **10**

6 Uncountable branching recurrence does not validate long production **15**

1 Introduction

Computability logic (CoL) is a long-term project for redeveloping logic on the basis of a constructive game semantics. The approach induces a rich collection of logical operators, standing for various natural operations on games. Among those are *recurrence operators*, the most basic sorts of which are *parallel recurrence* $\curlywedge$, (uncountable) *branching recurrence* $\curlyvee$, and *countable branching recurrence* $\curlyvee^{\aleph_0}$. Each recurrence operator ! comes with its dual ?, defied by $?F = \neg!\neg F$. The present paper shows that the logical behaviors of these three sorts of recurrences are pairwise distinct — that is, they validate different principles — even at the most basic level, namely, in combination with only *negation* $\neg$, *parallel conjunction* $\wedge$, and *parallel disjunction* $\vee$ (as always, either $\vee$ or $\wedge$, being definable, can be dropped).

Showing validity or non-validity of various principles in CoL tends to be far from easy. This is especially so for principles involving recurrence operators. Recent years ([4]-[6],[8]-[13],[15]-[17]) have seen rapid and sustained progress in finding sound and complete axiomatizations for many, often quite expressive, fragments of CoL, at both the propositional and the first-order levels. Those fragments, however, have typically been recurrence-free.[1] So, it would be accurate to say that, at this point, practically nothing is known about the logical behavior of recurrences, and finding syntactic descriptions (such as axiomatizations) of the logics induced by them remains among the greatest challenges in the entire CoL enterprise. The present paper attempts to bring some initial light into this otherwise

*Supported by Summer Research Fellowship from Villanova University

[1]The so called intuitionistic fragment of CoL, studied in [10, 11, 17], is the only exception. There, however, the usage of recurrence ! is limited to the very special form/context $!E \rightarrow F$.

completely dark picture. Its results constitute a necessary first step on the presumably long road of syntactically taming recurrences: before even considering looking for axiomatizations, one needs to know whether to expect for those axiomatizations to be common or different for the various sorts of recurrences naturally emerging in game semantics.

The logics induced by the three recurrences turn out to be separated by the following two principles, which we call **short production** and **long production**, respectively:

$$P \wedge !(P \to P \wedge P) \ \to !P; \tag{1}$$

$$P \wedge !(P \to P \wedge P) \wedge !(P \vee P \to P) \ \to !P. \tag{2}$$

Namely, the situation is as shown in Figure 1, with **validity** throughout this paper understood as what CoL calls *uniform*, or *strong* validity.[2]

	when $! = \curlywedge$	when $! = \curlywedge^{\aleph_0}$	when $! = \curlywedge$
short production	*valid*	*invalid*	*invalid*
long production	*valid*	*valid*	*invalid*

Figure 1

This result is by no means obvious. One could have just as well expected that the differences between the three types of recurrences are too subtle to induce non-identical logics, at least at the $(\neg, \wedge, \vee, !)$-level. For instance, as shown in [10, 13], the implicative logic induced by all three recurrences $! \in \{\curlywedge, \curlywedge, \curlywedge^{\aleph_0}\}$, with implication $E \supset F$ understood as $!E \to F$, is exactly the implicative fragment of Heyting's intuitionistic calculus. The fact that the seemingly "almost the same" operators $\curlywedge$ and $\curlywedge^{\aleph_0}$ induce different logics is especially surprising.

The intended audience for this relatively short (by the standards of CoL) and technical paper is expected to be familiar with the main concepts of CoL, such as those of static games, hard- and easy-play machines, the operators $\neg, \wedge, \vee, \curlywedge, \curlyvee, \curlywedge, \curlywedge, \bigsqcap, \bigsqcup$ (as always, $A \to B$ is an abbreviation of $\neg A \vee B$), interpretation, validity, and the related notions. If not, it would be both necessary and sufficient to read the first ten sections of [14] for a self-contained, tutorial-style introduction. The definition of $\curlywedge$ given in [14] is a little bit long and, for that reason, this paper re-introduces this operation, together with its "countable" counterpart $\curlywedge^{\aleph_0}$, through a shorter definition. No other operations and concepts will be reintroduced and, again, they are to be understood as defined or explained in [14].

2 Recurrence operations: a quick review

Officially, (uncountable) branching recurrence $\curlywedge$ was first introduced in [3], parallel recurrence $\curlywedge$ in [7], and countable branching recurrence $\curlywedge^{\aleph_0}$ in [13]. $\curlywedge$ is the author's favorite, as it permits reusing its argument (as a resource) in the strongest algorithmic sense possible, thus allowing us to claim that the compound operation $\curlywedge A \to B$ captures our most general intuition of algorithmically reducing B to A. The weaker $\curlywedge$ stands out as the simplest sort of a recurrence. $\curlywedge^{\aleph_0}$, by its strength strictly between $\curlywedge$ and $\curlywedge$, is, in a sense, the strongest of all possible nontrivial weakenings of $\curlywedge$. Our interest in $\curlywedge^{\aleph_0}$ is partly also historical. It is related to the apparent fact that $\curlywedge^{\aleph_0}$ is "equivalent" to Blass's [2] *repetition*

[2]Extensionally, weak validity (in most earlier papers on CoL simply called *validity*) typically coincides with strong validity ([4]-[5],[8]-[12],[16]), but tends to be harder to deal with in completeness proofs, even though a way of turning completeness proofs with respect to strong validity into completeness proofs with respect to weak validity appears to be more or less standard. Also, in all applications, it is strong (uniform) validity that matters, with weak validity being of purely theoretical interest. For these reasons, in the latest papers on CoL, including the present one, the interest has shifted towards strong validity, in completeness proofs no longer addressing the question on weak validity — at least temporarily so.

operator R, the idea of which, in fact, was already present in [1], fifteen years before a similar (in the overall logical spirit) idea was materialized in the form of the exponential operator ! of linear logic. Here the qualification "equivalent" lacks a precise meaning, because $⫰^{\aleph_0}$ and R operate in non-identical game-semantical contexts (among the differences is that Blass's games are strict while the CoL games are not), which have never been brought to a common denominator. In a precise yet weaker wording, it is believed that, at least, the logical behaviors of the two operators are indistinguishable. Such a claim was made in [17] and, while no proof has been attempted, the present author has hardly any doubts that it is correct.

As we probably remember, $⅄A$ is defined simply as the infinite $\wedge$-conjunction $A \wedge A \wedge A \wedge \ldots$. The operator $⫰$ is technically much more evolved. In semiformal terms, a play of $⫰A$ starts as an ordinary play of game A. At any time, however, player $\bot$ (the environment) is allowed to make a "replicative move", which creates two copies of the current position Φ of A. From that point on, the game turns into two games played in parallel, each continuing from position Φ. We use the bits 0 and 1 to denote those two threads, which have a common past (position Φ) but possibly diverging futures. Again, at any time, $\bot$ can further branch either thread, creating two copies of the current position in that thread. If thread 0 was branched, the resulting two threads will be denoted by 00 and 01; and if the branched thread was 1, then the resulting threads will be denoted by 10 and 11. And so on: at any time, $\bot$ may split any of the existing threads w into two threads $w0$ and $w1$. Each thread in the eventual run of the game will be thus denoted by a (possibly infinite) bit string. The game is considered won by $\top$ (the machine) if it wins A in each of the threads; otherwise the winner is $\bot$.

To each infinite bit string w may thus correspond a separate run of A in thread (represented by) w and, as there are uncountably many infinite bit strings, uncountably many parallel runs of A may be generated when playing $⫰A$. Let us call a bit string w **essentially finite** if it contains only a finite number of "1"s; otherwise we say that w is **essentially infinite**. We extend these terms from bit strings to the corresponding threads in the play of $⫰A$. The definition of $⫰A$ thus requires from $\top$ to win A in all — whether they be essentially finite or essentially infinite — threads. All it takes to turn that definition into a definition of $⫰^{\aleph_0}$ is to relax that requirement and, when determining the winner, only look at essentially finite threads. Since there are only countably many essentially finite bit strings, only countably many runs of A are generated — more precisely, only countably many runs of A are of relevance — in $⫰^{\aleph_0}A$. This completes our semiformal definition/explanation of $⫰$ and $⫰^{\aleph_0}$.

In fully formal terms, consider a (constant) game A. Both $⫰A$ and $⫰^{\aleph_0}A$ have the same sets of legal runs. There are two types of legal moves in (legal) positions of either game: (1) replicative and (2) non-replicative. To define these, let us agree that by an **actual node** of a (legal) position Φ of $⫰A$ or $⫰^{\aleph_0}A$ we mean a bit string w such that w is either empty,[3] or else is $u0$ or $u1$ for some bit string u such that Φ contains the move $u:$. An actual node is said to be a **leaf** iff it is not a proper prefix of any other actual node.[4] A replicative move can only be made by (is only legal for) $\bot$, and such a move in a given position Φ should be $w:$, where w is a leaf of Φ.[5] As for non-replicative moves, they can be made by either player. Such a move by a player $\wp$ in a given position Φ should be $w.\alpha$, where w is an actual node of Φ and α is a move such that, for any infinite bit string v, α is a legal move by $\wp$ in position $\Phi^{\preceq wv}$ of A.[6] Here, for a run Θ and a bit string x, $\Theta^{\preceq x}$ means the result of deleting from Θ all moves except those that look like $u.\beta$ for some initial segment u of x, and then further deleting the prefix "$u.$" from such moves.[7] A legal run Γ of $⫰A$ is considered won by $\top$ iff, for every infinite bit string v, $\Gamma^{\preceq v}$ is a $\top$-won run of A. And a legal run Γ of $⫰^{\aleph_0}A$ is considered won by $\top$ iff, for every infinite but essentially finite bit string v, $\Gamma^{\preceq v}$ is a $\top$-won run of A. This completes our definition of $⫰$ and $⫰^{\aleph_0}$.

[3] Intuitively, the empty bit string is the name/address of the initial thread; all other threads will be descendants of that thread.

[4] Intuitively, a leaf is the unique individual name of an already existing thread of a play over A, while an actual node w which is not a leaf is a "partial" common name of several already existing threads — particularly, all threads whose individual names look like wv for some v.

[5] The intuitive meaning of move $w:$ is splitting thread w into $w0$ and $w1$, thus "activating" these two new nodes/threads.

[6] The intuitive meaning of such a move $w.\alpha$ is making move α in thread w and all of its (current or future) descendants.

[7] Intuitively, $\Theta^{\preceq x}$ is the run of A that has been played in thread x, if such a thread exists (has been generated); otherwise, $\Theta^{\preceq x}$ is the run of A that has been played in (the unique) existing thread which (whose name, that is) is some initial segment of x.

What we here semiformally call a **thread** of a (legal) run Γ of $⫰A$ or $⫰^{\aleph_0}A$ is a generalized leaf. Each thread is named by — and is usually identified with — a bit string w. When w is finite, saying that it is (it names) a thread means the same as saying that it is a leaf.[8] And when w is infinite, saying that it is (it names) a thread means that every finite initial segment of w is an actual node of some finite initial segment Φ of Γ. In other words, a thread is nothing but what the paper [14] more technically refers to as a "*complete branch of the underlying bitstring tree*". Intuitively, however, when w is a thread, every initial segment of it we also see as "the same thread".

This was a brutally quick review, of course. See [14] for more explanations and illustrations. In our treatment, we shall hardly ever rely on the formal definitions of the relevant game operations. Rather, we will be using informal or semiformal explanations and intuitive reasoning. Again, it should be pointed out that the present paper is not meant for a newcomer to the area of CoL.

3 Parallel recurrence validates both production principles

We first want to set up a uniform (interpretation-independent) winning strategy for short production with ! understood as $⅄$. In fact, we can and will do so for the following, more general than (1), form of short production:

$$P \wedge !(P \to P \wedge Q) \ \to !Q. \tag{3}$$

Writing $\overline{X}$ for $\neg X$ and eliminating $\to$, (3) is rewritten as follows:

$$\overline{P} \vee ?\big(P \wedge (\overline{P} \vee \overline{Q})\big) \vee !Q. \tag{4}$$

For convenience of references, let us agree that $P_0, P_1, P_2, \ldots$ all mean the same as P, and that $Q_1, Q_2, \ldots$ all mean Q. Now, remembering that $⅄$ is nothing but an infinite $\wedge$-conjunction, and its dual $⅂$ is nothing but an infinite $\vee$-disjunction (and that disjunction is associative), (4) can be further rewritten as

$$\overline{P}_0 \ \vee \ \big(P_1 \wedge (\overline{P}_1 \vee \overline{Q}_1)\big) \ \vee \ \big(P_2 \wedge (\overline{P}_2 \vee \overline{Q}_2)\big) \ \vee \ \ldots \ \vee \ \big(Q_1 \wedge Q_2 \wedge \ldots\big). \tag{5}$$

Since our strategy does not depend on an interpretation * applied to (5), we typically omit it (here and later in similar cases) and write, say, P where, strictly speaking, P^* is meant. In other words, with some innocent abuse of concepts, we identify formulas with the games into which they turn after an interpretation is applied to them.

A strategy that solves (5) is rather simple, and is schematically shown in Figure 2.

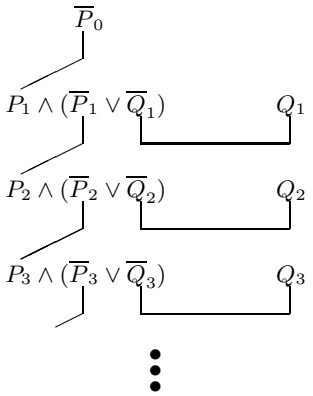


Figure 2

[8] More precisely, if Γ is infinite, w should be a leaf of every "sufficiently long" finite initial segment Φ of Γ.

The strategy consists in copycat routines between the pairs of subgames indicated in Figure 2 by arcs; we say that the corresponding two subgames X and $\overline{X}$ are **matched**, or **synchronized** by the strategy. Synchronizing means mimicking, in X, the moves made by the adversary in $\overline{X}$, and vice versa. If we see our strategy as an EPM ("Easy-Play Machine"), the synchronization is *perfect*, in the sense that the run taking place in $\overline{X}$ is the exact *negation* (all labels reversed) of the run taking place in X, meaning that exactly one of the two subgames will be eventually won by $\top$. If we see our strategy as an HPM ("Hard-Play Machine"), the synchronization is not necessarily perfect.[9] What is however still guaranteed is that the negation of the run taking place in $\overline{X}$ is a $\bot$-delay (See [14], Section 5) of the run taking place in X; taking into account that (as always in CoL) the games that we consider are static, this means that at least one of the subgames X, $\overline{X}$ will be won by $\top$, which, in view of the monotonicity of the winning conditions for the relevant game operations, is "even better than" when exactly one subgame is won. In view of this observation, for simplicity, here and later we will pretend that a synchronization is always perfect. We may not always be specific about whether the strategy that we consider is an HPM or an EPM, as these two models are equivalent for static games (See [14], Section 6). This equivalence, in turn, allows us to pretend (usually only implicitly) that the adversary of a given strategy always waits patiently until the strategy permits it to move.

Anyway, a simple combinatorial analysis of the situation convinces us that the strategy represented in Figure 2 wins the game: in view of its matching arrangements, one can see that if one of Q_i $(i \geq 1)$ is lost by $\top$, then either $\overline{P}_0$ or one of $P_j \wedge (\overline{P}_j \vee \overline{Q}_j)$ $(1 \leq j \leq i)$ is won, and hence so is the overall game.

As for long production, again, we can claim its being validated by $\curlywedge$ in forms more general than the originally given (2). One of such forms is

$$P \wedge !(P \to P \wedge Q) \wedge !(R \vee Q \to R) \ \to !R. \tag{6}$$

As shown in [14], with $! = \curlywedge$, everything provable in affine logic is uniformly valid; and uniform validity is closed under modus ponens. The formula (3) $\to$ (6) can easily be seen to be provable in affine logic. And, as we already know, its antecedent is uniformly valid. Hence so is its consequent.

4 Branching recurrences do not validate short production

We are going to show that the following instance of short production, with either $! \in \{\mathbin{\mathring{\downarrow}}, \mathbin{\mathring{\downarrow}}^{\aleph_0}\}$, is not valid, where p is a binary *elementary letter* (see Section 7 of [14]):

$$\sqcup x \sqcap y p(x,y) \wedge !\Big(\sqcup x \sqcap y p(x,y) \to \big(\sqcup x \sqcap y p(x,y) \wedge \sqcup x \sqcap y p(x,y)\big)\Big) \ \to \ !\sqcup x \sqcap y p(x,y).$$

Using $\overline{p}(x,y)$ for $\neg p(x,y)$, the above formula is rewritten as follows:

$$\sqcap x \sqcup y \overline{p}(x,y) \vee ?\Big(\sqcup x \sqcap y p(x,y) \wedge \big(\sqcap x \sqcup y \overline{p}(x,y) \vee \sqcap x \sqcup y \overline{p}(x,y)\big)\Big) \vee !\sqcup x \sqcap y p(x,y). \tag{7}$$

Let us fix an HPM $\mathcal{H}$ as an arbitrary strategy of the machine ($\top$). We want to construct a *counter-strategy* $\mathcal{C}$ such that, when the environment ($\bot$) follows it, $\mathcal{H}$ loses (7) under an appropriately selected interpretation, meaning that (7) is not uniformly valid (because $\mathcal{H}$ was picked arbitrarily). Technically, $\mathcal{C}$ is a function that takes a computation step i of $\mathcal{H}$ as an input, and returns a (possibly empty) sequence of moves that the environment should make during step i.

By a **literal** in this section we mean the formula $p(a,b)$ or $\overline{p}(a,b)$ for whatever constants a, b. The first literal is said to be **positive**, and the second literal is said to be **negative**. When we say that a given positive literal $p(a,b)$ occurs in a given formula, we always mean that it occurs without negation. For instance, the formula $p(a,b) \wedge \overline{p}(c,d)$ contains the literal $p(a,b)$ — as well as $\overline{p}(c,d)$ — but not $p(c,d)$. So, strictly speaking, a literal for us is a formula $p(a,b)$ or $\overline{p}(a,b)$ together with some fixed (usually clear from the context) positive occurrence. Unlike a literal, an **atom** always simply means

[9] After all, the environment can make any finite number of moves at once while the machine/strategy can make at most one move per computation step.

the formula $p(a,b)$ (for some constants a,b), no matter where and how it occurs. So, for instance, the above formula $p(a,b)\wedge\overline{p}(c,d)$ *does* contain the atom (but not the literal) $p(c,d)$. Any two literals $p(a,b)$ and $\overline{p}(a,b)$ (the same a,b) are said to be **opposite**.

Below is a description of the work of $\mathcal{C}$. In it, as in the previous section, terminologically we treat formulas as if they were games. The description assumes the context of a particular step of the play/interaction between $\mathcal{H}$ and $\mathcal{C}$. In that context, a **fresh** constant means a constant (decimal numeral) that has never been chosen by either player for the variables of (7) so far. A **component**, **part** or **subgame**, unless otherwise specified, always means one of (7). We refer to the three disjuncts of (7) as the **recurrence-free component**, the **?-component** and the **!-component**, respectively. By an **activated** literal we mean one to which the game has already been brought down in the recurrence-free component, or in one of the threads of the !-component, or in one of the parts of one of the threads of the ?-component. For instance, if, by a given time, in one of the threads of the ?-component, the game $\bigsqcup x\bigsqcap y p(x,y)\wedge\big(\bigsqcap x\bigsqcup y\overline{p}(x,y)\vee\bigsqcap x\bigsqcup y\overline{p}(x,y)\big)$ has been brought down to $\bigsqcap y p(a,y)\wedge\big(\overline{p}(b_1,c)\vee\bigsqcup y\overline{p}(b_2,y)\big)$, then $\overline{p}(b_1,c)$ — but not necessarily $p(b_1,c)$ — is an activated literal at that (and any later) time. We implicitly assume that $\mathcal{H}$ never makes illegal moves, or otherwise $\mathcal{C}$ wins immediately.

We define the counterstrategy $\mathcal{C}$ — or rather describe it in semiformal terms — through the following four "prescriptions":

Prescription (i): At the very beginning of the play, choose the constant 1 for x in the recurrence-free component, thus bringing it down to $\bigsqcup y\overline{p}(1,y)$.

Prescription (ii): Whenever, in any given thread w of the ?-component, $\mathcal{H}$ chooses a constant a for x in the $\bigsqcup x\bigsqcap y p(x,y)$ part, choose fresh constants b_1,b_2 ($b_1\neq b_2$) for the two occurrences of x in the $\bigsqcap x\bigsqcup y\overline{p}(x,y)\vee\bigsqcap x\bigsqcup y\overline{p}(x,y)$ part of the same thread. Thus, as a result, in thread w we will now have[10]

$$\bigsqcap y p(a,y)\wedge\big(\bigsqcup y\overline{p}(b_1,y)\vee\bigsqcup y\overline{p}(b_2,y)\big).$$

Prescription (iii): If and when $\mathcal{H}$ chooses a constant $\mathfrak{m}$ for x in the (so far) single thread of the !-component, split that thread, and choose fresh constants $\mathfrak{n}_1,\mathfrak{n}_2$ ($\mathfrak{n}_1\neq\mathfrak{n}_2$) for y in the two newly emerged threads. Thus, as a result, we will now have two threads in the !-component, one containing $p(\mathfrak{m},\mathfrak{n}_1)$ and the other containing $p(\mathfrak{m},\mathfrak{n}_2)$. From this point on, no moves in the !-component can or will ever be made again by either player.

Prescription (iv): Suppose, in a given thread w of the ?-component, by now the game has been brought down to one of the following forms:

$$\bigsqcap y p(a,y)\wedge\big(\bigsqcup y\overline{p}(b,y)\vee F\big); \tag{8}$$

$$\bigsqcap y p(a,y)\wedge\big(F\vee\bigsqcup y\overline{p}(b,y)\big) \tag{9}$$

(where F is either $\bigsqcup y\overline{p}(b',y)$ or $\overline{p}(b',c')$ for some b',c'), and $\mathcal{H}$ chooses a constant c for y in the $\bigsqcup y\overline{p}(b,y)$ subcomponent, such that the literal $p(b,c)$ is (already) activated. Then choose a fresh constant d for y in the $\bigsqcap y p(a,y)$ part of the same thread. Thus, depending on which of (8), (9) was the case, in thread w we will now have one of the following:

$$p(a,d)\wedge\big(\overline{p}(b,c)\vee F\big);$$

$$p(a,d)\wedge\big(F\vee\overline{p}(b,c)\big).$$

Consider the play of $\mathcal{H}$ in the scenario where the environment acts according to counterstrategy $\mathcal{C}$. There are two possibilities to be looked at separately.

One possibility is that it never comes to acting according to Prescription (iii). This means that the ! component remains unchanged throughout the play. Observe that then it never comes to acting

[10] I.e., the (sub)game $\bigsqcup x\bigsqcap y p(x,y)\wedge\Big(\bigsqcap x\bigsqcup y\overline{p}(x,y)\vee\bigsqcap x\bigsqcup y\overline{p}(x,y)\Big)$ will be brought down to ...

according to Prescription (iv) either, because Prescription (iii) is the only place where the *first-ever* positive activated literal can emerge — a literal whose presence is required in Prescription (iv). In this case, we choose an interpretation that makes every atom $p(a,b)$ true. A rather straightforward analysis of the situation convinces us that $\mathcal{H}$ loses (7) under this interpretation.

The other possibility, on which we focus throughout the rest of this section, is that, at some point, it comes to $\mathcal{C}$ acting according to Prescription (iii). Let us fix $\mathfrak{m}, \mathfrak{n}_1, \mathfrak{n}_2$ as the (unique) constants from Prescription (iii).

Let us say that a literal $p(a,d)$ is a **threadmate** of a literal $\overline{p}(b,c)$ iff there is a formula/game $p(a,d) \wedge \big(\overline{p}(b,c) \vee F\big)$ or $p(a,d) \wedge \big(F \vee \overline{p}(b,c)\big)$[11] to which the game has been brought down[12] in one of the threads of the ?-component.

We define a **chain** as a nonempty finite sequence of activated literals of the following form:

$$p(a_1,b_1),\ \overline{p}(a_1,b_1),\ldots,\ p(a_{n-1},b_{n-1}),\ \overline{p}(a_{n-1},b_{n-1}),\ p(a_n,b_n),\ \overline{p}(a_n,b_n) \tag{10}$$

($n \geq 1$), where (in addition to what can be seen from the above display — namely, that every literal at an odd position is positive and is followed by the opposite literal), for each i with $1 \leq i < n$, the literal $p(a_{i+1},b_{i+1})$ is a threadmate of the literal $\overline{p}(a_1,b_i)$.

What we call a **semichain**

$$p(a_1,b_1),\ \overline{p}(a_1,b_1),\ \ldots,\ p(a_{n-1},b_{n-1}),\ \overline{p}(a_{n-1},b_{n-1}),\ p(a_n,b_n) \tag{11}$$

satisfies exactly the same conditions as a chain, with the only difference that the last element $\overline{p}(a_n,b_n)$ is not present. Thus, chains are even-length while semichains are odd-length. Of course, if (10) is a chain, then (11) is a semichain. But not necessarily vice versa: if the literal $\overline{p}(a_n,b_n)$ is not activated, then (10) is not a chain even if (11) is a semichain.

We say that a constant a is **activated** iff it has been chosen for x or y at some point by either player in any (thread of any) part of the play. Using consecutive natural numbers $1,2,\ldots$ for the computation steps of $\mathcal{H}$, by the **activation time** $\mathbb{AT}(a)$ of such a constant we mean the earliest computation step of $\mathcal{H}$ during which a was first chosen by the corresponding player. Remember that, in the HPM model, the machine can make at most one move during a given computation step, while the environment can make any finite number of moves. We assume that the two moves that $\mathcal{C}$ makes according to Prescription (ii) — choosing b_1 and choosing b_2 — happen during the same step, so that $\mathbb{AT}(b_1) = \mathbb{AT}(b_2)$.

Where $i \in \{1,2\}$, a **$p(\mathfrak{m},\mathfrak{n}_i)$-headed chain** is a chain whose first literal is $p(\mathfrak{m},\mathfrak{n}_i)$. When we do not want to be specific about whether $i=1$ or $i=2$, we simply say "**a headed chain**". The same terminology extends from chains to semichains.

Lemma 4.1 *Suppose the following are headed semichains:*

$$p(a_1,b_1),\ \overline{p}(a_1,b_1),\ \ldots,\ p(a_{n-1},b_{n-1}),\ \overline{p}(a_{n-1},b_{n-1}),\ p(a_n,b_n);$$

$$p(a'_1,b'_1),\ \overline{p}(a'_1,b'_1),\ \ldots,\ p(a'_{n-1},b'_{n-1}),\ \overline{p}(a'_{n-1},b'_{n-1}),\ p(a'_n,b'_n).$$

Then $a_1 = a'_1, \ldots, a_n = a'_n$ *and* $\mathbb{AT}(a_1) > \ldots > \mathbb{AT}(a_n)$.

Proof. Induction on n. For $n=1$, the statement of the lemma is immediate, because $a_1 = a'_1 = \mathfrak{m}$. Now, suppose $n > 1$. By the definition of a semichain, there is a thread w in the ?-component where, for some F, at some point, the game was brought down to

$$p(a_n,b_n) \wedge \big(\overline{p}(a_{n-1},b_{n-1}) \vee F\big) \tag{12}$$

(or to $p(a_n,b_n) \wedge \big(F \vee \overline{p}(a_{n-1},b_{n-1})\big)$, but this case is similar); there is also a thread w' where, for some F', at some point, the game was brought down to

$$p(a'_n,b'_n) \wedge \big(\overline{p}(a'_{n-1},b'_{n-1}) \vee F'\big) \tag{13}$$

[11] In either case, F is either $\bigsqcup y\,\overline{p}(b',y)$ or $\overline{p}(b',c')$ for some b',c'.

[12] In this context meaning "has been brought down at some point in the play". The same applies to our usage of "activated" and similar terms.

(or to $p(a'_n, b'_n) \wedge \big(F' \vee \overline{p}(a'_{n-1}, b'_{n-1})\big)$, but this case is similar). Let us try to trace the history of how (12) and (13) emerged in the corresponding threads. At the beginning, both threads w and w' — just like all threads of the ?-component — had the formula/game $\sqcup x \sqcap y p(x,y) \wedge \big(\sqcap x \sqcup y \overline{p}(x,y) \vee \sqcap x \sqcup y \overline{p}(x,y)\big)$. Some time later, the game in w became

$$\sqcap y p(a_n, y) \wedge \big(\sqcap x \sqcup y \overline{p}(x,y) \vee \sqcap x \sqcup y \overline{p}(x,y)\big), \tag{14}$$

and the game in w' became

$$\sqcap y p(a'_n, y) \wedge \big(\sqcap x \sqcup y \overline{p}(x,y) \vee \sqcap x \sqcup y \overline{p}(x,y)\big). \tag{15}$$

In view of Prescription (ii), the former event was followed by $\mathcal{C}$ turning (14) into

$$\sqcap y p(a_n, y) \wedge \big(\sqcup y \overline{p}(a_{n-1}, y) \vee \sqcup y \overline{p}(k, y)\big) \tag{16}$$

for some k, and the latter event was followed by $\mathcal{C}$ turning (15) into

$$\sqcap y p(a'_n, y) \wedge \big(\sqcup y \overline{p}(a'_{n-1}, y) \vee \sqcup y \overline{p}(k', y)\big) \tag{17}$$

for some k'. Note that a_{n-1} (as well as a'_{n-1}, k and k') had to be fresh, for otherwise $\mathcal{C}$ would not have chosen it. So, $\mathbb{AT}(a_{n-1}) > \mathbb{AT}(a_n)$. We further claim that, by the time (16) and (17) emerged, the threads w and w' had not yet diverged; that is, (16) and (17) are the same and hence $a_n = a'_n$. Indeed, deny this. Note that then, as constants chosen by $\mathcal{C}$ in different threads, a_{n-1} and a'_{n-1} would be different from each other, because $\mathcal{C}$ always selects fresh constants. But, by the induction hypothesis, $a_{n-1} = a'_{n-1}$, which is a contradiction. $\square$

Let us say that a positive literal $p(a,b)$ is **reachable** iff it appears in (or, equivalently, is the last element of) some headed semichain.

By the **activation time** of an activated positive literal $p(a,d)$ we mean the time at which the constant d was (first) chosen for y by $\mathcal{C}$. In other words, this is the time (computation step of $\mathcal{H}$) at which the literal $p(a,d)$ first emerged in the overall play.

Lemma 4.2 *Every positive activated literal is reachable.*

Proof. Consider an arbitrary positive activated literal $p(a,d)$. We proceed by induction on the activation time of $p(a,d)$. Among all positive activated literals, $p(\mathfrak{m}, \mathfrak{n}_1)$ and $p(\mathfrak{m}, \mathfrak{n}_2)$ obviously have the smallest activation time; and they are reachable.

Now consider any other activated positive literal $p(a,d)$. Obviously it could have emerged in the play only according to Prescription (iv). Namely, at some point, in some thread of the ?-component, we had $\sqcap y p(a,y) \wedge \big(\sqcup y \overline{p}(b,y) \vee F\big)$ (or $\sqcap y p(a,y) \wedge \big(F \vee \sqcup y \overline{p}(b,y)\big)$, but this case is similar), and the event that triggered the application of Prescription (iv) was that the above became $\sqcap y p(a,y) \wedge \big(\overline{p}(b,c) \vee F\big)$, where the (positive) literal $p(b,c)$ had already been activated. To this event, $\mathcal{C}$ responded by further bringing the game in the thread down to $p(a,d) \wedge \big(\overline{p}(b,c) \vee F\big)$ for a fresh d, thus "activating" $p(a,d)$. The activation time of $p(b,c)$ is thus smaller than that of $p(a,d)$. Hence, by the induction hypothesis, $p(b,c)$ is reachable. But $p(a,d)$ is a threadmate of $\overline{p}(b,c)$. Hence $p(a,d)$ is also reachable: a semichain ending in $p(a,d)$ is obtained from the semichain ending in $p(b,c)$ by appending to it the two literals $\overline{p}(b,c)$ and $p(a,d)$. $\square$

We say that a constant a is **reachable** iff, for some b, the positive literal $p(a,b)$ is reachable.

Lemma 4.3 *Suppose b and c are two distinct reachable constants. Then $\mathbb{AT}(b) \neq \mathbb{AT}(c)$.*

Proof. Assume $b \neq c$ are reachable constants. In view of Lemma 4.1, with some thought, one can see that there is a finite sequence $a_1, \ldots, a_n$ of constants such that $a_1 = \mathfrak{m}$, $\mathbb{AT}(a_1) > \ldots > \mathbb{AT}(a_n)$ and any reachable constant is among $a_1, \ldots, a_n$ (hint: consider a longest headed semichain). So, we have $b = a_i$ and $c = a_j$ for some $1 \leq i \neq j \leq n$. And therefore $\mathbb{AT}(b) < \mathbb{AT}(c)$ or $\mathbb{AT}(c) > \mathbb{AT}(b)$, i.e., $\mathbb{AT}(b) \neq \mathbb{AT}(c)$. $\square$

Lemma 4.4 *Suppose*

$$p(a_1, b_1),\ \overline{p}(a_1, b_1),\ \ldots,\ p(a_n, b_n),\ \overline{p}(a_n, b_n)$$

is a $p(\mathfrak{m}, \mathfrak{n}_1)$-headed chain, and

$$p(a_1, b'_1),\ \overline{p}(a_1, b'_1),\ \ldots,\ p(a_n, b'_n),\ \overline{p}(a_n, b'_n)$$

is a $p(\mathfrak{m}, \mathfrak{n}_2)$-headed chain. Then $b_1 \neq b'_1, \ldots, b_n \neq b'_n$.

Proof. We show that $b_i \neq b'_i$ by induction on $i \in \{1, \ldots, n\}$. For $i = 1$ this is immediate, because $b_1 = \mathfrak{n}_1$ and $b'_1 = \mathfrak{n}_2$.

Now consider any $i \in \{2, \ldots, n\}$. Let us trace the history of the two threads w and w' of the ?-component in which $p(a_i, b_i)$ and $p(a_i, b'_i)$ were "activated" (i.e., first emerged), respectively. Originally, in both threads we had

$$\bigsqcup x \bigsqcap y\, p(x, y) \wedge \big(\bigsqcap x \bigsqcup y\, \overline{p}(x, y) \vee \bigsqcap x \bigsqcup y\, \overline{p}(x, y)\big),$$

which later evolved to

$$\bigsqcap y\, p(a_i, y) \wedge \big(\bigsqcap x \bigsqcup y\, \overline{p}(x, y) \vee \bigsqcap x \bigsqcup y\, \overline{p}(x, y)\big). \tag{18}$$

In both threads, $\mathcal{C}$'s response according to Prescription (ii) brought (18) down to

$$\bigsqcap y\, p(a_i, y) \wedge \big(\bigsqcup y\, \overline{p}(a_{i-1}, y) \vee \bigsqcup y\, \overline{p}(c, y)\big) \tag{19}$$

(or $\bigsqcap y\, p(a_i, y) \wedge \big(\bigsqcup y\, \overline{p}(c, y) \vee \bigsqcup y\, \overline{p}(a_{i-1}, y)\big)$, but this case is similar) for some c different from a_{i-1}. Here we see that at the time of the action that resulted in (19), the two threads w and w' were not separated yet, that is, whatever we have said so far, was happening in the common ancestor of the two threads. This is so because, otherwise, $\mathcal{C}$ would have chosen non-identical a_{i-1}s in the two threads.

But the two threads had to diverge at some point, because otherwise b_{i-1} and b'_{i-1}, chosen later by $\mathcal{H}$ for y in $\bigsqcup y\, \overline{p}(a_{i-1}, y)$, would have to be the same, which, however, is not the case by the induction hypothesis.

If the two threads diverged before anyhow modifying (19), then b_i and b'_i, as constants chosen later by $\mathcal{C}$ in different threads, are different, and we are done.

Now suppose some change happened in (19) before the two threads diverged. What could have been such a change? $\mathcal{C}$ would not have moved in (19) it until $\mathcal{H}$ had made a move there first. But $\mathcal{H}$ could not have moved within the $\bigsqcup y\, \overline{p}(a_{i-1}, y)$ part of (19) until the threads diverged (because in w it had to select b_{i-1} for y while in w' select b'_{i-1}; these two, by the induction hypothesis, are distinct). So, the only possible event is that $\mathcal{H}$ moved within the $\bigsqcup y\, \overline{p}(c, y)$ part of (19), namely, brought that part down to $\overline{p}(c, d)$ for some d. According to Prescription (iv), $\mathcal{C}$ responds to such a move only if the literal $p(c, d)$ is (already) activated, which, by Lemma 4.2, is the same as to say that $p(c, d)$ is reachable. But if $p(c, d)$ is reachable, then so is c; and, of course, a_{i-1} is also reachable. Therefore, by Lemma 4.3, $\mathbb{AT}(c) \neq \mathbb{AT}(a_{i-1})$. This is a contradiction, because a_{i-1} and c were "activated" simultaneously when $\mathcal{C}$ brought (18) down to (19). Thus, $\mathcal{C}$ did not respond to $\mathcal{H}$'s action, and what we now have in the two, not-yet-diverged threads w and w' is

$$\bigsqcap y\, p(a_i, y) \wedge \big(\bigsqcup y\, \overline{p}(a_{i-1}, y) \vee \overline{p}(c, d)\big). \tag{20}$$

Now, the only next event in the evolution of the two threads is that they, at last, diverge (so that $\mathcal{H}$ can choose the different constants b_{i-1} and b'_{i-1} for y in $\bigsqcup y\, \overline{p}(a_{i-1}, y)$). But once the threads diverge, as noted earlier, $\mathcal{C}$ will later choose different constants b_i and b'_i for y in the $\bigsqcap y\, p(a_i, y)$ component of the two threads. Showing that $b_i \neq b'_i$ was exactly our goal. $\square$

Let us say that a chain is **complete** iff its last literal is $\overline{p}(1, b)$ for some b.

Lemma 4.5 *Either there is no complete $p(\mathfrak{m}, \mathfrak{n}_1)$-headed chain, or there is no complete $p(\mathfrak{m}, \mathfrak{n}_2)$-headed chain.*

Proof. Assume, for a contradiction, that there is a complete $p(\mathfrak{m},\mathfrak{n}_1)$-headed chain

$$p(a_1,b_1),\ \overline{p}(a_1,b_1),\ \ldots,\ p(a_n,b_n),\ \overline{p}(a_n,b_n)$$

and also there is a complete $p(\mathfrak{m},\mathfrak{n}_2)$-headed chain

$$p(a'_1,b'_1),\ \overline{p}(a'_1,b'_1),\ \ldots,\ p(a'_{n'},b'_{n'}),\ \overline{p}(a'_{n'},b'_{n'}).$$

Without loss of generality here we may assume that $n \leq n'$. According to Lemma 4.1, we have $a_1 = a'_1, \ldots, a_n = a'_n$. And, by Lemma 4.4, $b_n \neq b'_n$. Thus, as $a_n = 1$, we have two non-identical activated literals $\overline{p}(1,b_n)$ and $\overline{p}(1,b'_n)$. This is however impossible. It is impossible because the constant 1 was chosen by $\mathcal{C}$ only in the recurrence-free component (when following Prescription (i)) which, as a result, was brought down to $\bigsqcup y\overline{p}(1,y)$; and, since that component is recurrence-free and thus cannot be replicated, $\mathcal{H}$ would not have a chance to make two different choices b_n and b'_n for y there to further bring it down to $\overline{p}(1,b_n)$ and $\overline{p}(1,b'_n)$. $\square$

To complete our proof, pick an $i \in \{1,2\}$ such that there is no complete $p(\mathfrak{m},\mathfrak{n}_i)$-headed chain (the existence of such an i is guaranteed by Lemma 4.5). Let us say that an atom is **i-reachable** iff it appears in some $p(\mathfrak{m},\mathfrak{n}_i)$-headed semichain. We choose an interpretation that makes all i-reachable atoms false, and makes any other atom true. It is left to the reader to convince himself or herself that, under this interpretation, (7) is lost by $\mathcal{H}$. In this exercise, whether ! means $\downarrow^{\aleph_0}$ or $\downarrow$ is of no relevance.

5 Countable branching recurrence validates long production

In this section we are going to show that, with $! = \downarrow^{\aleph_0}$, long production is valid in the strong form of (6), which we rewrite as

$$\overline{\mathcal{P}} \vee ?\big(P \wedge (\overline{P} \vee \overline{Q})\big) \vee ?\big((R \vee Q) \wedge \overline{R}\big) \vee !\mathcal{R}. \tag{21}$$

Here we have used the calligraphic $\mathcal{R}$ for one of the two occurrences of R in order to differentiate it from the other occurrence. Similarly for $\mathcal{P}$. This is merely for readability.

We refer to the four disjuncts of (21) as the **recurrence-free component**, the **left ?-component**, the **right ?-component** and the **!-component**, respectively. As before, we see formulas as games. Some other earlier terminology and conventions may as well apply.

For our purposes, we want to agree on a simplified way of schematically representing different stages of a play over (21). We explain this way in a semiformal fashion. Initially, both ?-components and the !-component have a single thread ϵ (ϵ stands for the **empty bit string**). To indicate this, we use ϵ as a subscript and, after omitting the external disjunction symbols as well as ? and !, we rewrite (21) as

$$\overline{\mathcal{P}} \qquad P_\epsilon \wedge (\overline{P}_\epsilon \vee \overline{Q}_\epsilon) \qquad (R_\epsilon \vee Q_\epsilon) \wedge \overline{R}_\epsilon \qquad \mathcal{R}_\epsilon. \tag{22}$$

Our purported uniform solution/strategy for (21) — let us call that strategy $\mathcal{K}$ — makes two initialization moves consisting in replicating the (so far the only) thread ϵ of both ?-components. This results in the position that we represent as

$$\overline{\mathcal{P}} \qquad P_0 \wedge (\overline{P}_0 \vee \overline{Q}_0) \quad P_1 \wedge (\overline{P}_1 \vee \overline{Q}_1) \qquad (R_0 \vee Q_0) \wedge \overline{R}_0 \quad (R_1 \vee Q_1) \wedge \overline{R}_1 \qquad \mathcal{R}_\epsilon. \tag{23}$$

Here we see the subscripts 0 and 1 because these are (the names of) the threads into which ϵ turns after it is split.

After the above initialization moves, $\mathcal{K}$ establishes synchronization between the following pairs of subgames: $(\overline{\mathcal{P}}, P_0)$, $(\overline{Q}_0, Q_0)$ and $(\overline{R}_0, \mathcal{R}\epsilon)$. This arrangement is shown in Figure 3, with synchronizations indicated by arcs.

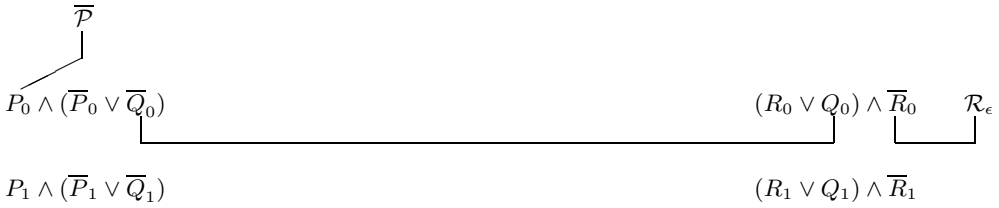


Figure 3

If the environment does not make any replicative moves in the !-component, the situation represented by Figure 3 will persist throughout the rest of the game. Of course, $\overline{\mathcal{P}}$ will probably no longer be the original $\overline{\mathcal{P}}$ at later stages of the play, but what matters is that, to whatever game the original $\overline{\mathcal{P}}$ evolves (which we continue denoting by $\overline{\mathcal{P}}$), it will essentially be the negation of to whatever game the original P_0 evolves; "essentially" in the sense explained in Section 3, which guarantees that at least one of the two games will be eventually won by $\mathcal{K}$ and — again as in Section 3 — we can safely pretend that exactly one of them will be in fact won. Similarly for the other components of the game shown in Figure 3. A straightforward analysis of the situation in the present scenario (the scenario where the environment does not make replications in the !-component) shows that $\mathcal{K}$ wins as desired. In this analysis, the presence of the two subgames $P_1 \wedge (\overline{P}_1 \vee \overline{Q}_1)$ and $(R_1 \vee Q_1) \wedge \overline{R}_1$ displayed at the bottom of Figure 3 is irrelevant. As we are going to see, the role of these two is to maintain "fresh" copies of the original $P \wedge (\overline{P} \vee \overline{Q})$ and $(R \vee Q) \wedge \overline{R}$. The same applies to the $\overline{P}_0$ and Q_0 components. Here and later, by "fresh" we mean that $\mathcal{K}$ has not made any moves in these (sub)games. Of course, $\mathcal{K}$ has no way to prevent the environment from making moves in these subgames. But such moves are harmless in the sense that they cannot create any problems for $\mathcal{K}$ later if it decides to start synchronizing such a "fresh" subgame X with another "fresh" subgame $\overline{X}$. The reason, again as explained in Section 3, is that we deal with static games (that is, for any interpretation *, the games P^*, Q^*, R^* are static).

We continue our description of the strategy $\mathcal{K}$. The case of the environment making no replications in the !-component has been already fully covered. Now, suppose the environment replicates the thread ϵ of the !-component (at this point, ϵ is the only thread there). That is, the environment splits the $\mathcal{R}_\epsilon$ component into two copies $\mathcal{R}_0$ and $\mathcal{R}_1$. In response, $\mathcal{K}$ replicates the $(R_0 \vee Q_0) \wedge \overline{R}_0$ component, turning it into two child copies $(R_{00} \vee Q_{00}) \wedge \overline{R}_{00}$ and $(R_{01} \vee Q_{01}) \wedge \overline{R}_{01}$. It does the same with $P_1 \wedge (\overline{P}_1 \vee \overline{Q}_1)$ and $(R_1 \vee Q_1) \wedge \overline{R}_1$. Figure 4 shows all components that we will be dealing with from now on, and also shows the synchronization arrangements that $\mathcal{K}$ will maintain. The "$\times$" under Q_{01} indicates that, from now on, this component is "wasted" in the sense that it will not and cannot be synchronized with anything.

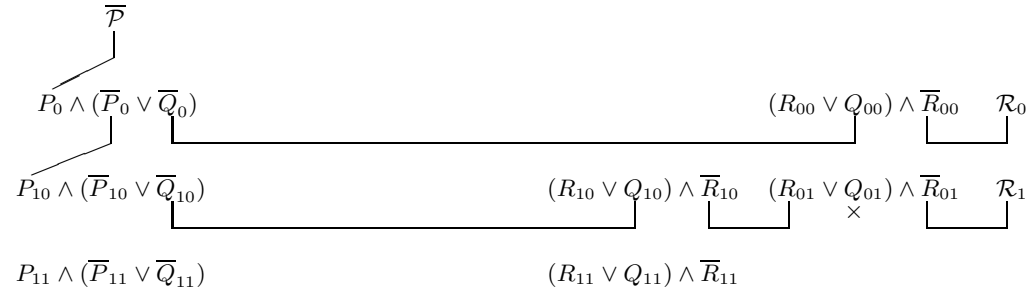


Figure 4

Again, if the environment makes no further replications, then the synchronization shown in Figure 4 obviously guarantees a win for $\mathcal{K}$. Let us now say $\mathcal{R}_1$ is replicated (the other possibility would be

replicating $\mathcal{R}_0$). $\mathcal{K}$'s reaction is replicating $(R_{01} \vee Q_{01}) \wedge \overline{R}_{01}$, $(R_{10} \vee Q_{10}) \wedge \overline{R}_{10}$, $P_{11} \wedge (\overline{P}_{11} \vee \overline{Q}_{11})$ and $(R_{11} \vee Q_{11}) \wedge \overline{R}_{11}$, followed by the synchronization arrangements shown in Figure 5.

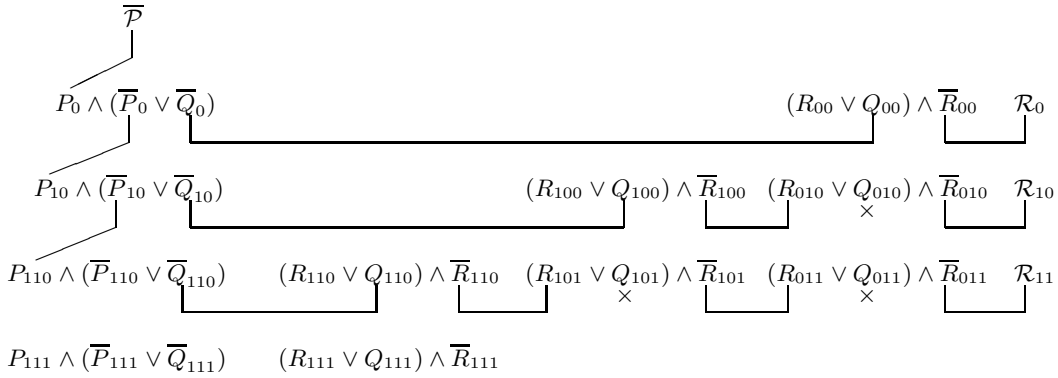


Figure 5

Assume that, next time, the environment replicates $\mathcal{R}_0$. The situation resulting from $\mathcal{K}$'s reaction is shown in Figure 6.

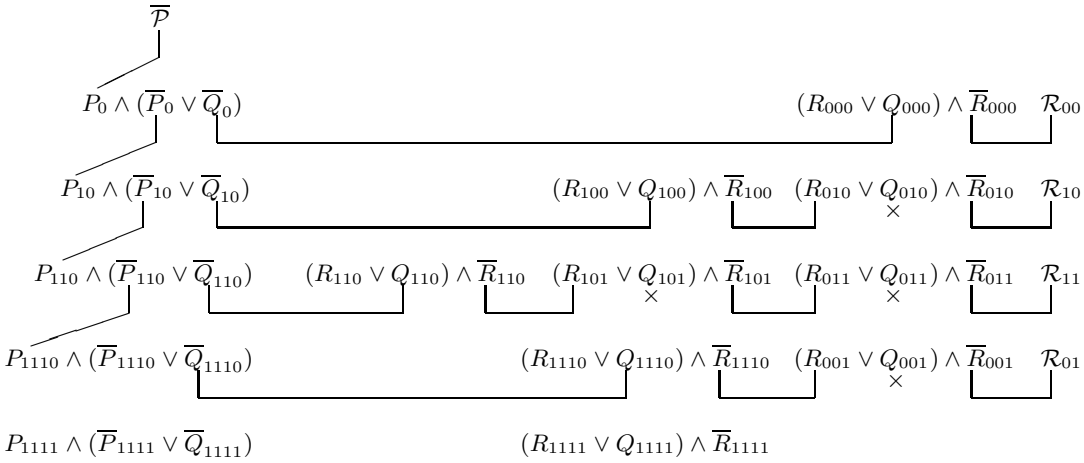


Figure 6

Do you see a pattern here? As an exercise, try to trace three more steps, namely, in the scenario where the environment replicates $\mathcal{R}_{00}$, then $\mathcal{R}_{10}$, and then $\mathcal{R}_{11}$. Once you are done, you have understood the strategy and there is no need to read our further — general — description of it.

In general terms, the work of $\mathcal{K}$ is divided into **stages**, with each stage represented by a diagram in the style of Figures 3-6. Figure 3 shows the first stage, which includes initialization as described earlier and subsequent maintainance of synchronization between three pairs of subgames.

Now let us consider stage #n for an arbitrary $n \geq 1$. The corresponding diagram will look like the one shown in Figure 7. Here we provide additional explanations for that diagram:

- $w_1, \ldots, w_n$ stand for the bit strings representing the "currently existing" threads of the !-component; there are exactly n such threads.
- z is 1^n (the string of n "1"s). Both of the ?-components have thread z, reserved for the purpose of keeping a "fresh copy" of the corresponding game (game $P \wedge (\overline{P} \vee \overline{Q})$ in the left ?-component and game $(R \vee Q) \wedge \overline{R}$ in the right ?-component).

- $u_1, \ldots, u_n$, in addition to z, stand for the bit strings representing the "currently existing" threads of the left ?-component; there are exactly $n+1$ such threads. Here each u_i is $1^{i-1}0$ (the string of $i-1$ "1"s followed by a "0").
- $k_1, \ldots, k_n$ are positive integers and, for each $j \in \{1, \ldots, n\}$ and $e \in \{1, \ldots, k_j\}$, v_j^e is a bit string that represents some thread of the right ?-component. Together with z, such v_j^es are all of the (pairwise distinct) threads of that component.

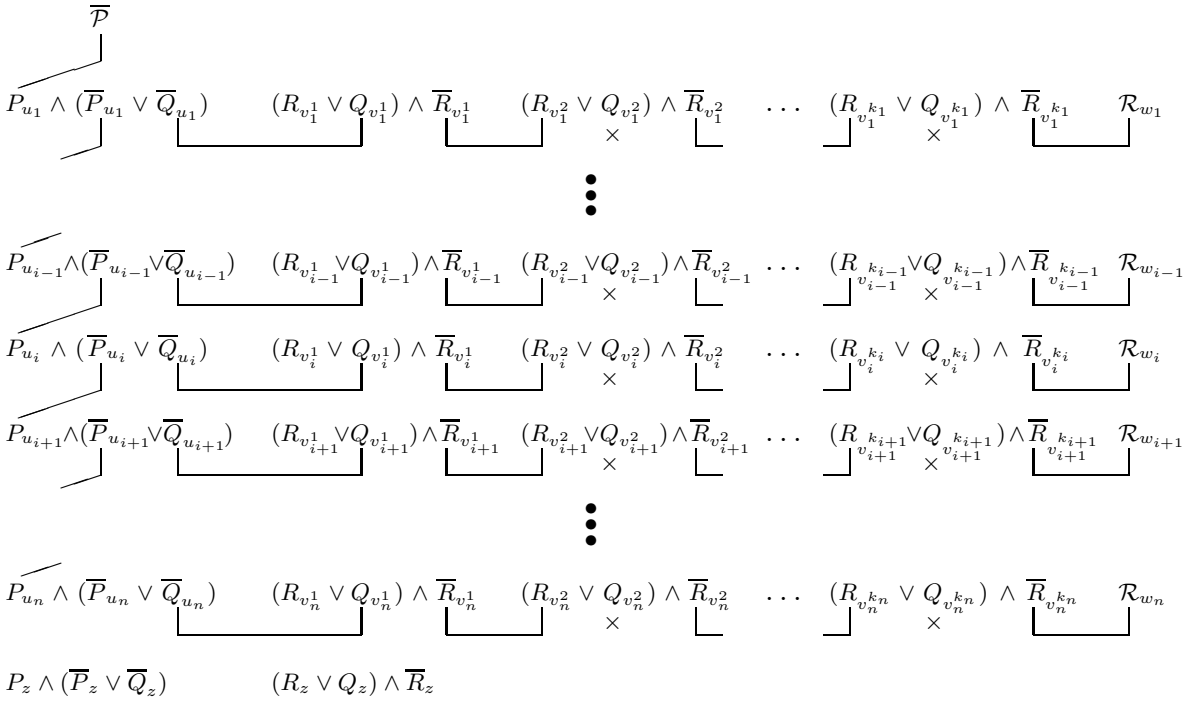


Figure 7

The work of $\mathcal{K}$ during stage $\#n$ consists in performing the synchronization routine represented by the arcs of Figure 7. Let us observe right now (through a routine analysis left to the reader) that, if stage $\#n$ lasts forever, $\mathcal{K}$ wins. As an aside, in this case $\mathcal{K}$ wins even if ! means $\mathrel{\mathring{\downarrow}}$ rather than $\mathrel{\mathring{\downarrow}}^{\aleph_0}$.

Stage $\#n$ will end if and when the environment splits one of the threads $w_1, \ldots, w_n$ of the !-component. Let us assume thread w_i is split, which now becomes two threads: w_i0 and w_i1. This triggers a transition to stage $\#(n+1)$. A diagram for that stage is shown in Figure 8. Note that we have placed the newly emerged thread w_i1 of the !-component at the bottom of the list of (non-z) threads of that component, while leaving thread w_i0 where thread w_i was previously found. In response to the above replicative move by the environment, $\mathcal{K}$ makes a series of replications. Namely, it replicates:

- The threads $v_i^1, \ldots, v_i^{k_i}$ of the right ?-component — the ones that were (in Figure 7) found in the same row as $\mathcal{R}_{w_i}$. This results in two series of new threads that replace the old ones: $v_i^1 0, \ldots, v_i^{k_i} 0$ and $v_i^1 1, \ldots, v_i^{k_i} 1$.
- The thread z of the right ?-component, which now turns into $z0$ (i.e. $1^n 0$) and $z1$ (i.e. 1^{n+1}).
- The thread z of the left ?-component, which now turns into $z0$ (i.e. $1^n 0$) and $z1$ (i.e. 1^{n+1}).

Where these newly born threads/copies are placed, and how the synchronization arrangements are set up or redefined for them (while preserving all other old matchings) can be seen from Figure 8.

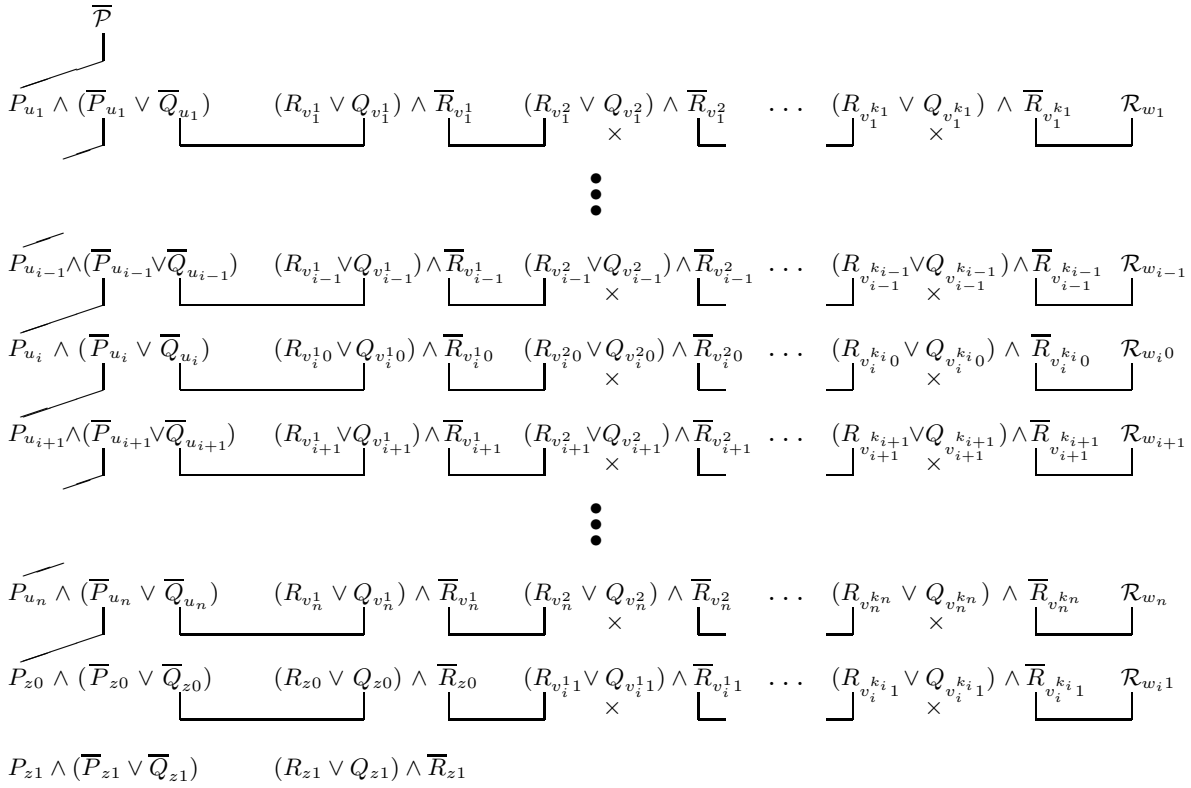


Figure 8

We have already observed that, if there are only finitely many stages (i.e., the environment only makes finitely many replications in the !-component), $\mathcal{K}$ is the winner. It remains to understand why $\mathcal{K}$ also wins the game in the cases where there is no last stage.

Let us use the term "**line**" for the rows of a diagram in the style of Figure 7, for the exception of the topmost row consisting of $\overline{\mathcal{P}}$ and the bottommost row consisting of the two "reserve" threads. We number the lines of a diagram consecutively from top to bottom. To see that the strategy $\mathcal{K}$ is successful, consider any essentially finite (but possibly infinite) bit string s such that the thread s has actually emerged in the !-component. Let v be the shortest (possibly empty) initial segment of s containing all "1"s that s contains. So, s looks like vu, where the u part entirely consists of (finitely or infinitely many) "0"s. Remember these v and u.

Imagine the stage — let it be stage $\#n$ — at which the component $\mathcal{R}_v$ first emerged in the corresponding diagram. At that time, $\mathcal{R}_v$ will be placed in line $\#n$. Temporarily assuming that n is the last stage that ever emerges in the play, and analyzing[13] the diagram for stage $\#n$, one can easily see that

> *if $\mathcal{K}$ loses in thread v, then it either wins in the recurrence-free component, or in one of the threads x of one of the two ?-components in line $\#m$ for some $1 \leq m \leq n$.* (24)

Now back to the case of infinitely many stages. An analysis of the situation, which the reader should partly undertake on his or her own after internalizing our construction, reveals that, in the continuously evolving diagram, everything — including matching arrangements — in lines $\#1$ through $\#n$ will remain "the same", and the only changes that may be occurring in those lines are that "0"s will be added to the names of the threads of the right (but not the left) ?-component and of the thread of the !-component.[14]

[13] In fact, we have already implicitly undertaken such an analysis earlier when observing that $\mathcal{K}$ is successful in the cases of finitely many stages.

[14] If we imagine an infinite diagram corresponding to the entire play, its lines $\#1$ through $\#n$ will remain "essentially the same" as in the diagram for stage $\#n$.

The above changes are of no relevance to our earlier argument for (24), in which we now should simply replace "v" by "vu" and "x" by "xy", where y, just like u, is a certain (possibly infinite) string entirely consisting of "0"s; as xy is essentially finite, we again find that $\mathcal{K}$ wins the overall game.

Since the above s was an arbitrary essentially finite string, we conclude that, if $\mathcal{K}$ loses in the !-component, then it wins the overall game. And, of course, so does it if it does not lose in the !-component.

6 Uncountable branching recurrence does not validate long production

An **enumeration game** is a game where any natural number, identified with its decimal representation, is a legal move by either player at any time (and there are no other legal moves). This way, either player can be seen to enumerate a set of numbers — the numbers made by it as moves during the play. The winner in a (legal) play of an enumeration game only depends on the two sets enumerated this way. That is, what matters is only *what* moves have been eventually made, regardless of *when* (in what order) and *how many times* those moves were made.

Let us rewrite long production (2) in the following form:

$$\overline{P}^1 \vee ?\big(P^2 \wedge (\overline{P}^3 \vee \overline{P}^4)\big) \vee ?\big((P^5 \vee P^6) \wedge \overline{P}^7\big) \vee !P^8. \tag{25}$$

Here we have used the superscripts 1 through 8 to differentiate between the different occurrences of P, and otherwise it is understood that each P^i simply means P. Throughout this section, ! stands for $\fatslash$, and ? for its dual $⚲$ ($= \neg\fatslash\neg$).

Let us fix an HPM $\mathcal{H}$ as an arbitrary strategy of the machine. We want to construct a *counterstrategy* $\mathcal{D}$ — in the same sense as in Section 4 — such that, when the environment follows it, $\mathcal{H}$ loses (25) with P interpreted as a certain enumeration game.

As was done earlier, terminologically we identify formulas with games. This does not create any confusion: because P is going to be interpreted as an enumeration game anyway, the legal moves of it — even if not the winner — are known even before we actually define that interpretation.

As in the preceding section, we refer to $\overline{P}^1$ as the *recurrence-free component*, refer to $?\big(P^2\wedge(\overline{P}^3\vee\overline{P}^4)\big)$ as the *left ?-component*, refer to $?\big((P^5 \vee P^6) \wedge \overline{P}^7\big)$ as the *right ?-component* and refer to $!P^8$ as the !-*component*. A **fresh move** means a number that has not yet been chosen in the play by either player as a move in any subgame P^i or $\overline{P}^j$.

The work of the counterstrategy $\mathcal{D}$ is very simple. It consists in repeating, over and over infinitely many times, the following routine:

Step (i): Split (make a replicative move in) each thread of the !-component.

Step (ii): Make a fresh move[15] in the recurrence-free component, in each thread of the left ?-component, in each thread of the right ?-component, and in each thread of the !-component.

The rest of our discussion is in the context of the run/play generated in the scenario where $\mathcal{H}$ plays as $\top$ and the environment ($\bot$) acts according to strategy $\mathcal{D}$. It is obvious that (as along as P is interpreted as an enumeration game) $\mathcal{D}$ plays legally. As always, we safely assume that its adversary $\mathcal{H}$ never makes illegal moves, either.

Let $\mathbb{T}^?_l$ be the set of all threads that eventually emerge in the left ?-component, $\mathbb{T}^?_r$ be the set of all threads that eventually emerge in the right ?-component, and $\mathbb{T}^!$ be the set of all threads that eventually emerge in the !-component. As an aside, notice that $\mathbb{T}^!$ is exactly the set of all infinite bit strings. The same is not necessarily the case for $\mathbb{T}^?_l$ and $\mathbb{T}^?_r$ though, which may even be finite.

What we here — by some abuse of terminology — call **literals** are the following objects (technically, these "objects" are nothing but superscript/subscript pairs, except for one case where we only have a superscript):

[15] Several moves are made during this step, and it is understood that the condition of their "freshness" implies that they are different not only from all earlier moves/numbers, but also from each other.

- $\overline{P}^1$;
- P^2_w, $\overline{P}^3_w$ and $\overline{P}^4_w$ for each $w \in \mathbb{T}^?_l$;
- P^5_w, P^6_w and $\overline{P}^7_w$ for each $w \in \mathbb{T}^?_r$;
- P^8_w for each $w \in \mathbb{T}^!$.

The literals of the form P^2_w, P^5_w, P^6_w, P^8_w are **positive**, and all other literals (the ones with an overline) are **negative**.

With each literal L we associate its **content**. The latter is a pair $(L_\top, L_\bot)$, where $L_\top$ is the set of all moves (numbers) made (enumerated) by $\mathcal{H}$ in the corresponding (sub)game, and $L_\bot$ is the set of all moves made by $\mathcal{D}$ there. Here the meaning of "the corresponding (sub)game" must be clear: in the case of $\overline{P}^1$, this is the (sub)game played in the recurrence-free component; in the case of P^2_w (resp. $\overline{P}^3_w$, $\overline{P}^4_w$), this is the (sub)game played in the P^2 (resp. $\overline{P}^3$, $\overline{P}^4$) part of thread w of the left ?-component; in the case of P^5_w (resp. P^6_w, $\overline{P}^7_w$), this is the (sub)game played in the P^5 (resp. P^6, $\overline{P}^7$) part of thread w of the right ?-component; and, in the case of P^8_w, this is the (sub)game played in thread w of the !-component.

Lemma 6.1 *If L and M are two distinct literals (meaning that either their superscripts, or their subscripts, or both, are non-identical), then their contents are also distinct.*

Proof. If L and M are two distinct literals, the (sub)games they represent happen to be in different parts of the overall game (including the possibility of being in different threads). But $\mathcal{D}$ keeps making fresh moves in all existing components. So, some number (in fact, infinitely many numbers) is in the set enumerated by $\mathcal{D}$ in one (sub)game but not in the set enumerated by $\mathcal{D}$ in the other (sub)game. □

Let L and M be literals. We say that L **matches** M iff, where $(L_\top, L_\bot)$ and $(M_\top, M_\bot)$ are the contents of L and M, we have $L_\top = M_\bot$ and $L_\bot = M_\top$. Next, we say that L is a **threadmate** of M iff $L \neq M$ and one of the following conditions is satisfied:

- For some $w \in \mathbb{T}^?_l$, both L and M are among $P^2_w, \overline{P}^3_w, \overline{P}^4_w$.
- For some $w \in \mathbb{T}^?_r$, both L and M are among $P^5_w, P^6_w, \overline{P}^7_w$.

We define a **chain** as a nonempty finite sequence $L_1, \ldots, L_n$ of literals satisfying the following conditions:

- L_1 (and only L_1) is $\overline{P}^1$.
- For each odd i with $1 \leq i < n$, L_{i+1} matches L_i.
- For each even i with $1 \leq i < n$, L_{i+1} is a threadmate of L_i.

Let

$$\overline{P}^{i_1}, \tilde{P}^{i_2}_{w_2}, \tilde{P}^{i_3}_{w_3}, \tilde{P}^{i_4}_{w_4}, \ldots, \tilde{P}^{i_n}_{w_n}$$

be a chain (here $\tilde{P}$ stands for P with or without an overline; and, of course, $i_1 = 1$). The **type** of such a chain is the sequence $i_1, i_2, i_3, i_4, \ldots, i_n$ of the superscripts of the above-displayed literals.

Lemma 6.2 *There are no two non-identical chains that have the same type.*

Proof. Rather immediately from Lemma 6.1. □

Let us say that a literal L is **reachable** iff there is a chain $L_1, \ldots, L_n$ with $L_n = L$.

Lemma 6.3 *There are only countably many reachable literals.*

Proof. The number of all possible types of chains is countable, because every type is a finite sequence of numbers. And, in view of Lemma 6.2, all reachable literals can be listed by listing the (unique) types of the corresponding chains. □

We select an interpretation * that interprets P as the enumeration game such that, whenever $(S_\top, S_\bot)$ is the pair of the sets enumerated (while playing P^*) by $\top$ and $\bot$, respectively, $\top$ is the winner if and only if $(S_\top, S_\bot)$ is the content of some reachable positive literal or $(S_\bot, S_\top)$ is the content of some reachable negative literal.

Implicitly relying on Lemma 6.1, we now claim that $\mathcal{H}$ loses the overall game under this interpretation. To see why, first observe that there is a thread u (in fact, uncountably many such threads) in the !-component such that P^8_u is not reachable. This is so because there are uncountably many threads in the !-component, of which, however, according to Lemma 6.3, only countably many are reachable. By our choice of interpretation, $\mathcal{H}$ loses the game P in thread u, meaning that it loses the entire !-component. Next, $\mathcal{H}$ loses the recurrence-free component because $\overline{P}^1$ is reachable yet negative. Next, $\mathcal{H}$ can be seen to lose in every thread w of the left ?-component. Namely, if one of (which is the same as to say that *all of*) the three literals $P^2_w, \overline{P}^3_w, \overline{P}^4_w$ is reachable, then $\mathcal{H}$ loses because it loses in the $\overline{P}^3 \vee \overline{P}^4$ part of the thread; otherwise $\mathcal{H}$ loses because it loses in the P^2 part. Finally, the threads of the right ?-component can be handled in a similar way.

References

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

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

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

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

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

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

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

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

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

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

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

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

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

[14] G. Japaridze. *In the beginning was game semantics*. **Games: Unifying Logic, Language, and Philosophy**. O. Majer, A.-V. Pietarinen and T. Tulenheimo, eds. Springer 2009, pp. 249-350. A preprint is available at http://arxiv.org/abs/cs.LO/0507045

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

[16] G. Japaridze. *Toggling operators in computability logic*. **Theoretical Computer Science** (to appear).

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