Playing with Repetitions in Data Words Using Energy Games

Diego Figueira*
CNRS, LaBRI, France

M. Praveen*
Chennai Mathematical Institute, India
UMI ReLaX

Abstract

We introduce two-player games which build words over infinite alphabets, and we study the problem of checking the existence of winning strategies. These games are played by two players, who take turns in choosing valuations for variables ranging over an infinite data domain, thus generating multi-attributed *data words*. The winner of the game is specified by formulas in the Logic of Repeating Values, which can reason about repetitions of data values in infinite data words. We prove that it is undecidable to check if one of the players has a winning strategy, even in very restrictive settings. However, we prove that if one of the players is restricted to choose valuations ranging over the Boolean domain, the games are effectively equivalent to *single-sided* games on vector addition systems with states (in which one of the players can change control states but cannot change counter values), known to be decidable and effectively equivalent to energy games.

Previous works have shown that the satisfiability problem for various variants of the logic of repeating values is equivalent to the reachability and coverability problems in vector addition systems. Our results raise this connection to the level of games, augmenting further the associations between logics on data words and counter systems.

ACM Reference Format:
Diego Figueira and M. Praveen. 2018. Playing with Repetitions in Data Words Using Energy Games. In *LICS '18: LICS '18: 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, July 9–12, 2018, Oxford, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3209108.3209154

*Authors partially supported by ANR project BRAVAS (grant ANR-17-CE40-0028), and by a grant from the Infosys foundation.

LICS '18, July 9–12, 2018, Oxford, United Kingdom

ACM ISBN 978-1-4503-5583-4/18/07…$15.00
https://doi.org/10.1145/3209108.3209154

1 Introduction

Words over an unbounded domain —known as *data words*— is a structure that appears in many scenarios, as abstractions of timed words, runs of counter automata, runs of concurrent programs with an unbounded number of processes, traces of reactive systems, and more broadly as abstractions of any record of the run of processes handling unbounded resources. Here, we understand data word as a (possibly infinite) word in which every position carries a vector of elements from a possibly infinite domain (*e.g.*, a vector of numbers).

Many specification languages have been proposed to specify properties of data words, both in terms of automata [16, 19] and logics [5, 12–14]. One of the most basic mechanisms for expressing properties on these structures is based on whether a data value at a given position is repeated either *locally* (*e.g.*, in the 2^{nd} component of the vector at the 4^{th} future position), or *remotely* (*e.g.*, in the 1^{st} component of a vector at some position in the past). This has led to the study of linear temporal logic extended with these kind of tests, called *Logic of Repeating Values* (LRV) [10]. The satisfiability problem for LRV is inter-reducible with the reachability problem for Vector Addition Systems with States (VASS), and when the logic is restricted to testing remote repetitions only in the future, it is inter-reducible with the coverability problem for VASS [10, 11]. These connections also extend to data trees and branching VASS [3].

Previous works on data words have been centered around the satisfiability, containment, or model checking problems. Here, we initiate the study of two-player *games* on such structures, motivated by the realizability problem of reactive systems (hardware, operating systems, communication protocols). A reactive system keeps interacting with the environment in which it is functioning, and a data word can be seen as a trace of this interaction. The values of some variables are decided by the system and some by the environment. The reactive system has to satisfy a specified property, given as a logical formula over data words. The *realizability* problem asks whether it is possible that there exists a system that always satisfies the specified property, irrespective of what the environment does. This can be formalized as the existence of a winning strategy for a two-player game that is defined to this end. In this game, there are two sets of variables. Valuations for one set of variables are decided by the system player (representing the reactive system) and for

the other set of variables, valuations are decided by the environment player (representing the environment in which the reactive system is functioning). The two players take turns giving valuations to their respective variables and build an infinite sequence of valuations. The system player wins a game if the resulting sequence satisfies the specified logical formula. Motivated by the *realizability problem* of Church [8], the question of existence of winning strategies in such games are studied extensively (starting from [17]) for the case where variables are Boolean and the logic used is propositional linear temporal logic. To the best of our knowledge there have been no works on the more general setup of infinite domains. This work can be seen as a first step towards considering richer structures, this being the case of an infinite set with an equivalence relation.

Contributions By combining known relations between satisfiability of (fragments of) LRV and (control state) reachability in VASS [10, 11] with existing knowledge about realizability games ([17] and numerous papers expanding on it), it is not difficult to show that realizability games for LRV are related to games on VASS. Using known results about undecidability of games on VASS, it is again not difficult to show that realizability games for LRV are undecidable. Among others, one way to get decidable games on VASS is to make the game asymmetric, letting one player only change control states, while the other player can additionally change values in counters, resulting in the so called single-sided VASS games [2]. Our first contribution in this paper is to identify that the corresponding asymmetry in LRV realizability is to give only Boolean variables to one of the players and let the logic test only for remote repetitions in the past (and disallow testing for remote repetitions in the future). Once this identification of the fragment is made, the proof of its inter-reducibility with single-sided VASS games follows more or less along expected lines by adapting techniques developed in [10, 11].

To obtain the fragment mentioned in the previous paragraph, we impose two restrictions; one is to restrict one of the players to Boolean variables and the other is to disallow testing for remote repetitions in the future. Our next contributions in this paper is to prove that lifting either of these restrictions lead to undecidability. A common feature in similar undecidability proofs (e.g., undecidability of VASS games [1]) is a reduction from the reachability problem for 2-counter machines (details follow in the next section) in which one of the players emulates the moves of the counter machine while the other player catches the first player in case of cheating. Our first undecidability proof uses a new technique where the two players cooperate to emulate the moves of the counter machine and one of the players has the additional task of detecting cheating. Another common feature of similar undecidability proofs is that emulating zero testing transitions of the counter machine is difficult while emulating incrementing and decrementing transitions are easy. Our second undecidability proof uses another new technique in which even emulating decrementing transitions is difficult and requires specific moves by the two players.

Related works The relations between satisfiability of various logics over data words and the problem of language emptiness for automata models have been explored before. In [5], satisfiability of the two variable fragment of first-order logic on data words is related to reachability in VASS. In [12], satisfiability of LTL extended with freeze quantifiers is related to register automata.

A general framework for games over infinite-state systems with a well-quasi ordering is introduced in [1] and the restriction of downward closure is imposed to get decidability. In [18], the two players follow different of rules, making the abilities of the two players asymmetric and leading to decidability. A possibly infinitely branching version of VASS is studied in [6], where decidability is obtained in the restricted case when the goal of the game is to reach a configuration in which one of the counters has the value zero. Games on VASS with inhibitor arcs are studied in [4] and decidability is obtained in the case where one of the players can only increment counters and the other player can not test for zero value in counters. In [7], energy games are studied, which are games on counter systems and the goal of the game is to play for ever without any counter going below zero in addition to satisfying parity conditions on the control states that are visited infinitely often. Energy games are further studied in [2], where they are related to single-sided VASS games, which restrict one of the players to not make any changes to the counters. Closely related perfect half-space games are studied in [9], where it is shown that optimal complexity upper bounds can be obtained for energy games by using perfect half space games.

Organization In Section 2 we define the logic LRV, counter machines, and VASS games. In Section 3 we introduce LRV games. Section 4 shows undecidability results for the fragment of LRV with data repetition tests restricted to past. Section 5 shows the decidability result of past-looking single-sided LRV games. Section 6 shows undecidability of future-looking single-sided LRV games, showing that in some sense the decidability result is maximal. We conclude in Section 7.

2 Preliminaries

We denote by $\mathbb{Z}$ the set of integers and by $\mathbb{N}$ the set of non-negative integers. For any set S, we denote by S^* (resp. S^ω) the set of all finite (resp. countably infinite) sequences of elements in S. For a sequence $\sigma \in S^*$, we denote its length by $|\sigma|$. We denote by $\mathcal{P}(S)$ (resp. $\mathcal{P}^+(S)$) the set of all subsets (resp. non-empty subsets) of S.

Logic of repeating values We recall the syntax and semantics of the logic of repeating values from [10, 11]. This

logic extends the usual propositional linear temporal logic with the ability to reason about repetitions of data values from an infinite domain. We let this logic use both Boolean variables (*i.e.*, propositions) and data variables ranging over an infinite data domain $\mathbb{D}$. The Boolean variables can be simulated by data variables. However, we need to consider fragments of the logic, for which explicitly having Boolean variables is convenient. Let $BVARS = \{q, t, \ldots\}$ be a countably infinite set of Boolean variables ranging over $\{\top, \bot\}$, and let $DVARS = \{x, y, \ldots\}$ be a countably infinite set of 'data' variables ranging over $\mathbb{D}$. We denote by LRV the logic whose formulas are defined as follows:[1]

$$\varphi ::= q \mid x \approx \mathsf{X}^j y \mid x \approx \langle\varphi?\rangle y \mid x \not\approx \langle\varphi?\rangle y \mid x \approx \langle\varphi?\rangle^{-1} y \mid x \not\approx \langle\varphi?\rangle^{-1} y \mid \varphi \wedge \varphi \mid \neg\varphi \mid \mathsf{X}\varphi \mid \varphi \mathsf{U} \varphi \mid \mathsf{X}^{-1}\varphi \mid \varphi \mathsf{S} \varphi \ , \text{ where } q \in BVARS,\ x, y \in DVARS,\ j \in \mathbb{Z}$$

A *valuation* is the union of a mapping from $BVARS$ to $\{\top, \bot\}$ and a mapping from $DVARS$ to $\mathbb{D}$. A *model* is a finite or infinite sequence of valuations. We use σ to denote models and $\sigma(i)$ denotes the i^{th} valuation in σ, where $i \in \mathbb{N} \setminus \{0\}$. For any model σ and position $i \in \mathbb{N} \setminus \{0\}$, the satisfaction relation $\models$ is defined inductively as follows. The temporal operators next (X), previous (X^{-1}), until (U) since (S) and its derived operators (F, G, F^{-1}, G^{-1}, etc.) and Boolean connectives are defined in the usual way and are skipped.

$$\sigma, i \models q : \sigma(i)(q) = \top$$

$$\sigma, i \models x \approx \mathsf{X}^j y \text{ iff } 1 \leq i + j \leq |\sigma|,\ \sigma(i)(x) = \sigma(i+j)(y)$$

$$\sigma, i \models x \approx \langle\varphi?\rangle y \text{ iff } \exists j > i \text{ s.t. } \sigma(i)(x) = \sigma(j)(y),\ \sigma, j \models \varphi$$

$$\sigma, i \models x \not\approx \langle\varphi?\rangle y \text{ iff } \exists j > i \text{ s.t. } \sigma(i)(x) \neq \sigma(j)(y),\ \sigma, j \models \varphi$$

$$\sigma, i \models x \approx \langle\varphi?\rangle^{-1} y \text{ iff } \exists j < i \text{ s.t. } \sigma(i)(x) = \sigma(j)(y),\ \sigma, j \models \varphi$$

$$\sigma, i \models x \not\approx \langle\varphi?\rangle^{-1} y \text{ iff } \exists j < i \text{ s.t. } \sigma(i)(x) \neq \sigma(j)(y),\ \sigma, j \models \varphi$$

for $q \in BVARS$, $x, y \in DVARS$. Intuitively, the formula $x \approx \mathsf{X}^j y$ tests that the data value mapped to the variable x at the current position repeats in the variable y after j positions. We use the notation $\mathsf{X}^i x \approx \mathsf{X}^j y$ as an abbreviation for the formula $\mathsf{X}^i(x \approx \mathsf{X}^{j-i} y)$ (assuming without any loss of generality that $i \leq j$). The formula $x \approx \langle\varphi?\rangle y$ tests that the data value mapped to x now repeats in y at a future position that satisfies the nested formula φ. The formula $x \not\approx \langle\varphi?\rangle y$ is similar but tests for disequality of data values instead of equality. If a model is being built sequentially step by step and these formulas are to be satisfied at a position, they create obligations (for repeating some data values) to be satisfied in some future step. The formulas $x \approx \langle\varphi?\rangle^{-1} y$ and $x \not\approx \langle\varphi?\rangle^{-1} y$ are similar but test for repetitions of data values in past positions.

We append symbols to LRV for denoting syntactic restrictions as shown in the following table. For example, LRV$[\top, \approx, \leftarrow]$ denotes the fragment of LRV in which nested formulas, disequality constraints and future obligations are not allowed. For clarity, we replace $\langle\top?\rangle$ with $\Diamond$ in formulas. E.g., we write $x \approx \langle\top?\rangle y$ as simply $x \approx \Diamond y$.

Symbol	Meaning
$\top$	φ has to be $\top$ in $x \approx \langle\varphi?\rangle y$ (no nested formulas)
$\approx$	disequality constraints ($x \not\approx \langle\varphi?\rangle y$ or $x \not\approx \langle\varphi?\rangle^{-1} y$) are not allowed
$\rightarrow$	past obligations ($x \approx \langle\varphi?\rangle^{-1} y$ or $x \not\approx \langle\varphi?\rangle^{-1} y$) are not allowed
$\leftarrow$	future obligations ($x \approx \langle\varphi?\rangle y$ or $x \not\approx \langle\varphi?\rangle y$) are not allowed

Parity games on integer vectors We recall the definition of games on Vector Addition Systems with States (VASS) from [2]. The game is played between two players: system and environment. A VASS game is a tuple (Q, C, T, π) where Q is a finite set of states, C is a finite set of counters, T is a finite set of transitions and $\pi : Q \to \{1, \ldots, p\}$, for some integer p, is a colouring function that assigns a number to each state. The set Q is partitioned into two parts Q^e (states of environment) and Q^s (states of system). A transition in T is a tuple (q, op, q') where $q, q' \in Q$ are the origin and target states and op is an operation of the form $x++$, $x--$ or nop, where $x \in C$ is a counter. We say that a transition of a VASS game belongs to environment if its origin belongs to environment; similarly for system. A VASS game is *single-sided* if every environment transition is of the form (q, nop, q'). It is assumed that every state has at least one outgoing transition.

A configuration of the VASS game is an element $(q, \vec{n})$ of $Q \times \mathbb{N}^C$, consisting of a state q and a valuation $\vec{n}$ for the counters. A play of the VASS game begins at a designated initial configuration. The player owning the state of the current configuration (say $(q, \vec{n})$) chooses an outgoing transition (say (q, op, q')) and changes the configuration to $(q', \vec{n'})$, where $\vec{n'}$ is obtained from $\vec{n}$ by incrementing (resp. decrementing) the counter x once, if op is $x++$ (resp. $x--$). If $op = nop$, then $\vec{n'} = \vec{n}$. We denote this update as $(q, \vec{n}) \xrightarrow{(q, op, q')} (q', \vec{n'})$. The play is then continued similarly by the owner of the state of the next configuration. If any player wants to take a transition that decrements some counter, that counter should have a non-zero value before the transition. Note that in a single-sided VASS game, environment cannot change the value of the counters. The game continues forever and results in an infinite sequence of configurations $(q_0, \vec{n_0})(q_1, \vec{n_1}) \cdots$. System wins the game if the maximum colour occurring infinitely often in $\pi(q_0)\pi(q_1)\pi(q_2)\cdots$ is even. We assume without loss of generality that from any configuration, at least one transition is enabled (if this condition is not met, we can add extra states and transitions to create an infinite loop ensuring that the owner of the deadlocked configuration loses). In our constructions, we use a generalized form of transitions $q \xrightarrow{\vec{u}} q'$ where $\vec{u} \in \mathbb{Z}^C$, to indicate that each counter

[1] In a previous work [11] this logic was denoted by PLRV (LRV + Past).

c should be updated by adding $\vec{u}(c)$. Such VASS games can be effectively translated into ones of the form defined in the previous paragraph, preserving winning regions.

A strategy se for environment in a VASS game is a mapping $se : (Q \times \mathbb{N}^C)^* \cdot (Q^e \times \mathbb{N}^C) \to T$ such that for all $\gamma \in (Q \times \mathbb{N}^C)^*$, all $q^e \in Q^e$ and all $\vec{n} \in \mathbb{N}^C$, $se(\gamma \cdot (q^e, \vec{n}))$ is a transition whose source state is q^e. A strategy ss for system is a mapping $ss : (Q \times \mathbb{N}^C)^* \cdot (Q^s \times \mathbb{N}^C) \to T$ satisfying similar conditions. Environment plays a game according to a strategy se if the resulting sequence of configurations $(q_0, \vec{n_0})(q_1, \vec{n_1}) \cdots$ is such that for all $i \in \mathbb{N}$, $q_i \in Q^e$ implies $(q_i, \vec{n}_i) \xrightarrow{se((q_0,\vec{n}_0)(q_1,\vec{n}_1)\cdots(q_i,\vec{n}_i))} (q_{i+1}, \vec{n}_{i+1})$. The notion is extended to system player similarly. A strategy ss for system is winning if system wins all the games that she plays according to ss, irrespective of the strategy used by environment. It was shown in [2] that it is decidable to check whether system has a winning strategy in a given single-sided VASS game and an initial configuration. An optimal double exponential upper bound was shown for this problem in [9].

Counter machines A 2-counter machine is a tuple (Q, δ), where Q is a finite set of states and δ is a finite set of transitions. Each transition is a triple of the form (q_1, u, q_2), where $q_1, q_2 \in Q$ and u is either '$c_i --$', '$c_i ++$', or '$c_i = 0?$' for some $i \in \{1, 2\}$. The symbols c_1, c_2 denote counters that the transitions can update. A configuration of the 2-counter machine is a triple (q, n_1, n_2) where $q \in Q$ and $n_1, n_2 \in \mathbb{N}$. The transition relation $\rightarrow$ on configurations is defined as follows. We have $(q_1, n_1, n_2) \rightarrow (q', n_1', n_2')$ iff either: (1) $(q, c_i + +, q') \in \delta$ for $i \in \{1, 2\}$ and $n_i' = n_i + 1$, $n_{3-i}' = n_{3-i}$; (2) $(q, c_i - -, q') \in \delta$ for $i \in \{1, 2\}$ and $n_i > 0$, $n_i' = n_i - 1$, $n_{3-i}' = n_{3-i}$; or (3) $(q, c_i = 0?, q') \in \delta$ for $i \in \{1, 2\}$ and $n_i = 0, (n_1', n_2') = (n_1, n_2)$. A counter machine is *deterministic* if for every configuration (q, n_1, n_2) there exists at most one configuration (q', n_1', n_2') so that $(q, n_1, n_2) \rightarrow (q', n_1', n_2')$. For our undecidability results we will use deterministic 2-counter machines, henceforward just "counter machines". Given a counter machine (Q, δ) and two of its states $q_{init}, q_{fin} \in Q$, the reachability problem is to determine if there is a sequence of transitions of the 2-counter machine starting from the configuration $(q_{init}, 0, 0)$ and ending at the configuration (q_{fin}, n_1, n_2) for some $n_1, n_2 \in \mathbb{N}$. It is known that the reachability problem for 2-counter machines is undecidable [15]. To simplify our undecidability results we further assume, without any loss of generality, that there exists a transition $\hat{t} = (q_{fin}, c_1 + +, q_{fin}) \in \delta$.

3 Game of repeating values

The game of repeating values is played between two players, called environment and system. The set $BVARS$ is partitioned as $BVARS^e, BVARS^s$, owned by environment and system respectively. The set $DVARS$ is partitioned similarly. Let BY^e (resp. DY^e, BY^s, DY^s) be the set of all mappings $bv^e : BVARS^e \to \{\top, \bot\}$ (resp., $dv^e : DVARS^e \to \mathbb{D}$, $bv^s : BVARS^s \to \{\top, \bot\}$, $dv^s : DVARS^s \to \mathbb{D}$). Given two mappings $v_1 : V_1 \to \mathbb{D} \cup \{\top, \bot\}$, $v_2 : V_2 \to \mathbb{D} \cup \{\top, \bot\}$ for disjoint sets of variables V_1, V_2, we denote by $v = v_1 \oplus v_2$ the mapping defined as $v(x_1) = v_1(x_1)$ for all $x_1 \in V_1$ and $v(x_2) = v_2(x_2)$ for all $x_2 \in V_2$. Let Υ^e (resp., Υ^s) be the set of mappings $\{bv^e \oplus dv^e \mid bv^e \in BY^e, dv^e \in DY^e\}$ (resp. $\{bv^s \oplus dv^s \mid bv^s \in BY^s, dv^s \in DY^s\}$). The first round of a game of repeating values is begun by environment choosing a mapping $v_1^e \in \Upsilon^e$, to which system responds by choosing a mapping $v_1^s \in \Upsilon^s$. Then environment continues with the next round by choosing a mapping from Υ^e and so on. The game continues forever and results in an infinite model $\sigma = (v_1^e \oplus v_1^s)(v_2^e \oplus v_2^s) \cdots$. The winning condition is given by a LRV formula φ — system wins iff $\sigma, 1 \models \varphi$.

Let Υ be the set of all valuations. For any model σ and $i > 0$, let $\sigma \restriction i$ denote the valuation sequence $\sigma(1) \cdots \sigma(i)$, and $\sigma \restriction 0$ denote the empty sequence. A strategy for environment is a mapping $te : \Upsilon^* \to \Upsilon^e$. A strategy for system is a mapping $ts : \Upsilon^* \cdot \Upsilon^e \to \Upsilon^s$. We say that environment plays according to a strategy te if the resulting model $(v_1^e \oplus v_1^s)(v_2^e \oplus v_2^s) \cdots$ is such that $v_i^e = te(\sigma \restriction (i-1))$ for all positions $i \in \mathbb{N} \setminus \{0\}$. System plays according to a strategy ts if the resulting model is such that $v_i^s = ts(\sigma \restriction (i-1) \cdot v_i^e)$ for all positions $i \in \mathbb{N} \setminus \{0\}$. A strategy ts for system is winning if system wins all games that she plays according to ts, irrespective of the strategy used by environment. Given a formula φ in (some fragment of) LRV, we are interested in the decidability of checking whether system has a winning strategy in the game of repeating values whose winning condition is φ.

We illustrate the utility of this game with an example. Consider a scenario in which the system is trying to schedule tasks on processors. The number of tasks can be unbounded and task identifiers can be data values. Assuming the variable init carries identifiers of tasks that are initialized and proc carries identifiers of tasks that are processed, the formula G (proc $\approx \Diamond^{-1}$init) specifies that all tasks that are processed must have been initialized beforehand. Assuming the variable log carries identifiers of tasks that have been processed and are being logged into an audit table, the formula G (proc $\approx$ X log) specifies that all processed tasks are logged into the audit table in the next step. Suppose there is a Boolean variable lf belonging to the environment. The formula G ($\neg$lf $\Rightarrow \neg(\log \approx \mathsf{X}^{-1}$ proc)) specifies that if lf is false (denoting that the logger is not working), then the logger can not put the task that was processed in the previous step into the audit table in this step. The combination of the last two specifications is not realizable by any system since as soon as the system processes a task, the environment can make the logger non-functional in the next step. This can be algorithmically determined by the fact that for the conjunction of the last two formulas, there is no winning strategy for system in the game of repeating values.

4 Undecidability of LRV[$\top, \approx, \leftarrow$] games

Here we establish that determining if system has a winning strategy in the LRV[$\top, \approx, \leftarrow$] game is undecidable. This uses a fragment of LRV in which there are no future demands, no disequality demands $\not\approx$, and every sub-formula $x \approx \langle\varphi?\rangle^{-1} y$ is such that $\varphi = \top$. Further, this undecidability result holds even for the case where each player owns only one data variable, and where the distance of local demands is bounded by 3, that is, all local demands of the form $x \approx X^i y$ are so that $-3 \leq i \leq 3$. Simply put, the result shows that bounding the distance of local demands and the number of data variables does not help in obtaining decidability.

Theorem 4.1. *The winning strategy existence problem for the* LRV[$\top, \approx, \leftarrow$] *game is undecidable, even when each player owns only one variable, and the distance of local demands is bounded by* 3.

As we shall see in the next section, if we further restrict the game so that environment does not own any data variable, we obtain decidability.

Undecidability is shown by reduction from the reachability problem for counter machines. The reduction will be first shown for the case where environment owns a data variable y and system owns a data variable x plus some other Boolean variables encoding *labels*. In a second part we show how to eliminate these Boolean variables.

4.1 Reduction with Boolean variables

Lemma 4.2. *The winning strategy existence problem for the* LRV[$\top, \approx, \leftarrow$] *game is undecidable when environment owns one data variable and unboundedly many Boolean variables, and system one (data) variable.*

Proof idea. For convenience, we name the counters of the 2-counter machines c_x and c_y instead of c_1 and c_2. To simulate counters c_x and c_y, we use the environment variable x and system variable y. There are a few more Boolean variables that environment uses for the simulation. We define a LRV[$\top, \approx, \leftarrow$] formula to force environment and system to simulate runs of 2-counter machines as follows. Suppose σ is the concrete model built during a game. The value of counter c_x (resp. c_y) before the i^{th} transition is the cardinality of the set $\{d \in \mathbb{D} \mid \exists j \in \{1, \ldots, i\} : \sigma(j)(x) = d, \forall j' \in \{1, \ldots, i\} : \sigma(j')(y) \neq d\}$ (resp. $\{d \in \mathbb{D} \mid \exists j \in \{1, \ldots, i\}, \sigma(j)(y) = d, \forall j' \in \{1, \ldots, i\}, \sigma(j')(x) \neq d\}$). Intuitively, the value of counter c_x is the number of data values that have appeared under variable x but not under y. In each round, environment chooses the transition of the 2-counter machine to be simulated and sets values for its variables accordingly. If everything is in order, system cooperates and sets the value of the variable y to complete the simulation. Otherwise, system can win immediately by setting the value of y to a value that certifies that the actions of environment violate the semantics of the 2-counter machine. If any player deviates from this behavior at any step, the other player wins immediately. The only other way system can win is by reaching the halting state and the only other way environment can win is by properly simulating the 2-counter machine for ever and never reaching the halting state. □

4.2 Getting rid of Boolean variables

The reduction above makes use of some Boolean variables to encode transitions of the 2-counter machine. However, one can modify the reduction above to do the encoding inside equivalence classes of the variable x. Suppose there are $m-1$ labels that we want to encode. A data word prefix of the form

$$\begin{array}{rccccc} label: & l_1 & l_2 & & & l_n \\ x: & x_1 & x_2 & \cdots & & x_n \\ y: & y_1 & y_2 & & & y_n \end{array}$$

where l_i, x_i, y_i are, respectively, the label, value of x, and value of y at position i, is now encoded as

$$\begin{array}{lccccccccccccc} x: & d & d & & x_1 & d & d & & x_2 & d & d & \cdots & d & d & & x_n & d & d \\ y: & d & d & w_1 & y_1 & d & d & w_2 & y_2 & d & d & & d & d & w_n & y_n & d & d \end{array}$$

where each w_i is a data word of the form $(d_1, d_1) \cdots (d_m, d_m)$; further the data values of w_i are so that $d \notin \{d_1, \ldots, d_m\}$, and so that every pair of w_i, w_j with $i \neq j$ has disjoint sets of data values. The purpose of w_i is to encode the label l_i; the purpose of the repeated data value (d, d) is to delimit the boundaries of each encoding of a label, which we will call a 'block'; the purpose of repeating (d, d) at each occurrence is to avoid confusing this position with the encoding position (x_i, y_i) —*i.e.*, a boundary position is one whose data value is repeated at distance m+3 *and* at distance 1.

This encoding can be enforced using a LRV formula. Further, the encoding of values of counters in the reduction before is not broken since the additional positions have the property of having the same data value under x as under y, and in this the encoding of counter c_x —*i.e.*, the number of data values that have appeared under x but not under y— is not modified; similarly for counter c_y.

Lemma 4.3. *The winning strategy existence problem for the* LRV[$\top, \approx, \leftarrow$] *game is undecidable when system and environment owns one (data) variable each and no Boolean variables.*

Unbounded local tests The previous undecidability use either an unbounded number of variables or a bounded number of variables but an unbounded X-distance of local demands. However, through a more clever encoding one can avoid testing whether two positions at distance n have the same data value by a chained series of tests. This is a standard coding which does not break the 2-counter machine reduction. Then we obtain the following, which proves the theorem.

Lemma 4.4. *The winning strategy existence problem for the* LRV[$\top, \approx, \leftarrow$] *game is undecidable when system and environment own only one variable each, and the distance of local data repetition demands is bounded by 3.*

5 Decidability of single-sided LRV[⊤, ←]

In this section we show that the single-sided LRV[⊤, ←]-game is decidable. We first observe that we do not need to consider $\not\approx$ formulas for our decidability argument, since there is a reduction of the winning strategy existence problem that removes all sub-formulas of the from $x \not\approx \Diamond^{-1} y$.

Proposition 5.1. *There is a polynomial-time reduction from the winning strategy existence problem for* LRV[⊤, ←] *into the problem on* LRV[⊤, ≈, ←].

This is done as it was done for the satisfiability problem [11, Proposition 4]. The key observation is that

- $\neg(x \not\approx \Diamond^{-1} y)$ is equivalent to $\neg \mathsf{X}^{-1}\top \vee (x \approx \mathsf{X}^{-1} y \wedge \mathsf{G}^{-1}(\neg \mathsf{X}^{-1}\top \vee y \approx \mathsf{X}^{-1} y))$;
- $x \not\approx \Diamond^{-1} y$ can be translated into $\neg(x \approx x_{\approx \Diamond^{-1} y}) \wedge x_{\approx \Diamond^{-1} y} \approx \Diamond^{-1} y$ for a new variable $x_{\approx \Diamond^{-1} y}$ belonging to the same player as x.

Given a formula φ in negation normal form (*i.e.*, negation is only applied to boolean variables and data tests), consider the formula φ' resulting from the replacements listed above. It follows that φ' does not make use of $\not\approx$. It is easy to see that there is a winning strategy for system in the game with winning condition φ if and only if she has a winning strategy for the game with condition φ'.

We consider games where environment has only Boolean variables while system player has data variables. We call this the single-sided LRV[⊤, ←] games and show that winning strategy existence problem is decidable. The main concept we use for decidability is a symbolic representation of models, introduced in [10]. The building blocks of the symbolic representation are *frames*, which we adapt here. We finally show effective reductions between single-sided LRV[⊤, ←] games and single-sided VASS games. This implies decidability of single-sided LRV[⊤, ←] games. From Proposition 5.1, it suffices to show effective reductions between single-sided LRV[⊤, ≈, ←] games and single-sided VASS games.

Given a formula in LRV[⊤, ≈, ←], we replace sub-formulas of the form $x \approx \mathsf{X}^{-j} y$ with $\mathsf{X}^{-j}(y \approx \mathsf{X}^{j} x)$ if $j > 0$. For a formula φ obtained after such replacements, let l be the maximum i such that a term of the form $\mathsf{X}^i x$ appears in φ. We call l the X-length of φ. Let $BVARS^\varphi \subseteq BVARS$ and $DVARS^\varphi \subseteq DVARS$ be the set of Boolean and data variables used in φ. Let Ω_l^φ be the set of constraints of the form $\mathsf{X}^i q$, $\mathsf{X}^i x \approx \mathsf{X}^j y$ or $\mathsf{X}^i(x \approx \Diamond^{-1} y)$, where $q \in BVARS^\varphi$, $x, y \in DVARS^\varphi$ and $i, j \in \{0, \ldots, l\}$. For $e \in \{0, \ldots, l\}$, an (e, φ)-frame is a set of constraints $fr \subseteq \Omega_l^\varphi$ that satisfies the following conditions:

(F0) For all constraints $\mathsf{X}^i q, \mathsf{X}^i x \approx \mathsf{X}^j y, \mathsf{X}^i(x \approx \Diamond^{-1} y) \in fr$, $i, j \in \{0, \ldots, e\}$.
(F1) For all $i \in \{0, \ldots, e\}$ and $x \in DVARS^\varphi$, $\mathsf{X}^i x \approx \mathsf{X}^i x \in fr$.
(F2) For all $i, j \in \{0, \ldots, e\}$ and $x, y \in DVARS^\varphi$, $\mathsf{X}^i x \approx \mathsf{X}^j y \in fr$ iff $\mathsf{X}^j y \approx \mathsf{X}^i x \in fr$.
(F3) For all $i, j, j' \in \{0, \ldots, e\}$ and $x, y, z \in DVARS^\varphi$, if $\{\mathsf{X}^i x \approx \mathsf{X}^j y, \mathsf{X}^j y \approx \mathsf{X}^{j'} z\} \subseteq fr$, then $\mathsf{X}^i x \approx \mathsf{X}^{j'} z \in fr$.
(F4) For all $i, j \in \{0, \ldots, e\}$ and $x, y \in DVARS^\varphi$ such that $\mathsf{X}^i x \approx \mathsf{X}^j y \in fr$:
  - if $i = j$, then for every $z \in DVARS^\varphi$ we have $\mathsf{X}^i(x \approx \Diamond^{-1} z) \in fr$ iff $\mathsf{X}^j(y \approx \Diamond^{-1} z) \in fr$.
  - if $i < j$, then $\mathsf{X}^j(y \approx \Diamond^{-1} x) \in fr$ and for any $z \in DVARS^\varphi$, $\mathsf{X}^j(y \approx \Diamond^{-1} z) \in fr$ iff either $\mathsf{X}^i(x \approx \Diamond^{-1} z) \in fr$ or there exists $i \le j' < j$ with $\mathsf{X}^j y \approx \mathsf{X}^{j'} z \in fr$.

The condition (F0) ensures that a frame can constrain at most $(e + 1)$ contiguous valuations. The next three conditions ensure that equality constraints in a frame form an equivalence relation. The last condition ensures that obligations for repeating values in the past are consistent among various variables.

A pair of (l, φ)-frames (fr, fr') is said to be one-step consistent iff

(O1) for all $\mathsf{X}^i x \approx \mathsf{X}^j y \in \Omega_l^\varphi$ with $i, j > 0$, we have $\mathsf{X}^i x \approx \mathsf{X}^j y \in fr$ iff $\mathsf{X}^{i-1} x \approx \mathsf{X}^{j-1} y \in fr'$,
(O2) for all $\mathsf{X}^i(x \approx \Diamond^{-1} y) \in \Omega_l^\varphi$ with $i > 0$, we have $\mathsf{X}^i(x \approx \Diamond^{-1} y) \in fr$ iff $\mathsf{X}^{i-1}(x \approx \Diamond^{-1} y) \in fr'$ and
(O3) for all $\mathsf{X}^i q \in \Omega_l^\varphi$ with $i > 0$, we have $\mathsf{X}^i q \in fr$ iff $\mathsf{X}^{i-1} q \in fr'$.

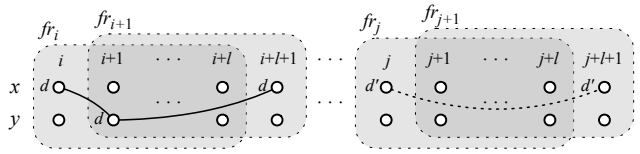
For $e \in \{0, \ldots, l-1\}$, an (e, φ) frame fr and an $(e+1, \varphi)$ frame fr', the pair (fr, fr') is said to be one step consistent iff $fr \subseteq fr'$ and for every constraint in fr' of the form $\mathsf{X}^i x \approx \mathsf{X}^j y$, $\mathsf{X}^i q$ or $\mathsf{X}^i(x \approx \Diamond^{-1} y)$ with $i, j \in \{0, \ldots, e\}$, the same constraint also belongs to fr.

An (infinite) (l, φ)-symbolic model ρ is an infinite sequence of (l, φ)-frames such that for all $i \in \mathbb{N}$, the pair $(\rho(i), \rho(i+1))$ is one-step consistent. Let us define the symbolic satisfaction relation $\rho, i \models_{symb} \varphi'$ where φ' is a sub-formula of φ. The relation $\models_{symb}$ is defined in the same way as $\models$ for LRV, except that for every element φ' of Ω_l^φ, we have $\rho, i \models_{symb} \varphi'$ whenever $\varphi' \in \rho(i)$. We say that a concrete model σ realizes a symbolic model ρ if for every $i \in \mathbb{N} \setminus \{0\}$, $\rho(i) = \{\varphi' \in \Omega_l^\varphi \mid \sigma, i \models \varphi'\}$. The next result follows easily from definitions.

Lemma 5.2 (symbolic vs. concrete models)**.** *Suppose φ is a* LRV[⊤, ≈, ←] *formula of* X*-length l, ρ is a (l, φ)-symbolic model and σ is a concrete model realizing ρ. Then ρ symbolically satisfies φ iff σ satisfies φ.*

We fix a LRV[⊤, ≈, ←] formula φ of X-length l. For $e \in \{0, \ldots, l\}$, an (e, φ)-frame fr, $i \in \{0, \ldots, e\}$ and a variable x, the set of past obligations of the variable x at level i in fr is defined to be the set $\mathrm{PO}_{fr}(x, i) = \{y \in DVARS^\varphi \mid \mathsf{X}^i(x \approx \Diamond^{-1} y) \in fr\}$. The equivalence class of x at level i in fr is defined to be $[(x, i)]_{fr} = \{y \in DVARS^\varphi \mid \mathsf{X}^i x \approx \mathsf{X}^i y \in fr\}$.

Consider a concrete model σ restricted to two variables x, y as shown below. The top row indicates the positions $i, (i+1), \ldots, (i+l), (i+l+1), j, (j+1), \ldots, (j+l), (j+l+1)$.



The left column indicates the two variables x, y and the remaining columns indicate valuations. E.g., $\sigma(i+1)(y) = d$ and $\sigma(j+l+1)(x) = d'$. Let $fr_i = \{\varphi' \in \Omega_l^\varphi \mid \sigma, i \models \varphi'\}$. We have indicated this pictorially by highlighting the valuations that determine the contents of fr_i. The data values for x at positions i and $(i+l+1)$ are equal, but the positions are too far apart to be captured by any one constraint of the form $\mathsf{X}^\alpha x \approx \mathsf{X}^\beta x$ in Ω_l^φ. However, the intermediate position $(i+1)$ has the same data value and is less than l positions apart from both positions. One constraint from Ω_l^φ can capture the data repetition between positions i and $(i+1)$ while another one captures the repetition between positions $(i+1)$ and $(i+l+1)$, thus indirectly capturing the repetition between positions i and $(i+l+1)$. For $e \in \{0, \ldots, l\}$, an (e, φ)-frame fr, $i \in \{0, \ldots, e\}$ and a variable x, we say that there is a forward (resp. backward) reference from (x, i) in fr if $\mathsf{X}^i x \approx \mathsf{X}^{i+j} y \in fr$ (resp. $\mathsf{X}^i x \approx \mathsf{X}^{i-j} y \in fr$) for some $j > 0$ and $y \in DVARS^\varphi$. The constraint $x \approx \mathsf{X}y$ in fr_i above is a forward reference from $(x, 0)$ in fr_i, while the constraint $\mathsf{X}^l x \approx y$ is a backward reference from (x, l) in fr_{i+1}.

In the above picture, the data values of x at positions j and $(j+l+1)$ are equal, but the two positions are too far apart to be captured by any constraint of the form $\mathsf{X}^\alpha z \approx \mathsf{X}^\beta w$ in Ω_l^φ. Neither are there any intermediate positions with the same data value to capture the repetition indirectly. We maintain a counter to keep track of the number of such remote data repetitions. Let $X \subseteq DVARS^\varphi$ be a set of variables. A *point of decrement* for counter X in an (e, φ)-frame fr is an equivalence class of the form $[(x, e)]_{fr}$ such that there is no backward reference from (x, e) in fr and $\mathsf{PO}_{fr}(x, e) = X$. In the above picture, the equivalence class $[(x, l)]_{fr_{j+1}}$ in the frame fr_{j+1} is a point of decrement for $\{x\}$. A *point of increment for X in an (l, φ)-frame fr* is an equivalence class of the form $[(x, 0)]_{fr}$ such that there is no forward reference from $(x, 0)$ in fr and $[(x, 0)]_{fr} \cup \mathsf{PO}_{fr}(x, 0) = X$. In the above picture, the equivalence class $[(x, 0)]_{fr_j}$ in the frame fr_j is a point of increment for $\{x\}$. Points of increment are not present in (e, φ)-frames for $e < l$ since such frames do not contain complete information about constraints in the next l positions. We denote by $inc(fr)$ the vector indexed by non-empty subsets of $DVARS^\varphi$, where each coordinate contains the number of points of increment in fr for the corresponding subset of variables. Similarly, we have the vector $dec(fr)$ for points of decrement.

Given a LRV$[\top, \approx, \leftarrow]$ formula φ in which $DVARS^e = \emptyset = BVARS^s$, we construct a single-sided VASS game as follows. Let l be the X-length of φ and FR be the set of all (e, φ)-frames for all $e \in \{0, \ldots, l\}$. Let A^φ be a deterministic parity automaton that accepts a symbolic model iff it symbolically satisfies φ, with set of states Q^φ and initial state q_{init}^φ. The single-sided VASS game will have set of counters $\mathcal{P}^+(DVARS^\varphi)$, set of environment states $\{-1, 0, \ldots, l\} \times Q^\varphi \times (\mathsf{FR} \cup \{\bot\})$ and set of system states $\{-1, 0, \ldots, l\} \times Q^\varphi \times (\mathsf{FR} \cup \{\bot\}) \times \mathcal{P}(BVARS^\varphi)$. Every state will inherit the colour of its Q^φ component. For convenience, we let $\bot$ to be the only $(-1, \varphi)$-frame and $(\bot, fr')$ be one-step consistent for every 0-frame fr'. The initial state is $(-1, q_{init}^\varphi, \bot)$, the initial counter values are all 0 and the transitions are as follows ($\lceil\cdot\rceil l$ denotes the mapping that is identity on $\{-1, 0, \ldots, l-1\}$ and maps all others to l).

- $(e, q, fr) \xrightarrow{\vec{0}} (e, q, fr, V)$ for every $e \in \{-1, 0, \ldots, l\}$, $q \in Q^\varphi$, $fr \in \mathsf{FR} \cup \{\bot\}$ and $V \subseteq BVARS^\varphi$.
- $(e, q_{init}^\varphi, fr, V) \xrightarrow{inc(fr) - dec(fr')} (e+1, q_{init}^\varphi, fr')$ for every $V \subseteq BVARS^\varphi$, $e \in \{-1, 0, \ldots, l-2\}$, (e, φ)-frame fr and $(e+1, \varphi)$-frame fr', where the pair (fr, fr') is one-step consistent and $\{p \in BVARS^\varphi \mid \mathsf{X}^{e+1} p \in fr'\} = V$.
- $(e, q, fr, V) \xrightarrow{inc(fr) - dec(fr')} (\lceil e+1 \rceil l, q', fr')$ for every $e \in \{l-1, l\}$, (e, φ)-frame fr, $V \subseteq BVARS^\varphi$, $q, q' \in Q^\varphi$ and $(\lceil e+1 \rceil l, \varphi)$-frame fr', where the pair (fr, fr') is one-step consistent, $\{p \in BVARS^\varphi \mid \mathsf{X}^{\lceil e+1 \rceil l} p \in fr'\} = V$ and $q \xrightarrow{fr'} q'$ is a transition in A^φ.

Transitions of the form $(e, q, fr) \xrightarrow{\vec{0}} (e, q, fr, V)$ let the environment choose any subset V of $BVARS^\varphi$ to be true in the next round. In transitions of the form $(e, q, fr, V) \xrightarrow{inc(fr) - dec(fr')} (\lceil e+1 \rceil l, q', fr')$, the condition $\{p \in BVARS^\varphi \mid \mathsf{X}^{\lceil e+1 \rceil l} p \in fr'\} = V$ ensures that the frame fr' chosen by the system is compatible with the subset V of $BVARS^\varphi$ chosen by the environment in the preceding step. By insisting that the pair (fr, fr') is one-step consistent, we ensure that the sequence of frames built during a game is a symbolic model. The condition $q \xrightarrow{fr'} q'$ ensures that the symbolic model is accepted by A^φ and hence symbolically satisfies φ. The update vector $inc(fr) - dec(fr')$ ensures that symbolic models are realizable, as explained in the proof of the following result.

Lemma 5.3 (repeating values to VASS)**.** *Let φ be a* LRV$[\top, \approx, \leftarrow]$ *formula with $DVARS^e = BVARS^s = \emptyset$. Then system has a winning strategy in the corresponding single-sided* LRV$[\top, \approx, \leftarrow]$ *game iff she has a winning strategy in the single-sided VASS game constructed above.*

Proof idea. A game on the single-sided VASS game results in a sequence of frames. The single-sided VASS game embeds automata which check that these sequences are symbolic models that symbolically satisfy φ. This in conjunction with Lemma 5.2 (symbolic vs. concrete models) will prove the result, provided the symbolic models are also realizable. Some symbolic models are not realizable since frames contain too many constraints about data values repeating in the past and no concrete model can satisfy all those constraints. To

avoid this, the single-sided VASS game maintains counters for keeping track of the number of such constraints. Whenever a frame contains such a past repetition constraint that is not satisfied locally within the frame itself, there is an absence of backward references in the frame and it results in a point of decrement. Then the $-dec(fr')$ part of transitions of the form $(e, q, fr, V) \xrightarrow{inc(fr)-dec(fr')} (\lceil e+1 \rceil_l, q', fr')$ will decrement the corresponding counter. In order for this counter to have a value of at least 0, the counter should have been incremented earlier by $inc(fr)$ part of earlier transitions. This ensures that symbolic models resulting from the single-sided VASS games are realizable. □

Corollary 5.4. *The winning strategy existence problem for single-sided* LRV$[\top, \approx, \leftarrow]$ *game of repeating values (without past-time temporal modalities) is in* 4Exptime.

Proof. For a LRV$[\top, \approx, \leftarrow]$ formula with $DVARS^e = BVARS^s = \emptyset$ and no past-time temporal modalities, the single-sided VASS game defined above can be constructed in 2Exptime. Hence, the double exponential time upper bound for energy games (and for single-sided VASS games) given in [9] translates to 4Exptime for single-sided LRV$[\top, \approx, \leftarrow]$ games. □

Our decidability proof thus depends ultimately on energy games, as hinted in the title of this paper. Next we show that single-sided VASS games can be effectively reduced to single-sided LRV$[\top, \approx, \leftarrow]$ games.

Theorem 5.5. *Given a single-sided VASS game, a single-sided* LRV$[\top, \approx, \leftarrow]$ *game can be constructed in polynomial time so that the system player has a winning strategy in the first game iff the system player has a winning strategy in the second one.*

Proof idea. We will simulate runs of single-sided VASS games with models of formulas in LRV. The formulas satisfied at position i of the concrete model will contain information about counter values before the i^{th} transition and the identity of the i^{th} transition chosen by the environment and the system players in the run of the single-sided VASS game. For simulating a counter x, we use two system variables x and $\overline{x}$. The data values assigned to these variables from positions 1 to i in a concrete model σ will represent the counter value that is equal to the cardinality of the set $\{d \in \mathbb{D} \mid \exists j \in \{1, \ldots, i\}, \sigma(j)(x) = d, \forall j' \in \{1, \ldots, i\}, \sigma(j')(\overline{x}) \neq d\}$. Using formulas in LRV$[\top, \approx, \leftarrow]$, the two players can be enforced to correctly update the concrete model to faithfully reflect the moves in the single-sided VASS game. A formula can also be written to ensure that system wins the single-sided LRV$[\top, \approx, \leftarrow]$ game iff the single-sided VASS game being simulated satisfies the parity condition. □

6 Single-sided LRV$[\top, \approx, \rightarrow]$ is undecidable

In this section we show that the positive decidability result for the single-sided LRV$[\top, \leftarrow]$ game cannot be replicated for the future demands fragment, even in a restricted setting.

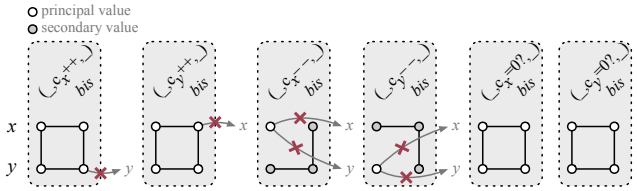
Theorem 6.1. *The existence of winning strategy for single-sided* LRV$[\top, \approx, \rightarrow]$ *games is undecidable, even when environment has 1 Boolean variable and system has 3 data variables.*

We don't know the decidability status for the case where system has less than three data variables.

As in the previous undecidability results in Section 4, the result is proven by a reduction from the reachability problem for 2-counter machines. System makes use of *labels* to encode the sequence of transitions of a witnessing run of the counter machine. This time, system has 3 data variables x, y, z (in addition to a number of Boolean variables which encode the labels); and environment has just one Boolean variable b. Variables x, y are used to encode the counters c_x and c_y as before, and variables z, b are used to ensure that there are no 'illegal' transitions — namely, no decrements of a zero-valued counter, and no tests for zero for a non-zero-valued counter.

Each transition in the run of the 2-counter machine will be encoded using *two* consecutive positions of the game. Concretely, while in the previous coding of Section 4 a witnessing reachability run $t_1\, t_2 \cdots t_n \in \delta^*$ was encoded with the label sequence $begin\, t_1\, t_2 \cdots t_n \hat{t}^\omega$, in this encoding transitions are interspersed with a special *bis* label, and thus the run is encoded as $t_1\, bis\, t_2\, bis \cdots t_n\, bis\, (\hat{t}\, bis)^\omega \in (\delta \cup \{bis\})^\omega$.

Suppose a position has the label of a c_x + + transition and the variable x has the data value d. Our encoding will ensure that if the data value d repeats in the future, it will be only once and at a position that has the label of a c_x − − transition. A symmetrical property holds for c_y and variable y. The value of counter c_x (resp. c_y) before the i^{th} transition (encoded in the $2i^{th}$ and $(2i+1)^{st}$ positions) is the number of positions $j < 2i$ satisfying the following two conditions: i) the position j should have the label of a c_x + + transition and ii) $\sigma(j)(x) \notin \{\sigma(j')(x) \mid j+1 < j' < 2i\}$. Intuitively, if $2i$ is the current position, the value of c_x (resp. c_y) is the number of previous positions that have the label of a c_x + + transition whose data value is not yet matched by a position with the label of a c_x − − transition. In this reduction we assume that system plays first and environment plays next at each round, since it is easier to understand (the reduction also holds for the game where turns are inverted by shifting environment behavior by one position). At each round, system will play a label *bis* if the last label played was a transition. Otherwise, she will choose the next transition of the 2-counter machine to simulate and she will chose the values for variables x, y, z in such a way that the aforementioned encoding for counters c_x and c_y is preserved. To this end, system is bound by the following set of rules, described here pictorially:



The first (leftmost) rule, for example, reads that whenever there is a $c_x{+}{+}$ transition label, then all four values for x and y in both positions (*i.e.*, the transition position and the next *bis* position) must have the same data value d (which we call 'principal'), which does not occur in the future under variable y. The third rule says that $c_x - -$ is encoded by having x on the first position to carry the 'principal' data value d of the transition, which is *final* (that is, it is not repeated in the future under x or y), and all three remaining positions have the same data value d' different from d. In this way, system can make sure that the value of c_x is decremented, by playing a data value d that has occurred in a $c_x + +$ position that is not yet matched. (While system could also play some data value which does not match any previous $c_x + +$ position, this 'illegal' move leads to a losing play for system, as we will show.) In this rule, the usage of two positions per transition becomes essential: it ensures that the data value d' of y (for which $d' \neq d$) appears in the future both in x and y. Thus, the presence of d' doesn't affect the value of c_y or c_x —to affect either, the data value should repeat in only one variable.

From these rules, it follows that every $c_k + +$ can be matched to at most one future $c_k - -$. However, there can be two ways in which this coding can fail: a) there could be invalid tests $c_k = 0?$, that is, a situation in which the preceding positions of the test contain a $c_k + +$ transition which is not matched with a $c_k - -$ transition; and b) there could be some $c_k - -$ with no previous matching $c_k + +$. As we will see next, variables z and b play a crucial role in the game whenever any of these two cases occurs. In all the rounds, environment always plays $\top$, except if he detects that one of these two situations, a) or b), have arisen, in which case he plays $\bot$. In the following rounds system plays a value in z that will enable to test, with an LRV formula, if there was indeed an a) or b) situation, in which case system will lose, or if environment was just 'bluffing', in which case system will win. Since this is the most delicate point in the reduction, we dedicate the remaining of this section to the explanation of how these two situations a) and b) are treated.

Remember that environment has just *one bit* of information to play with. The LRV property we build ensures that the sequence of b-values must be from the set $\top^*\bot^*\top^\omega$.

a) Avoiding illegal tests for zero. Suppose that at some point of the 2-counter machine simulation, system decides to play a $c_k = 0?$ transition. Suppose there is some preceding $c_k + +$ transition for which either: a1) there is no matching $c_k - -$ transition; or a2) there is a matching $c_k - -$ transition but it occurs after the $c_k = 0?$ transition. Situation a1 can be easily avoided by ensuring that any winning play must satisfy the formula $\mu = \mathsf{G}(\tau_{(c_k++)} \wedge \mathsf{F}\tau_{(c_k=0?)} \Rightarrow k \approx \Diamond k)$ for every $k \in \{x, y\}$. Here, τ_{inst} tests if the current position is labelled with an instruction of type $inst$. On the other hand, Situation a2 requires environment to play a certain strategy (represented in Figure 1-a2). This means that c_k is non-zero at the position of the $c_k = 0?$ transition, and that this is an illegal transition; thus, environment must respond accordingly. Further, suppose this is the *first* illegal transition that has occurred so far. Environment, who so far has been playing only $\top$, decides to play $\bot$ to mark the cheating point. Further, he will continue playing $\bot$ until the matching $c_k - -$ transition is reached (if it is never reached, it is situation a1 and system loses as explained before), after which he will play $\top$ forever afterwards. In some sense, environment provides a *link* between the illegal transition and the proof of its illegality through a $\bot^*$-path. The following characterizes environment's denouncement of an illegal test for zero:

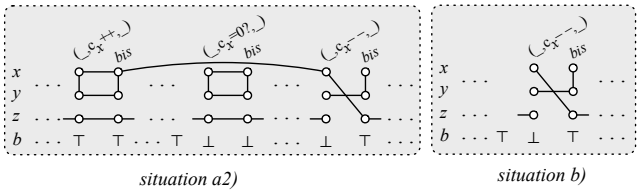


Figure 1. Depiction of best strategies in both situations.

Property 1: b becomes $\bot$ at a $c_k = 0?$ position and stops being $\bot$ at a $c_k - -$ position thereafter.

Note that Property 1 is clearly definable by a formula π_1 of LRV$[\top, \approx, \rightarrow]$. If Property 1 holds, a formula φ_1 can constrain system to play z according to the following: z always carries the same data value, distinct from the values of all other variables, but as soon as the last $\bot$ value is played, which has to be on a $c_k - -$ position, the value of z changes and holds the principal value of that $c_k - -$ transition,[2] and it continues to hold that value forever after (*cf.* Figure 1-*a*2). Further, if environment cheated in his denouncement by linking a $c_k = 0?$ transition with a future $c_k - -$ with a matching $c_k + +$ that falls in-between the test for zero and the decrement, then a property π_1' can catch this: there exists a $c_k + +$ with $\bot$ whose principal value matches that of a future z-value.

Finally, assuming environment correctly denounced an illegal test for zero and system played accordingly on variable z, a property φ_1' can test that environment exposed an illegal transition, by testing that there exists a $c_k + +$ transition whose principal value corresponds to the z-value of some future position. Thus, the encoding for this situation is expressed with the formula $\psi_1 = \mu \wedge ((\pi_1 \wedge \neg\pi_1') \Rightarrow (\varphi_1 \wedge \neg\varphi_1'))$.

b) Avoiding illegal decrements. Suppose now that at some point of the 2-counter machine simulation, system decides to play a $c_k - -$ transition for which there is no preceding $c_k{+}{+}$ transition matching its final data value. This is a form of cheating, and thus environment should respond accordingly. Further, suppose this is the first cheating that has occurred so

[2]To make sure it is the *last* $\bot$ element, system has to wait for $\top$ to appear, hence variable z changes its value at the next position after the last $\bot$.

far. Environment, who so far has been playing only $\top$, decides then to mark this position with $\bot$; and for the remaining of the play environment plays only $\top$ (even if more illegal transitions are performed in the sequel). Summing up, for this situation environment's best strategy has a value sequence from $\top^*\bot\top^\omega$, and this property characterizes environment's denouncement of an illegal decrement (*cf.* Figure 1-*b*).

Property 2: b becomes $\bot$ at a c_k-- position and stops being $\bot$ immediately after.

A formula π_2 can test Property 2; and a formula φ_2 can constrain variable z to always have the same data value — distinct from all other data values played on variables x, y— while b contains $\top$ values; and as soon as b turns to $\bot$ on a c_k-- position, then z at the next position takes the value of the current variable k, and maintains that value (*cf.* Figure 1-*b*). Further, a formula φ_2' tests that in this case there must be some c_k++ position with a data value equal to variable z of a future position. The formula corresponding to this case is then $\psi_2 = \pi_2 \Rightarrow \varphi_2 \wedge \varphi_2'$.

The final formula to test is then of the form $\varphi = \varphi_{lab} \wedge \varphi_{x,y} \wedge \psi_1 \wedge \psi_2$, where φ_{lab} ensures the finite-automata behavior of labels, and in particular that a final state can be reached, and $\varphi_{x,y}$ asserts the correct behavior of the variables x, y relative to the labels. It follows that system has a winning strategy for the game with input φ if, and only if, there is a positive answer to the reachability problem for the 2-counter machine. Finally, labels can be eliminated by means of special data values encoding *blocks* exactly as done in Section 4.2, and in this way Theorem 6.1 follows.

7 Conclusion

It remains open whether the 4Exptime upper bound given in Corollary 5.4 is optimal. The satisfiability problem for propositional LTL is complete for Pspace and the realizability problem is complete for exponential of alternating Pspace, which is 2Exptime. Since the satisfiability of LRV is complete for 2Expspace, it would be surprising if games on LRV have upper bounds smaller than exponential of alternating 2Expspace, which is 4Exptime. For the decidability result in Section 5, we assumed that in any sub-formula of the form $x \approx \langle\varphi?\rangle^{-1}y$, φ is $\top$. We believe that this assumption can possibly be removed if we maintain counters for (variable, formula) pairs instead of variables. We leave the technical details of this extension for future work. An open question is the decidability status of single-sided games with future obligations restricted to only two data variables; the reduction we have in Section 6 needs three.

Some future directions for research on this topic include finding restrictions other than single-sidedness to get decidability. For the decidable cases, the structure of winning strategies can be studied, e.g., whether memory is needed and if yes, how much.

Acknowledgments The authors thank Stéphane Demri and Prakash Saivasan for useful discussions, and anonymous reviewers for their constructive feedback, which helped us correct some omissions and improve the presentation.

References

[1] P. A. Abdulla, A. Bouajjani, and J. D'orso. 2008. Monotonic and Downward Closed Games. *Journal of Logic and Computation* 18, 1 (2008), 153–169. https://doi.org/10.1093/logcom/exm062

[2] P. A. Abdulla, R. Mayr, A. Sangnier, and J. Sproston. 2013. Solving Parity Games on Integer Vectors. In *CONCUR 2013 (LNCS)*, Vol. 8052. Springer Berlin Heidelberg, 106–120. https://doi.org/10.1007/978-3-642-40184-8_9

[3] S. Abriola, D. Figueira, and S. Figueira. 2017. Logics of Repeating Values on Data Trees and Branching Counter Systems. In *FOSSACS'17*, LNCS Vol. 10203. 196–212. https://doi.org/10.1007/978-3-662-54458-7_12

[4] B. Bérard, S. Haddad, M. Sassolas, and N. Sznajder. 2012. Concurrent Games on VASS with Inhibition. In CONCUR'12, LNCS Vol. 7454. Springer, 39–52. https://doi.org/10.1007/978-3-642-32940-1_5

[5] M. Bojańczyk, C. David, A. Muscholl, Th. Schwentick, and L. Segoufin. 2011. Two-variable logic on data words. *ACM Transactions on Computational Logic* 12, 4 (2011), 27. http://doi.acm.org/10.1145/1970398.1970403

[6] T. Brázdil, P. Jančar, and A. Kučera. 2010. *Reachability Games on Extended Vector Addition Systems with States.* In *ICALP'10*, LNCS Vol. 6199. Springer, 478–489. https://doi.org/10.1007/978-3-642-14162-1_40

[7] K. Chatterjee, M. Randour, and J.-F. Raskin. 2014. Strategy synthesis for multi-dimensional quantitative objectives. *Acta Informatica* 51, 3 (June 2014), 129–163. https://doi.org/10.1007/s00236-013-0182-6

[8] A. Church. 1962. Logic, arithmetic and automata. In *Proceedings of the international congress of mathematicians*, Vol. 1962. 23–35.

[9] T. Colcombet, M. Jurdziński, R. Lazić, and S. Schmitz. 2017. Perfect half space games. In *LICS 2017*. 1–11. https://doi.org/10.1109/LICS.2017.8005105

[10] S. Demri, D. D'Souza, and R. Gascon. 2012. Temporal Logics of Repeating Values. *Journal of Logic and Computation* 22, 5 (2012), 1059–1096. https://doi.org/10.1093/logcom/exr013

[11] S. Demri, D. Figueira, and M. Praveen. 2016. Reasoning about Data Repetitions with Counter Systems. *Logical Methods in Computer Science* Volume 12, Issue 3, 1–55. https://doi.org/10.2168/LMCS-12(3:1)2016

[12] S. Demri and R. Lazić. 2009. LTL with the freeze quantifier and register automata. *ACM Transactions on Computational Logic* 10, 3 (2009). https://doi.org/10.1145/1507244.1507246

[13] D. Figueira. 2011. A decidable two-way logic on data words. In *LICS'11*. IEEE, 365–374. https://doi.org/10.1109/LICS.2011.18

[14] A. Kara, Th. Schwentick, and Th. Zeume. 2010. Temporal Logics on Words with Multiple Data Values. In *FST&TCS'10*. LZI, 481–492. https://doi.org/10.4230/LIPIcs.FSTTCS.2010.481

[15] M. L. Minsky. 1961. Recursive unsolvability of Post's problem of 'tag' and other topics in the theory of Turing Machines. *Annals of Mathematics* 74 (1961), 437–455.

[16] F. Neven, T. Schwentick, and V. Vianu. 2004. Finite State Machines for Strings Over Infinite Alphabets. *ACM Transactions on Computational Logic* 5, 3 (2004), 403–435. http://doi.acm.org/10.1145/1013560.1013562

[17] A. Pnueli and R. Rosner. 1989. On the synthesis of an asynchronous reactive module. *Automata, Languages and Programming* (1989), 652–671.

[18] J.-F. Raskin, M. Samuelides, and L. Van Begin. 2005. Games for Counting Abstractions. *Electronic Notes in Theoretical Computer Science* 128, 6 (2005), 69 – 85. AVoCS 2004. https://doi.org/10.1016/j.entcs.2005.04.005

[19] L. Segoufin. 2006. Automata and Logics for Words and Trees over an Infinite Alphabet. In *CSL'06*, LNCS Vol. 4207. Springer, 41–57. https://doi.org/10.1007/11874683_3