Reasoning about Data Repetitions with Counter Systems

Stéphane Demri, Diego Figueira, M. Praveen

Reasoning about Data Repetitions with Counter Systems

Stéphane Demri
NYU & LSV, CNRS

Diego Figueira
University of Edinburgh

M. Praveen
LaBRI, Univ. Bordeaux, France

***Abstract*—We study linear-time temporal logics interpreted over data words with multiple attributes. We restrict the atomic formulas to equalities of attribute values in successive positions and to repetitions of attribute values in the future or past. We demonstrate correspondences between satisfiability problems for logics and reachability-like decision problems for counter systems. We show that allowing/disallowing atomic formulas expressing repetitions of values in the past corresponds to the reachability/coverability problem in Petri nets. This gives us 2EXPSPACE upper bounds for several satisfiability problems. We prove matching lower bounds by reduction from a reachability problem for a newly introduced class of counter systems. This new class is a succinct version of vector addition systems with states in which counters are accessed via pointers, a potentially useful feature in other contexts. We strengthen further the correspondences between data logics and counter systems by characterizing the complexity of fragments, extensions and variants of the logic. For instance, we precisely characterize the relationship between the number of attributes allowed in the logic and the number of counters needed in the counter system.**

I. Introduction

Words with multiple data: Finite data words [4] are ubiquitous structures that include timed words, runs of counter automata or runs of concurrent programs with an unbounded number of processes. These are finite words in which every position carries a label from a finite alphabet and a data value from some infinite alphabet. More generally, structures over an infinite alphabet provide an adequate abstraction for objects from several domains: for example, infinite runs of counter automata can be viewed as infinite data words, finite data trees model XML documents with attribute values [9] and so on. A wealth of specification formalisms for data words (or slight variants) has been introduced stemming from automata, see e.g. [26], [29], to adequate logical languages such as first-order logic [1], [5] or temporal logics [23], [19], [9], [17], [10] (see also a related formalism in [12]). Depending on the type of structures, other formalisms have been considered such as XPath [9] or monadic second-order logic [3]. In full generality, most formalisms lead to undecidable decision problems and a well-known research trend consists of finding a good trade-off between expressiveness and decidability. Restrictions to regain decidability are protean: bounding the models (from trees to words for instance), restricting the number of variables, see e.g. [1], limiting the set of the temporal operators or the use of the data manipulating operator, see e.g. [11], [6]. Moreover, interesting and surprising results have been exhibited about relationships between logics for data words and counter automata [1], [7], [2], leading to a first classification of automata on data words [2]. This is why logics for data words are not only interesting for their own sake but also for their deep relationships with data automata or with counter automata. Herein, we pursue further this line of work and we work with words in which every position contains a *vector* of data values.

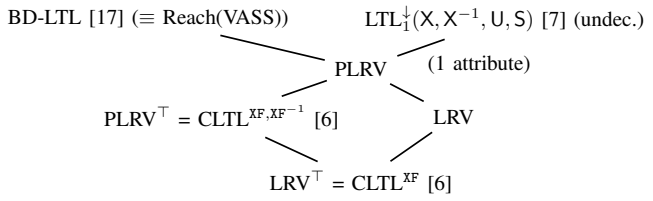
Work supported by project REACHARD ANR-11-BS02-001, and FET-Open Project FoX, grant agreement 233599. M. Praveen was supported by the ERCIM "Alain Bensoussan" Fellowship Programme.

Motivations: In [6], a decidable logic interpreted over (finite or infinite) sequences of variable valuations (understood as words with multiple data) is introduced in which the atomic formulae are of the form either $\mathtt{x} \approx \mathsf{X}^i\mathtt{y}$ or $\mathtt{x} \approx \langle\top?\rangle\mathtt{y}$. The formula $\mathtt{x} \approx \mathsf{X}^i\mathtt{y}$ states that the current value of variable $\mathtt{x}$ is the same as the value of $\mathtt{y}$ i steps ahead (local constraint) whereas $\mathtt{x} \approx \langle\top?\rangle\mathtt{y}$ states that the current value of $\mathtt{x}$ is repeated in a future value of $\mathtt{y}$ (future obligation). Such atomic properties can be naturally expressed with a *freeze* operator that stores a data value for later comparison, and in [6], it is shown that the satisfiability problem is decidable with the temporal operators in $\{\mathsf{X}, \mathsf{X}^{-1}, \mathsf{U}, \mathsf{S}\}$. The freeze operator allows to store a data value in a register and then to test later equality between the value in the register and a data value at some other position. This is a powerful mechanism but the logic in [6] uses it in a limited way: only repetitions of data values can be expressed and it restricts very naturally the use of the freeze operator. The decidability result is robust since, for instance, it holds with the addition of atomic formulas of the form $\mathtt{x} \approx \langle\top?\rangle^{-1}\mathtt{y}$ stating that the current value of $\mathtt{x}$ is repeated in a past value of $\mathtt{y}$ (past obligation). Decidability can be shown either by reduction into $\mathrm{FO}^2(\sim, <, +\omega)$, a first-order logic over data words introduced in [1] or by reduction into the verification of fairness properties in Petri nets, shown decidable in [16]. In both cases, an essential use of the decidability of the reachability problem for Petri nets is made, for which no primitive recursive algorithm is known, see e.g. [21]. Hence, even though the logics shown decidable in [6] poorly use the freeze operator (the only properties about data are related to controlled repetitions), the complexity of their satisfiability problems is unknown. Moreover, it is unclear whether the reductions into the reachability problem for Petri nets are really needed; this would be the case if reductions in the other direction exist. Note that in [17], a richer logic has been introduced and it has been shown that satisfiability

is equivalent to the reachability problem for Petri nets.

Our main motivation is to investigate logics that express repetitions of values, revealing the correspondence between expressivity of the logic and reachability problems for counter machines, including well-known problems for Petri nets. This work can be seen as a study of the precision with which counting needs to be done as a consequence of having a mechanism for demanding "*the current data value is repeated in the future/past*" in a logic. Hence, this is not the study of yet another logic, but of a natural feature shared by most studied logics on data words [6], [1], [7], [11], [17], [9], [10]: the property of demanding that a data value be repeated. We consider different ways in which one can demand the repetition of a value, and study the repercussion in terms of the "precision" with which we need to count in order to solve the satisfiability problem. Our measurement of precision here distinguishes the reachability versus the coverability problem for Petri nets and the number of counters needed as a function of the number of variables used in the logic.

Our contribution: We introduce the logic LRV ("Logic of Repeating Values") interpreted over finite words with multiple data, equipped with atomic formulas of the form either $\mathtt{x} \approx \mathsf{X}^i\mathtt{y}$ or $\mathtt{x} \approx \langle\phi?\rangle\mathtt{y}$ [resp. $\mathtt{x} \not\approx \langle\phi?\rangle\mathtt{y}$], where $\mathtt{x} \approx \langle\phi?\rangle\mathtt{y}$ [resp. $\mathtt{x} \not\approx \langle\phi?\rangle\mathtt{y}$] states that the current value of $\mathtt{x}$ is repeated [resp. is not repeated] in some future value of $\mathtt{y}$ in a position where ϕ holds true. When we impose $\phi = \top$, the logic introduced in [6] is obtained and it is denoted by $\text{LRV}^\top$. The syntax for future obligations is freely inspired from PDL with its test operator '?'. Even though LRV contains the past-time temporal operators X^{-1} and S, it has no past obligations. We write PLRV to denote the extension of LRV with past obligations of the form $\mathtt{x} \approx \langle\phi?\rangle^{-1}\mathtt{y}$ or $\mathtt{x} \not\approx \langle\phi?\rangle^{-1}\mathtt{y}$. Below, we illustrate how LRV and variants are compared to existing data logics.



Our main results are listed below.

1. We begin where [6] stopped: the reachability problem for Petri nets is reduced to the satisfiability problem of PLRV (i.e., the logic with past obligations).
2. Without past obligations, the satisfiability problem is much easier: we reduce the satisfiability problem of $\text{LRV}^\top$ and LRV to the control-state reachability problem for VASS, via a detour to a reachability problem on gainy VASS. But the number of counters in the VASS is exponential in the number of variables used in the formula. This gives us a 2EXPSPACE upper bound.
3. The exponential blow up mentioned above is unavoidable: we show a polynomial-time reduction in the converse direction, starting from a linear sized counter machine (without zero tests) that can access exponentially many counters. This gives us a matching 2EXPSPACE lower bound.
4. Several augmentations to the logic do not alter the complexity: we show that complexity is preserved when MSO-definable temporal operators are added or when infinite words with multiple data are considered.
5. The power of nested testing formulas: we show that the complexity of the satisfiability problem for $\text{LRV}^\top$ reduces to PSPACE-complete when the number of variables in the logic is bounded by a constant, while the complexity of the satisfiability of LRV does not reduce even when only one variable is allowed. Recall that the difference between $\text{LRV}^\top$ and LRV is that the later allows any ϕ in $\mathtt{x} \approx \langle\phi?\rangle\mathtt{y}$ while the former restricts ϕ to just $\top$.
6. The power of pairs of repeating values: we show that the satisfiability problem of $\text{LRV}^\top$ augmented with $\langle\mathtt{x},\mathtt{y}\rangle \approx \langle\top?\rangle\langle\mathtt{x}',\mathtt{y}'\rangle$ (repetitions of pairs of data values) is undecidable, even when $\langle\mathtt{x},\mathtt{y}\rangle \approx \langle\top?\rangle^{-1}\langle\mathtt{x}',\mathtt{y}'\rangle$ is not allowed (i.e., even when past obligations are not allowed).
7. Implications for classical logics: we show a 3EXPSPACE upper bound for the satisfiability problem for *forward*-$\text{EMSO}^2(+1,<,\sim)$ over data words, using results on LRV.

For proving the result mentioned in point 3 above, we introduce a new class of counter machines that we call *chain systems* and show a key hardness result for them. This class is interesting for its own sake and could be used in situations where the power of binary encoding needs to be used. We prove the $(k+1)$EXPSPACE-completeness of the control state reachability problem for chain systems of level k (we only use $k = 1$ in this paper but the proof for arbitrary k is no more complex than the proof for the particular case of $k = 1$). In chain systems, the number of counters is equal to an exponential tower of height k but we cannot access the counters directly in the transitions. Instead, we have a pointer that we can move along a chain of counters. The $(k+1)$EXPSPACE lower bound is obtained by a non-trivial extension of the EXPSPACE-hardness result from [22], [8]. Then we show that the control state reachability problem for the class of chain systems with $k = 1$ can be reduced to the satisfiability problem for LRV (see Section V). It was known that data logics are strongly related to classes of counter automata, see e.g. [1], [7], [2] but herein, we show how varying the expressive power of logics leads to correspondence with different reachability problems for counter machines.

II. Preliminaries

We write $\mathbb{N}$ [resp. $\mathbb{Z}$] to denote the set of non-negative integers [resp. integers] and $[i,j]$ to denote $\{k \in \mathbb{Z} : i \leq k \text{ and } k \leq j\}$. For $\mathbf{v} \in \mathbb{Z}^n$, $\mathbf{v}(i)$ denotes the i^{th} element of $\mathbf{v}$ for every $i \in [1,n]$. We write $\mathbf{v} \preceq \mathbf{v}'$ whenever for every $i \in [1,n]$, we have $\mathbf{v}(i) \leq \mathbf{v}'(i)$. For a (possibly infinite) alphabet Σ, Σ^* represents the set of finite words over Σ, Σ^+ the set of finite non-empty words over Σ. For a finite word or sequence $u = a_1 \dots a_k$ over Σ, we write $|u|$ to denote its *length* k. For $0 \leq i < |u|$, $u(i)$ represents the $(i+1)$-th letter of the word, here a_{i+1}.

Logics of Repeating Values: Let $\mathrm{VAR} = \{\mathtt{x}_1, \mathtt{x}_2, \ldots\}$ be a countably infinite set of *variables*. We denote by LRV the logic whose formulas are defined as follows, where $\mathtt{x}, \mathtt{y} \in \mathrm{VAR}$, $i \in \mathbb{N}$.

$$\phi ::= \mathtt{x} \approx \mathsf{X}^i\mathtt{y} \mid \mathtt{x} \approx \langle\phi?\rangle\mathtt{y} \mid \mathtt{x} \not\approx \langle\phi?\rangle\mathtt{y} \mid \phi \wedge \phi \mid \neg\phi \mid \mathsf{X}\phi \mid \phi\mathsf{U}\phi \mid \mathsf{X}^{-1}\phi \mid \phi\mathsf{S}\phi$$

A *valuation* is a map from VAR to $\mathbb{N}$, and a *model* is a finite non-empty sequence σ of valuations. For every model σ and $0 \leq i < |\sigma|$, the satisfaction relation $\models$ is defined:

- $\sigma, i \models \mathtt{x} \approx \mathsf{X}^j\mathtt{y}$ iff $i+j < |\sigma|$ and $\sigma(i)(\mathtt{x}) = \sigma(i+j)(\mathtt{y})$,
- $\sigma, i \models \mathtt{x} \approx \langle\phi?\rangle\mathtt{y}$ iff there exists j such that $i < j < |\sigma|$, $\sigma(i)(\mathtt{x}) = \sigma(j)(\mathtt{y})$ and $\sigma, j \models \phi$.

The semantics $\mathtt{x} \not\approx \langle\phi?\rangle\mathtt{y}$ is defined similarly but asking for a different data value. The temporal operators next (X), previous (X^{-1}), until (U) and since (S) and Boolean connectives are defined in the usual way. We also use the standard derived temporal operators (G, F, F^{-1}, ...) and constants $\top$, $\bot$. We write $\sigma \models \phi$ if $\sigma, 0 \models \phi$. Given a set of temporal operators $\mathcal{O}$, we write LRV($\mathcal{O}$) to denote the fragment of LRV restricted to formulas with operators from $\mathcal{O}$. The *satisfiability problem* for LRV (written SAT(LRV)) is to check for an LRV formula ϕ, whether there is σ such that $\sigma \models \phi$. Let PLRV be the extension of LRV with additional atomic formulas of the form $\mathtt{x} \approx \langle\phi?\rangle^{-1}\mathtt{y}$ and $\mathtt{x} \not\approx \langle\phi?\rangle^{-1}\mathtt{y}$. The satisfaction relation is extended as expected: $\sigma, i \models \mathtt{x} \approx \langle\phi?\rangle^{-1}\mathtt{y}$ iff there is $0 \leq j < i$ such that $\sigma(i)(\mathtt{x}) = \sigma(j)(\mathtt{y})$ and $\sigma, j \models \phi$, and similarly for $\mathtt{x} \not\approx \langle\phi?\rangle^{-1}\mathtt{y}$. We write LRV$^\top$ [resp. PLRV$^\top$] to denote the fragment of LRV [resp. PLRV] in which atomic formulas are restricted to $\mathtt{x} \approx \mathsf{X}^i\mathtt{y}$ and $\mathtt{x} \approx \langle\top?\rangle\mathtt{y}$ [resp. to $\mathtt{x} \approx \mathsf{X}^i\mathtt{y}$, $\mathtt{x} \approx \langle\top?\rangle\mathtt{y}$ and $\mathtt{x} \approx \langle\top?\rangle^{-1}\mathtt{y}$]. In [6], LRV$^\top$ and PLRV$^\top$ were shown to be decidable by reduction into the reachability problem for Petri nets. However, the characterization of their complexity remained open.

Properties: In the table below, we justify our choices for atomic formulae by presenting several abbreviations (with their obvious semantics). By contrast, we include in LRV both $\mathtt{x} \approx \langle\phi?\rangle\mathtt{y}$ and $\mathtt{x} \not\approx \langle\phi?\rangle\mathtt{y}$ when ϕ is an arbitrary formula since there is no obvious way to express one with the other.

Abbreviation	Definition
$\mathtt{x} \not\approx \mathsf{X}^i\mathtt{y}$	$\neg(\mathtt{x} \approx \mathsf{X}^i\mathtt{y}) \wedge \overbrace{\mathsf{X}\cdots\mathsf{X}}^{i \text{ times}}\top$
$\mathtt{x} \approx \mathsf{X}^{-i}\mathtt{y}$	$\overbrace{\mathsf{X}^{-1}\cdots\mathsf{X}^{-1}}^{i \text{ times}}(\mathsf{X}^i\mathtt{x} \approx \mathtt{y})$
$\mathtt{x} \not\approx \mathsf{X}^{-i}\mathtt{y}$	$\neg(\mathtt{x} \approx \mathsf{X}^{-i}\mathtt{y}) \wedge \overbrace{\mathsf{X}^{-1}\cdots\mathsf{X}^{-1}}^{i \text{ times}}\top$
$\mathtt{x} \not\approx \langle\top?\rangle\mathtt{y}$	$(\mathtt{x} \not\approx \mathsf{X}\mathtt{y}) \vee \mathsf{X}((\mathtt{y} \approx \mathsf{X}\mathtt{y})\mathsf{U}(\mathtt{y} \not\approx \mathsf{X}\mathtt{y}))$
$\mathtt{x} \not\approx \langle\top?\rangle^{-1}\mathtt{y}$	$(\mathtt{x} \not\approx \mathsf{X}^{-1}\mathtt{y}) \vee \mathsf{X}^{-1}((\mathtt{y} \approx \mathsf{X}^{-1}\mathtt{y})\mathsf{S}(\mathtt{y} \not\approx \mathsf{X}^{-1}\mathtt{y}))$

Models for LRV can be viewed as finite data words in $(\Sigma \times \mathbb{D})^*$, where Σ is a finite alphabet and $\mathbb{D}$ is an infinite domain. E.g., equalities between dedicated variables can simulate that a position is labelled by a letter from Σ; moreover, we may assume that the data values are encoded with the variable $\mathtt{x}$. Let us express that whenever there are $i < j$ s.t. i and j [resp. $i+1$ and $j+1$, $i+2$ and $j+2$] are labelled by a [resp. a', a''], $\sigma(i+1)(\mathtt{x}) \neq \sigma(j+1)(\mathtt{x})$. This can be stated in LRV by: $\mathsf{G}(a \wedge \mathsf{X}(a' \wedge \mathsf{X}a'') \Rightarrow \mathsf{X}\neg(\mathtt{x} \approx \langle\mathsf{X}^{-1}a \wedge a' \wedge \mathsf{X}a''?\rangle\mathtt{x}))$. This is an example of key constraints, see e.g. [27, Definition 2.1] and the paper contains also numerous examples of properties that can be captured by LRV.

Basics on VASS: A *vector addition system with states* (VASS) is a tuple $\mathcal{A} = \langle Q, C, \delta\rangle$ where Q is a finite set of *control states*, C is a finite set of *counters* and δ is a finite set of *transitions* in $Q \times \mathbb{Z}^C \times Q$. A *configuration* of $\mathcal{A}$ is a pair $\langle q, \mathbf{v}\rangle \in Q \times \mathbb{N}^C$. We write $\langle q, \mathbf{v}\rangle \to \langle q', \mathbf{v}'\rangle$ if there is $(q, \mathbf{u}, q') \in \delta$ such that $\mathbf{v}' = \mathbf{v} + \mathbf{u}$. The *reachability problem* for VASS (written Reach(VASS)) consists in checking whether $\langle q_0, \mathbf{v_0}\rangle \xrightarrow{*} \langle q_f, \mathbf{v_f}\rangle$, given two configurations $\langle q_0, \mathbf{v_0}\rangle$ and $\langle q_f, \mathbf{v_f}\rangle$. The reachability problem for VASS is decidable with non-primitive recursive complexity algorithms [25], [18], [21], but the best known lower bound is only EXPSPACE [22], [8]. The *control state reachability problem* is the following EXPSPACE-complete problem [22], [28]: given a configuration $\langle q_0, \mathbf{v_0}\rangle$ and a control state q_f, check whether $\langle q_0, \mathbf{v_0}\rangle \xrightarrow{*} \langle q_f, \mathbf{v}\rangle$ for some $\mathbf{v} \in \mathbb{N}^C$. We also use two other kinds of computations: with gains ($\to_{\text{gainy}}$) or losses ($\to_{\text{lossy}}$). We write $\langle q, \mathbf{v}\rangle \to_{\text{gainy}} \langle q', \mathbf{v}'\rangle$ [resp. $\langle q, \mathbf{v}\rangle \to_{\text{lossy}} \langle q', \mathbf{v}'\rangle$] if there is a transition $(q, \mathbf{u}, q') \in \delta$ and $\mathbf{w} \in \mathbb{N}^C$ such that $\mathbf{v} \preceq \mathbf{w}$ and $\mathbf{w} + \mathbf{u} \preceq \mathbf{v}'$ [resp. $\mathbf{w} \preceq \mathbf{v}$ and $\mathbf{v}' \preceq \mathbf{w} + \mathbf{u}$].

III. The Power of Past: From Reach(VASS) to SAT(PLRV)

While [6] concentrated on decidability results, here we begin with a hardness result. When past obligations are allowed as in PLRV, SAT(PLRV) is equivalent to the very difficult problem of reachability in VASS. Combined with the result of the next section where we prove that removing past obligations leads to a reduction into the control state reachability problem for VASS, this means that reasoning with past obligations is much more complicated.

Theorem 1. There is a polynomial-space reduction from Reach(VASS) into SAT(PLRV).

The proof of Theorem 1 is analogous to the proof of [1, Theorem 16] except that properties are expressed in PLRV instead of being expressed in $\mathrm{FO}^2(\sim, <, +1)$. In Section IV-A, we show how to eliminate test formulas ϕ from $\mathtt{x} \approx \langle\phi?\rangle\mathtt{y}$ and $\mathtt{x} \approx \langle\phi?\rangle^{-1}\mathtt{y}$. Combining this with the decidability proof for PLRV$^\top$ satisfiability from [6], we get that SAT(PLRV) [resp. for SAT(PLRV$^\top$)] is equivalent to Reach(VASS) modulo polynomial-space reductions.

IV. Leaving the Past Behind Simplifies Things: From SAT(LRV) to Control State Reachability

In this section, we show the reduction from SAT(LRV) to the control state reachability problem in VASS. We obtain a 2EXPSPACE upper bound for SAT(LRV) as a consequence. This is done in two steps: (1) simplifying formulas of the form $\mathtt{x} \approx \langle\phi?\rangle\mathtt{y}$ to remove the test formula ϕ (i.e., a reduction from SAT(LRV) into SAT(LRV$^\top$)); and (2) reducing from SAT(LRV$^\top$) into the control state reachability pb. in VASS.

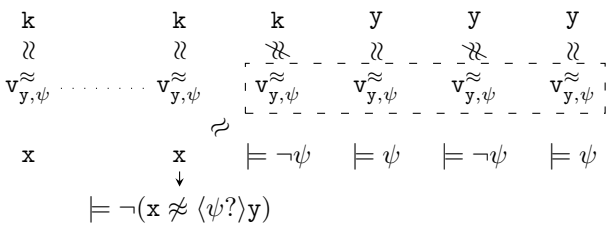
A. Elimination of Test Formulas

We give a polynomial-time algorithm that for every $\varphi \in$ LRV computes a formula $\varphi' \in \text{LRV}^\top$ that preserves satisfiability: there is a model σ s.t. $\sigma \models \varphi$ iff there is a model σ' s.t. $\sigma' \models \varphi'$. We give the reduction in two steps. First, we eliminate formulas with inequality tests of the form $\mathtt{x} \not\approx \langle\varphi?\rangle\mathtt{y}$ using only positive tests of the form $\mathtt{x} \approx \langle\varphi?\rangle\mathtt{y}$. We then eliminate formulas of the form $\mathtt{x} \approx \langle\varphi?\rangle\mathtt{y}$, using only formulas of the form $\mathtt{x} \approx \langle\top?\rangle\mathtt{y}$. Let $\text{LRV}^{\approx}$ be the logic LRV where there are no appearances of formulas of the form $\mathtt{x} \not\approx \langle\varphi?\rangle\mathtt{y}$; and let $\text{PLRV}^{\approx}$ be PLRV without $\mathtt{x} \not\approx \langle\varphi?\rangle\mathtt{y}$ or $\mathtt{x} \not\approx \langle\varphi?\rangle^{-1}\mathtt{y}$.

Proposition 2 (from LRV to $\text{LRV}^{\approx}$)**.** There is a polynomial-time reduction from SAT(LRV) into SAT($\text{LRV}^{\approx}$); and from SAT(PLRV) into SAT($\text{PLRV}^{\approx}$).

Proof sketch: For every $\varphi \in$ LRV, we compute $\varphi' \in \text{LRV}^{\approx}$ in polynomial time, which preserves satisfiability. Besides all the variables from φ, φ' uses a distinguished variable $\mathtt{k}$ (which will have a constant value, different from all the values of the variables of φ) and variables $\mathtt{v}^{\approx}_{\mathtt{y},\psi}$, $\mathtt{v}_{\mathtt{x}\not\approx\langle\psi?\rangle\mathtt{y}}$ for every subformula ψ of φ and variables $\mathtt{x}, \mathtt{y}$ of φ.

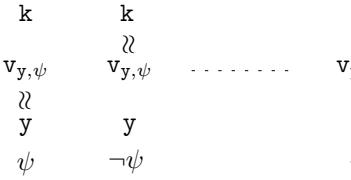
Each subformula $\mathtt{x} \not\approx \langle\psi?\rangle\mathtt{y}$ is replaced by $\mathtt{x} \not\approx \mathtt{v}_{\mathtt{x}\not\approx\langle\psi?\rangle\mathtt{y}} \wedge \mathtt{v}_{\mathtt{x}\not\approx\langle\psi?\rangle\mathtt{y}} \approx \langle\psi?\rangle\mathtt{y}$, where $\mathtt{v}_{\mathtt{x}\not\approx\langle\psi?\rangle\mathtt{y}}$ is a fresh variable. On the other hand, for each subformula $\neg(\mathtt{x} \not\approx \langle\psi?\rangle\mathtt{y})$, we use the variable $\mathtt{v}^{\approx}_{\mathtt{y},\psi}$ in conjunction with $\mathtt{k}$ as shown below.



In the beginning, $\mathtt{v}^{\approx}_{\mathtt{y},\psi} \approx \mathtt{k}$, which is broken at the first position where all future positions where ψ holds have the same value for $\mathtt{y}$. At this position (say $i+1$), $\mathtt{v}^{\approx}_{\mathtt{y},\psi} \not\approx \mathtt{k}$ and $\mathtt{v}^{\approx}_{\mathtt{y},\psi}$ maintains the same value (say n) at all future positions, illustrated as a dashed box above. In all positions after i that satisfy ψ, $\mathtt{y} \approx \mathtt{v}^{\approx}_{\mathtt{y},\psi}$. These conditions can be enforced without using $\neg(\mathtt{x} \not\approx \langle\psi?\rangle\mathtt{y})$. Suppose $\mathtt{x}$ also has value n at position i as shown above. Now in any position after i that satisfies ψ, $\mathtt{y}$ has the value n, which is the value of $\mathtt{x}$ in position i. This is exactly the semantics of $\neg(\mathtt{x} \not\approx \langle\psi?\rangle\mathtt{y})$. Hence, $\neg(\mathtt{x} \not\approx \langle\psi?\rangle\mathtt{y})$ can be replaced by $\neg\mathsf{X}\mathsf{F}\psi \vee \mathtt{x} \approx \mathsf{X}\mathtt{v}^{\approx}_{\mathtt{y},\psi}$. Past obligations are treated in a symmetrical way. ∎

Proposition 3 (from $\text{LRV}^{\approx}$ to $\text{LRV}^\top$)**.** There is a polynomial-time reduction from SAT($\text{LRV}^{\approx}$) into SAT($\text{LRV}^\top$); and from SAT($\text{PLRV}^{\approx}$) into SAT($\text{PLRV}^\top$).

Proof sketch: For every $\varphi \in \text{LRV}^{\approx}$, we compute in polynomial time $\varphi' \in \text{LRV}^\top$ that preserves satisfiability. Besides all the variables from φ, φ' uses a new distinguished variable $\mathtt{k}$, and a variable $\mathtt{v}_{\mathtt{y},\psi}$ for every subformula ψ of φ and every variable $\mathtt{y}$ of φ. We enforce $\mathtt{k}$ to have a constant value different from all values of variables of φ. At every position, we enforce ψ to hold if $\mathtt{v}_{\mathtt{y},\psi} \approx \mathtt{y}$, and ψ not to hold if $\mathtt{v}_{\mathtt{y},\psi} \approx \mathtt{k}$ as shown above. Then $\mathtt{x} \approx \langle\psi?\rangle\mathtt{y}$ is replaced by $\mathtt{x} \approx \langle\top?\rangle\mathtt{v}_{\mathtt{y},\psi}$. ∎



Corollary 4. There is a polynomial-time reduction from SAT(LRV) into SAT($\text{LRV}^\top$).

Since SAT($\text{PLRV}^\top$) is decidable [6], we obtain the decidability of SAT(PLRV).

Corollary 5. SAT(PLRV) is decidable.

B. From $\text{LRV}^\top$ Satisfiability to Control State Reachability

In [6], SAT($\text{LRV}^\top$) is reduced to the reachability problem for a subclass of VASS. Herein, this is refined by introducing *incremental errors* in order to improve the complexity.

In [6], the standard concept of atoms from the Vardi-Wolper construction of automaton for LTL is used. Refer to the diagram at the top of Figure 1. The formula $\mathtt{x} \approx \langle\top?\rangle\mathtt{y}$ in the left atom creates an obligation for the current value of $\mathtt{x}$

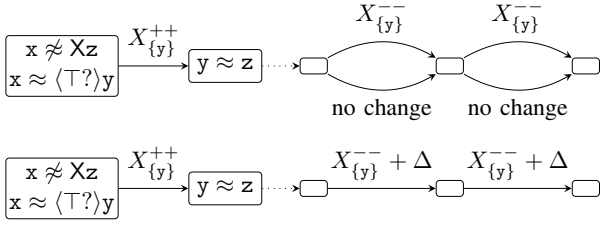


Fig. 1. Automaton constructions from [6] (top) and from this paper (bottom).

to appear some time in the future in $\mathtt{y}$. This obligation cannot be satisfied in the second atom, since $\mathtt{y}$ has to satisfy some other constraint there ($\mathtt{y} \approx \mathtt{z}$). To remember this unsatisfied obligation about $\mathtt{y}$ while taking the transition from the first atom to the second, the counter $X_{\{\mathtt{y}\}}$ is incremented. The counter can be decremented later in transitions that allow the repetition in $\mathtt{y}$. If several transitions allow such a repetition, only one of them needs to decrement the counter (since there was only one obligation at the beginning). The other transitions which should not decrement the counter can take the alternative labelled "no change" in the right part of Figure 1.

The idea here is to replace the combination of the decrementing transition and the "no change" transition in the top of Figure 1 with a single transition with incremental errors as shown in the bottom. After Lemma 6 below that formalises ideas from [6, Section 7], we prove that the transition with incremental errors is sufficient.

Lemma 6 ([6])**.** For a $\text{LRV}^\top$ formula ϕ that uses the variables $\{\mathtt{x}_1, \ldots, \mathtt{x}_k\}$, a VASS $\mathcal{A}_\phi = \langle Q, C, \delta\rangle$ can be defined, along with sets $Q_0, Q_f \subseteq Q$ of *initial* and *final* states resp., s.t.

- the set of counters C consists of all nonempty subsets of $\{\mathtt{x}_1, \ldots, \mathtt{x}_k\}$.
- For all $q, q' \in Q$, either $\{\mathbf{u} \mid (q, \mathbf{u}, q') \in \delta\} = [n_1, m_1] \times \cdots \times [n_{|C|}, m_{|C|}]$ for some $n_1, m_1, \ldots, n_{|C|}, m_{|C|} \in [-k, k]$, or $\{\mathbf{u} \mid (q, \mathbf{u}, q') \in \delta\} = \emptyset$. (We call this property *closure under component-wise interpolation*.)
- If $\delta \cap (\{q\} \times [-k, k]^C \times \{q'\})$ is not empty, then for every $X \in C$ there is $(q, \mathbf{u}, q') \in \delta$ so that $\mathbf{u}(X) \geq 0$. (We call this property *optional decrement*.)
- Let $\mathbf{0}$ be the counter valuation that assigns 0 to all counters. Then ϕ is satisfiable iff $\langle q_0, \mathbf{0}\rangle \xrightarrow{*} \langle q_f, \mathbf{0}\rangle$ for some $q_0 \in Q_0$ and $q_f \in Q_f$.

At the top of Figure 1, only one counter $X_{\{\mathtt{y}\}}$ is shown and is decremented by 1 for simplicity. In general, multiple counters can be changed and they can be incremented/decremented by any number up to k, depending on the initial and target atoms of the transition. If a counter can be incremented by k_1 and can be decremented by k_2, then there will also be transitions between the same pair of atoms allowing changes of $k_1 - 1, \ldots, 1, 0, -1, \ldots - (k_2 - 1)$. This corresponds to the closure under component-wise interpolation mentioned in the lemma above. The optional decrement property corresponds to the fact that there will always be a "no change" transition that does not decrement any counter.

Now, we show that a single transition that decrements all counters by the maximal possible number can simulate the set of all transitions between two atoms, using incremental errors. Let $\mathcal{A}_{\text{inc}} = \langle Q, C, \delta^{min}\rangle$ and $Q_0, Q_f \subseteq Q$, where Q, Q_0, Q_f and C are same as those of $\mathcal{A}_\phi$ and δ^{min} is defined as follows: $(q, minup_{q,q'}, q') \in \delta^{min}$ iff $\delta \cap (\{q\} \times [-k, k]^C \times \{q'\})$ is not empty and $minup_{q,q'}(X) = \min_{\mathbf{u}:(q,\mathbf{u},q')\in\delta}\{\mathbf{u}(X)\}$ for all $X \in C$.

Lemma 7. If $\langle q, \mathbf{v}\rangle \to \langle q', \mathbf{v}'\rangle$ in $\mathcal{A}_\phi$, then $\langle q, \mathbf{v}\rangle \to_{\text{gainy}} \langle q', \mathbf{v}'\rangle$ in $\mathcal{A}_{\text{inc}}$.

Proof sketch: If $\mathcal{A}_\phi$ takes one of the transitions in the top of Figure 1, $\mathcal{A}_{\text{inc}}$ takes the corresponding transition at the bottom, adjusting the incremental error Δ accordingly. ∎

Lemma 8. If $\langle q_1, \mathbf{v}_1\rangle \xrightarrow{*}_{\text{gainy}} \langle q_2, \mathbf{0}\rangle$ in $\mathcal{A}_{\text{inc}}$ and $\mathbf{v}'_1 \preceq \mathbf{v}_1$, then $\langle q_1, \mathbf{v}'_1\rangle \xrightarrow{*} \langle q_2, \mathbf{0}\rangle$ in $\mathcal{A}_\phi$.

Proof sketch: If $\mathcal{A}_{\text{inc}}$ takes a transition at the bottom of Figure 1, $\mathcal{A}_\phi$ takes the corresponding decrementing transition at the top, ignoring any incremental errors. This may result in $\mathcal{A}_\phi$ reaching $\mathbf{0}$ earlier in the run, in which case "no change" transitions are used in the rest of the run. ∎

Theorem 9. SAT(LRV$^\top$) is in 2EXPSPACE.

Proof sketch: The proof is in four steps (standard arguments are used for 3. and 4.).

1 From [6], a LRV$^\top$ formula ϕ is satisfiable iff $\langle q_0, \mathbf{0}\rangle \xrightarrow{*} \langle q_f, \mathbf{0}\rangle$ in $\mathcal{A}_\phi$ for some $q_0 \in Q_0$ and $q_f \in Q_f$.
2 This is the step that requires new insight. From Lemmas 7 and 8, $\langle q_0, \mathbf{0}\rangle \xrightarrow{*} \langle q_f, \mathbf{0}\rangle$ in $\mathcal{A}_\phi$ iff $\langle q_0, \mathbf{0}\rangle \xrightarrow{*}_{\text{gainy}} \langle q_f, \mathbf{0}\rangle$ in $\mathcal{A}_{\text{inc}}$.
3 Let $\mathcal{A}_{\text{dec}}$ be the VASS obtained from $\mathcal{A}_{\text{inc}}$ by "reversing" each transition. By replacing each gainy transition of $\mathcal{A}_{\text{inc}}$ by the reverse lossy transtion of $\mathcal{A}_{\text{dec}}$, we infer that $\langle q_0, \mathbf{0}\rangle \xrightarrow{*}_{\text{gainy}} \langle q_f, \mathbf{0}\rangle$ in $\mathcal{A}_{\text{inc}}$ iff $\langle q_f, \mathbf{0}\rangle \xrightarrow{*}_{\text{lossy}} \langle q_0, \mathbf{0}\rangle$ in $\mathcal{A}_{\text{dec}}$.
4 $\langle q_f, \mathbf{0}\rangle \xrightarrow{*}_{\text{lossy}} \langle q_0, \mathbf{0}\rangle$ iff $\langle q_f, \mathbf{0}\rangle \xrightarrow{*} \langle q_0, \mathbf{v}\rangle$ for some $\mathbf{v}$. Checking the latter condition is precisely the control state reachability problem for VASS.

The number of control states and counters in $\mathcal{A}_\phi$ and in $\mathcal{A}_{\text{dec}}$ is exponential in $|\phi|$ (size of ϕ). Each control state and transition function of $\mathcal{A}_{\text{dec}}$ can be represented in space polynomial in $|\phi|$. Given a control state, testing if it is an initial or a final state can be done in linear time in the size of the state. The automaton $\mathcal{A}_{\text{dec}}$ can be constructed in exponential time in $|\phi|$. Hence, the EXPSPACE upper bound for the control state reachability problem in VASS [28] gives the 2EXPSPACE upper bound for SAT(LRV$^\top$). ∎

Corollary 10. SAT(LRV) is in 2EXPSPACE.

V. Simulating Exponentially Many Counters

In Section IV, the reduction from SAT(LRV) to control state reachability involves an exponential blow up, since we use one counter for each subset of variables. The question of whether this can be avoided depends on whether LRV is powerful enough to reason about subsets of variables or whether there is a smarter reduction without a blow-up. Similar questions are open in other related areas [24].

Here we prove that LRV is indeed powerful enough to reason about subsets of variables. We establish a 2EXPSPACE lower bound, which leverages the power of LRV to access exponentially many counters through binary encoding. Consider the variables and their values shown in the table below. The conditions in the third row are about the equality of variable values in a single position.

Variable	$\mathtt{x}_1$	$\mathtt{x}_2$	$\mathtt{x}_3$	$\mathtt{z}$	inc	dec
Value	20	30	20	20	10	20
Condition	$\mathtt{x}_1 \approx \mathtt{z}$	$\mathtt{x}_2 \not\approx \mathtt{z}$	$\mathtt{x}_3 \approx \mathtt{z}$		inc $\not\approx \mathtt{z}$	dec $\approx \mathtt{z}$
Encodes	1	0	1			decrement

The combination of all the conditions can be thought of as encoding "decrement 101^{th} counter", considering 101 as the binary encoding of 5, where the i-th bit is 1 if $\mathtt{x}_i \approx \mathtt{z}$. A total of 8 counters can be manipulated with $\mathtt{x}_1, \mathtt{x}_2$ and $\mathtt{x}_3$. Using the power of LRV to reason about values that repeat in the future, we can encode the condition "for every counter, an increment is followed by a decrement in the future", which then ensures control state reachability.

The rest of this section is divided into three parts. The first part defines chain systems, which are like VASS where transition rules like "increment counter X" are replaced by "increment 101^{th} counter". The second part shows lower bounds for the control state reachability problem for chain systems. The third part shows that LRV can reason about chain systems of level 1.

A. Chain Systems

We introduce a new class of counter systems that is instrumental to show that SAT(LRV$^\top$) is 2EXPSPACE-hard. This is an intermediate formalism between counter automata with zero-tests with counters bounded by triple exponential values (having 2EXPSPACE-hard control state reachability problem) and properties expressed in LRV$^\top$. Systems with chained counters have no zero-tests and the only updates are increments and decrements. However, the systems are equipped with a finite family of counters, each family having an exponential number of counters. Let $\mathtt{exp}(0,n) \stackrel{\text{def}}{=} n$ and $\mathtt{exp}(k+1,n) \stackrel{\text{def}}{=} 2^{\mathtt{exp}(k,n)}$ for every $k \geq 0$.

Definition 11. A *counter system with chained counters* (herein called a *chain system*) is a tuple $\mathcal{A} = \langle Q, f, k, Q_0, Q_F, \delta \rangle$ where (1) $f : [1,n] \to \mathbb{N}$ where $n \geq 1$ is the *number of chains* and $\mathtt{exp}(k, f(\alpha))$ is the number of counters for the chain α where $k \geq 0$, (2) Q is a non-empty finite set of *states*, (3) $Q_0 \subseteq Q$ is the set of *initial states* and $Q_F \subseteq Q$ is the set of *final states*, (4) δ is the set of *transitions* in $Q \times I \times Q$ where

$$I = \{\mathrm{inc}(\alpha), \mathrm{dec}(\alpha), \mathrm{next}(\alpha), \mathrm{prev}(\alpha), \mathrm{first}(\alpha)?, \overline{\mathrm{first}(\alpha)}?, \\ \mathrm{last}(\alpha)?, \overline{\mathrm{last}(\alpha)}? : \alpha \in [1,n]\}.$$

The system $\mathcal{A} = \langle Q, f, k, Q_0, Q_F, \delta \rangle$ is said to be at *level* k. In order to encode the natural numbers from f and the value k, we use a unary representation. We say that a transition containing $\mathrm{inc}(\alpha)$ is α-*incrementing*, and a transition containing $\mathrm{dec}(\alpha)$ is α-*decrementing*. The idea is that for each chain $\alpha \in [1,n]$, we have $\mathtt{exp}(k, f(\alpha))$ counters, but we cannot access them directly in the transitions like we do in VASS. Instead, we have a pointer to a counter that we can move. We can ask if we are pointing to the first counter ($\mathrm{first}(\alpha)?$) or not ($\overline{\mathrm{first}(\alpha)}?$), or the last counter ($\mathrm{last}(\alpha)?$) or not ($\overline{\mathrm{last}(\alpha)}?$), and we can change the pointer to the next ($\mathrm{next}(\alpha)$) or previous ($\mathrm{prev}(\alpha)$) counter.

A *run* is a finite sequence ρ in δ^* such that

1) for every two $\rho(i) = q \xrightarrow{\mathrm{instr}} r$ and $\rho(i+1) = q' \xrightarrow{\mathrm{instr}'} r'$ we have $r = q'$,
2) for every chain $\alpha \in [1,n]$, for every $i \in [1, |\rho|]$, we have $0 \leq c_i^\alpha < \mathtt{exp}(k, f(\alpha))$ where

$$c_i^\alpha = \mathrm{card}(\{i' < i \mid \rho(i') = q \xrightarrow{\mathrm{next}(\alpha)} r\}) \; - \\ \mathrm{card}(\{i' < i \mid \rho(i') = q \xrightarrow{\mathrm{prev}(\alpha)} r\}), \tag{1}$$

3) for every $i \in [1, |\rho|]$ and for every chain $\alpha \in [1,n]$,
   (a) if $\rho(i) = q \xrightarrow{\mathrm{first}(\alpha)?} q'$, then $c_i^\alpha = 0$;
   (b) if $\rho(i) = q \xrightarrow{\overline{\mathrm{first}(\alpha)}?} q'$, then $c_i^\alpha \neq 0$;
   (c) if $\rho(i) = q \xrightarrow{\mathrm{last}(\alpha)?} q'$, then $c_i^\alpha = \mathtt{exp}(k, f(\alpha)) - 1$;
   (d) if $\rho(i) = q \xrightarrow{\overline{\mathrm{last}(\alpha)}?} q'$, then $c_i^\alpha \neq \mathtt{exp}(k, f(\alpha)) - 1$.

A run is *accepting* whenever $\rho(1)$ starts with an initial state from Q_0 and $\rho(|\rho|)$ ends with a final state from Q_F. A run is *perfect* iff for every $\alpha \in [1,n]$, there is some injective function

$$\gamma : \{i \mid \rho(i) \text{ is } \alpha\text{-decrementing}\} \to \{i \mid \rho(i) \text{ is } \alpha\text{-incrementing}\}$$

such that for every $\gamma(i) = j$ we have that $j < i$ and $c_i^\alpha = c_j^\alpha$. A run is *gainy and ends at zero* iff for every chain $\alpha \in [1,n]$, there is some injective function

$$\gamma : \{i \mid \rho(i) \text{ is } \alpha\text{-incrementing}\} \to \{i \mid \rho(i) \text{ is } \alpha\text{-decrementing}\}$$

such that for every $\gamma(i) = j$ we have that $j > i$ and $c_i^\alpha = c_j^\alpha$. In the sequel, we shall simply say that the run is gainy. Below, we define two problems for which we shall characterize the computational complexity.

PROBLEM: Existence of a perfect accepting run of level $k \geq 0$ (Per(k))
INPUT: A chain system $\mathcal{A}$ of level k. QUESTION: Does $\mathcal{A}$ have a *perfect* accepting run?

PROBLEM: Existence of a gainy accepting run of level $k \geq 0$ (Gainy(k))
INPUT: A chain system $\mathcal{A}$ of level k. QUESTION: Does $\mathcal{A}$ have a *gainy* accepting run?

Per(k) is actually a control state reachability problem in VASS where the counters are encoded succinctly. Gainy(k) is a reachability problem in VASS with incrementing errors and the reached counter values are equal to zero. Here, the counters are encoded succinctly too.

Lemma 12. For every $k \geq 0$, Per(k) and Gainy(k) are interreducible in logarithmic space.

Lemma 13. Per(k) is in $(k+1)$EXPSPACE.

The proof of Lemma 13 consists of simulating perfect runs by the runs of a VASS in which the control states record the positions of the pointers in the chains.

B. Hardness Results for Chain Systems

We show that Per(k) is $(k+1)$EXPSPACE-hard. The implication is that replacing direct access to counters by a pointer that can move along a chain of counters does not decrease the power of VASS, while providing access to more counters. To demonstrate this, we extend Lipton's EXPSPACE-hardness proof for the control state reachability problem in VASS [22] (see also its exposition in [8]). Since our pointers can be moved only one step at a time, this extension involves new insights into the control flow of algorithms used in Lipton's proof, allowing us to implement it even with the limitations imposed by step-wise pointer movements.

Lipton's proof starts from the standard result in computability theory that a Turing Machine using space 2^{n^γ} can be simulated by a counter automaton equipped with 4 counters whose values are bounded by $2^{2^{n^\gamma}}$. Lipton's proof shows that such a counter automaton can be simulated with a VASS. Unlike counter automata, VASS does not have zero-test transitions. Each counter $\mathtt{c}$ of the automaton is complemented with an extra counter $\overline{\mathtt{c}}$ in the VASS. The VASS is designed such that the sum of the values in $\mathtt{c}$ and $\overline{\mathtt{c}}$ is $2^{2^{n^\gamma}}$ in any reachable configuration. Now testing $\mathtt{c}$ for zero is equivalent to testing that $\overline{\mathtt{c}}$ is $2^{2^{n^\gamma}}$. This later test is implemented in a VASS as explained next.

Figure 2 illustrates how a counter $\mathtt{s}_i$ is decremented 2^{2^i} times. There are two nested loops indexed by $\mathtt{y}_{i-1}$ and $\mathtt{z}_{i-1}$, initialised to $2^{2^{i-1}}$. When all the iterations are finished, we would have decremented $\mathtt{s}_i$ exactly $2^{2^{i-1}} \times 2^{2^{i-1}} = 2^{2^i}$ times. Testing $\mathtt{y}_{i-1}$ and $\mathtt{z}_{i-1}$ for zero is done by a similar gadget, where $i, i-1$ are replaced by $i-1, i-2$. The resulting VASS is composed of n^γ constant sized VASS, each one implementing the nested loops for an i between 1 and n^γ. The idea behind our $(k+1)$EXPSPACE lower bound is to replace the VASS with a chain system and n^γ with $\exp(k, n^\gamma)$. This will not work as it is, since a chain system composed of $\exp(k, n^\gamma)$ constant sized portions will not give a polynomial-time reduction. We reduce the size of the chain system by observing that the decrementing algorithm for $i-1$ is the same as the one for i, except that $i, i-1$ are replaced by $i-1, i-2$ respectively. We can write a single sequence of instructions for the decrementing algorithm and invoke it for any i by placing the pointers at the appropriate counters. Roughly speaking, $\exp(k, n^\gamma)$ copies of the code for large decrements is replaced by one copy of the code plus a chain of implicit length $\exp(k, n^\gamma)$ equipped with a pointer to refer to a specific copy.

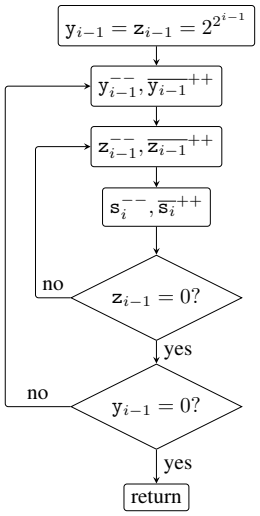


Fig. 2. Control flow of algorithm that decrements $\mathtt{s}_i$ 2^{2^i} times.

The main difficulty in the implementation is to ensure that the multiple invocations of the algorithm return to the correct point. Unlike Lipton's proof that uses VASS, we have to work with chain systems and this introduces some more constructions, the technical details of which are rather tedious.

Theorem 14. Per(k) is $(k+1)$EXPSPACE-hard.

C. Reasoning About Chain Systems with LRV

Given a chain system $\mathcal{A}$ of level 1, we construct a formula in LRV that is satisfiable iff $\mathcal{A}$ has a gainy accepting run. Hence, we get a 2EXPSPACE lower bound for SAT(LRV). The main idea is to encode runs of chain systems with LRV formulas that access the counters using binary encoding, so that the formulas can handle exponentially many counters.

Lemma 15. There is a polynomial-time reduction from Gainy(1) into SAT(LRV(X, F)).

Proof sketch: Let $\mathcal{A} = \langle Q, f, 1, Q_0, Q_F, \delta \rangle$ be a chain system of level 1 with $f : [1, n] \to \mathbb{N}$, having thus n chains of counters, of respective size $2^{f(1)}, \ldots, 2^{f(n)}$.

We encode a word $\rho \in \delta^*$ that represents an accepting run. For this, we use the alphabet δ of transitions. We can simulate the labels $\delta = \{t_1, \ldots, t_m\}$ with variables $\mathtt{t}_0, \ldots, \mathtt{t}_m$, where a node has an encoding of the label t_i iff the formula $\langle t_i \rangle = \mathtt{t}_0 \approx \mathtt{t}_i$ holds true. We build an LRV formula φ so that there is an accepting gainy run $\rho \in \delta^*$ of $\mathcal{A}$ if, and only if, there is a model σ so that $\sigma \models \varphi$ and σ encodes ρ.

The counter-blind conditions to check are: (a) Every position satisfies $\langle t \rangle$ for some $t \in \delta$; (b) the first position satisfies $\langle (q_0, \mathrm{instr}, q) \rangle$ for some $q_0 \in Q_0$, $\mathrm{instr} \in I$, $q \in Q$; (c) the last position satisfies $\langle (q, \mathrm{instr}, q') \rangle$ for some $q \in Q$, $\mathrm{instr} \in I$, $q' \in Q_F$; (d) no two consecutive positions i and $i+1$ satisfy $\langle (q, u, q') \rangle$ and $\langle (p, u', p') \rangle$ respectively, with $q' \neq p$. The difficulty is then in checking that the values of the counters encode indeed a correct gainy run.

We say that a position i is *α-incrementing* [resp. *α-decrementing*] if it satisfies $\langle (q, u, q') \rangle$ for some $q, q' \in Q$ and $u = \mathrm{inc}(\alpha)$ [resp. $u = \mathrm{dec}(\alpha)$]. We use a label (α, i) and variables $\mathtt{x}^\alpha_{inc}$, $\mathtt{x}^\alpha_{dec}$ for every $\alpha \in [1, n]$ and $i \in [1, f(\alpha)]$. We say that a position i operates on the α-counter $\mathtt{c}$, if $\langle (\alpha, j) \rangle$ holds (i.e., position i encodes the label (α, j)) for every position j of the representation of $\mathtt{c}$ in base 2 containing a '1', and $\neg\langle (\alpha, j) \rangle$ for every position j containing a '0'. Note that we can encode every value $0 \leq \mathtt{c} < 2^{f(\alpha)}$.

For every chain α, let us consider the following properties:

- Every two positions of σ have different values of $\mathtt{x}^\alpha_{inc}$ [resp. of $\mathtt{x}^\alpha_{dec}$].
- For every position i of σ operating on an α-counter $\mathtt{c}$ with an instruction '$\mathrm{first}(\alpha)$?' [resp. '$\overline{\mathrm{first}(\alpha)}$?', '$\mathrm{last}(\alpha)$?', '$\overline{\mathrm{last}(\alpha)}$?'] , we have $\mathtt{c} = 0$ [resp. $\mathtt{c} \neq 0$, $\mathtt{c} = 2^{f(\alpha)} - 1$, $\mathtt{c} \neq 2^{f(\alpha)} - 1$].
- For every position i of σ operating on an α-counter $\mathtt{c}$, if the position contains an instruction '$\mathrm{next}(\alpha)$' [resp. '$\mathrm{prev}(\alpha)$'], then the next position $i+1$ operates on the α-counter $\mathtt{c}+1$ [resp. $\mathtt{c}-1$]; otherwise, the position $i+1$ operates on the α-counter $\mathtt{c}$.
- For every α-incrementing position i of σ operating on an α-counter $\mathtt{c}$ there is a future α-decrementing position $j > i$ on the same α-counter, so that $\sigma(i)(\mathtt{x}^\alpha_{inc}) = \sigma(j)(\mathtt{x}^\alpha_{dec})$.

In fact, these properties together with (a)–(d) are sufficient and necessary to encode a gainy and accepting run of $\mathcal{A}$, and they can be all expressed in LRV. Then, we obtain a polynomial-time reduction from Gainy(1) into the satisfiability problem for LRV(X, F). ∎

VI. A Robust Equivalence

We have seen that the satisfiability problem for LRV is equivalent to the control state reachability problem in an exponentially larger VASS. In this section we evaluate how robust is this result with LRV variants or fragments.

A. Infinite Words with Multiple Data

So far, we have considered only finite words with multiple data. It is also natural to consider the variant with infinite words, but it is known that this sometimes leads to undecidability. However, in this case the decidability and complexity results are preserved. Let LRV$_\omega$ be the variant of LRV in which infinite models of length ω are taken into account

instead of finite ones. The logics PLRV_ω, $\text{LRV}_\omega^\top$, etc. are defined accordingly.

Proposition 16. (I) $\text{SAT}(\text{PLRV}_\omega)$ is decidable.
(II) $\text{SAT}(\text{LRV}_\omega)$ is 2EXPSPACE-complete.

B. Adding MSO-Definable Temporal Operators

It is standard to extend LTL with MSO-definable temporal operators (see e.g. [30], [13]), and the same can be done with LRV. A temporal operator $\oplus$ of arity n is *MSO-definable* whenever there is a formula $\phi(\mathtt{x}, \mathtt{P_1}, \ldots, \mathtt{P_n})$ from monadic second-order logic with a unique free position variable $\mathtt{x}$ and with n free unary predicates such that $\sigma, i \models \oplus(\phi_1, \ldots, \phi_n)$ iff $\sigma \models \phi(i, X_1, \ldots, X_n)$ in MSO where each set X_j is equal to the set of positions from σ satisfying the formula ϕ_j, see e.g. [13]. The 2EXPSPACE upper bound is preserved with a fixed finite set of MSO-definable operators.

Theorem 17. Let $\{\oplus_1, \ldots, \oplus_N\}$ be a finite set of MSO-definable temporal operators. Satisfiability problem for LRV extended with $\{\oplus_1, \ldots, \oplus_N\}$ is 2EXPSPACE-complete.

Note that PLRV augmented with MSO-definable temporal operators is decidable too.

C. The Now Operator or the Effects of Moving the Origin

The satisfiability problem for Past LTL with the temporal operator Now is known to be EXPSPACE-complete [20]. The satisfaction relation is parameterized by the current position of the origin and past-time temporal operators use that position. For instance, $\sigma, i \models_o \mathtt{Now}\ \phi \overset{\text{def}}{\Leftrightarrow} \sigma, i \models_i \phi$ and $\sigma, i \models_o \phi_1 \mathsf{S} \phi_2 \overset{\text{def}}{\Leftrightarrow}$ there is $j \in [o, i]$ such that $\sigma, j \models_o \phi_2$ and for all $j' \in [j-1, i]$, we have $\sigma, j' \models_o \phi_1$. The powerful operator Now can be obviously defined in MSO but not with the above definition since it requires *two* free position variables, one of which refers to the current position of the origin.

Theorem 18. $\text{SAT}(\text{LRV} + \mathtt{Now})$ is 2EXPSPACE-complete.

This contrasts with the undecidability results from [17, Theorem 5] in presence of the operator Now.

D. Bounding the Number of Variables

Given the relationship between the number of variables in a formula and the number of counters needed in the corresponding VASS, we investigate the consequences of fixing the number of variables. Interestingly, this classical restriction has an effect only for $\text{LRV}^\top$, i.e., when test formulas ϕ are restricted to $\top$ in $\mathtt{x} \approx \langle \phi ? \rangle \mathtt{y}$. Let LRV_k [resp. $\text{LRV}_k^\top$, $\text{PLRV}_k^\top$] be the restriction to formulas with at most k variables. In [6, Theorem 5], it is shown that $\text{SAT}(\text{LRV}_1^\top)$ is PSPACE-complete by establishing a reduction into the reachability problem for VASS when counter values are linearly bounded. Below, we generalize this result for any $k \geq 1$ by using the proof of Theorem 9 and the fact that the control state reachability problem for VASS with at most k counters is in PSPACE.

Proposition 19. For every $k \geq 1$, $\text{SAT}(\text{LRV}_k^\top)$ is PSPACE-complete.

This does not imply that LRV_k is in PSPACE, since the reduction from LRV into $\text{LRV}^\top$ in Section IV-A introduces new variables. In fact, it introduces a number of variables that depends on the size of the formula. It turns out that this is unavoidable, and that its satisfiability problem is 2EXPSPACE-hard, by the following reduction.

Lemma 20. There is a polynomial-time reduction from $\text{SAT}(\text{LRV}^\top)$ into $\text{SAT}(\text{LRV}_1)$ [resp. $\text{SAT}(\text{PLRV}^\top)$ and $\text{SAT}(\text{PLRV}_1)$].

Proof sketch: Let $\varphi \in \text{LRV}^\top$ using k variables $\mathtt{x}_1, \ldots, \mathtt{x}_k$ so that $\sigma \models \varphi$. We will encode σ restricted to $\mathtt{x}_1, \ldots, \mathtt{x}_k$ inside a model σ_φ with only one variable, say $\mathtt{x}$. To this end, σ_φ is divided into N segments $s_1 \cdots s_N$ of equal length, where $N = |\sigma|$. A special data value d not used in σ plays the role of *delimiter* between segments and between positions that code values from σ. Suppose that d is a data value that is not in σ. Then, each segment s_i has length $k' = 2k + 1$, and is defined as the data values "$d\ d_1\ d\ d_2\ \ldots d\ d_k\ d$", where $d_j = \sigma(i)(\mathtt{x}_j)$. Figure 3 contains an example. We can force that the model has this shape with LRV_1. With this coding, we can tell that we are between two segments if there are two consecutive equal data values. In fact, we are at a position corresponding to $\mathtt{x}_i$ (for $i \in [1, k]$) inside a segment if we are standing at the $2i$-th element of a segment, and we can test this with the formula $\gamma_i = \mathsf{X}^{k'-2i}\mathtt{x} \approx \mathsf{X}^{k'-2i+1}\mathtt{x} \vee (\mathsf{X}^{k'-2i}\top \wedge \neg \mathsf{X}^{k'-2i+1}\top)$. Using the γ_i's, we translate $\mathtt{x}_i \approx \langle \top ? \rangle \mathtt{x}_j$ in φ into a formula that: moves to the position $2i$ of the segment (the one corresponding to $\mathtt{x}_i$), and tests $\mathtt{x} \approx \langle \gamma_j ? \rangle \mathtt{x}$. We can do this similarly with all formulas. ∎

Corollary 21. For all $k \geq 1$, $\text{SAT}(\text{LRV}_k)$ is 2EXPSPACE-complete.

Corollary 22. For all $k \geq 1$, $\text{SAT}(\text{PLRV}_k)$ is as hard as Reach(VASS).

E. The Power of Pairs of Repeating Values

Let us consider an LRV variant so that repetition of tuples of values is possible: we add formulas of the form $(\mathtt{x}_1, \ldots, \mathtt{x}_k) \approx \langle \phi ? \rangle (\mathtt{y}_1, \ldots, \mathtt{y}_k)$ where $\mathtt{x}_1, \ldots, \mathtt{x}_k, \mathtt{y}_1, \ldots, \mathtt{y}_k \in \text{VAR}$. This extends $\mathtt{x} \approx \langle \varphi ? \rangle \mathtt{y}$ by testing whether the vector of data values from the variables $(\mathtt{x}_1, \ldots, \mathtt{x}_k)$ of the current position coincides with that of $(\mathtt{y}_1, \ldots, \mathtt{y}_k)$ in a future position.

We call this extension LRV_{vec}. Unfortunately, we can show that $\text{SAT}(\text{LRV}_{vec})$ is undecidable, even when only tuples of dimension 2 are allowed. This is proved by reduction from a variant of Post's Correspondence Problem (PCP). In order to code solutions of PCP instances, we adapt a proof technique used in [1] for first-order logic with two variables and two equivalence relations on words. However, our proof uses only *future* modalities (unlike the proof of [1, Proposition 27]) and no past obligations (unlike the proof of [17, Theorem 4]). To prove this result, we work with a variant of the PCP problem in which solutions $u_{i_1} \cdots u_{i_n} = v_{i_1} \cdots v_{i_n}$ have to satisfy $|u_{i_1} \cdots u_{i_j}| \leq |v_{i_1} \cdots v_{i_j}|$ for every j.

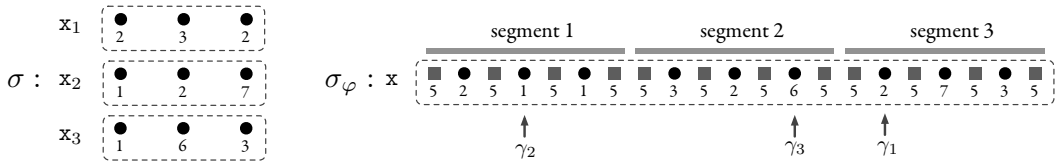


Fig. 3. Example of the reduction from SAT(LRV$^\top$) into SAT(LRV$_1$), for $k = 3$, $N = 3$ and $d = 5$.

Theorem 23. SAT(LRV$_{vec}$(X, U)) is undecidable.

Furthermore, our proof can be adapted to show that SAT(LRV$^\top_{vec}$) is also undecidable.

VII. Implications for Logics on Data Words

A data word is an element of $(\Sigma \times \mathbb{D})^*$, where Σ is a finite alphabet and $\mathbb{D}$ is an infinite domain. We focus here on first-order logic with two variables, and on a temporal logic.

Two-variable Logics: We study a fragment of EMSO$^2(+1, <, \sim)$ on data words, and we show that it has a satisfiability problem in 3EXPSPACE, as a consequence of our results on the satisfiability for LRV. The satisfiability problem for EMSO$^2(+1, <, \sim)$ is known to be decidable, equivalent to the reachability problem for VASS [1], with no known primitive-recursive algorithm. Here we show a large fragment with elementary complexity.

Consider the fragment of EMSO$^2(+1, <, \sim)$ —that is, first-order logic with two variables, with a prefix of existential quantification over monadic relations— where all formulas are of the form $\exists X_1, \ldots, X_n\ \varphi$ with

$$\begin{aligned} \varphi ::= & \ \mathit{atom} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \\ & \exists x \exists y\ \varphi \mid \forall x \forall y\ \varphi \mid \forall x \exists y\ (x \leq y\ \wedge\ \varphi), \text{ where} \\ \mathit{atom} ::= & \ \zeta = \zeta' \mid \zeta \neq \zeta' \mid \zeta \sim \zeta' \mid \zeta < \zeta' \mid \zeta \leq \zeta' \mid \\ & +1(\zeta, \zeta') \mid X_i(\zeta) \mid a(\zeta) \end{aligned}$$

for any $a \in \Sigma$, $i \in [1, n]$, and $\zeta, \zeta' \in \{x, y\}$. The relation $x < y$ tests that the position y appears after x in the word; $+1(x, y)$ tests that y is the next position to x; and $x \sim y$ tests that positions x and y have the same data value. We call this fragment *forward*-EMSO$^2(+1, <, \sim)$. In fact, *forward*-EMSO$^2(+1, <, \sim)$ captures EMSO$^2(+1, <)$ (i.e., all regular languages on the finite labeling of the data word).[1] However, it seems to be strictly less expressive than EMSO$^2(+1, <, \sim)$, since it does not appear to be able to express, for example, *there are exactly two occurrences of every data value*. Yet, it can express *there are at most two occurrences of every data value* (with $\exists X\ \forall x \forall y.\ x \sim y \wedge x < y \rightarrow X(x) \wedge \neg X(y)$), and *there is exactly one occurrence of every data value*. For the same reason, it would neither capture EMSO$^2(+1, \sim)$.

By an exponential reduction into SAT(LRV), we obtain that *forward*-EMSO$^2(+1, <, \sim)$ is decidable in elementary time.

[1]Indeed, note that one can easily test whether a word is accepted by a finite automaton with the help of a monadic relation X_{fst} that holds only at the first position. Further, the property of X_{fst} can be expressed in *forward*-EMSO$^2(+1, <, \sim)$ as $(\exists y\ .\ X_{fst}(y)) \wedge (\forall x \forall y\ .\ x < y \rightarrow \neg X_{fst}(y))$.

Proposition 24. The satisfiability problem for *forward*-EMSO$^2(+1, <, \sim)$ is in 3EXPSPACE.

Proof sketch: Through a standard translation we can bring any formula of *forward*-EMSO$^2(+1, <, \sim)$ into a formula of the form

$$\varphi = \exists X_1, \ldots, X_n\ \left(\forall x \forall y\ \chi\ \wedge\ \bigwedge_k \forall x \exists y\ (x \leq y \wedge \psi_k) \right)$$

that preserves satisfiability, where χ and all ψ_k's are quantifier-free formulas, and there are no tests for labels. Furthermore, this is a polynomial-time translation. This translation is just the Scott normal form of EMSO$^2(+1, <, \sim)$ [1] adapted to *forward*-EMSO$^2(+1, <, \sim)$, and can be done in the same way.

We now give an exponential-time translation tr : *forward*-EMSO$^2(+1, <, \sim) \rightarrow$ LRV. For any formula φ of *forward*-EMSO$^2(+1, <, \sim)$, $tr(\varphi)$ is an equivalent (in the sense of satisfiability) LRV formula.

The translation makes use of: a distinguished variable $\mathtt{x}$ that encodes the data values of any data word satisfying φ; variables $\mathtt{x}_0, \ldots, \mathtt{x}_n$ that are used to encode the monadic relations $X_1, \ldots, X_n$; and a variable $\mathtt{x}_{prev}$ whose purpose will be explained later on. We give now the translation. To translate $\forall x \forall y\ \chi$, we first bring the formula to a form

$$\bigwedge_{m \in M} \neg\exists x \exists y\ (\ x \leq y\ \wedge\ \chi_m\ \wedge\ \chi^x_m\ \wedge\ \chi^y_m\ ), \qquad (\star)$$

where every χ^x_m (resp. χ^y_m) is a conjunction of (negations of) atoms of monadic relations on x (resp. y); and $\chi_m = \mu \wedge \nu$ where $\mu \in \{x{=}y, +1(x, y), \neg(+1(x, y) \vee x{=}y)\}$ and $\nu \in \{x \sim y, \neg(x \sim y)\}$. As an example, if $\chi = \neg(x \sim y) \vee X(x) \vee X(y)$, then the corresponding formula would be

$$\bigwedge_{\mu, \chi^x, \chi^y} \begin{array}{l} \big((\neg\exists x \exists y\ x \leq y \wedge \mu \wedge \neg X(x) \wedge \chi^y)\ \wedge \\ \ (\neg\exists x \exists y\ x \leq y \wedge \mu \wedge \chi^x \wedge \neg X(y))\big) \end{array}$$

for all $\mu \in \{x{=}y, +1(x, y), \neg(+1(x, y) \vee x{=}y)\}$, $\chi^x \in \{X(x), \neg X(x)\}$, $\chi^y \in \{X(y), \neg X(y)\}$. We can bring the formula into this normal form in exponential time. We define $tr(\chi^x_m)$ as the conjunction of all the formulas $\mathtt{x}_0 \approx \mathtt{x}_i$ so that $X_i(x)$ is a conjunct of χ^x_m, and all the formulas $\neg(\mathtt{x}_0 \approx \mathtt{x}_i)$ so that $\neg X_i(x)$ is a conjunct of χ^x_m; we do similarly for $tr(\chi^y_m)$. If $\mu = +1(x, y)$ and $\nu = x \sim y$ we define $tr\left(\exists y\big(x \leq y \wedge \chi_m \wedge \chi^x_m \wedge \chi^y_m\big)\right)$ as $\mathtt{x} \approx \mathsf{X}\mathtt{x} \wedge tr(\chi^x_m) \wedge \mathsf{X} tr(\chi^y_m)$.

If $\mu = (x = y)$ and $\nu = x \sim y$ we translate

$$tr\left(\exists y\big(x \leq y \wedge \chi_m \wedge \chi^x_m \wedge \chi^y_m\big)\right)\ =\ tr(\chi^x_m) \wedge tr(\chi^y_m).$$

We proceed similarly for $\mu = +1(x, y)$, $\nu = \neg(x \sim y)$; and the translation is of course $\bot$ (false) if $\mu = (x{=}y)$, $\nu = \neg(x{\sim}y)$. The difficult cases are the remaining ones. Suppose $\mu = \neg(+1(x, y) \vee x{=}y)$, $\nu = x{\sim}y$. In other words, x is at least two positions before y, and they have the same data value. Observe that the formula $tr(\chi^x_m) \wedge \mathtt{x} \approx \langle tr(\chi^y_m)?\rangle \mathtt{x}$ does not encode precisely this case, as it would correspond to a weaker condition $x < y \wedge x \sim y$. In order to properly translate this case we make use of the variable $\mathtt{x}_{prev}$, ensuring that it always has the data value of the variable $\mathtt{x}$ in the previous position

$$prev \;=\; \mathsf{G}\big(\; \mathsf{X}\top \;\Rightarrow\; \mathtt{x} \approx \mathsf{X}\mathtt{x}_{prev} \;\big).$$

We then define $tr\left(\exists y \;\big(\; x \leq y \;\wedge\; \chi_m \;\wedge\; \chi^x_m \;\wedge\; \chi^y_m \;\big)\right)$ as

$$tr(\chi^x_m) \;\wedge\; \mathtt{x} \approx \langle \mathtt{x}_{prev} \approx \langle tr(\chi^y_m)?\rangle \mathtt{x}?\rangle\, \mathtt{x}_{prev}.$$

Note that by nesting twice the future obligation we ensure that the target position where $tr(\chi^y_m)$ must hold is at a distance of at least two positions. For $\nu = \neg(x{\sim}y)$ we produce a similar formula, replacing the innermost appearance of $\approx$ with $\not\approx$ in the formula above. We then define $tr(\forall x \forall y\ \chi)$ as

$$prev \;\wedge\; \bigwedge_{m \in M} \neg\, \mathsf{F}\; tr\big(\exists y\ (x \leq y \;\wedge\; \chi_m \wedge \chi^x_m \wedge \chi^y_m)\big).$$

To translate $\forall x \exists y\ (x \leq y \wedge \psi_k)$ we proceed in a similar way. One can show that tr preserves satisfiability. By Corollary 10 we can decide the satisfiability of the translation in 2EXPSPACE, and since the translation is exponential, this gives us a 3EXPSPACE bound for the satisfiability of *forward*-EMSO$^2(+1, <, \sim)$. ∎

Temporal Logics: Our results have also implications for a fragment of the logic LTL extended with one register for storing and comparing data values (called the *freeze* operator), noted $\mathrm{LTL}^{\downarrow}_1$. Its satisfiability problem was shown to be decidable, but with non-primitive-recursive complexity [7].

Our results yield a fragment of $\mathrm{LTL}^{\downarrow}_1$ with elementary 2EXPSPACE upper bound.

VIII. Conclusion

We introduced the logic LRV and variants by allowing data value repetitions thanks to formulas of the form $\mathtt{x} \approx \langle \phi? \rangle \mathtt{y}$. LRV extends the main logic from [6] but it is also a fragment of BD-LTL from [17] whose satisfiability is equivalent to Reach(VASS). We showed that SAT(LRV) is 2EXPSPACE-complete by reduction into the control-state reachability problem for VASS (via a detour to gainy VASS) and by introducing a new class of counter machines (the chain systems) in order to get the complexity lower bound. This new class of counter machines is interesting for its own sake and could be used to establish other hardness results thanks to our results that extend non-trivially the proof from [22], [8]. Correspondences between extensions of LRV (such as PLRV, PLRV$^\top$ and PLRV$_1$) and reachability problem for VASS are also established, strengthening further the relationships between data logics and reachability problems for counter machines. Other results for variants are presented in the paper and a summary can be found below.

$$\begin{aligned} \mathrm{LRV}^\top_k &: \text{PSPACE-complete} \\ \mathrm{LRV} \equiv \mathrm{LRV}^\top \equiv \mathrm{LRV}_1 \equiv \mathrm{LRV} + \{\oplus_1, \ldots, \oplus_k\} &: \text{2EXPSPACE-complete} \\ \mathrm{PLRV} \equiv \mathrm{PLRV}^\top \equiv \mathrm{PLRV}_1 &\equiv \text{Reach(VASS)} \\ \mathrm{LRV}^\top_{vec} &: \text{undecidable} \end{aligned}$$

References

[1] M. Bojańczyk, C. David, A. Muscholl, Th. Schwentick, and L. Segoufin. Two-variable logic on data words. *ACM ToCL*, 12(4):27, 2011.

[2] M. Bojańczyk and S. Lasota. An extension of data automata that captures XPath. In *LICS'10*, pages 243–252. IEEE, 2010.

[3] B. Bollig, A. Cyriac, P. Gastin, and K. Narayan Kumar. Model checking languages of data words. In *FoSSaCS'12*, volume 7213 of *LNCS*, pages 391–405. Springer, 2012.

[4] P. Bouyer. A logical characterization of data languages. *IPL*, 84(2):75–85, 2002.

[5] C. David. *Analyse de XML avec données non-bornées*. PhD thesis, LIAFA, 2009.

[6] S. Demri, D. D'Souza, and R. Gascon. Temporal logics of repeating values. *JLC*, 22(5):1059–1096, 2012.

[7] S. Demri and R. Lazić. LTL with the freeze quantifier and register automata. *ACM ToCL*, 10(3), 2009.

[8] J. Esparza. Decidability and complexity of Petri net problems — an introduction. In *Advances in Petri Nets 1998*, volume 1491 of *LNCS*, pages 374–428. Springer, Berlin, 1998.

[9] D. Figueira. *Reasoning on words and trees with data*. PhD thesis, ENS Cachan, 2010.

[10] D. Figueira. A decidable two-way logic on data words. In *LICS'11*, pages 365–374. IEEE, 2011.

[11] D. Figueira and L. Segoufin. Future-looking logics on data words and trees. In *MFCS'09*, volume 5734 of *LNCS*, pages 331–343, 2009.

[12] M. Fitting. Modal logic between propositional and first-order. *JLC*, 12(6):1017–1026, 2002.

[13] P. Gastin and D. Kuske. Satisfiability and model checking for MSO-definable temporal logics are in PSPACE. In *CONCUR'03*, volume 2761 of *LNCS*, pages 222–236. Springer, 2003.

[14] J. Hopcroft, R. Motwani, and J. Ullman. *Introduction to Automata Theory, Languages, and Computation (2nd. ed.)*. Addison Wesley, 2001.

[15] J. Hopcroft and J.J. Pansiot. On the reachability problem for 5-dimensional vector addition systems. *TCS*, 8:135–159, 1979.

[16] P. Jančar. Undecidability of bisimilarity for Petri Nets and some related problems. *TCS*, 148:281–301, 1995.

[17] A. Kara, Th. Schwentick, and Th. Zeume. Temporal logics on words with multiple data values. In *FST&TCS'10*, pages 481–492. LZI, 2010.

[18] S. Rao Kosaraju. Decidability of reachability in vector addition systems. In *STOC'82*, pages 267–281, 1982.

[19] O. Kupferman and M. Vardi. Memoryful Branching-Time Logic. In *LICS'06*, pages 265–274. IEEE, 2006.

[20] F. Laroussinie, N. Markey, and Ph. Schnoebelen. Temporal logic with forgettable past. In *LICS'02*, pages 383–392. IEEE, 2002.

[21] J. Leroux. Vector addition system reachability problem: a short self-contained proof. In *POPL'11*, pages 307–316, 2011.

[22] R. J. Lipton. The reachability problem requires exponential space. Technical Report 62, Dept. of Computer Science, Yale University, 1976.

[23] A. Lisitsa and I. Potapov. Temporal logic with predicate λ-abstraction. In *TIME'05*, pages 147–155. IEEE, 2005.

[24] A. Manuel and R. Ramanujam. Counting multiplicity over infinite alphabets. In *RP'09*, volume 5797 of *LNCS*, pages 141–153, 2009.

[25] E.W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM Journal of Computing*, 13(3):441–460, 1984.

[26] F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM ToCL*, 5(3):403–435, 2004.

[27] M. Niewerth and Th. Schwentick. Two-variable logic and key constraints on data words. In *ICDT'11*, pages 138–149. ACM, 2011.

[28] C. Rackoff. The covering and boundedness problems for vector addition systems. *TCS*, 6(2):223–231, 1978.

[29] L. Segoufin. Automata and logics for words and trees over an infinite alphabet. In *CSL'06*, volume 4207 of *LNCS*, pages 41–57. Springer, 2006.

[30] P. Wolper. Temporal logic can be more expressive. *I&C*, 56:72–99, 1983.

APPENDIX

A. Proof of Theorem 1

Proof: First, the reachability problem for VASS can be reduced in polynomial-space to its restriction such that the initial and final configurations have all the counters equal to zero and each transition can only increment or decrement a unique counter. In the sequel, we consider an instance of this subproblem: $\mathcal{A} = \langle Q, C, \delta \rangle$ is a VASS, the initial configuration is $\langle q_i, \vec{0} \rangle$ and the final configuration is $\langle q_f, \vec{0} \rangle$.

Now, we build a formula ϕ in PLRV such that $\langle q_f, \vec{0} \rangle$ is reachable from $\langle q_i, \vec{0} \rangle$ iff ϕ is satisfiable. To do so, we encode runs of $\mathcal{A}$ by data words following exactly the proof of [1, Theorem 16] except that the properties are expressed in PLRV instead of FO2$(\sim, <, +1)$.

The objective is to encode a word $\rho \in \delta^*$ that represents an accepting run from $\langle q_i, \vec{0} \rangle$ to $\langle q_f, \vec{0} \rangle$. We use the alphabet δ of transitions, that we code using a logarithmic number of variables. One can simulate m different labels in PLRV, by using $\lceil \log(m) + 1 \rceil$ variables and its equivalence classes. In order to simulate the alphabet δ, we use the variables $\mathtt{x}_0, \ldots, \mathtt{x}_N$, with $N = \lceil \log(\mathrm{card}(\delta)) \rceil$. For any $t \in \delta$, let $\langle t \rangle \in$ PLRV be the formula that tests for label t at the current position. More precisely, for any fixed injective function $\lambda : \delta \to 2^{[1,N]}$ we define

$$\langle t \rangle \quad = \quad \bigwedge_{i \in \lambda(t)} \mathtt{x}_0 = \mathtt{x}_i \quad \wedge \quad \bigwedge_{1 \leq i \leq N, i \notin \lambda(t)} \mathtt{x}_0 \neq \mathtt{x}_i.$$

Note that $\langle t \rangle$ uses exclusively variables $\mathtt{x}_0, \ldots, \mathtt{x}_N$, that is of size logarithmic in $\mathrm{card}(\delta)$, and that $\langle t \rangle$ holds at a position for at most one $t \in \delta$. We build a PLRV formula ϕ so that any word from δ^* corresponding to a model of ϕ is an accepting run for $\mathcal{A}$ from $\langle q_i, \vec{0} \rangle$ to $\langle q_f, \vec{0} \rangle$. And conversely, for any accepting run of $\mathcal{A}$ from $\langle q_i, \vec{0} \rangle$ to $\langle q_f, \vec{0} \rangle$ there is a model of ϕ corresponding to the run. The following are standard counter-blind conditions to check.

1) Every position satisfies $\langle t \rangle$ for some $t \in \delta$.
2) The first position satisfies $\langle \langle q_i, \mathrm{instr}, q \rangle \rangle$ for some $q \in Q$.
3) The last position satisfies $\langle \langle q, \mathrm{instr}, q_f \rangle \rangle$ for some $q \in Q$.
4) There are no two consecutive positions i and $i+1$ satisfying $\langle \langle q_1, \mathrm{instr}, q_2 \rangle \rangle$ and $\langle \langle q_1', \mathrm{instr}', q_2' \rangle \rangle$ respectively, with $q_2 \neq q_1'$.

In the formula ϕ, we use the variable $\mathtt{x}$ to relate increments and decrements. Here are the main properties to satisfy.

1) For every counter $\mathtt{c} \in C$, there are no two positions labelled by a transition with instruction $\mathrm{inc}(\mathtt{c})$ having the same same value for $\mathtt{x}$:

$$\mathsf{G}(\mathrm{inc}(\mathtt{c}) \Rightarrow \neg(\mathtt{x} \approx \langle \mathrm{inc}(\mathtt{c})? \rangle \mathtt{x}))$$

   where $\mathrm{inc}(\mathtt{c})$ is a shortcut for $\bigvee_{\langle q, \mathrm{inc}(\mathtt{c}), q' \rangle \in \delta} \langle t \rangle$. A similar constraint can be expressed with $\mathrm{dec}(\mathtt{c})$.
2) For every counter $\mathtt{c} \in C$, for every position labelled by a transition with instruction $\mathrm{inc}(\mathtt{c})$, there is a future position labelled by a transition with instruction $\mathrm{dec}(\mathtt{c})$ with the same value for $\mathtt{x}$:

$$\mathsf{G}(\mathrm{inc}(\mathtt{c}) \Rightarrow \mathtt{x} \approx \langle \mathrm{dec}(\mathtt{c})? \rangle \mathtt{x})$$

   where $\mathrm{dec}(\mathtt{c})$ is a shortcut for $\bigvee_{\langle q, \mathrm{dec}(\mathtt{c}), q' \rangle \in \delta} \langle t \rangle$. This will guarantee that the final configuration ends with all counters equal to zero.
3) Similarly, for every counter $\mathtt{c} \in C$, for every position labelled by a transition with instruction $\mathrm{dec}(\mathtt{c})$, there is a past position labelled by a transition with instruction $\mathrm{inc}(\mathtt{c})$ with the same value for $\mathtt{x}$:

$$\mathsf{G}(\mathrm{dec}(\mathtt{c}) \Rightarrow \mathtt{x} \approx \langle \mathrm{inc}(\mathtt{c})? \rangle^{-1} \mathtt{x})$$

   This will guarantee that every decrement follows a corresponding increment, satisfying that counter values are never negative.

Let ϕ be the conjunction of all the formulas defined above. Since all the properties considered herein are those used in the proof of [1, Theorem 16] (but herein they are expressed in PLRV instead of FO2$(\sim, <, +1)$), it follows that ϕ is satisfiable in PLRV iff there is an accepting run of $\mathcal{A}$ from $\langle q_i, \vec{0} \rangle$ to $\langle q_f, \vec{0} \rangle$. ■

B. Proofs of Propositions 2 and 3

Henceforward, $\mathit{vars}(\varphi)$ denotes the set of all variables in φ, and $\mathit{sub}_{\langle\rangle}(\varphi)$ the set of all subformulas ψ such that $\mathtt{x} \approx \langle\psi?\rangle\mathtt{y}$ or $\mathtt{x} \not\approx \langle\psi?\rangle\mathtt{y}$ appears in φ for some $\mathtt{x}, \mathtt{y} \in \mathit{vars}(\varphi)$.

In both reductions we make use of the following easy lemma.

Lemma 25. There is a polynomial-time satisfiability-preserving translation t : LRV $\to$ LRV [resp. t : LRV$^{\approx} \to$ LRV$^{\approx}$] such that for every φ and $\psi \in \mathit{sub}_{\langle\rangle}(t(\varphi))$, we have $\mathit{sub}_{\langle\rangle}(\psi) = \emptyset$.

Proof: This is a standard reduction. Indeed, given a formula φ with subformula $\mathtt{x} \approx \langle\psi?\rangle\mathtt{y}$, φ is satisfiable iff $\mathsf{G}(\psi \Leftrightarrow \mathtt{x}_{new} \approx \mathtt{y}_{new}) \wedge \varphi[\mathtt{x} \approx \langle\psi?\rangle\mathtt{y} \leftarrow \mathtt{x} \approx \langle\mathtt{x}_{new} \approx \mathtt{y}_{new}?\rangle\mathtt{y}]$ is satisfiable, where $\mathtt{x}_{new}, \mathtt{y}_{new} \notin \mathit{vars}(\varphi)$, and $\varphi[\mathtt{x} \approx \langle\psi?\rangle\mathtt{y} \leftarrow \mathtt{x} \approx \langle\mathtt{x}_{new} \approx \mathtt{y}_{new}?\rangle\mathtt{y}]$ is the result of replacing every occurrence of $\mathtt{x} \approx \langle\psi?\rangle\mathtt{y}$ by $\mathtt{x} \approx \langle\mathtt{x}_{new} \approx \mathtt{y}_{new}?\rangle\mathtt{y}$ in φ. ■

Proof of Proposition 2: We show how to compute in polynomial time, for every $\varphi \in$ LRV, a formula $\varphi' \in$ LRV$^{\approx}$ that preserves satisfiability. The formula φ' uses, besides all the variables from φ, a distinguished variable $\mathtt{k} \notin \mathit{vars}(\varphi)$, and variables $\mathtt{v}^{\approx}_{\mathtt{x},\psi}$, $\mathtt{v}_{\mathtt{x}\not\approx\langle\psi?\rangle\mathtt{y}}$ for every subformula ψ of φ and variables $\mathtt{x}, \mathtt{y} \in \mathit{vars}(\varphi)$.

The idea behind the coding is the following. Any positive test $\mathtt{x} \not\approx \langle\psi?\rangle\mathtt{y}$ can be always converted to a test $\mathtt{x} \not\approx \mathtt{v}_{\mathtt{x}\not\approx\langle\psi?\rangle\mathtt{y}} \wedge \mathtt{v}_{\mathtt{x}\not\approx\langle\psi?\rangle\mathtt{y}} \approx \langle\psi?\rangle\mathtt{y}$, where $\mathtt{v}_{\mathtt{x}\not\approx\langle\psi?\rangle\mathtt{y}}$ is a fresh variable. At any given position, we only need at most as many fresh variables as there are subformulas ψ and source and target variables $\mathtt{x}, \mathtt{y} \in \mathit{vars}(\varphi)$, and we hence index these fresh variables by $\mathtt{x}$, $\mathtt{y}$ and ψ. On the other hand, when a negative test $\neg(\mathtt{x} \not\approx \langle\psi?\rangle\mathtt{y})$ holds at a position i, it means that all positions satisfying ψ to the right of i have the same data value in the variable $\mathtt{y}$. Since this data value is determined for every $\psi, \mathtt{y}$, there is only boundedly many data values to remember, one for every subformula and variable. We use separate variables $\mathtt{v}^{\approx}_{\mathtt{x},\psi}$ to remember this data value for every $\mathtt{x}, \psi$. This variable is used in conjunction with $\mathtt{k}$. Whenever $\mathtt{v}^{\approx}_{\mathtt{x},\psi} \not\approx \mathtt{k}$ at a position i, it means that $\neg(\mathtt{v}^{\approx}_{\mathtt{x},\psi} \not\approx \langle\psi?\rangle\mathtt{x})$, and we can make sure of this without using $\not\approx$. Indeed, we can do this by letting $\mathtt{v}^{\approx}_{\mathtt{x},\psi}$ maintain the same value from position i onwards —until the last element verifying ψ—, we can test that whenever ψ holds, we have $\mathtt{x} \approx \mathtt{v}^{\approx}_{\mathtt{x},\psi}$.

For any model σ, let σ_φ be so that

(a) $|\sigma| = |\sigma_\varphi|$;
(b) for every $0 \le i < |\sigma|$ and $\mathtt{x} \in \mathit{vars}(\varphi)$, $\sigma(i)(\mathtt{x}) = \sigma_\varphi(i)(\mathtt{x})$;
(c) there is some data value $d \notin \{\sigma(i)(\mathtt{x}) \mid \mathtt{x} \in \mathit{vars}(\varphi), 0 \le i < |\sigma|\}$ such that for every $0 \le i < |\sigma_\varphi|$, $\sigma_\varphi(i)(\mathtt{k}) = d$;
(d) for every $0 \le i < |\sigma|$, $\mathtt{x} \in \mathit{vars}(\varphi)$, and $\psi \in \mathit{sub}_{\langle\rangle}(\varphi)$,
   - if for some $j \ge i$, $\sigma, j \models \psi$ and for every $i \le j' < |\sigma|$ such that $\sigma, j' \models \psi$ we have $\sigma(j)(\mathtt{x}) = \sigma(j')(\mathtt{x})$, then $\sigma_\varphi(i)(\mathtt{v}^{\approx}_{\mathtt{x},\psi}) = \sigma(j)(\mathtt{x})$,
   - otherwise, $\sigma_\varphi(i)(\mathtt{v}^{\approx}_{\mathtt{x},\psi}) = \sigma_\varphi(i)(\mathtt{k})$;
(e) for every $0 \le i < |\sigma|$, $\mathtt{x}, \mathtt{y} \in \mathit{vars}(\varphi)$, and $\psi \in \mathit{sub}_{\langle\rangle}(\varphi)$,
   - if for some $j > i$, $\sigma(j)(\mathtt{y}) \ne \sigma(i)(\mathtt{x})$ and $\sigma, j \models \psi$, then let j_0 be the first such j, and $\sigma_\varphi(i)(\mathtt{v}_{\mathtt{x}\not\approx\langle\psi?\rangle\mathtt{y}}) = \sigma(j_0)(\mathtt{y})$,
   - otherwise, $\sigma_\varphi(i)(\mathtt{v}_{\mathtt{x}\not\approx\langle\psi?\rangle\mathtt{y}}) = \sigma_\varphi(i)(\mathtt{k})$.

For every other unmentioned variable, σ and σ_φ coincide. It is evident that for every φ and σ, a model σ_φ with the aforementioned properties exists. Next, we define $\varphi' \in$ LRV$^{\approx}$ so that

$$\sigma \models \varphi \text{ if, and only if, } \sigma_\varphi \models \varphi'. \qquad (\dagger)$$

We assume that for every $\psi \in \mathit{sub}_{\langle\rangle}(\varphi)$, we have that $\mathit{sub}_{\langle\rangle}(\psi) = \emptyset$; this is without any loss of generality due to Lemma 25. We will also assume that φ is in negation normal form, that is, negation appears only in subformulas of the type $\neg(\mathtt{x} \approx \mathsf{X}^{\ell}\mathtt{y})$ or $\neg(\mathtt{x} \approx \langle\psi?\rangle\mathtt{y})$ [resp. $\not\approx$]. In particular, this means that we introduce a dual operator for each temporal operator.

- First, the variable $\mathtt{k}$ will act as a constant; it will always have the same data value at any position of the model, which must be different from those of all variables of φ.

$$\mathit{const} \;=\; \mathsf{G}\left((\mathtt{k} \approx \mathsf{X}\mathtt{k} \vee \neg\mathsf{X}\top) \;\wedge \bigwedge_{\mathtt{x}\in\mathit{vars}(\varphi)} (\mathtt{k} \not\approx \mathtt{x})\right)$$

- Second, for every position we ensure that if $\mathtt{v}^{\approx}_{\mathtt{x},\psi}$ is different from $\mathtt{k}$, then it preserves its value until the last element verifying ψ; and if ψ holds at any of these positions then $\mathtt{v}^{\approx}_{\mathtt{x},\psi}$ contains the value of $\mathtt{x}$.

$$\mathit{val}\text{-}\mathtt{v}^{\approx}_{\mathtt{x},\psi} \;=\; \mathsf{G}\left(\begin{array}{rcl} \mathtt{k} \not\approx \mathtt{v}^{\approx}_{\mathtt{x},\psi} & \Rightarrow & \mathtt{v}^{\approx}_{\mathtt{x},\psi} \approx \mathsf{X}\mathtt{v}^{\approx}_{\mathtt{x},\psi} \vee \neg\mathsf{X}\mathsf{F}\psi \quad \wedge \\ \mathtt{k} \not\approx \mathtt{v}^{\approx}_{\mathtt{x},\psi} \wedge \psi & \Rightarrow & \mathtt{v}^{\approx}_{\mathtt{x},\psi} \approx \mathtt{x} \end{array}\right)$$

- Finally, let $\varphi^{\approx}$ be the result of replacing
  (a) every appearance of $\neg(\mathtt{x} \not\approx \langle\psi?\rangle\mathtt{y})$ by $\neg\mathsf{X}\mathsf{F}\psi \;\vee\; \mathtt{x} \approx \mathsf{X}\mathtt{v}^{\approx}_{\mathtt{y},\psi}$, and
  (b) every positive appearance of $\mathtt{x} \not\approx \langle\psi?\rangle\mathtt{y}$ by $\mathtt{x} \not\approx \mathtt{v}_{\mathtt{x}\not\approx\langle\psi?\rangle\mathtt{y}} \wedge \mathtt{v}_{\mathtt{x}\not\approx\langle\psi?\rangle\mathtt{y}} \approx \langle\psi?\rangle\mathtt{y}$
  in φ.

We then define the translation φ' as follows.

$$\varphi' \;=\; \varphi^{\approx} \;\wedge\; \mathit{const} \;\wedge \bigwedge_{\substack{\psi\in \mathit{sub}_{\langle\rangle}(\phi),\\ \mathtt{x}\in \mathit{vars}(\varphi)}} \mathit{val}\text{-}\mathtt{v}^{\approx}_{\mathtt{x},\psi}$$

Notice that φ' can be computed from φ in polynomial time.

Claim 26. φ is satisfiable if, and only if, φ' is satisfiable.

Proof:

[$\Rightarrow$] Note that one direction would follow from (†): if φ is satisfiable by some σ, then φ' is satisfiable in σ_φ. In order to establish (†), we show that

$$\text{for every subformula } \gamma \text{ of } \varphi \text{ and } 0 \le i < |\sigma_\varphi|, \text{ we have } \sigma, i \models \gamma \text{ iff } \sigma_\varphi, i \models \gamma^{\approx}. \qquad (\ddagger)$$

Since $\sigma_\varphi \models \mathit{const}$ by condition (c), and $\sigma_\varphi \models \mathit{val}\text{-}\mathtt{v}^{\approx}_{\mathtt{x},\psi}$ for every ψ due to (d), this is sufficient to conclude that $\sigma \models \varphi$ iff $\sigma_\varphi \models \varphi'$.

We show (‡) by structural induction. If $\gamma = \mathtt{x} \not\approx \langle\psi?\rangle\mathtt{y}$, then $\sigma, i \models \gamma$ if there is some $j > i$ where $\sigma(j)(\mathtt{y}) \neq \sigma(i)(\mathtt{x})$ and $\sigma, j \models \psi$. Let j_0 be the first such j. Then, by condition (e), $\sigma(i)(\mathtt{x}) \neq \sigma_\varphi(i)(\mathtt{v}_{\mathtt{x}\not\approx\langle\psi?\rangle\mathtt{y}}) = \sigma(j_0)(\mathtt{y})$. Hence, $\sigma_\varphi, i \models \mathtt{v}_{\mathtt{x}\not\approx\langle\psi?\rangle\mathtt{y}} \approx \langle\psi?\rangle\mathtt{y}$ and $\sigma_\varphi, i \models \mathtt{x} \not\approx \mathtt{v}_{\mathtt{x}\not\approx\langle\psi?\rangle\mathtt{y}}$ and thus $\sigma_\varphi, i \models \gamma^{\approx}$.

If, on the other hand, $\gamma = \neg(\mathtt{x} \not\approx \langle\psi?\rangle\mathtt{y})$ then either

- there is no $j > i$ so that $\sigma, j \models \psi$, or, otherwise,
- for every $j' \ge i+1$ so that $\sigma, j' \models \psi$ we have $\sigma(j')(\mathtt{y}) = \sigma(i)(\mathtt{x})$.

In the first case we have that $\sigma_\varphi, i \models \neg\mathsf{X}\mathsf{F}\psi$, and in the second case we have that, by (d), $\sigma_\varphi(i')(\mathtt{v}^{\approx}_{\mathtt{y},\psi}) = \sigma_\varphi(i)(\mathtt{x})$ for every $i' > i$, and in particular for $i' = i+1$. Hence, $\sigma_\varphi, i \models \neg\mathsf{X}\mathsf{F}\psi \;\vee\; \mathtt{x} \approx \mathsf{X}\mathtt{v}^{\approx}_{\mathtt{y},\psi}$ and thus $\sigma_\varphi, i \models \gamma^{\approx}$.

Finally, the proof for the base case of the form $\gamma = \mathtt{x} \approx \mathsf{X}^\ell\mathtt{y}$ [resp. $\not\approx$] and for all boolean and temporal operators are by an easy verification, since $(\cdot)^{\approx}$ is homomorphic for these. Hence, (‡) holds.

[$\Leftarrow$] Now suppose that $\sigma \models \varphi'$. Since $\sigma \models \mathit{const}$, we have that σ verifies condition (c). And since $\sigma \models \mathit{val}\text{-}\mathtt{v}^{\approx}_{\mathtt{x},\psi}$ for every $\psi \in \mathit{sub}_{\langle\rangle}(\varphi)$, $\mathtt{x} \in \mathit{vars}(\varphi)$, we have that σ verifies (d). We prove by structural induction that for every subformula γ of φ, if $\sigma, i \models \gamma^{\approx}$ then $\sigma, i \models \gamma$.

- If $\gamma = \mathtt{x} \approx \mathtt{y}$ [resp. $\mathtt{x} \not\approx \mathtt{y}$, $\mathtt{x} \approx \mathsf{X}^\ell\mathtt{y}$, $\mathtt{x} \not\approx \mathsf{X}^\ell\mathtt{y}$] it is immediate since $\gamma^{\approx} = \gamma$.
- If $\gamma = \mathtt{x} \not\approx \langle\psi?\rangle\mathtt{y}$ and thus $\gamma^{\approx} = \mathtt{x} \not\approx \mathtt{v}_{\mathtt{x}\not\approx\langle\psi?\rangle\mathtt{y}} \wedge \mathtt{x}^{\not\approx}_{\mathtt{y},\psi} \approx \langle\psi?\rangle\mathtt{y}$. Then, $\sigma(i)(\mathtt{x}) \neq \sigma(i)(\mathtt{v}_{\mathtt{x}\not\approx\langle\psi?\rangle\mathtt{y}}) = \sigma(j)(\mathtt{y})$ for some $j > i$ where $\sigma, j \models \psi$. Hence, $\sigma, i \models \mathtt{x} \not\approx \langle\psi?\rangle\mathtt{y}$.
- If $\gamma = \neg(\mathtt{x} \not\approx \langle\psi?\rangle\mathtt{y})$, and thus $\gamma^{\approx} = \neg\mathsf{X}\mathsf{F}\psi \;\vee\; \mathtt{x} \approx \mathsf{X}\mathtt{v}^{\approx}_{\mathtt{y},\psi}$. This means that either
  - there is no $j > i$ so that $\sigma, j \models \psi$ and hence $\sigma, i \models \neg(\mathtt{x} \not\approx \langle\psi?\rangle\mathtt{y})$, or
  - $\sigma, j \models \psi$ for some $j > i$ and $\sigma(i)(\mathtt{x}) = \sigma(i+1)(\mathtt{v}^{\approx}_{\mathtt{y},\psi})$. By condition (d), we have that for all $i < j'$, so that $\sigma, j' \models \psi$, we have $\sigma(j')(\mathtt{v}^{\approx}_{\mathtt{y},\psi}) = \sigma(j')(\mathtt{y})$. Then, $\sigma, i \models \neg(\mathtt{x} \not\approx \langle\psi?\rangle\mathtt{y})$.
- If $\gamma = \mathsf{F}\psi$ and $\gamma^{\approx} = \mathsf{F}\psi^{\approx}$, there must be some position $i' \ge i$ so that $\sigma, i' \models \psi^{\approx}$. By inductive hypothesis, $\sigma, i' \models \psi$ and hence $\sigma, i \models \mathsf{F}\psi$. We proceed similarly for all temporal operators and their dual, as well as for the boolean operators $\wedge$, $\vee$. This is because $(\cdot)^{\approx}$ is homomorphic for all temporal and positive boolean operators.

■

We can easily extend this coding allowing for past obligations. We only need to use some extra variables $\mathtt{v}^{-1,\approx}_{\mathtt{x},\psi}$, $\mathtt{v}^{-1}_{\mathtt{x}\not\approx\langle\psi?\rangle^{-1}\mathtt{y}}$ that behave in the same way as the previously defined, but with past obligations. That

is, we also define a *val*-$\mathtt{v}^{-1,\approx}_{\mathtt{x},\psi}$ as *val*-$\mathtt{v}^{\approx}_{\mathtt{x},\psi}$, but making use of $\mathtt{v}^{-1,\approx}_{\mathtt{x},\psi}$, X^{-1} and F^{-1}. And finally, we have to further replace

(c) every appearance of $\neg(\mathtt{x} \not\approx \langle\psi?\rangle^{-1}\mathtt{y})$ by $\neg\mathsf{X}^{-1}\mathsf{F}^{-1}\psi \ \vee \ \mathtt{x} \approx \mathsf{X}\mathtt{v}^{-1,\approx}_{\mathtt{y},\psi}$, and

(d) every positive appearance of $\mathtt{x} \not\approx \langle\psi?\rangle^{-1}\mathtt{y}$ by $\mathtt{x} \not\approx \mathtt{v}^{-1}_{\mathtt{x}\not\approx\langle\psi?\rangle^{-1}\mathtt{y}} \wedge \mathtt{v}^{-1}_{\mathtt{x}\not\approx\langle\psi?\rangle^{-1}\mathtt{y}} \approx \langle\psi?\rangle^{-1}\mathtt{y}$

in φ to obtain $\varphi^{\approx}$. ■

Proof of Proposition 3: We show how to compute in polynomial time, for every $\varphi \in \mathrm{LRV}^{\approx}$, a formula $\varphi' \in \mathrm{LRV}^{\top}$ that preserves satisfiability. The formula φ' uses, besides all the variables from φ, a distinguished variable $\mathtt{k}$ that does not appear in φ, and a variable $\mathtt{v}_{\mathtt{x},\psi}$ for every subformula ψ of φ and every variable $\mathtt{x}$ appearing in φ.

The idea is that $\mathtt{k}$ acts as a constant, it preserves the same data value throughtout the model, and we enforce, for every subformula ψ and variable $\mathtt{x}$ of φ, that: ψ holds at a position if $\mathtt{v}_{\mathtt{x},\psi} \approx \mathtt{x}$, and ψ does not hold at a position if $\mathtt{v}_{\mathtt{x},\psi} \approx \mathtt{k}$. Now, instead of testing $\mathtt{x} \approx \langle\psi?\rangle\mathtt{y}$, one can simply test $\mathtt{x} \approx \langle\top?\rangle\mathtt{v}_{\mathtt{y},\psi}$.

Let $\varphi \in \mathrm{LRV}^{\approx}$. We define *vars*$(\varphi)$ as the set of all variables in φ, and *sub*$_{\langle\rangle}(\varphi)$ as the set of all subformulas ψ such that $\mathtt{x} \approx \langle\psi?\rangle\mathtt{y}$ appears in φ for some $\mathtt{x}, \mathtt{y}$. We show how to build a formula $\varphi' \in \mathrm{LRV}^{\top}$ which preserves satisfiability. For any model σ, let σ_φ be so that

(a) $|\sigma| = |\sigma_\varphi|$,

(b) for every $0 \leq i < |\sigma|$ and $\mathtt{x} \in$ *vars*(φ), $\sigma(i)(\mathtt{x}) = \sigma_\varphi(i)(\mathtt{x})$,

(c) there is some data value $d \notin \{\sigma_\varphi(i)(\mathtt{x}) \mid \mathtt{x} \in \mathit{vars}(\varphi), 0 \leq i < |\sigma|\}$ such that for every $0 \leq i < |\sigma|$, $\sigma_\varphi(i)(\mathtt{k}) = d$, and

(d) for every $0 \leq i < |\sigma|$, $\sigma_\varphi(i)(\mathtt{v}_{\mathtt{x},\psi}) = \sigma_\varphi(i)(\mathtt{x})$ if $\sigma, i \models \psi$, and $\sigma_\varphi(i)(\mathtt{v}_{\mathtt{x},\psi}) = \sigma_\varphi(i)(\mathtt{k})$ otherwise.

For every other unmentioned variable, σ and σ_φ coincide. It is evident that for every φ and σ, a model σ_φ with the aforementioned properties exists. Next, we define $\varphi' \in \mathrm{LRV}^{\top}$ so that

$$\sigma \models \varphi \ \text{ if, and only if, } \ \sigma_\varphi \models \varphi'. \tag{†}$$

We assume that for every $\psi \in$ *sub*$_{\langle\rangle}(\varphi)$, we have that *sub*$_{\langle\rangle}(\psi) = \emptyset$. This is without any loss of generality by Lemma 25.

- First, the variable $\mathtt{k}$ will act as a constant; it will always have the same data value at any position of the model, which must be different from those of all variables of φ.

$$\mathit{const} \ = \ \mathsf{G}\left((\mathtt{k} \approx \mathsf{X}\mathtt{k} \vee \neg\mathsf{X}\top) \ \wedge \bigwedge_{\mathtt{x}\in\mathit{vars}(\varphi)} (\mathtt{k} \not\approx \mathtt{x})\right)$$

- Second, any variable $\mathtt{v}_{\mathtt{x},\psi}$ has either the value of $\mathtt{k}$ or that of $\mathtt{x}$. Further, the latter holds if, and only if, ψ is true.

$$\mathit{val}\text{-}\mathtt{v}_{\mathtt{x},\psi} \ = \ \mathsf{G}(\mathtt{v}_{\mathtt{x},\psi} \approx \mathtt{k} \ \vee \ \mathtt{v}_{\mathtt{x},\psi} \approx \mathtt{x}) \ \wedge \ \mathsf{G}(\mathtt{v}_{\mathtt{x},\psi} \approx \mathtt{x} \ \Leftrightarrow \ \psi)$$

- Finally, let $\varphi^{\top}$ be the result of replacing every appearance of $\mathtt{x} \approx \langle\psi?\rangle\mathtt{y}$ by $\mathtt{x} \approx \langle\top?\rangle\mathtt{v}_{\mathtt{y},\psi}$ in φ.

We then define φ' as follows.

$$\varphi' \ = \ \varphi^{\top} \ \wedge \ \mathit{const} \ \wedge \bigwedge_{\psi\in\mathit{sub}_{\langle\rangle}(\varphi)} \mathit{val}\text{-}\mathtt{v}_{\mathtt{x},\psi}$$

Notice that φ' can be computed from φ in polynomial time.

Claim 27. φ is satisfiable if, and only if, φ' is satisfiable.

Proof:

[⇒] Note that one direction would follow from (†): if φ is satisfiable by some σ, then φ' is satisfiable in σ_φ. In order to establish (†), we show that

$$\text{for every subformula } \gamma \text{ of } \varphi \text{ and } 0 \leq i < |\sigma_\varphi|, \text{ we have } \sigma, i \models \gamma \text{ iff } \sigma_\varphi, i \models \gamma^{\approx}. \tag{‡}$$

Since $\sigma_\varphi \models \mathit{const}$ by condition (c) and $\sigma_\varphi \models \mathit{val}\text{-}\mathtt{v}_{\mathtt{x},\psi}$ for every ψ due to (d), this is sufficient to conclude then $\sigma \models \varphi$ iff $\sigma_\varphi \models \varphi'$. We show it by structural induction. If $\sigma, i \models \mathtt{x} \approx \langle\psi?\rangle\mathtt{y}$ then there is some $j > i$ where $\sigma(j)(\mathtt{y}) = \sigma(i)(\mathtt{x})$ and $\sigma, j \models \psi$. Then, by condition (d), $\sigma_\varphi(j)(\mathtt{v}_{\mathtt{y},\psi}) = \sigma_\varphi(j)(\mathtt{y})$ and thus $\sigma_\varphi, i \models \mathtt{x} \approx \langle\top?\rangle\mathtt{v}_{\mathtt{y},\psi}$. Conversely, if $\sigma_\varphi, i \models \mathtt{x} \approx \langle\top?\rangle\mathtt{v}_{\mathtt{y},\psi}$ this means that there is some $j > i$ such that $\sigma_\varphi(i)(\mathtt{x}) = \sigma_\varphi(j)(\mathtt{v}_{\mathtt{y},\psi})$. By (c), we have that $\sigma_\varphi(j)(\mathtt{v}_{\mathtt{y},\psi}) \neq \sigma_\varphi(j)(\mathtt{k})$, and by (d) this means that $\sigma_\varphi(j)(\mathtt{v}_{\mathtt{y},\psi}) = \sigma_\varphi(j)(\mathtt{y})$ and $\sigma_\varphi, j \models \psi$. Therefore, $\sigma, i \models \mathtt{x} \approx \langle\psi?\rangle\mathtt{y}$. The proof for the base case of

the form $\psi = \mathtt{x} \approx \mathsf{X}^i \mathtt{y}$ and for all the cases of the induction step are by an easy verification since $(\cdot)^\top$ is homomorphic. Hence, (‡) holds.

[⇐] Suppose that $\sigma \models \varphi'$. Note that due to *const* condition (c) holds in σ, and due to *val*-$\mathtt{v}_{\mathtt{x},\psi}$, condition (d) holds. We show that for every position i and subformula γ of φ, if $\sigma, i \models \gamma^\top$, then $\sigma, i \models \gamma$.

The only interesting case is when $\gamma = \mathtt{x} \approx \langle \psi ? \rangle \mathtt{y}$, and $\gamma^\top = \mathtt{x} \approx \langle \top ? \rangle \mathtt{v}_{\mathtt{y},\psi}$, since for all boolean and temporal operators $(\cdot)^\top$ is homomorphic. If $\sigma, i \models \mathtt{x} \approx \langle \top ? \rangle \mathtt{v}_{\mathtt{y},\psi}$ there must be some $j > i$ so that $\sigma(j)(\mathtt{v}_{\mathtt{y},\psi}) = \sigma(i)(\mathtt{x})$. By condition (c), $\sigma(j)(\mathtt{k}) \neq \sigma(j)(\mathtt{v}_{\mathtt{y},\psi}) = \sigma(i)(\mathtt{x})$, and by condition (d), this means that $\sigma, j \models \psi$ and $\sigma(j)(\mathtt{v}_{\mathtt{y},\psi}) = \sigma(j)(\mathtt{y})$. Hence, $\sigma, i \models \mathtt{x} \approx \langle \psi ? \rangle \mathtt{y}$. The remaining cases are straightforward since $(\cdot)^\top$ is homomorphic on temporal and boolean operators. ∎

Finally, note that we can extend this coding to treat past obligations in the obvious way. ∎

C. Proofs of Lemmas 7, 8 and Theorem 9

We define the VASS $\mathcal{A}_{\text{inc}}$ as $\langle Q, C, \delta^{min} \rangle$ and $Q_0, Q_f \subseteq Q$, where Q, Q_0, Q_f and C are same as those of $\mathcal{A}_\phi$ and δ^{min} is defined as follows: $(q, minup_{q,q'}, q') \in \delta^{min}$ iff $\delta \cap (\{q\} \times [-k, k]^C \times \{q'\})$ is not empty and

$$minup_{q,q'}(X) = \min_{\mathbf{u}:(q,\mathbf{u},q') \in \delta} \{\mathbf{u}(X)\} \text{ for all } X \in C.$$

We also use the "maximal" transition in some lemmas:

$$maxup_{q,q'}(X) = \max_{\mathbf{u}:(q,\mathbf{u},q') \in \delta} \{\mathbf{u}(X)\}.$$

Proof of Lemma 7: Since $\langle q, \mathbf{v} \rangle \to \langle q', \mathbf{v}' \rangle$, there is a transition $(q, \mathbf{u}, q') \in \delta$ such that $\mathbf{u} + \mathbf{v} = \mathbf{v}'$. By definition of $\mathcal{A}_{\text{inc}}$, there is a transition $(q, minup_{q,q'}, q') \in \delta^{min}$ where $\mathbf{u} - minup_{q,q'} \succeq \mathbf{0}$. Let $incer = \mathbf{u} - minup_{q,q'}$ (this will be the incremental error used in $\mathcal{A}_{\text{inc}}$). Now we have $\langle q, \mathbf{v} + incer \rangle \xrightarrow{minup_{q,q'}} \langle q', \mathbf{v} + \mathbf{u} \rangle = \langle q', \mathbf{v}' \rangle$ in $\mathcal{A}_{\text{inc}}$. Hence, $\langle q, \mathbf{v} \rangle \to_{\text{gainy}} \langle q', \mathbf{v}' \rangle$ in $\mathcal{A}_{\text{inc}}$. ∎

Proof of Lemma 8: By induction on the length n of the run $\langle q_1, \mathbf{v}_1 \rangle \xrightarrow{*}_{\text{gainy}} \langle q_2, \mathbf{0} \rangle$. The base case $n = 0$ is trivial since there is no change in the configuration.

Induction step: the idea is to simulate the first gainy transition by a normal transition that decreases each counter as much as possible while ensuring that (1) the resulting value is non-negative and (2) we can apply the induction hypothesis to the resulting valuation. We calculate the update required for each counter individually and by closure under component-wise interpolation, there will always be a transition with the required update function. Let $\langle q_1, \mathbf{v}_1 \rangle \to_{\text{gainy}} \langle q_3, \mathbf{v}_3 \rangle \xrightarrow{*}_{\text{gainy}} \langle q_2, \mathbf{0} \rangle$ and $\mathbf{v}'_1 \preceq \mathbf{v}_1$. We will define an update function $\mathbf{u}'$ such that $\langle q_1, \mathbf{v}'_1 \rangle \to \langle q_3, \mathbf{v}'_3 \rangle$, $\mathbf{v}'_3(X) = \mathbf{v}'_1(X) + \mathbf{u}'(X)$ for each $X \in C$ and $\mathbf{v}'_3 \preceq \mathbf{v}_3$. For each counter $X \in C$, $\mathbf{u}'(X)$ is defined as follows:

Case 1: $\mathbf{v}'_1(X) + minup_{q_1,q_3}(X) \geq 0$. Let $\mathbf{u}'(X) = minup_{q_1,q_3}(X)$.

$$\begin{aligned}
\mathbf{v}_3(X) &\geq \mathbf{v}_1(X) + minup_{q_1,q_3}(X) && \text{[From the semantics of } \mathcal{A}_{\text{inc}}\text{]} \\
&\geq \mathbf{v}'_1(X) + minup_{q_1,q_3}(X) && \text{[Since } \mathbf{v}'_1 \preceq \mathbf{v}_1\text{]} \\
&= \mathbf{v}'_1(X) + \mathbf{u}'(X) && \text{[By definition of } \mathbf{u}'(X)\text{]} \\
&= \mathbf{v}'_3(X)
\end{aligned}$$

Case 2: $\mathbf{v}'_1(X) + minup_{q_1,q_3}(X) < 0$. Therefore, $minup_{q_1,q_3}(X) < -\mathbf{v}'_1(X)$. Moreover, since $maxup_{q_1,q_3}(X) \geq 0$ (due to the optional decrement property) and $\mathbf{v}'_1(X) \geq 0$, $-\mathbf{v}'_1(X) \leq maxup_{q_1,q_3}(X)$. Let $\mathbf{u}'(X) = -\mathbf{v}'_1(X)$. Now, $\mathbf{v}'_3(X) = \mathbf{v}'_1(X) + \mathbf{u}'(X) = 0 \preceq \mathbf{v}_3(X)$.

By definition of $minup_{q_1,q_3}$ and the closure of the set of transitions δ of $\mathcal{A}_\phi$ under component-wise interpolation, we have $(q_1, \mathbf{u}', q_3) \in \delta$ and hence $\langle q_1, \mathbf{v}'_1 \rangle \to \langle q_3, \mathbf{v}'_3 \rangle$. Since $\mathbf{v}'_3 \preceq \mathbf{v}_3$ and $\langle q_3, \mathbf{v}_3 \rangle \xrightarrow{*}_{\text{gainy}} \langle q_2, \mathbf{0} \rangle$, we can use the induction hypothesis to conclude that $\langle q_3, \mathbf{v}'_3 \rangle \xrightarrow{*} \langle q_2, \mathbf{0} \rangle$. So we conclude that $\langle q_1, \mathbf{v}'_1 \rangle \to \langle q_3, \mathbf{v}'_3 \rangle \xrightarrow{*} \langle q_2, \mathbf{0} \rangle$. ∎

Proof of Theorem 9: The proof is in four steps.

Step 1: From [6], a LRV$^\top$ formula ϕ is satisfiable iff $\langle q_0, \mathbf{0}\rangle \xrightarrow{*} \langle q_f, \mathbf{0}\rangle$ in $\mathcal{A}_\phi$ for some $q_0 \in Q_0$ and $q_f \in Q_f$.

Step 2: This is the step that requires new insight. From Lemmas 7 and 8, $\langle q_0, \mathbf{0}\rangle \xrightarrow{*} \langle q_f, \mathbf{0}\rangle$ in $\mathcal{A}_\phi$ iff $\langle q_0, \mathbf{0}\rangle \xrightarrow{*}_{\text{gainy}} \langle q_f, \mathbf{0}\rangle$ in $\mathcal{A}_{\text{inc}}$.

Step 3: This is a standard trick. Let $\mathcal{A}_{\text{dec}} = \langle Q, C, \delta^{rev}\rangle$ be a VASS such that for every transition $(q, \mathbf{u}, q') \in \delta^{min}$ of $\mathcal{A}_{\text{inc}}$, $\mathcal{A}_{\text{dec}}$ has a transition $(q', -\mathbf{u}, q) \in \delta^{rev}$, where $-\mathbf{u} : C \to \mathbb{Z}$ is the function such that $-\mathbf{u}(X) = -1 \times \mathbf{u}(X)$ for all $X \in C$. We infer that $\langle q_0, \mathbf{0}\rangle \xrightarrow{*}_{\text{gainy}} \langle q_f, \mathbf{0}\rangle$ in $\mathcal{A}_{\text{inc}}$ iff $\langle q_f, \mathbf{0}\rangle \xrightarrow{*}_{\text{lossy}} \langle q_0, \mathbf{0}\rangle$ in $\mathcal{A}_{\text{dec}}$ (we can simply reverse every transition in the run of $\mathcal{A}_{\text{inc}}$ to get a run of $\mathcal{A}_{\text{dec}}$ and vice-versa).

Step 4: This is another standard trick. Since $\mathcal{A}_{\text{dec}}$ does not have zero-tests, we can remove all decrementing errors from a run of $\mathcal{A}_{\text{dec}}$ from $\langle q_f, \mathbf{0}\rangle$ to $\langle q_0, \mathbf{0}\rangle$, to get another run from $\langle q_f, \mathbf{0}\rangle$ to $\langle q_0, \mathbf{v}\rangle$, where $\mathbf{v}$ is some counter valuation (possibly different from $\mathbf{0}$). Using decremental errors at the last configuration, $\mathcal{A}_{\text{dec}}$ can then reach the configuration $\langle q_0, \mathbf{0}\rangle$. In other words, $\langle q_f, \mathbf{0}\rangle \xrightarrow{*}_{\text{lossy}} \langle q_0, \mathbf{0}\rangle$ iff $\langle q_f, \mathbf{0}\rangle \xrightarrow{*} \langle q_0, \mathbf{v}\rangle$ for some counter valuation $\mathbf{v}$. Checking the latter condition is precisely the control state reachability problem for VASS.

If the control state in the above instance is reachable, then Rackoff gives a bound on the length of a shortest run reaching it [28]. The bound is doubly exponential in the size of the VASS. Since in our case, the size of the VASS is exponential in the size of the LRV$^\top$ formula ϕ, the bound is triply exponential. A non-deterministic Turing machine can maintain a binary counter to count up to this bound, using doubly exponential space. The machine can start by guessing some initial state q_0 and a counter valuation set to $\mathbf{0}$. (this can be done in polynomial space). In one step, the machine guesses a transition to be applied next and updates the current configuration accordingly, while incrementing the binary counter. At any step, the space required to store the current configuration is at most doubly exponential. By the time the binary counter reaches its triply exponential bound, if a final control state is not reached, the machine rejects its input. Otherwise, it accepts. Since this non-deterministic machine operates in doubly exponential space, an application of Savitch's Theorem gives us the required 2EXPSPACE upper bound for the satisfiability problem of LRV$^\top$. ∎

D. Proofs of Lemmas 12 and 13

Let $\mathcal{A} = \langle Q, f, k, Q_0, Q_F, \delta\rangle$ be a chain automaton of level k. A run ρ *ends in zero* whenever for every chain $\alpha \in [1, n]$, $c_L^\alpha = 0$ with $L = |\rho|$. We write Per$^{\text{zero}}(k)$ and Gainy$^{\text{zero}}(k)$ to denote the variant problems of Per(k) and Gainy(k), respectively, in which runs that end in zero are considered. First, note that Per$^{\text{zero}}(k)$ and Per(k) are interreducible in logarithmic space since it is always possible to add adequately self-loops when a final state is reached in order to guarantee that $c_L^\alpha = 0$ for every chain $\alpha \in [1, n]$. Similarly, Gainy$^{\text{zero}}(k)$ and Gainy(k) are interreducible in logarithmic space.

Proof of Lemma 12: Below, we show that Per$^{\text{zero}}(k)$ and Gainy$^{\text{zero}}(k)$ are interreducible in logarithmic space, which allows us to get the proof of the lemma since logarithmic-space reductions are closed under composition. From $\mathcal{A}$, let us define a reverse counter automaton $\widetilde{\mathcal{A}}$ of level k, where the reverse operation $\widetilde{\cdot}$ is defined on instructions, transitions, sets of transitions and automata as follows:

- $\widetilde{\text{inc}(\alpha)} \stackrel{\text{def}}{=} \text{dec}(\alpha)$; $\widetilde{\text{dec}(\alpha)} \stackrel{\text{def}}{=} \text{inc}(\alpha)$; $\widetilde{\text{next}(\alpha)} \stackrel{\text{def}}{=} \text{prev}(\alpha)$; $\widetilde{\text{prev}(\alpha)} \stackrel{\text{def}}{=} \text{next}(\alpha)$,
- $\widetilde{\text{first}(\alpha)?} \stackrel{\text{def}}{=} \text{last}(\alpha)?$; $\widetilde{\text{last}(\alpha)?} \stackrel{\text{def}}{=} \text{first}(\alpha)?$; $\widetilde{\overline{\text{first}(\alpha)}?} \stackrel{\text{def}}{=} \overline{\text{last}(\alpha)}?$; $\widetilde{\overline{\text{last}(\alpha)}?} \stackrel{\text{def}}{=} \overline{\text{first}(\alpha)}?$,
- $q \xrightarrow{\text{instr}} q' \stackrel{\text{def}}{=} q' \xrightarrow{\widetilde{\text{instr}}} q$,
- $\widetilde{\mathcal{A}} \stackrel{\text{def}}{=} \langle Q, f, k, Q_F, Q_0, \widetilde{\delta}\rangle$ with $\widetilde{\delta} \stackrel{\text{def}}{=} \{q \xrightarrow{\widetilde{\text{instr}}} q' : q \xrightarrow{\text{instr}} q' \in \delta\}$. Note that Q_0 and Q_F have been swapped.

The reverse operation can be extended to sequences of transitions as follows: $\widetilde{\varepsilon} \stackrel{\text{def}}{=} \varepsilon$ and $\widetilde{t \cdot u} \stackrel{\text{def}}{=} \widetilde{u} \cdot \widetilde{t}$. Note that $\widetilde{\cdot}$ extends the reverse operation defined in the proof of Theorem 9 (step 3).

One can show the following implications, for any run ρ fo $\mathcal{A}$ that ends in zero:

1) ρ is perfect and accepting for $\mathcal{A}$ implies $\widetilde{\rho}$ is gainy and accepting for $\widetilde{\mathcal{A}}$.
2) ρ is gainy and accepting for $\widetilde{\mathcal{A}}$ implies $\widetilde{\rho}$ is perfect and accepting for $\mathcal{A}$.
3) ρ is gainy and accepting for $\mathcal{A}$ implies $\widetilde{\rho}$ is perfect and accepting for $\widetilde{\mathcal{A}}$.
4) ρ is perfect and accepting for $\widetilde{\mathcal{A}}$ implies $\widetilde{\rho}$ is gainy and accepting for $\mathcal{A}$.

(1) and (4) [resp. (2) and (3)] have very similar proofs because $\widetilde{\cdot}^{-1}$ is actually equal to $\widetilde{\cdot}$. (1)–(4) are sufficient to establish that $\mathrm{Per}^{\mathrm{zero}}(k)$ and $\mathrm{Gainy}^{\mathrm{zero}}(k)$ are interreducible in logarithmic space.

■

Proof of Lemma 13: It is sufficient to show that $\mathrm{Per}^{\mathrm{zero}}(k)$ is in $(k+1)$EXPSPACE. Let $\mathcal{A} = \langle Q, f, k, Q_0, Q_F, \delta\rangle$ be a chain automaton with $f : [1,n] \to \mathbb{N}$. We reduce this instance of $\mathrm{Per}^{\mathrm{zero}}(k)$ into several instances of the control state reachability problem for VASS such that the number of instances is bounded by $\mathcal{O}(|\mathcal{A}|^2)$ and the size of each instance is in $\mathcal{O}(\mathtt{exp}(k,|\mathcal{A}|)^{|\mathcal{A}|})$, which provides the $(k+1)$EXPSPACE upper bound by [28]. The only instructions in the VASS $\mathcal{A}'$ defined below are: increment a counter, decrement a counter or the `skip` action, which just changes the control state without modifying the counter values (which can be obviously simulated by an increment followed by a decrement).

Let us define a VASS $\mathcal{A}' = \langle Q', C', \delta'\rangle$ with

$$C' = \{\mathtt{c}^1_0, \ldots, \mathtt{c}^1_{\mathtt{exp}(k,f(1))-1}, \ldots, \mathtt{c}^n_0, \ldots, \mathtt{c}^n_{\mathtt{exp}(k,f(n))-1}\}$$

In $\mathcal{A}'$ it will be possible to access the counters directly by encoding the positions of the pointers in the states. Let $Q' = Q \times [0, \mathtt{exp}(k,f(1))-1] \times \cdots \times [0, \mathtt{exp}(k,f(n))-1]$. It remains to define the transition relation δ':

- Whenever $q \xrightarrow{\mathrm{inc}(\alpha)} q' \in \delta$, we have $\langle q, \beta_1, \ldots, \beta_n\rangle \xrightarrow{\mathrm{inc}(\mathtt{c}^\alpha_{\beta_\alpha})} \langle q', \beta_1, \ldots, \beta_n\rangle \in \delta'$ (and similarly with decrements),
- Whenever $q \xrightarrow{\mathrm{next}(\alpha)} q' \in \delta$, we have

  $$\langle q, \beta_1, \ldots, \beta_\alpha, \ldots, \beta_n\rangle \xrightarrow{\mathtt{skip}} \langle q', \beta_1, \ldots, \beta_\alpha + 1, \ldots, \beta_n\rangle \in \delta'$$

  if $\beta_\alpha + 1 < \mathtt{exp}(k, f(\alpha))$ (and similarly with $\mathrm{prev}(\alpha)$),
- When $q \xrightarrow{\mathrm{first}(\alpha)?} q' \in \delta$, $\langle q, \beta_1, \ldots, \beta_n\rangle \xrightarrow{\mathtt{skip}} \langle q', \beta_1, \ldots, \beta_n\rangle \in \delta'$ if $\beta_\alpha = 0$ (and similarly with $\overline{\mathrm{first}(\alpha)}?$, $\mathrm{last}(\alpha)?$ and $\overline{\mathrm{last}(\alpha)}?$).

It is easy to show that there is a perfect accepting run that ends in zero iff there are $\mathbf{q}_0 \in Q_0 \times \{0\}^n$ and $\mathbf{q}_F \in Q_F \times \{0\}^n$ such that there is a run from the configuration $\langle \mathbf{q}_0, \mathbf{0}\rangle$ to the configuration $\langle \mathbf{q}_F, \mathbf{x}\rangle$ for some $\mathbf{x}$.

In order to prove the above equivalence, we can use the transformations (I) and (II) stated below.

(I) Let $\rho = t_1 \cdots t_L$ be a run of $\mathcal{A}$ such that for every $i \in [1, L-1]$, $t_i = q_{i-1} \xrightarrow{\mathrm{instr}_i} q_i$ and the values c^α_i are defined as before (1). One can show that $\rho' = t'_1 \cdots t'_n$ with $t'_i = \langle q_{i-1}, \mathbf{x}_{i-1}\rangle \xrightarrow{\mathrm{instr}'_i} \langle q_i, \mathbf{x}_i\rangle$ for every i, is a run of $\mathcal{A}'$ where

- $\mathbf{x}_0 = \mathbf{0}$ and for every $i \in [1, L]$, $\mathbf{x}_i = \langle c^1_i, \ldots, c^n_i\rangle$,
- for every $i \in [1, L]$,
  - if $\mathrm{instr}_i = \mathrm{inc}(\alpha)$ then $\mathrm{instr}'_i = \mathrm{inc}(\mathtt{c}^\alpha_{c^\alpha_i})$ (a similar clause holds for decrements),
  - otherwise (i.e., if instr_i is not $\mathrm{inc}(\alpha)$ nor $\mathrm{dec}(\alpha)$ for any α), $\mathrm{instr}'_i = \mathtt{skip}$.

(II) Similarly, let $\rho' = t'_1 \cdots t'_L$ be a run of $\mathcal{A}'$ where $t'_i = (q_{i-1}, \mathbf{x}_{i-1}) \xrightarrow{\mathrm{instr}'_i} (q_i, \mathbf{x}_i)$ for every i, with $\mathbf{x}_0 = \mathbf{0}$. One can show that $\rho = t_1 \cdots t_L$ with $t_i = q_{i-1} \xrightarrow{\mathrm{instr}_i} q_i$ for every i, is a perfect accepting run of $\mathcal{A}$ such that for every $i \in [1, L]$,

- if $\mathrm{instr}'_i = \mathrm{inc}(\mathtt{c}^\alpha_j)$ then $\mathrm{instr}_i = \mathrm{inc}(\alpha)$ (a similar clause holds for decrements),
- otherwise, if $\mathbf{x}_{i+1}(\alpha) = \mathbf{x}_i(\alpha) + 1$ for some α, then $\mathrm{instr}_i = \mathrm{next}(\alpha)$ (a similar clause holds for previous),
- otherwise, if $\mathbf{x}_i(\alpha) = 0$ and $q_i \xrightarrow{\mathrm{first}(\alpha)?} q_{i+1} \in \delta$ for some α, then $\mathrm{instr}_i = \mathrm{first}(\alpha)?$ (a similar clause holds for $\overline{\mathrm{first}(\alpha)}?$),
- otherwise, if $\mathbf{x}_i(\alpha) = \mathtt{exp}(k, f(\alpha)) - 1$ and $q_i \xrightarrow{\mathrm{last}(\alpha)?} q_{i+1} \in \delta$ for some α, then $\mathrm{instr}_i = \mathrm{last}(\alpha)?$ (a similar clause holds for $\overline{\mathrm{last}(\alpha)}?$).

■

E. Details of the Hardness Result for Chain Automata

This section is dedicated to prove Theorem 14 by extending Lipton's proof [22] (see also its presentation in [8]) that shows that control state reachability problem for VASS is EXPSPACE-hard. In Sub-section E1, we recall principles of Lipton's proof without entering into the implementation details; indeed when dealing with chain automata the coding has to be refined. In Sub-section E2, we explain

how large decrements can be encoded with chain automata (a key point in the proof). Sub-section E3 is dedicated to the initialization of the counter values (another key point in the proof). In Lipton's proof, initialization amounts to performing a series of large increments using principles analogous to those for encoding decrements. When chain automata are used as a low level programming language, we need to maintain additional data structures (in chain automata, this means chains of counters). Unlike [22], [8], we do not use a return mechanism and we simulate directly the call/return stack. Hence, our construction is more general but at the cost of a more careful analysis. Sub-section E4 contains the statements and proofs of the main lemmas to conclude the correctness of our simulation. Finally, Sub-section E5 summarizes the different steps of the proof and concludes this section.

The proof below is complicated and sometimes we may repeat arguments for the sake of clarity and to present arguments from different viewpoints. We believe this might help the reader to grasp the present simulation.

1) Principles of Lipton's Proof: In this section, we present the principles behind the proof showing that the control state reachability problem for VASS is EXPSPACE-hard [22], [8].

Lipton's proof starts from the standard result in computability theory that a Turing Machine using space 2^{n^γ} can be simulated by a counter automaton equipped with 4 counters whose values are bounded by $2^{2^{n^\gamma}}$. Lipton's proof shows that such a counter automaton can be simulated with a VASS. Unlike counter automata, VASS does not have zero-test transitions. Each counter $\mathtt{c}$ of the automaton is complemented with an extra counter $\overline{\mathtt{c}}$ in the VASS. The VASS is designed such that the sum of the values in $\mathtt{c}$ and $\overline{\mathtt{c}}$ is $2^{2^{n^\gamma}}$ in any reachable configuration. Now testing $\mathtt{c}$ for zero is equivalent to testing that $\overline{\mathtt{c}}$ is $2^{2^{n^\gamma}}$. This later test is performed as explained next.

Testing that $\overline{\mathtt{c}}$ is $2^{2^{n^\gamma}}$ is done by decrementing $\overline{\mathtt{c}}$ by $2^{2^{n^\gamma}}$. To do so, the value of $\overline{\mathtt{c}}$ is transferred to another counter $\mathtt{s}$ and then $\mathtt{s}$ is decremented by $2^{2^{n^\gamma}}$. This whole operation results in the values of $\mathtt{c}$ and $\overline{\mathtt{c}}$ getting swapped, while all other counters retain their original values. To rectify the swapping between $\mathtt{c}$ and $\overline{\mathtt{c}}$, we repeat the operation except that the value of $\mathtt{c}$ is transferred to $\mathtt{s}$ and then $\mathtt{s}$ is decremented by $2^{2^{n^\gamma}}$.

Decrementing $\mathtt{s}$ by $2^{2^{n^\gamma}}$ is done by having a series of counters $\mathtt{s}_i, \overline{\mathtt{s}}_i$ for each i between 1 and n^γ. For each i, the sum of values in $\mathtt{s}_i$ and $\overline{\mathtt{s}}_i$ is maintained at 2^{2^i}. Below, we illustrate how a counter $\mathtt{s}_i$ is decremented 2^{2^i} times.

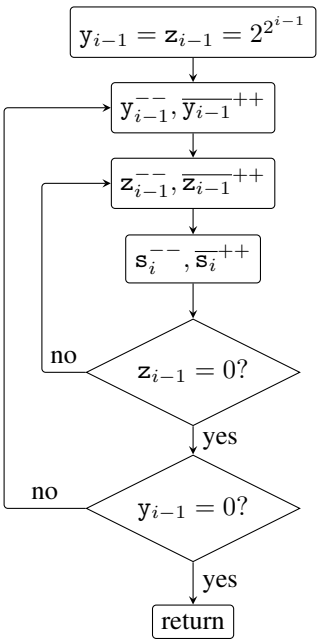


Fig. 4. Algorithm for decrementing $\mathtt{s}_i$ by 2^{2^i}

There are two nested loops indexed by $\mathtt{y}_{i-1}$ and $\mathtt{z}_{i-1}$, initialised to $2^{2^{i-1}}$. When all the iterations are finished, we would have decremented $\mathtt{s}_i$ exactly $2^{2^{i-1}} \times 2^{2^{i-1}} = 2^{2^i}$ times. Testing $\mathtt{y}_{i-1}$ and $\mathtt{z}_{i-1}$ for zero is done by a similar gadget, where $i, i-1$ are replaced by $i-1, i-2$. The resulting VASS is

composed of n^γ constant sized VASS, each one implementing the nested loops for an i between 1 and n^γ.

In the above explanation of the decrementing algorithm, we have assumed that the counters $\mathtt{y}_{i-1}$ and $\mathtt{z}_{i-1}$ have the exact value $2^{2^{i-1}}$ in the beginning. This can be achieved if we first inductively initialize all counters $\mathtt{y}_j$ and $\mathtt{z}_j$ to have the value 2^{2^j} for each j between 1 and $i-2$. Assuming this initialization has been done, the algorithm shown in Fig. 5 initializes $\mathtt{y}_{i-1}$ and $\mathtt{z}_{i-1}$ to $2^{2^{i-1}}$. The algorithm shown in

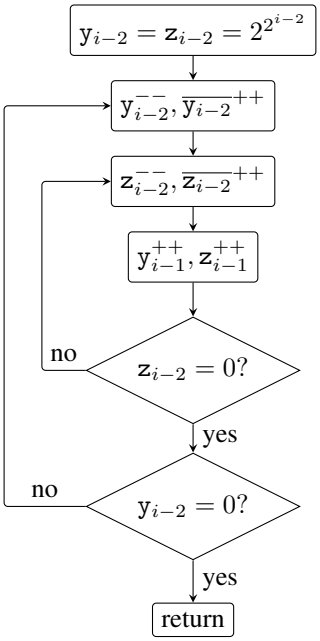


Fig. 5. Algorithm for initializing $\mathtt{y}_{i-1}, \mathtt{z}_{i-1}$

Fig. 5 is based on the same principle as the one in Fig. 4, except that $i, i-1$ are replaced by $i-1, i-2$ and that instead of decrementing $\mathtt{s}_i$ inside the nested loops, $\mathtt{y}_{i-1}, \mathtt{z}_{i-1}$ are incremented.

Now that we know how zero-tests are implemented in a VASS, let us see how a VASS simulates counter automata with bounded counter values. For every transition $\langle q, \mathtt{c} \leftarrow \mathtt{c}+1, q' \rangle$ of the counter automaton, the VASS will have the transition $\langle q, (\mathrm{inc}(\mathtt{c}); \mathrm{dec}(\overline{\mathtt{c}})), q' \rangle$. Thus, for every incrementing transition of the counter automaton, the VASS can execute a corresponding incrementing transition. Conversely, for every incrementing transition executed by the VASS, the counter automaton can execute the corresponding incrementing transition.

For every transition $\langle q :$ if $\mathtt{c} = 0$ goto q_1 else $\mathtt{c} \leftarrow \mathtt{c}-1$ goto $q_2 \rangle$ of the counter automaton, the VASS will have transitions corresponding to the following program.

q: goto nonzero or zero
nonzero: $\mathrm{dec}(\mathtt{c}); \mathrm{inc}(\mathtt{c})$; goto q_2
zero: transfer($\overline{\mathtt{c}}$, $\mathtt{s}$); goto $\mathrm{Dec}_{\textbf{zerorep}}$
zerorep: transfer($\mathtt{c}$, $\mathtt{s}$); goto Dec_{q_1}

The macro transfer($\overline{\mathtt{c}}$, $\mathtt{s}$) is intended to be a shortcut for "$\mathtt{s} := \mathtt{s} + \overline{\mathtt{c}}$; $\overline{\mathtt{c}} := 0$" but its actual implementation is described in the next section. Its code guarantees that at least one of its executions performs the sequence "$\mathtt{s} := \mathtt{s} + \overline{\mathtt{c}}$; $\overline{\mathtt{c}} := 0$". The macro transfer($\mathtt{c}$, $\mathtt{s}$) is defined analogously.

In the above, the control state $\mathrm{Dec}_{\textbf{zerorep}}$ launches a program that decrements $\mathtt{s}$ by $2^{2^{n^\gamma}}$ and then transfers the control to **zerorep**. Similarly, Dec_{q_1} launches a program that decrements $\mathtt{s}$ by $2^{2^{n^\gamma}}$ and then transfers the control to q_1.

The idea behind our $(k+1)$EXPSPACE lower bound is that a Turing Machine using space $\mathtt{exp}(k+1, n^\gamma)$ can be simulated by a counter automaton equipped with 4 counters whose values are bounded by $\mathtt{exp}(k+2, n^\gamma)$. We shall show that such a counter automaton can be simulated with a chain automaton of level k with a map f such that for each chain α, $f(\alpha) = n^\gamma$. Each chain α of the chain automaton will be complemented with another chain $\overline{\alpha}$. We will denote the counters in these chains by $\alpha_1, \overline{\alpha}_1, \ldots, \alpha_{\mathtt{exp}(k,n^\gamma)}, \overline{\alpha}_{\mathtt{exp}(k,n^\gamma)}$. The chain automaton will be designed such that the sum of the

values in α_i and $\overline{\alpha}_i$ is 2^{2^i} in any reachable configuration and for any i between 1 and $\mathtt{exp}(k, n^\gamma)$. Now testing α_i for zero is equivalent to testing that $\overline{\alpha}_i$ is 2^{2^i}. This later test will not work directly as explained above, since a chain automaton composed of $\exp(k, n^\gamma)$ constant sized portions will not give a polynomial-time reduction. We reduce the size of the chain automaton by observing that the decrementing algorithm for $i-1$ is the same as the one for i, except that $i, i-1$ are replaced by $i-1, i-2$ respectively. We can write a single sequence of instructions for the decrementing algorithm and invoke it for any i by placing the pointers at the appropriate counters. The main difficulty in the implementation is to ensure that the multiple invocations of the algorithm return to the correct point, which is handled in the next section.

2) Encoding Large Decrements with Chain Automata: In the ideas explained in the previous section, we assumed that a call to a procedure will return to the correct program point. In Lipton's proof [22] and its exposition contained in [8], the decrementing algorithms are described as subroutines, and a return mechanism (which involves remembering return addresses) is implemented in a VASS. Since we do not have direct access to counters in chain automata, it is cumbersome to implement a general purpose return mechanism. In this section, we explain how decrements can be implemented with a chain automaton without using a return mechanism.

Recall the following high-level algorithm used in Lipton's proof.

outerloop: Decrement y_{i-1}.
innerloop: Decrement z_{i-1}.
decrement: Decrement s_i.
innerexit: If $z_{i-1} \neq 0$, go to innerloop. Otherwise, set z_{i-1} back to $2^{2^{i-1}}$ and go to outerexit.
outerexit: If $y_{i-1} \neq 0$, go to outerloop. Otherwise, set y_{i-1} back to $2^{2^{i-1}}$ and return.

Whenever we have to return from the decrementing algorithm for stage $i-1$, we have to return to the step labeled either innerexit or outerexit of stage i, in the high-level algorithm described above. To remember which one, we maintain a pair of chains $\mathtt{stack}$ and $\overline{\mathtt{stack}}$, each having $\mathtt{exp}(k, n^\gamma)$ counters, so that the sum of the values of $\mathtt{stack}_i$ and $\overline{\mathtt{stack}}_i$ is 1. For convenience, we will use $N = \mathtt{exp}(k, n^\gamma)$ from here onwards. We follow the convention that if at stage i, the counter in the chain $\mathtt{stack}$ has value 1, then it is the step labeled innerexit that invoked the decrementing algorithm for stage $i-1$. A $\mathtt{stack}$ value of 0 at stage i implies that the step labeled outerexit invoked the decrementing algorithm for stage $i-1$. In addition, we have six chains $\mathtt{y}, \overline{\mathtt{y}}, \mathtt{z}, \overline{\mathtt{z}}, \mathtt{s}$ and $\overline{\mathtt{s}}$, each with N counters. Recall that for the decrementing algorithm to work properly at stage i, $\mathtt{y}_{i-1}$ and $\mathtt{z}_{i-1}$ both should have the value $2^{2^{i-1}}$ and $\mathtt{s}_{i-1}$ should have the value 0 (so $\overline{\mathtt{s}_{i-1}}$ should have the value $2^{2^{i-1}}$). We also need to have $\mathtt{stack}_{i-1}$ to have the value 0 (so $\overline{\mathtt{stack}_{i-1}}$ should have the value 1).

To understand our implementation of the decrementing algorithm, it will be helpful to think of the counters in the chains arranged as follows.

Chain\Stage	N	$N-1$	$N-2$	$\ldots$	i	$i-1$	$\cdots$	1
y						$2^{2^{i-1}}$		
z						$2^{2^{i-1}}$		
s					2^{2^i}	0		
stack	0	0	1	$\cdots$	1		$\cdots$	0
Pointer				Nextstage $\leftarrow$	$\uparrow$	$\rightarrow$ Prevstage		

The fact that all chains have their pointers at stage i is shown by the arrow $\uparrow$ in the last row of the above table. We will frequently move the pointers. The macro **Nextstage** is short for the sequence of transitions $\mathrm{next}(\mathtt{y}); \mathrm{next}(\overline{\mathtt{y}}); \ldots; \mathrm{next}(\mathtt{stack}); \mathrm{next}(\overline{\mathtt{stack}})$, which moves all the pointers one stage up as shown above. The macro **Prevstage** similarly moves all the pointers one stage down. The macro **transfer($\overline{\mathtt{z}}$, s)** is short for the following sequence of transitions.

transfer($\overline{\mathtt{z}}$, s) ⟨Next control state⟩
{
  zero: $\mathrm{inc}(\mathtt{z}); \mathrm{dec}(\overline{\mathtt{z}}); \mathrm{inc}(\mathtt{s}); \mathrm{dec}(\overline{\mathtt{s}})$; goto zero or fin
  fin: goto ⟨Next control state⟩
}

In the above description, ⟨Next control state⟩ represents the control state that immediately follows the macro **transfer($\overline{\mathtt{z}}$, s)** in a listing. The macro **transfer($\overline{\mathtt{y}}$, s)** is similar to the above, with y and $\overline{\mathtt{y}}$ replacing z and $\overline{\mathtt{z}}$ respectively. The macro **inc(next(z))** moves the pointer of the chain z one stage up, increments z

once and moves the pointer back one stage down: $\mathrm{next}(\mathtt{z}); \mathrm{inc}(\mathtt{z}); \mathrm{prev}(\mathtt{z})$. The macros **inc(next(**$\mathtt{y}$**))** and **inc(next(**$\overline{\mathtt{s}}$**))** are similar to **inc(next(**$\mathtt{z}$**))** with $\mathtt{y}$ and $\overline{\mathtt{s}}$ replacing $\mathtt{z}$ respectively. The macro **dec(next(**$\mathtt{s}$**))** is similarly defined to decrement the next counter in the chain $\mathtt{s}$. The macro `stack == 1` tests if the counter in the current stage in the chain `stack` is greather than 0: $\mathrm{dec}(\mathtt{stack}); \mathrm{inc}(\mathtt{stack})$.

Our returning mechanism is specific to the decrementing algorithm. There will be one copy for every zero testing transition in the counter automaton that we simulate (the size of the chain automaton will be linear in the size of the automaton even after creating these copies). The listing below is written in the form of a program in a low level programming language for readability. It can be easily translated into a set of transitions of a chain automaton. There is a control state between every two consecutive instructions below, but only the important ones are given names like "outerloop2". A line such as "innernonzero2: $\mathrm{dec}(\mathtt{z})$; $\mathrm{inc}(\mathtt{z})$; goto innerloop2" actually represents the set of transitions $\{\langle \text{innernonzero2}, \mathrm{dec}(\mathtt{z}), q\rangle, \langle q, \mathrm{inc}(\mathtt{z}), \text{innerloop2}\rangle\}$. The instruction "q: if $\mathrm{first}(\alpha)$? then {program 1} else {program 2}" represents the set of transitions $\{\langle q, \mathrm{first}(\alpha)?, q_1\rangle, \langle q, \overline{\mathrm{first}(\alpha)}?, q_2\rangle\} \cup$ {transitions for program 1 from q_1} $\cup$ {transitions for program 2 from q_2}. An instruction of the form "q : $\mathrm{inc}(\overline{\mathtt{stack}})$; goto outernonzero2 or outerzero2" represents the set of transitions $\{\langle q, \mathrm{inc}(\overline{\mathtt{stack}}), \text{outernonzero2}\rangle, \langle q, \mathrm{inc}(\overline{\mathtt{stack}}), \text{outerzero2}\rangle\}$. Depending on the non-deterministic choices made at control states that have multiple transitions enabled, there will be several different runs. We prove later that there is one run which has the intended effect and the other runs will never reach the final state. To understand the flow of the algorithm below, whenever the pointer is at stage i and the instruction "goto Dec" is executed, the reader can assume that $\mathtt{s}$ is decremented 2^{2^i} times and the control is magically transferred to the instruction immediately following the "goto Dec". We later prove formally that this actually happens. Referring to the tabular arrangement of the counters shown above would also help understand the following implementation and proofs of correctness.

Dec:
if $\mathrm{first}(\mathtt{stack})$? then

$(\mathrm{dec}(\mathtt{s}))^4; (\mathrm{inc}(\overline{\mathtt{s}}))^4$; goto DecFinished

else

Prevstage
outerloop2: $\mathrm{dec}(\mathtt{y}); \mathrm{inc}(\overline{\mathtt{y}})$ */* $\mathtt{y}$ is the index for outer loop */*
innerloop2: $\mathrm{dec}(\mathtt{z}); \mathrm{inc}(\overline{\mathtt{z}})$ */* $\mathtt{z}$ is the index for inner loop */*
dec(next(s)); inc(next($\overline{\mathtt{s}}$))
innertest2: goto innernonzero2 or innerzero2
innernonzero2: $\mathrm{dec}(\mathtt{z}); \mathrm{inc}(\mathtt{z})$; goto innerloop2 */* inner loop not yet complete */*
innerzero2: transfer($\overline{\mathtt{z}}$, s); $\mathrm{inc}(\mathtt{stack}); \mathrm{dec}(\overline{\mathtt{stack}})$; goto Dec */* inner loop complete. Assume control magically goes to outertest2 */*
outertest2: $\mathrm{dec}(\mathtt{stack}); \mathrm{inc}(\overline{\mathtt{stack}})$; goto outernonzero2 or outerzero2
outernonzero2: $\mathrm{dec}(\mathtt{y}); \mathrm{inc}(\mathtt{y})$; goto outerloop2 */* outer loop not yet complete */*
outerzero2: transfer($\overline{\mathtt{y}}$, s); goto Dec */* outer loop complete. Assume control magically goes to outerexit2 */*
outerexit2: Nextstage; goto DecFinished

fi
DecFinished: goto backtrack

backtrack:
if ($\overline{\mathrm{last}(\mathtt{stack})}$?) then

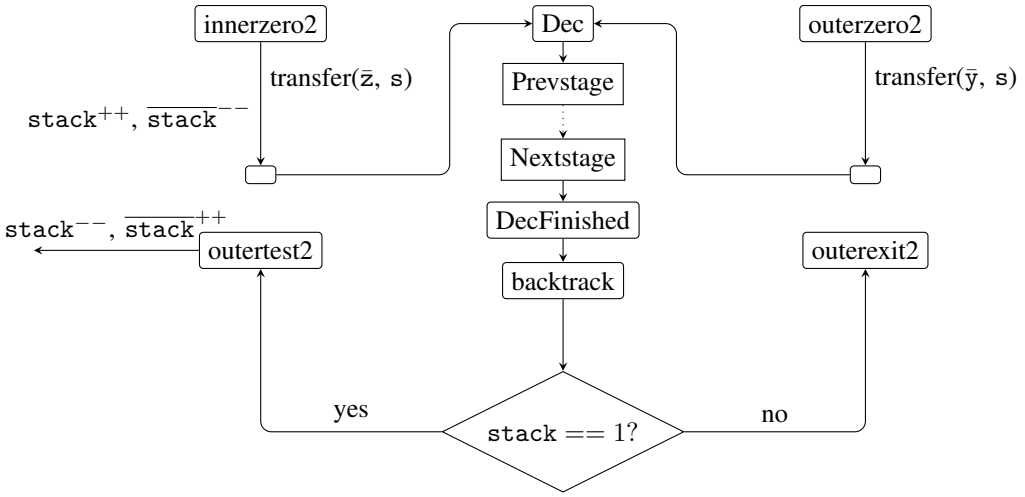
if (stack == 1) then goto outertest2
else goto outerexit2

else

goto ⟨Next control state⟩

fi

In the diagram below, we explain how the "magical transfers" happen after recursive calls to "Dec".



As can be seen from the listing, there are two recursive calls to "Dec" from inside "Dec". The first one is from "innerzero2", which should return to "outertest2". The second one is from "outerzero2", which should return to "outerexit2". As seen in the listing and also in the diagram above, the call from "innerzero2" is made just after incrementing `stack`. When the recursive call to "Dec" finishes (after possibly moving the pointers to previous stage and moving them back to current stage multiple times) and comes to backtrack, it will check if `stack` $== 1$. Since this is the case, the control goes to "outertest2", where the stack is set back to 0 and rest of the computation continues.

On the other hand, when a recursive call to "Dec" is made from "outerzero2", `stack` will be set to 0. Hence, when the recursive call to "Dec" returns and checks whether `stack` $== 1$, the answer will be no and the control goes to "outerexit2".

3) Initialisation Phase: In this section, we explain how to get counters to their required initial values (at the beginning of the runs all the values are equal to zero). We briefly recall what are these values after initialization:

1) each counter c has value zero,
2) for every $i \in [1, N]$, $\overline{\mathtt{y}}_i$, $\overline{\mathtt{z}}_i$ and $\mathtt{s}_i$ are equal to zero,
3) for every $i \in [1, N]$, $\mathtt{stack}_i$ is equal to zero and $\overline{\mathtt{stack}}_i$ is equal to one,
4) each complement counter $\overline{\mathtt{c}}$ has value 2^{2^N},
5) for every $i \in [1, N]$, $\mathtt{y}_i$, $\mathtt{z}_i$ and $\overline{\mathtt{s}}_i$ have the value 2^{2^i}.

Initialization for the points 1)–3) is easy to perform and below we focus on the initialization for the point 5). Initialization of the complement counter in point 4) will be dealt with in Section E5 by simply adjusting what is done below.

To help achieve these initial values for 5), we have another chain `init`, with N counters. We follow the convention that if at stage i, the counter in the chain `init` has value 1, then all the counters in all the chains at stage i or below have been properly initialized. By convention, the condition $\mathrm{last}(\mathtt{init})?$ is true if the pointer in the chain `init` is at stage N.

To understand our implementation of the decrementing algorithm, again, it will be helpful to think of the counters in the chains arranged as follows (this time we added the chain `init`)

Chain\Level	N	$N-1$	$N-2$	$\cdots$	i	$i-1$	$\ldots$	1
y						$2^{2^{i-1}}$		
z						$2^{2^{i-1}}$		
s					2^{2^i}	0		
stack	0	0	1	$\cdots$	1		$\cdots$	0
init	0	0	1	$\cdots$	1		$\cdots$	1
Pointer				Nextstage $\leftarrow$	$\uparrow$	$\rightarrow$ Prevstage		

The macro **Nextstage** is short for the sequence of transitions $\mathrm{next}(\mathtt{y}); \mathrm{next}(\overline{\mathtt{y}}); \ldots; \mathrm{next}(\mathtt{init}); \mathrm{next}(\overline{\mathtt{init}})$, which moves all the pointers one stage up as shown above. The macro **Prevstage** similarly moves all the pointers one stage down. The macro **next(init) == 1** tests if the value of the counter one stage up in the chain `init` is greater than 0: $\mathrm{next}(\mathtt{init}); \mathrm{dec}(\mathtt{init}); \mathrm{inc}(\mathtt{init}); \mathrm{prev}(\mathtt{init})$.

We give below the code used to initialize $\mathtt{y}, \mathtt{z}, \mathtt{s}$ and $\mathtt{stack}$ to the required values, assuming that all counters are initially set to 0 and all the pointers in all the chains are pointing to stage 0. To ease the reading, we have reproduced the code for Dec from Section E2.

As explained in Section E1, the principle for initialization is again nested loops. Below, the outer loop is implemented between the states "outerinit" and "outernonzero1", while the inner loop is between "innerinit" and "innernonzero1". Inside these loops, we have "inc(next(y)); inc(next(z));inc(next($\overline{\mathtt{s}}$))", which will increment counters in the next stage. This will work properly provided the counters in the current stage and below have already been initialized. We will prove later that initialization of counters at some stage will start only after all stages below have been initialized. We will also need to test counters at lower stages for zero, for which we will use the Dec algorithm from the previous section. Apart from the recursive calls to Dec from inside Dec itself, now there are also calls to Dec from "innerzero1" and "outerzero1" below. To handle these additional return locations, "backtrack" part of the code below has been updated that also checks if the call has been made from initialization code.

begininit: $(\mathrm{inc}(\mathtt{y}))^4; (\mathrm{inc}(\mathtt{z}))^4; (\mathrm{inc}(\overline{\mathtt{s}}))^4; \mathrm{inc}(\mathtt{init}); \mathrm{inc}(\overline{\mathtt{stack}})$
initialise: If $\mathrm{last}(\mathtt{init})?$ goto beginsim else goto outerinit
outerinit: $\mathrm{dec}(\mathtt{y}); \mathrm{inc}(\overline{\mathtt{y}})$ */* y is the index for outer loop */*
innerinit: $\mathrm{dec}(\mathtt{z}); \mathrm{inc}(\overline{\mathtt{z}})$ */* z is the index for inner loop */*
INC: inc(next(y)); inc(next(z)); inc(next($\overline{\mathtt{s}}$))
innertest1: goto innernonzero1 or innerzero1
innernonzero1: $\mathrm{dec}(\mathtt{z}); \mathrm{inc}(\mathtt{z})$; goto innerinit */* inner loop not yet complete */*
innerzero1: transfer($\overline{\mathtt{z}}$, s); $\mathrm{inc}(\mathtt{stack}); \mathrm{dec}(\overline{\mathtt{stack}})$; goto Dec */* inner loop complete. Assume control magically goes to outertest1 */*
outertest1: $\mathrm{dec}(\mathtt{stack}); \mathrm{inc}(\overline{\mathtt{stack}})$; goto outernonzero1 or outerzero1
outernonzero1: $\mathrm{dec}(\mathtt{y}); \mathrm{inc}(\mathtt{y})$; goto outerinit */* outer loop not yet complete */*
outerzero1: transfer($\overline{\mathtt{y}}$, s); goto Dec */* outer loop complete. Assume control magically goes to outerexit1 */*
outerexit1: Nextstage; $\mathrm{inc}(\mathtt{init}); \mathrm{inc}(\overline{\mathtt{stack}})$; goto initialise
beginsim: start simulating the counter machine (details follow but this is standard)

Dec:
if $\mathrm{first}(\mathtt{init})?$ then
    $(\mathrm{dec}(\mathtt{s}))^4; (\mathrm{inc}(\overline{\mathtt{s}}))^4$; goto DecFinished
else
    Prevstage
    outerloop2: $\mathrm{dec}(\mathtt{y}); \mathrm{inc}(\overline{\mathtt{y}})$ */* y is the index for outer loop */*
    innerloop2: $\mathrm{dec}(\mathtt{z}); \mathrm{inc}(\overline{\mathtt{z}})$ */* z is the index for inner loop */*
    dec(next(s)); inc(next($\overline{\mathtt{s}}$))
    innertest2: goto innernonzero2 or innerzero2
    innernonzero2: $\mathrm{dec}(\mathtt{z}); \mathrm{inc}(\mathtt{z})$; goto innerloop2 */* inner loop not yet complete */*
    innerzero2: transfer($\overline{\mathtt{z}}$, s); $\mathrm{inc}(\mathtt{stack}); \mathrm{dec}(\overline{\mathtt{stack}})$; goto Dec */* inner loop complete. Assume control magically goes to outertest2 */*
    outertest2: $\mathrm{dec}(\mathtt{stack}); \mathrm{inc}(\overline{\mathtt{stack}})$; goto outernonzero2 or outerzero2
    outernonzero2: $\mathrm{dec}(\mathtt{y}); \mathrm{inc}(\mathtt{y})$; goto outerloop2 */* outer loop not yet complete */*
    outerzero2: transfer($\overline{\mathtt{y}}$, s); goto Dec */* outer loop complete. Assume control magically goes to outerexit2 */*
    outerexit2: Nextstage; goto DecFinished
fi
DecFinished: goto backtrack
backtrack:
if (next(init) == 1) then */* we are not at the end of recursion */*
    if (stack == 1) then goto outertest2
    else goto outerexit2
else */* we are at the end of recursion */*
    if (stack == 1) then goto outertest1
    else goto outerexit1
fi

The control flow of the Dec procedure is presented in Figure 6.

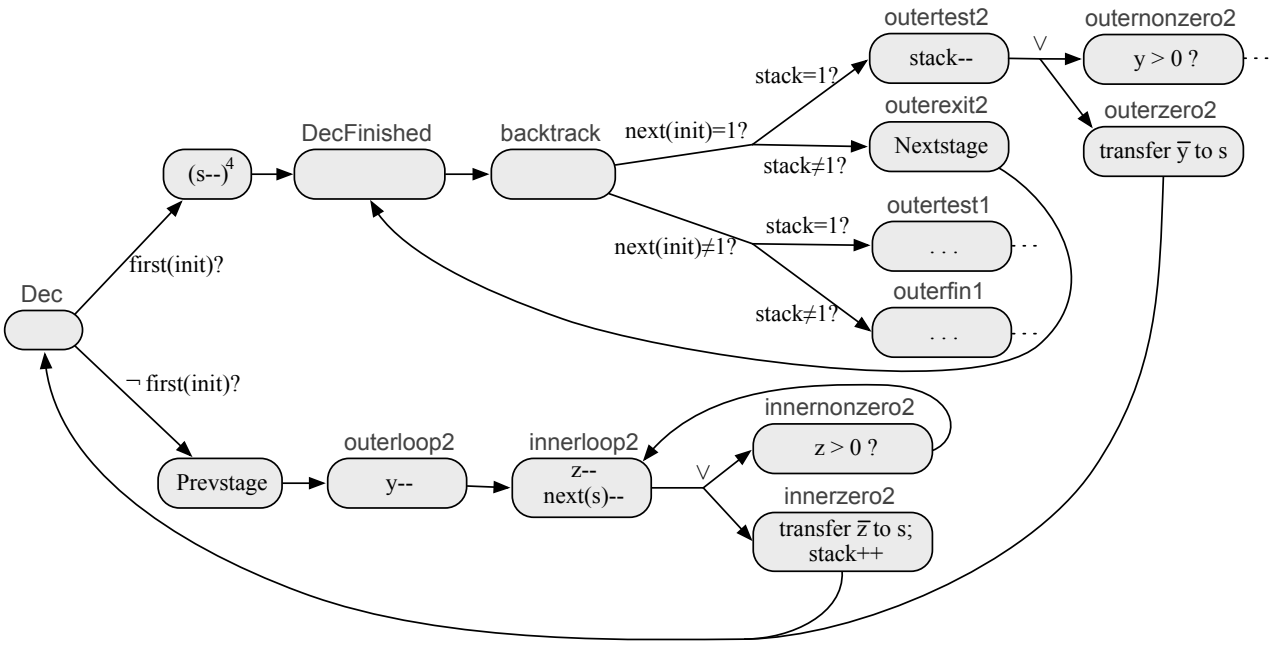


Fig. 6. Depiction of the control flow of the Dec procedure. In the figure we simplify some of the instructions, for example instead of $\mathrm{dec}(s); \mathrm{inc}(\bar{s})$ we just put s--, instead of $\mathrm{dec}(z); \mathrm{inc}(z)$ we just put $z > 0$?, etc.

4) Proof of Correctness: In the following lemmas, we refer to "a configuration of the chain automaton", by which we mean a configuration of the VASS $\mathcal{A}'$ defined in the proof of Lemma 13. A configuration has the pointer in chain α at stage i if the control state of $\mathcal{A}'$ is $\langle q, \beta_1, \ldots, \beta_\alpha, \ldots \rangle$ for some $q \in Q$ and $\beta_\alpha = i$. A configuration has the value x in the counter in the chain α at stage i if the counter $\mathtt{c}_i^\alpha$ has the value x.

Lemma 28. Suppose in a configuration of the chain automaton, all the pointers in all the chains are at stage i, the counters in the chains `init` and `s` at stage i have values 1 and 2^{2^i} respectively and all the counters in all the chains at or below stage $i-1$ have been initialised. If the control state is at "Dec", then there is a run ρ_i ending at the control state "DecFinished" (just before "backtrack" in the above listing) with the pointers at stage i in all the chains and no change to any counter at any stage, except that the counter at stage i in the chain `s` has the value 0 (intuitively, there is a run which decreases `s` by 2^{2^i} and does nothing else).

Proof: By induction on i. For the base case $i = 1$, we note that the test $\mathrm{first}(\mathtt{init})?$ is true. So ρ_1 is the run that decrements `s` four times, increments $\overline{\mathtt{s}}$ four times and goes to "DecFinished".

Assuming the lemma is true up to value i, we prove it for $i+1$. The run ρ_{i+1} does not perform the test $\mathrm{first}(\mathtt{init})?$ at the beginning of "Dec" but instead goes to the "else" branch. Following is the run ρ_{i+1}.

$$\text{Prevstage}; (\text{outerloop2}; (\text{innerloop2}; \mathrm{dec}(\mathrm{next}(\mathtt{s})); \mathrm{inc}(\mathrm{next}(\overline{\mathtt{s}})); \text{innertest2})^{2^{2^i}}$$
$$\text{innerzero2}; \text{transfer}(\overline{\mathtt{z}}, \mathtt{s}); \rho_i; \text{DecFinished}; \text{backtrack}; \text{outertest2})^{2^{2^i}}$$
$$\text{outerzero2}; \text{transfer}(\overline{\mathtt{y}}, \mathtt{s}); \rho_i; \text{DecFinished}; \text{backtrack}; \text{outerexit2}; \text{Nextstage}; \text{DecFinished}$$

Note that in the above run, we have omitted some instructions and program points but we have kept those that are the most essential for our reasoning.

The first transition is to move all pointers one stage down to i. Since all counters at stage i or below have been initialised, `y` and `z` both have the value 2^{2^i}. The first execution of outerloop2 results in `y` decreasing and $\overline{\mathtt{y}}$ increasing. The first execution of innerloop2 results in `z` decreasing and $\overline{\mathtt{z}}$ increasing. Then dec(next(`s`)); inc(next($\overline{\mathtt{s}}$)) results in the counter at stage $i+1$ in the chain `s` decreasing and $\overline{\mathtt{s}}$ increasing (recall that the pointer is now at stage i). Then our run chooses to go to innernonzero2, decreases and increases `z` and goes back to innerloop2. This inner loop is repeated 2^{2^i} times, at the end of which the counter at stage i in the chain `z` has value 0, $\overline{\mathtt{z}}$ has value 2^{2^i} and the counter at stage

$i+1$ in the chain s has been decremented 2^{2^i} times. Now our run chooses to go to innerzero2, where the first macro transfers the value of $\overline{\mathtt{z}}$ into s at stage i. So, the counters at stage i in the chains s and z both have the value 2^{2^i}. Then $\mathrm{inc}(\mathtt{stack}); \mathrm{dec}(\overline{\mathtt{stack}})$; sets stack to 1 at stage i and the control is transferred to "Dec". Since all the counters at or below stage $i-1$ have been initialised (they were initialised in the beginning and we did not touch any counter at or below stage $i-1$ so far) and the counter at stage i in the chain init is 1 (recall that all counters at stage i were initialised at the beginning and we have not touched init so far), we can apply the induction hypothesis. So our run now goes to "DecFinished" with s at stage i set to 0 and no other changes, and all the pointers in all the chains at stage i. At DecFinished, the control is transferred to "backtrack". Here the test (next(init) == 1) passes (since the counter at stage $i+1$ in the chain init has the value 1) and so does (stack == 1) (since we increased stack at stage i before going to Dec) and so the control is transferred to outertest2. At outertest2, our run sets stack back to 0 and chooses to go to outernonzero2. Here, y is increased, then decreased and the control is transferred back to outerloop2. Here y is decreased and the whole innerloop is again executed to decrease s at stage $i+1$ 2^{2^i} times. This way, the inner loop is executed 2^{2^i} times, each time decreasing y at stage i once and decreasing s at stage $i+1$, 2^{2^i} times. At the end, s at stage $i+1$ has been decremented $2^{2^i} \times 2^{2^i} = 2^{2^{i+1}}$ times and now our run chooses to go to outerzero2.

Now, the first macro transfers the value of $\overline{\mathtt{y}}$ into s at stage i. So, the counters at stage i in the chains s and y both have the value 2^{2^i}. Then the control is transferred to "Dec". Since all counters at or below stage $i-1$ have been initialised (they were initialised in the beginning and our calls to Dec till now left them intact by induction hypothesis) and the counter at stage i in the chain init is 1 (recall that all counters at stage i were initialised at the beginning and we have not changed init so far), we can apply the induction hypothesis. So our run now goes to "DecFinished" with s at stage i set to 0 and no other changes, and all the pointers in all the chains at stage i. At DecFinished, the control is transferred to "backtrack". Here the test (next(init) == 1) passes (since the counter at stage $i+1$ in the chain init has the value 1) but the test (stack == 1) does not go through. So the control is transferred to outerexit2. Here, all the pointers are moved to stage $i+1$ and the control is transferred to backtrack, exactly as stated in the lemma. Recall that we have reset the values in z and y back to 2^{2^i}, so we have not made any changes to any counter except decreasing the counter in the chain s at stage $i+1$ by $2^{2^{i+1}}$. This proves the induction step and hence the lemma. ■

Lemma 29. Suppose the control state is "begininit", all the pointers in all the chains are at stage 0 and all the counters at all stages have the value 0. For any i between 1 and N, there is a run ρ'_i ending at "initialise", such that the pointer is at stage i in all the chains and all the counters at or below stage i have been initialised.

Proof: By induction on i. For the base case $i = 1$, ρ'_1 is the run that executes the instructions immediately after "begininit" and ends at "initialise".

Now we assume the lemma is true up to i and prove it for $i+1$. By induction hypothesis, there is a run ρ'_i ending at "initialise", with all the pointers in all the chains at stage i and all the counters at or below stage i initialised. The run ρ'_{i+1} is obtained by appending the following sequence to ρ'_i. It uses the runs ρ_i constructed from the previous lemma.

$$\begin{aligned}(\text{outerinit}; (\text{innerinit}; \mathrm{inc}(\mathrm{next}(\mathtt{y})); \mathrm{inc}(\mathrm{next}(\mathtt{z})); \mathrm{inc}(\mathrm{next}(\overline{\mathtt{s}})); \text{innertest1})^{2^{2^i}};\\ \text{innerzero1}; \text{transfer}(\overline{\mathtt{z}}, \mathtt{s}); \rho_i; \text{DecFinished}; \text{backtrack}; \text{outertest1})^{2^{2^i}};\\ \text{outerzero1}; \text{transfer}(\overline{\mathtt{y}}, \mathtt{s}); \rho_i; \text{DecFinished}; \text{backtrack}; \text{outerexit1}; \text{initialise}\end{aligned}$$

Note that in the above run, we have omitted some instructions and program points but we have kept those that are the most essential for our reasoning.

The run ρ'_i is continued by testing if i is the last stage. If not, we go to "outerinit", decrement y, then go to "innerinit", decrement z, then increment the counters in the chains $\mathtt{y}, \mathtt{z}$ and $\overline{\mathtt{s}}$ at stage $i+1$. Then our run chooses "innernonzero1", decrements and increments z and goes back to "innerinit". This inner loop is executed 2^{2^i} times, at the end of which the counter at stage i in the chain z has value 0, $\overline{\mathtt{z}}$ has value 2^{2^i} and the counters at stage $i+1$ in the chains $\mathtt{y}, \mathtt{z}$ and $\overline{\mathtt{s}}$ have been incremented 2^{2^i} times. Now our run chooses to go to innerzero1, where the first macro transfers the value of $\overline{\mathtt{z}}$ into s at stage i. So, the counters at stage i in the chains s and z both have the value 2^{2^i}. Then $\mathrm{inc}(\mathtt{stack}); \mathrm{dec}(\overline{\mathtt{stack}})$; sets stack to 1 at stage i and the control is transferred to "Dec". Since all the counters at or below stage $i-1$ have been initialised and the counter at stage i in the chain init is 1 (recall our convention that counters at stage i or below initialised implies that the counter in the chain init at stage i has

the value 1), we can apply Lemma 28. So our run now goes to "DecFinished" with s at stage i set to 0 and no other changes, and all the pointers in all the chains at stage i. At DecFinished, the control is transferred to "backtrack". Here the test (next(init) == 1) does not go through and the test (stack == 1) passes, so control is transferred to "outertest1". At outertest1, our run sets stack back to 0 and chooses to go to outernonzero1. Here, y is increased, then decreased and the control is transferred back to outerloop1. Here y is decreased and the whole innerloop is again executed to increment y, z and $\overline{\text{s}}$ at stage $i+1$ 2^{2^i} times. This way, the inner loop is executed 2^{2^i} times, each time decreasing y at stage i once and incrementing y, z and $\overline{\text{s}}$ stage $i+1$ 2^{2^i} times. At the end, y, z and $\overline{\text{s}}$ at stage $i+1$ have been incremented $2^{2^i} \times 2^{2^i} = 2^{2^{i+1}}$ times and now our run chooses to go to outerzero1.

Now, the first macro at "outerzero1" transfers the value of $\overline{\text{y}}$ into s at stage i. So, the counters at stage i in the chains s and y both have the value 2^{2^i}. Then the control is transferred to "Dec". Since all counters at or below stage $i-1$ have been initialised (they were initialised in the beginning and our calls to Dec till now left them intact by Lemma 28) and the counter at stage i in the chain init is 1 , we can apply Lemma 28. So our run now goes to "DecFinished" with s at stage i set to 0 and no other changes, and all the pointers in all the chains at stage i. At DecFinished, the control is transferred to "backtrack". Here the test (next(init) == 1) does not go through and neither does the test (stack == 1). So the control is transferred to "outerexit1". Here, all the pointers are moved to stage $i+1$, init and $\overline{\text{stack}}$ at stage $i+1$ are set to 1 and the control is transferred to "initialise", exactly as in the lemma. This proves the induction step and hence the lemma. ■

The macros Nextstage and Prevstage move all the pointers in all the chains synchronously. Hence, in any run of the chain automaton, all the pointers are at the same stage. This condition can be temporarily violated while executing macros like inc(next(y)), but if all the pointers are at the same stage before the macro is executed, then any run that executes the macro is forced to regain the condition after the macro is executed (there is no non-deterministic choice inside the macro).

Lemma 30. Suppose that in a configuration of the chain automaton, all the pointers in all the chains are at stage i, all the counters at or below stage $i-1$ have been initialised, the counter in the chain init at stage i has the value 1 and the control state is at "Dec". If any run starts from this configuration (say $conf_1$) and ends at a configuration $conf_2$ with control state at "DecFinished" and all pointers at stage i (with no other intermediate configurations satisfying these two properties), then the run decreases the counter in the chain s at stage i by 2^{2^i}.

Proof: By induction on i. For the base case $i = 1$, if a run chooses the "else" branch of Dec, it gets stuck since the next instruction is to move all the pointers one stage down, which is not possible. So the run has to choose the only other option — the "if" branch, where it is forced to decrease s at stage 1 by 4 to reach "DecFinished".

Now we assume the lemma is true up to i and prove it for $i+1$. So suppose that in the configuration $conf_1$, all the pointers are at stage $i+1$, all the counters at or below stage i are initialised, the counter in the chain init at stage $i+1$ is 1 and the control state is at Dec. Suppose a run starts at $conf_1$ and reaches $conf_2$ with the control state at "DecFinished" and all pointers at stage $i+1$. First observe that if between $conf_1$ and $conf_2$, the pointers are moved below stage $i+1$, then the only way to get the pointers back to stage $i+1$ is through the macro Nextstage at control state "outerexit2", which can only be executed if the control state was at "backtrack", which means before $conf_2$, there is a configuration in which the control state is "DecFinished" and pointers are at stage i.

Now let us track the instructions executed by this run. In the first transition after $conf_1$, this run cannot choose the "if" branch of Dec since the test first(init)? does not go through. So the run goes to the "else" branch. It is forced to move all the pointers one stage down to i, decrement y and z at stage i, decrement s at stage $i+1$ and make a choice between "innernonzero2" and "innerzero2". If "innerzero2" is chosen at this stage, then in transfer($\overline{\text{z}}$, s), the value of s can be incremented at most once and then the control is forced to Dec. Then by induction hypothesis, no run can reach "DecFinished". So the only way to continue is to execute the innerloop2 2^{2^i} times, which requires decrementing s at stage $i+1$ 2^{2^i} times. At the end of this, when the control goes to Dec (after setting stack at stage i to 1), we know it has to reach "DecFinished" with pointers at stage i before reaching $conf_2$. By induction hypothesis, we know that when the run first comes to "DecFinished" with pointers at stage i, it would have decreased s at stage i by 2^{2^i}. Now the control is forced to "backtrack", and then to "outertest2" (since init at stage $i+1$ is 1 and we had set stack at stage i to 1). Here, the run is forced to set stack back to 0 and make a choice between outernonzero2 or outerzero2. As before, the run cannot continue if it chooses outerzero2 at this stage, so it is forced to choose outernonzero2. In fact, the run is forced to choose

outernonzero2 2^{2^i} times, which forces decreasing $\mathtt{s}$ at stage i by $2^{2^{i+1}}$. At the end of this, the run is forced to choose outerzero2 (choosing outernonzero2 now will make the run get stuck since $\mathrm{dec}(\mathtt{y})$ no longer possible with $\mathtt{y}$ having the value 0), increment $\mathtt{s}$ at stage i 2^{2^i} times (to avoid getting stuck in the Dec that follows), go to Dec, then by induction hypothesis forced to go to DecFinished with $\mathtt{s}$ at stage i decreased by 2^{2^i} and all ther pointers at stage i. Now the control is forced to backtrack and then to outerexit2 (since $\mathtt{init}$ at stage $i+1$ has the value 1 and $\mathtt{stack}$ at stage i has the value 0). Here, the pointers are moved one stage up to $i+1$ and the control goes to DecFinished. ■

5) Concluding the Proof of Theorem 14: We give a reduction from the control state reachability problem for counter automata with zero tests with counters bounded by 2^{2^N}, where $N = \mathtt{exp}(k, n^\gamma)$ and n is the size of counter automaton. Without any loss of generality, we can assume that the initial counter values are equal zero (then these values will be updated during the initialization phase). Given such a counter automaton, we construct a chain automaton, with each chain having N counters. It will have the chains $\mathtt{s}, \overline{\mathtt{s}}, \mathtt{y}, \overline{\mathtt{y}}, \mathtt{z}, \overline{\mathtt{z}}, \mathtt{stack}, \overline{\mathtt{stack}}, \mathtt{init}$ and $\overline{\mathtt{init}}$ required for zero tests as explained previously. There will be two more chains $\mathtt{counters}$ and $\overline{\mathtt{counters}}$ to simulate the counters of the counter automaton. The right number of counters can be obtained by imposing that for every chain α, $f(\alpha)$ is set to n^γ.

The initial state of the chain automaton is "begininit", the control state that launches the program for initialising the zero testing counters. The chain automaton will have all the control states of the counter automaton (in addition to others). The final state of the chain automaton is equal to the accepting state of the counter automaton. At the control state "beginsim", the chain automaton launches a program to initialise all the counters in the chain $\overline{\mathtt{counters}}$ to 2^{2^N}, similar to the program for initialising the zero testing counters.

In order to initialize the first $\delta \leq n$ counters from $\mathtt{counters}$ and $\overline{\mathtt{counters}}$ that correspond to the counters of the original counter automaton, we replace the code for **INC** from the code for the initialization phase by the following one:

INC: inc(next($\mathtt{y}$)); inc(next($\mathtt{z}$)); inc(next($\overline{\mathtt{s}}$));
$\mathrm{next}(\mathtt{init})$;
if $\mathrm{last}(\mathtt{init})$? then
  $\mathrm{inc}(\overline{\mathtt{counters}})$;
  $(\mathrm{next}(\overline{\mathtt{counters}});\mathrm{inc}(\overline{\mathtt{counters}}))^{\delta-1}$;
  $(\mathrm{prev}(\overline{\mathtt{counters}}))^{\delta-1}$;
$\mathrm{prev}(\mathtt{init})$;

After initialization, the next control state will be the initial state of the counter automaton and the pointers of all the chains $\mathtt{s}, \overline{\mathtt{s}}, \mathtt{y}, \overline{\mathtt{y}}, \mathtt{z}, \overline{\mathtt{z}}, \mathtt{stack}, \overline{\mathtt{stack}}, \mathtt{init}$ will be at stage $N = \mathtt{exp}(k, n^\gamma)$. The pointers of the chains $\mathtt{counters}$ and $\overline{\mathtt{counters}}$ will be at stage 0. From Lemma 29, there is a run that does all initialisations correctly and reaches the initial state of the counter automaton. From Lemma 30, any run that does not properly initialise all the counters gets stuck and hence will not reach the final state.

We denote the counters in the chains $\mathtt{counters}, \overline{\mathtt{counters}}$ by $\mathtt{c}_0, \overline{\mathtt{c}}_0, \ldots, \mathtt{c}_{\delta-1}, \overline{\mathtt{c}}_{\delta-1}$. For every transition $\langle q, \mathtt{c}_j \leftarrow \mathtt{c}_j + 1, q' \rangle$ of the counter automaton, the chain automaton will have the transition

$$\langle q, (\mathrm{next}(\mathtt{counters}); \mathrm{next}(\overline{\mathtt{counters}}))^j; \mathrm{inc}(\mathtt{counters}); \mathrm{dec}(\overline{\mathtt{counters}}); \\ (\mathrm{prev}(\mathtt{counters}); \mathrm{prev}(\overline{\mathtt{counters}}))^j, q' \rangle$$

Thus, for every incrementing transition of the counter automaton, the chain automaton can execute a corresponding incrementing transition. Conversely, for every incrementing transition executed by the chain automaton, the counter automaton can execute the corresponding incrementing transition.

For every transition $\langle q : \text{if } \mathtt{c}_j = 0 \text{ goto } q_1 \text{ else } \mathtt{c}_j \leftarrow \mathtt{c}_j - 1 \text{ goto } q_2 \rangle$ of the counter automaton, the chain automaton will have transitions corresponding to the following program. If the jth counter is equal to zero, then the content of the complement counter is equal to 2^{2^N}, which we transfer to $\mathtt{s}$ and then invoke Dec. We consider a slight variant of Dec so that the return address is q_1 after passing the zero-test successfully. In addition, since we need to restore the value of the complement counter, we perform a second transfer from the counter to $\mathtt{s}$ so that the total effect is null.

q: goto nonzero or zero
nonzero: $(\mathrm{next}(\mathtt{counters}); \mathrm{next}(\overline{\mathtt{counters}}))^j$; $\mathrm{dec}(\mathtt{counters}); \mathrm{inc}(\overline{\mathtt{counters}})$;
$(\mathrm{prev}(\mathtt{counters}); \mathrm{prev}(\overline{\mathtt{counters}}))^j$; goto q_2

zero: $(\mathrm{next}(\texttt{counters}); \mathrm{next}(\overline{\texttt{counters}}))^j$; transfer($\overline{\texttt{counters}}$, s); goto $\mathrm{Dec}_{\textbf{zerorep}}$
zerorep: transfer(counters, s); goto $\mathrm{Dec}_{\textbf{zeropass}}$
zeropass: $(\mathrm{prev}(\texttt{counters}); \mathrm{prev}(\overline{\texttt{counters}}))^j$; goto q_1

In the above, the control state $\mathrm{Dec}_{\textbf{zerorep}}$ launches a program very similar to Dec explained previously, except that instead of backtrack, the following $\mathrm{backtrack}_{\textbf{zerorep}}$ is called.

$\mathrm{backtrack}_{\textbf{zerorep}}$:
if ($\overline{\mathrm{last}(\texttt{stack})?}$) then */* we are not at the end of recursion */*
    if (stack == 1) then goto outertest2
    else goto outerexit2
else */* we are at the end of recursion */*
    goto zerorep
fi

The difference between backtrack and $\mathrm{backtrack}_{\textbf{zerorep}}$ is that the later transfers control to zerorep after completion. The control state $\mathrm{Dec}_{\textbf{zeropass}}$ launches a program very similar to Dec explained previously, except that instead of backtrack, $\mathrm{backtrack}_{\textbf{zeropass}}$ is called, which transfers control to zeropass after completion.

Now we explain the working of the above program in detail, starting from the state **zero**. If the value in the j^{th} counter of the counter automaton is 0 when the control state is q, then the value in the counter at stage j of the chain $\overline{\texttt{counters}}$ in the chain automaton will have the value 2^{2^N}. The macro transfer($\overline{\texttt{counters}}$, s) will set the j^{th} counter in the chain $\overline{\texttt{counters}}$ to 0 and set the j^{th} counter in the chain counters to 2^{2^N}, while swapping the values of s and $\overline{\texttt{s}}$ at stage $N = \texttt{exp}(k, n^\gamma)$. Then by Lemma 28, there is a run that swaps back the values of s and $\overline{\texttt{s}}$ at stage N and ends in the control state zerorep. Now the macro transfer(counters, s) will set the j^{th} counter in the chain $\overline{\texttt{counters}}$ back to 2^{2^N} and set the j^{th} counter in the chain counters back to 0 while swapping the values of s and $\overline{\texttt{s}}$ at stage N. Now the call to the decrementing algorithm at $\mathrm{Dec}_{\textbf{zeropass}}$ swaps back the values of s and $\overline{\texttt{s}}$ at stage N and the control is at zeropass, with no changes in any of the counters. Hence, for any zero testing transition executed by the counter automaton, the chain automaton can execute the corresponding sequence of transitions. Conversely, by Lemma 30, for any run of the chain automaton that begins at q and ends at q_1, the value of the j^{th} counter in the chain $\overline{\texttt{counters}}$ was 2^{2^N} and hence the value of the j^{th} counter in the chain counters was 0. Therefore, for any zero testing sequence of transitions executed by the chain automaton, the counter automaton can execute the corresponding zero testing transition.

We conclude that the chain automaton has a perfect run ending at its final state iff the counter automaton has an accepting run.

F. Proof of Lemma 15

Proof: Let $\mathcal{A} = \langle Q, f, 1, Q_0, Q_F, \delta\rangle$ be an chain system of level 1 with $f : [1, n] \to \mathbb{N}$, having thus n chains of counters, of respective size $2^{f(1)}, \ldots, 2^{f(n)}$.

The objective is to encode a word $\rho \in \delta^*$ that represents an accepting run. For this, we use the alphabet δ of transitions, that we code using a logarithmic number of variables.

We encode a word $\rho \in \delta^*$ that represents an accepting run. For this, we use the alphabet δ of transitions. Note that we can easily simulate the labels $\delta = \{t_1, \ldots, t_m\}$ with the variables $\mathtt{t}_0, \ldots, \mathtt{t}_m$, where a node has an encoding of the label t_i iff the formula $\langle t_i\rangle = \mathtt{t}_0 \approx \mathtt{t}_i$ holds true. We build a LRV formula φ so that there is an accepting gainy run $\rho \in \delta^*$ of $\mathcal{A}$ if, and only if, there is a model σ so that $\sigma \models \varphi$ and σ encodes the run ρ.

The following are standard counter-blind conditions to check.

(a) Every position satisfies $\langle a\rangle$ for some $a \in \delta$.
(b) The first position satisfies $\langle(q_0, \mathrm{instr}, q)\rangle$ for some $q_0 \in Q_0$, instr $\in I$, $q \in Q$.
(c) The last position satisfies $\langle(q, \mathrm{instr}, q')\rangle$ for some $q \in Q$, instr $\in I$, $q' \in Q_F$.
(d) There are no two consecutive positions i and $i+1$ satisfying $\langle(q, u, q')\rangle$ and $\langle(p, u', p')\rangle$ respectively, with $q' \neq p$.

We use a variable x, and variables $\mathtt{x}^\alpha_{inc}, \mathtt{x}^\alpha_{dec}, \mathtt{x}^\alpha_i$ for every chain α and $i \in [1, f(\alpha)]$. Let us fix the bijections $\chi^\alpha : [0, 2^{f(\alpha)} - 1] \to 2^{[1, f(\alpha)]}$ for every $\alpha \in [1, n]$, that assign to each number m the set of 1-bit positions of the representation of m in base 2. We say that a position i is *α-incrementing* [resp. *α-decrementing*] if it satisfies $\langle(q, u, q')\rangle$ for some $q, q' \in Q$ and $u = \mathrm{inc}(\alpha)$ [resp. $u = \mathrm{dec}(\alpha)$].

In the context of a model σ with the properties (a)–(d), we say that a position i operates on the α-counter $\mathtt{c}$, if $\chi^\alpha(\mathtt{c}) = X_i$, where

$$X_i = \{b \in [1, f(\alpha)] \mid \sigma(i)(\mathtt{x}) = \sigma(i)(\mathtt{x}^\alpha_b)\}. \qquad (\dagger)$$

Note that thus $0 \leq \mathtt{c} < 2^{f(\alpha)}$.

For every chain α, let us consider the following properties:

(1) Any two positions of σ have different values of $\mathtt{x}^\alpha_{inc}$ [resp. of $\mathtt{x}^\alpha_{dec}$].

(2) For every position i of σ operating on an α-counter $\mathtt{c}$ containing an instruction '$\mathrm{first}(\alpha)$?' [resp. '$\overline{\mathrm{first}(\alpha)}$?', '$\mathrm{last}(\alpha)$?', '$\overline{\mathrm{last}(\alpha)}$?'], we have $\mathtt{c} = 0$ [resp. $\mathtt{c} \neq 0$, $\mathtt{c} = 2^{f(\alpha)} - 1$, $\mathtt{c} \neq 2^{f(\alpha)} - 1$].

(3) For every position i of σ operating on an α-counter $\mathtt{c}$,
- if the position contains an instruction '$\mathrm{next}(\alpha)$' [resp. '$\mathrm{prev}(\alpha)$'], then the next position $i + 1$ operates on the α-counter $\mathtt{c} + 1$ [resp. $\mathtt{c} - 1$],
- otherwise, the position $i + 1$ operates on the α-counter $\mathtt{c}$.

(4) For every α-incrementing position i of σ operating on an α-counter $\mathtt{c}$ there is a future α-decrementing position $j > i$ operating on the same α-counter $\mathtt{c}$, such that $\sigma(i)(\mathtt{x}^\alpha_{inc}) = \sigma(j)(\mathtt{x}^\alpha_{dec})$.

Claim 31. There is a model satisfying the conditions above if, and only if, $\mathcal{A}$ has a gainy and accepting run.

Proof: **(Only if)** Suppose we have a model σ that verifies all the properties above. Since by (1), all the positions have different data values for $\mathtt{x}^\alpha_{dec}$, it then follows that the position j to which property (4) makes reference is unique. Since, also by (1), all the positions have different data values for $\mathtt{x}^\alpha_{inc}$, we have that the α-decrement corresponds to at most one α-increment position in condition (4). Moreover, it is an α-decrement of the same counter, that is, $X_i = X_j$ (cf. (†)). For these reasons, for every α-increment there is a future α-decrement of the same counter which corresponds in a unique way to that increment. More precisely, for every chain α, the function

$$\gamma^\alpha_\sigma : \{0 \leq i < |\sigma| : i \text{ is } \alpha\text{-incrementing}\} \to \{0 \leq i < |\sigma| : i \text{ is } \alpha\text{-decrementing}\}$$

where $\gamma^\alpha_\sigma(i) = j$ iff $\sigma(i)(\mathtt{x}^\alpha_{inc}) = \sigma(j)(\mathtt{x}^\alpha_{dec})$ is well-defined and injective, and for every $\gamma^\alpha_\sigma(i) = j$ we have that $j > i$ and $X_i = X_j$.

Consider $\rho \in \delta^*$ as having the same length as σ and such that $\rho(i) = (q, \mathrm{instr}, q')$ whenever $\sigma, i \models \langle(q, \mathrm{instr}, q')\rangle$. From conditions (2) and (3), it follows that the counter c^α_i corresponding to position i from ρ, is equal to $\chi^{-1}(X_i)$. Therefore, the functions $\{\gamma^\alpha_\sigma\}_{\alpha \in [1,n]}$ witness the fact that ρ is a gainy and accepting run of $\mathcal{A}$.

(If) Let us now focus on the converse, by exhibiting a model satisfying the above properties, assuming that $\mathcal{A}$ has an accepting gainy run $\rho \in \delta^*$. We therefore have, for every $\alpha \in [1, n]$, an injective function

$$\gamma^\alpha : \{i \mid \rho(i) \text{ is } \alpha\text{-incrementing}\} \to \{i \mid \rho(i) \text{ is } \alpha\text{-decrementing}\}$$

where for every $\gamma(i) = j$ we have that $j > i$ and $c^\alpha_i = c^\alpha_j$.

We build a model σ of the same length of ρ, and we now describe its data values. Let us fix two distinct data values d, e. For any position i, we have $\sigma(i)(l_0) = d$; and $\sigma(i)(l_j) = d$ if $j \in \lambda(\rho[i])$, and $\sigma(i)(l_j) = e$ otherwise. In this way we make sure that the properties (a)–(d) hold.

We use distinct data values $d_0, \ldots, d_{|\sigma|-1}$ and $d'_0, \ldots, d'_{|\sigma|-1}$ (all different from d, e). We define the data values of the remaining variables for any position i:

- For every α, $\sigma(i)(\mathtt{x}^\alpha_{inc}) = d_i$.
- For every α,
  - if i is α-decrementing, we define $\sigma(i)(\mathtt{x}^\alpha_{dec}) = \sigma(i')(\mathtt{x}^\alpha_{inc})$ where $i' = (\gamma^\alpha)^{-1}(i)$; if such $(\gamma^\alpha)^{-1}(i)$ does not exist, then $\sigma(i)(\mathtt{x}^\alpha_{dec}) = d'_i$;
  - otherwise, if i is not α-decrementing, $\sigma(i)(\mathtt{x}^\alpha_{dec}) = d'_i$.
- $\sigma(i)(\mathtt{x}) = d$.
- For every α, $\sigma(i)(\mathtt{x}^\alpha_l) = \sigma(i)(\mathtt{x}) = d$ if $l \in \chi(c^\alpha_i)$, otherwise $\sigma(i)(\mathtt{x}^\alpha_l) = e$.

Observe that these last two items ensure that the properties (2) and (3) hold.

By definition, every position of σ has a different data value for $\mathtt{x}^\alpha_{inc}$. Let us show that the same holds for $\mathtt{x}^\alpha_{dec}$. Note that at any position i, $\mathtt{x}^\alpha_{dec}$ has: the data value of $\sigma(i')(\mathtt{x}^\alpha_{inc}) = d_{i'}$ for some $i' < i$; or the data value d'_i. If there were two positions $i < j$ with $\sigma(i)(\mathtt{x}^\alpha_{dec}) = \sigma(j)(\mathtt{x}^\alpha_{dec})$ it would then be because: (i) $d'_i = d'_j$; (ii) $d_{i'} = d'_j$ for some $i' < i$; (iii) $d'_i = d_{j'}$ for some $j' < j$; or (iv) $d_{i'} = d_{j'}$ for some

$i' < i$, $j' < j$. It is evident that none of (i), (ii), (iii) can hold since all $d_0, \ldots, d_{|\sigma|-1}, d'_0, \ldots, d'_{|\sigma|-1}$ are distinct. For this reason, if (iv) holds, it means that $i' = j'$, and hence that $(\gamma^\alpha)^{-1}(i) = (\gamma^\alpha)^{-1}(j)$, implying that γ^α is not injective, which is a contradiction. Hence, all the positions have different data values for $\mathtt{x}^\alpha_{dec}$. Therefore, σ has the property (1).

To show that σ has property (4), let i be a α-incrementing position of σ, remember that $\sigma(i)(\mathtt{x}^\alpha_{inc}) = d_i$. Note that position $\gamma^\alpha(i) = j$ must be α-decrementing on the same counter. By definition of the value of $\mathtt{x}^\alpha_{dec}$, we have that $\sigma(j)(\mathtt{x}^\alpha_{dec})$ must be equal to $\sigma((\gamma^\alpha(j))^{-1})(\mathtt{x}^\alpha_{inc}) = \sigma(i)(\mathtt{x}^\alpha_{inc}) = d_i$. Thus, property (4) holds. ■

We complete the reduction by showing that all the properties expressed before can be efficiently encoded in our logic.

Claim 32. Properties (a)–(d) and (1)–(4) can be expressed by formulas of LRV, that can be constructed in polynomial time in the size of $\mathcal{A}$.

Proof: Along the proof, we use the following formulas, for any $\alpha \in [1, n]$ and $i \in [1, f(\alpha)]$,

$$\mathit{bit}^\alpha_i \stackrel{\text{def}}{=} \mathtt{x} \approx \mathtt{x}^\alpha_i.$$

We now show how to code each of the properties.

- Properties (a)–(d) are easy to express in LRV and present no complications.
- For expressing (1), we force $\mathtt{x}^\alpha_{inc}$ and $\mathtt{x}^\alpha_{dec}$ to have a different data value for every position of the model with the formula

$$\neg \mathsf{F}\ \mathtt{x}^\alpha_{inc} \approx \langle \top ? \rangle \mathtt{x}^\alpha_{inc} \quad \wedge \quad \neg \mathsf{F}\ \mathtt{x}^\alpha_{dec} \approx \langle \top ? \rangle \mathtt{x}^\alpha_{dec}.$$

- Property (2) is straightforward to express in LRV.
- We express property (3) by first determining which is the bit i that must be flipped for the increment of the counter pointer of the chain α.

$$\mathit{flip}^\alpha_i \quad = \quad \neg \mathit{bit}^\alpha_i \ \wedge \ \bigwedge_{j>i} \mathit{bit}^\alpha_j$$

Then by making sure that all the bits before i are preserved.

$$\mathit{copy}^\alpha_i \quad = \quad \bigwedge_{j<i} (\mathit{bit}^\alpha_j \Leftrightarrow \mathsf{X}(\mathit{bit}^\alpha_j))$$

And by making every bit greater or equal to i be a zero.

$$\mathit{zero}^\alpha_i \quad = \quad \bigwedge_{j>i} \mathsf{X}(\neg \mathit{bit}^\alpha_j)$$

And finally by swapping bit i.

$$\mathit{swap}^\alpha_i \quad = \quad \mathsf{X}\ \mathit{bit}^\alpha_i$$

Hence, the property to check is, for every $\alpha \in [1, n]$,

$$\bigvee_{(q, \mathrm{next}(\alpha), q') \in \delta} \langle (q, \mathrm{next}(\alpha), q') \rangle \quad \Rightarrow \quad \bigwedge_{i \in [1,n]} \big(\mathit{flip}^\alpha_i \quad \Rightarrow \quad \mathit{copy}^\alpha_i \ \wedge \ \mathit{zero}^\alpha_i \ \wedge \ \mathit{swap}^\alpha_i \big).$$

The formula expressing the property for decrements of counter pointers ($\mathrm{prev}(\alpha)$) is analogous.

- Finally, we express property (4) by testing, for every α-incrementing position,

$$\mathtt{x}^\alpha_{inc} \approx \langle \alpha\text{-}\mathit{dec}? \rangle \mathtt{x}^\alpha_{dec} \qquad \text{where} \quad \alpha\text{-}\mathit{dec} \ = \bigvee_{(q, \mathrm{dec}(\alpha), q') \in \delta} \langle (q, \mathrm{dec}(\alpha), q') \rangle$$

and for every $i \in [1, f(\alpha)]$,

$$\mathit{bit}^\alpha_i \ \Rightarrow \ \mathtt{x}^\alpha_{inc} \approx \langle \mathit{bit}^\alpha_i ? \rangle \mathtt{x}^\alpha_{dec} \quad \wedge \quad \neg \mathit{bit}^\alpha_i \ \Rightarrow \ \mathtt{x}^\alpha_{inc} \approx \langle \neg \mathit{bit}^\alpha_i ? \rangle \mathtt{x}^\alpha_{dec}.$$

If the α-increment has some data value d at variable $\mathtt{x}^\alpha_{inc}$, there must be only one future position j where $\mathtt{x}^\alpha_{dec}$ carries the data value d —since every $\mathtt{x}^\alpha_{dec}$ has a different value, by (1). For this reason, both positions (the α-increment and the α-decrement) operate on the same counter, and thus the formula faithfully expresses property (4).

■

As a corollary of Claims 31 and 32 we obtain a polynomial-time reduction from Gainy(1) into the satisfiability problem for LRV(X, F). ■

G. Proof of Proposition 16

Proof: (I) The developments from Section IV-A apply to the infinite case and since $\text{PLRV}_\omega^\top$ is shown decidable in [6], we get decidability of the satisfiability problem for PLRV_ω.
(II) First, note that LRV_ω is 2EXPSPACE-hard. Indeed, there is a simple logarithmic-space reduction from the satisfiability problem for LRV into the satisfiability problem for LRV_ω, which can be performed as for standard LTL. Indeed, it is sufficient to introduce two new variables $\mathtt{x}_{new}$ and $\mathtt{y}_{new}$, to state that $\mathtt{x}_{new} \approx \mathtt{y}_{new}$ is true at a finite prefix of the model (herein $\mathtt{x}_{new} \approx \mathtt{y}_{new}$ plays the role of a new propositional variable) and to relativize all the temporal operators and obligations to positions on which $\mathtt{x}_{new} \approx \mathtt{y}_{new}$ holds true.

Concerning the complexity upper bound, from Section IV-A, we can conclude that there is a polynomial-time reduction from LRV_ω to $\text{LRV}_\omega^\top$. The satisfiability problem for $\text{LRV}_\omega^\top$ is shown decidable in [6] and we can adapt developments from Section IV-B to get also an 2EXPSPACE upper bound for $\text{LRV}_\omega^\top$. This is the purpose of the rest of the proof.

By analyzing the constructions from [6, Section 7], one can show that ϕ built over the variable $\{\mathtt{x}_1, \ldots, \mathtt{x}_k\}$ is $\text{LRV}_\omega^\top$ satisfiable iff there are $Z \subseteq \mathcal{P}^+(\{\mathtt{x}_1, \ldots, \mathtt{x}_k\})$, a VASS $\mathcal{A}_\phi^Z = \langle Q, Z, \delta \rangle$ along with sets $Q_0, Q_f \subseteq Q$ of *initial* and *final* states and a Büchi automaton $\mathcal{B}^Z$ such that:

1) $\mathcal{A}_\phi^Z$ is the restriction of $\mathcal{A}_\phi$ defined in Section IV-B and $\langle q_0, \mathbf{0} \rangle \xrightarrow{*} \langle q_f, \mathbf{0} \rangle$ in $\mathcal{A}_\phi^Z$ for some $q_0 \in Q_0$ and $q_f \in Q_f$.
2) $\mathcal{B}^Z$ accepts a non-empty language.

By similar arguments from Section IV-B, existence of a run $\langle q_0, \mathbf{0} \rangle \xrightarrow{*} \langle q_f, \mathbf{0} \rangle$ can be checked in 2EXPSPACE. Observe that in the construction in [6, Section 7], counters in $\mathcal{P}^+(\{\mathtt{x}_1, \ldots, \mathtt{x}_k\}) \setminus Z$ are also updated in $\mathcal{A}_\phi^Z$ (providing the VASS $\mathcal{A}_\phi$) but one can show that this is not needed because of *optional decrement* condition and because $\mathcal{B}^Z$ is precisely designed to take care of the future obligations in the infinite related to the counters in $\mathcal{P}^+(\{\mathtt{x}_1, \ldots, \mathtt{x}_k\}) \setminus Z$. $\mathcal{B}^Z$ can be built in exponential time and it is of exponential size in the size of ϕ. Hence, non-emptiness of $\mathcal{B}^Z$ can be checked in EXPSPACE. Finally, the number of possible Z is only at most double exponential in the size of ϕ, which allows to get a nondeterministic algorithm in 2EXPSPACE and provides a 2EXPSPACE upper bound by Savitch's Theorem. ∎

H. Proof of Theorem 17

Proof: 2EXPSPACE-hardness is inherited from LRV. In order to establish the complexity upper bound, first note that LTL extended with a fixed finite set of MSO-definable temporal operators preserves the nice properties about LTL (see e.g. [13]):

- Model-checking and satisfiability problems are PSPACE-complete.
- Given a formula ϕ from such an extension, one can build a Büchi automaton $\mathcal{A}_\phi$ accepting exactly the models for ϕ and the size of $\mathcal{A}_\phi$ is in $\mathcal{O}(2^{p(|\phi|)})$ for some polynomial $p(\cdot)$ (depending on the finite set of MSO-definable operators).

All the results hold because the set of MSO-definable operators is finite and fixed, otherwise the complexity for satisfiability and the size of the Büchi automata are of non-elementary magnitude in the worst case. Moreover, this holds for finite and infinite models.

In order to obtain the 2EXPSPACE, the following properties are now sufficient:

1) Following developments from Section IV-A, it is straightforward to show that there is a logarithmic-space reduction from the satisfiability problem for $\text{LRV} + \{\oplus_1, \ldots, \oplus_N\}$ into the satisfiability problem for $\text{LRV}^\top + \{\oplus_1, \ldots, \oplus_N\}$.
2) By combining [13] and [6, Theorem 4], a formula ϕ in $\text{LRV}^\top + \{\oplus_1, \ldots, \oplus_N\}$ is satisfiable iff $\langle q_0, \mathbf{0} \rangle \xrightarrow{*} \langle q_f, \mathbf{0} \rangle$ for some $q_0 \in Q_0$ and $q_f \in Q_f$ in some VASS $\mathcal{A}_\phi$ such that the number of states in exponential in $|\phi|$. The relatively small size for $\mathcal{A}_\phi$ is due to the fact that $\mathcal{A}_\phi$ is built as the product of a VASS checking obligations (of exponential size in the number of variables and in the size of the local equalities) and of a finite-state automaton accepting the symbolic models of ϕ of exponential size thanks to [13] (see also more details in Section IV-B) .

By using arguments from Section IV-B, we can then reduce existence of a run $\langle q_0, \mathbf{0} \rangle \xrightarrow{*} \langle q_f, \mathbf{0} \rangle$ to an instance of the control state reachability problem in some VASS of linear size in the size of $\mathcal{A}_\phi$, whence the 2EXPSPACE upper bound. ∎

I. Proof of Theorem 18

Proof: Again, 2EXPSPACE-hardness is inherited from LRV. In order to get the 2EXPSPACE, we use arguments from Proposition 19. Indeed, the decidability proof from [6, Theorem 4] can be adapted to LRV + Now. The only difference is that the finite-state automaton is of double exponential size, see details below. Despite this exponential blow-up, the 2EXPSPACE upper bound can be preserved.

Let ϕ be a formula with k variables. There exist a VASS $\mathcal{A}_{\mathrm{dec}} = \langle Q, C, \delta\rangle$ and $Q_0, Q_f \subseteq Q$ such that ϕ is satisfiable iff there are $q_f \in Q_f$ and $q_0 \in Q_0$ such that $\langle q_f, \mathbf{0}\rangle \xrightarrow{*} \langle q_0, \mathbf{v}\rangle$ for some counter valuation $\mathbf{v}$. Note that the number of counters in $\mathcal{A}_{\mathrm{dec}}$ is bounded by 2^k, $\mathrm{card}(Q)$ is double exponential in $|\phi|$ and the maximal value for an update in a transition of $\mathcal{A}_{\mathrm{dec}}$ is k. Indeed, a formula from Past LTL+Now is equivalent to a Büchi automaton of double exponential size in its size [20]. Moreover, deciding whether a state in Q belongs to Q_0 [resp. Q_f] can be checked in exponential space in $|\phi|$ and δ can be decided in exponential space too. By using [28] (see also [2]), we can easily conclude that $\langle q_f, \mathbf{0}\rangle \xrightarrow{*} \langle q_0, \mathbf{v}\rangle$ for some counter valuation $\mathbf{v}$ iff $\langle q_f, \mathbf{0}\rangle \xrightarrow{*} \langle q_0, \mathbf{v}'\rangle$ for some $\mathbf{v}'$ such that the length of the run is bounded by $p(|\mathcal{A}_{\mathrm{dec}}| + \mathtt{max}(\mathcal{A}_{\mathrm{dec}}))^{\alpha(k)}$ where $p(\cdot)$ is a polynomial, α is a map of double exponential growth and $\mathtt{max}(\mathcal{A}_{\mathrm{dec}})$ denotes the maximal absolute value in an update (bounded by k presently). In order to take advantage of the results on VAS, we use the translation from VASS to VAS introduced in [15]: if a VASS has N_1 control states, the maximal absolute value in an update is N_2 and it has N_3 counters, then we can build a VAS (being able to preserve coverability properties) such that it has $N_3 + 3$ counters, the maximal absolute value in an update is $\max(N_2, N_1^2)$. From $\mathcal{A}_{\mathrm{dec}}$, we can indeed build an equivalent VAS with a number of counters bounded by $2^k + 3$ and with a maximal absolute value in an an update at most double exponential in the size of $|\phi|$. So, the length of the run is at most triple exponential in $|\phi|$. Consequently, the satisfiability problem for LRV + Now is in 2EXPSPACE. ∎

J. Proof of Proposition 19

Proof of Proposition 19: PSPACE-hardness is due the fact that LTL with a single propositional variable is PSPACE-hard, which can be easily simulated with the atomic formula $\mathtt{x} \approx \mathtt{y}$. Let $k \geq 1$ be some fixed value and $\phi \in \mathrm{LRV}_k^{\top}$. In the proof of Theorem 9, we have seen that there exist a VASS $\mathcal{A}_{\mathrm{dec}} = \langle Q, C, \delta\rangle$ and $Q_0, Q_f \subseteq Q$ such that ϕ is satisfiable iff there are $q_f \in Q_f$ and $q_0 \in Q_0$ such that $\langle q_f, \mathbf{0}\rangle \xrightarrow{*} \langle q_0, \mathbf{v}\rangle$ for some counter valuation $\mathbf{v}$. Note that the number of counters in $\mathcal{A}_{\mathrm{dec}}$ is bounded by 2^k, $\mathrm{card}(Q)$ is exponential in $|\phi|$ and the maximal value for an update in a transition of $\mathcal{A}_{\mathrm{dec}}$ is k. Moreover, deciding whether a state in Q belongs to Q_0 [resp. Q_f] can be checked in polynomial space in $|\phi|$ and δ can be decided in polynomial space too. By using [28] (see also [3]), we can easily conclude that $\langle q_f, \mathbf{0}\rangle \xrightarrow{*} \langle q_0, \mathbf{v}\rangle$ for some counter valuation $\mathbf{v}$ iff $\langle q_f, \mathbf{0}\rangle \xrightarrow{*} \langle q_0, \mathbf{v}'\rangle$ for some $\mathbf{v}'$ such that the length of the run is bounded by $p(|\mathcal{A}_{\mathrm{dec}}| + \mathtt{max}(\mathcal{A}_{\mathrm{dec}}))^{\alpha(k)}$ where $p(\cdot)$ is a polynomial, α is a map of double exponential growth and $\mathtt{max}(\mathcal{A}_{\mathrm{dec}})$ denotes the maximal absolute value in an update (bounded by k presently). Since k is fixed, the length of the run is at most exponential in $|\phi|$. Consequently, the following polynomial-space nondeterministic algorithm allows to check whether ϕ is satisfiable. Guess $q_f \in Q_f$, $q_0 \in Q_0$ and guess on-the-fly a run of length at most $p(|\mathcal{A}_{\mathrm{dec}}| + \mathtt{max}(\mathcal{A}_{\mathrm{dec}}))^{\alpha(k)}$ from $\langle q_f, \mathbf{0}\rangle$ to some $\langle q_0, \mathbf{v}'\rangle$. Note that counter valuations can be represented in polynomial space too. By Savitch's Theorem, we conclude that the satisfiability problem for $\mathrm{LRV}_k^{\top}$ is in PSPACE. ∎

K. Proof of Lemma 20

Proof of Lemma 20: The idea of the coding is the following. Suppose we have a formula $\varphi \in \mathrm{LRV}^{\top}$ using k variables $\mathtt{x}_1, \ldots, \mathtt{x}_k$ so that $\sigma \models \varphi$.

We will encode σ in a model σ_φ that encodes in only one variable, say $\mathtt{x}$ the whole model of σ restricted to $\mathtt{x}_1, \ldots, \mathtt{x}_k$. To this end, σ_φ is divided into N segments $s_1 \cdots s_N$ of equal length, where $N = |\sigma|$. A special fresh data value is used as a special constant. Suppose that d is a data value that is not in σ. Then, each segment s_i has length $k' = 2k + 1$, and is defined as the data values "$d\ d_1\ d\ d_2\ \ldots d\ d_k\ d$", where $d_j = \sigma(i)(\mathtt{x}_j)$. Figure 7 contains an example for $k = 3$ and $N = 3$. In fact, we can force what the model has this shape with LRV_1. Note that with this coding, we can tell that we are between two segments if there are two consecutive equal data values. In fact, we are at a position corresponding to $\mathtt{x}_i$ (for $i \in [1, k]$) inside a segment if we are standing at the $2i$-th element of

[2] S. Demri, M. Jurdziński, O. Lachish, and R. Lazić. The covering and boundedness problems for branching VAS. *JCSS*, 79(1):23–38, 2013.
[3] S. Demri, M. Jurdziński, O. Lachish, and R. Lazić. The covering and boundedness problems for branching VAS. *JCSS*, 79(1):23–38, 2013.

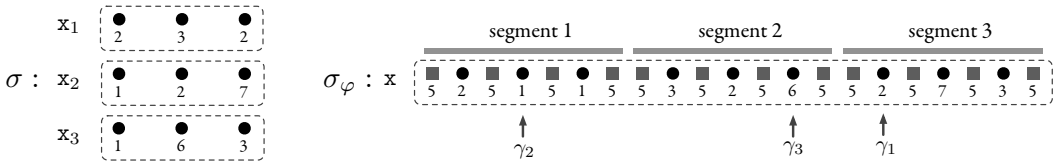


Fig. 7. Example of the encoding for $k = 3$ and $N = 3$.

a segment, and we can test this with the formula

$$\gamma_i \quad = \quad \mathsf{X}^{k'-2i}\mathtt{x} \approx \mathsf{X}^{k'-2i+1}\mathtt{x} \quad \vee \quad (\mathsf{X}^{k'-2i}\top \wedge \neg \mathsf{X}^{k'-2i+1}\top).$$

Using these formulas γ_i, we can translate any test $\mathtt{x}_i \approx \langle \top ? \rangle \mathtt{x}_j$ into a formula that

1) moves to the position $2i$ of the segment (the one corresponding to the $\mathtt{x}_i$ data value),
2) tests $\mathtt{x} \approx \langle \gamma_j ? \rangle \mathtt{x}$.

We can do this similarly with all formulas.

Let us now explain in more detail how to do the translation, and why it works.

Consider the following property of a given position i multiple of k' of a model σ_φ:

- for every $0 \leq j \leq k'-1$, $\sigma_\varphi(i+j)(\mathtt{x}) = \sigma_\varphi(i)(\mathtt{x})$ if, and only if, j is even, and
- either $i + k' \geq |\sigma_\varphi|$ or $\sigma_\varphi(i+k') = \sigma_\varphi(i)$.

This property can be easily expressed with a formula

$$\textit{segment-}k' \quad = \quad \bigwedge_{\substack{0 \leq j \leq k'-1, \\ j \text{ is even}}} \mathtt{x} \approx \mathsf{X}^j \mathtt{x} \quad \wedge \quad \bigwedge_{\substack{0 \leq j \leq k'-1, \\ j \text{ is odd}}} \mathtt{x} \not\approx \mathsf{X}^j \mathtt{x} \quad \wedge \quad (\neg \mathsf{X}^{k'} \top \vee \mathtt{x} \approx \mathsf{X}^{k'} \mathtt{x}).$$

Note that this formula tests that we are standing at the beginning of a segment in particular. It is now straightforward to produce a LRV_1 formula that tests

- $\sigma_\varphi, 0 \models \textit{segment-}k'$
- for every position i so that $\sigma_\varphi(i)(\mathtt{x}) = \sigma_\varphi(i+1)(\mathtt{x})$, we have $\sigma, i+1 \models \textit{segment-}k'$.

Let us call *many-segments* the formula expressing such a property. Note that the property implies that σ_φ is a succession of segments, as the one of Figure 7.

We give now the translation of a formula φ of $\mathrm{LRV}^\top$ with k variables into a formula φ' of LRV_1 with 1 variable.

$$\begin{aligned}
tr(\mathsf{X}\psi) &= \overbrace{\mathsf{X} \cdots \mathsf{X}}^{k' \text{ times}} tr(\psi) \\
tr(\mathsf{X}^{-1}\psi) &= \overbrace{\mathsf{X}^{-1} \cdots \mathsf{X}^{-1}}^{k' \text{ times}} tr(\psi) \\
tr(\mathsf{F}\psi) &= \mathsf{F}(\textit{segment-}k' \wedge tr(\psi)) \\
tr(\psi \mathsf{U} \gamma) &= (\textit{segment-}k' \Rightarrow tr(\psi)) \ \mathsf{U} \ (\textit{segment-}k' \Rightarrow tr(\gamma)) \\
tr(\psi \mathsf{S} \gamma) &= (\textit{segment-}k' \Rightarrow tr(\psi)) \ \mathsf{S} \ (\textit{segment-}k' \Rightarrow tr(\gamma)) \\
tr(\mathtt{x}_i \approx \mathsf{X}^\ell \mathtt{x}_j) &= \mathsf{X}^{2i-1}\mathtt{x} \approx \mathsf{X}^{\ell \cdot k' + 2j - 1}\mathtt{x} \qquad \text{(and similarly for } \not\approx)
\end{aligned}$$

Now we have to translate $\mathtt{x}_i \approx \langle \top ? \rangle \mathtt{x}_j$. This would be translated in our encoding by saying that the i-th position of the current segment is equal to the j-th position of a future segment. Note that the following formula

$$\xi_{i,j} \ = \ \mathsf{X}^{2i-1}(\mathtt{x} \approx \langle \gamma_j ? \rangle \mathtt{x})$$

does not exactly encode this property. For example, consider that we would like to test $\mathtt{x}_2 \approx \langle \top ? \rangle \mathtt{x}_3$ at the first element of the model σ_φ depicted in Figure 7. Although ξ holds, the property is not true, there is no *future* segment with the data value 1 in the position encoding $\mathtt{x}_3$. In fact, the formula ξ encodes correctly the property only when $\mathtt{x}_i \not\approx \mathtt{x}_j$. However, this is not a problem since when $\mathtt{x}_i \approx \mathtt{x}_j$ the formula $\mathtt{x}_i \approx \langle \top ? \rangle \mathtt{x}_j$ is equivalent to $\mathtt{x}_j \approx \langle \top ? \rangle \mathtt{x}_j$. We can then translate the formula as follows.

$$tr(\mathtt{x}_i \approx \langle \top ? \rangle \mathtt{x}_j) \ = \ (tr(\mathtt{x}_i \approx \mathtt{x}_j) \wedge \xi_{j,j}) \quad \vee \quad (tr(\mathtt{x}_i \not\approx \mathtt{x}_j) \wedge \xi_{i,j})$$

We then define $\varphi' = \textit{many-segments} \wedge tr(\varphi)$.

Claim 33. φ is satisfiable if, and only if, φ' is satisfiable.

Proof: In fact, if $\sigma \models \varphi$, then by the discussion above, $\sigma_\varphi \models \varphi'$. If, on the other hand, $\sigma' \models \varphi'$ for some σ', then since $\sigma' \models \textit{many-segments}$ it has to be a succession of segments of size $2k+1$, and we can recover a model σ of size $|\sigma'|/2k+1$ where $\sigma(i)(\mathtt{x}_j)$ is the data value of the $2j$-th position of the i-th segment of σ'. In this model, we have that $\sigma \models \varphi$. ∎

This coding can also be extended with past obligations in a straightforward way,

$$\begin{aligned} tr(\mathtt{x}_i \approx \langle \phi ? \rangle^{-1} \mathtt{x}_j) &= (tr(\mathtt{x}_i \approx \mathtt{x}_j) \wedge \xi_{j,j}^{-1}) \quad \vee \quad (tr(\mathtt{x}_i \not\approx \mathtt{x}_j) \wedge \xi_{i,j}^{-1}) \quad \text{where} \\ \xi_{i,j}^{-1} &= \mathsf{X}^{2i-1}(\mathtt{x} \approx \langle \gamma_j ? \rangle^{-1} \mathtt{x}). \end{aligned}$$

Therefore, there is also a reduction from PLRV$^\top$ into PLRV$_1$. ∎

L. Proof of Theorem 23

Let us introduce the problem below as a variant of PCP:

PROBLEM:	Modified Directed Post's Correspondence Problem (MPCP^{dir})
INPUT:	A finite alphabet Σ, $n \in \mathbb{N}$, $u_1, \ldots, u_n, v_1, \ldots, v_n \in \Sigma^+$, $\lvert u_1\rvert, \lvert u_2\rvert \leq 2$ and $\lvert v_1\rvert, \lvert v_2\rvert \geq 3$.
QUESTION:	Are there indices $1 \leq i_1, \ldots, i_m \leq n$ so that $u_{i_1} \cdots u_{i_m} = v_{i_1} \cdots v_{i_m}$, $i_1 \in \{1, 2\}$, $\lvert u_{i_1} \cdots u_{i_m}\rvert$ is even and for every $\lvert u_{i_1}\rvert < j < \lvert u_{i_1} \cdots u_{i_m}\rvert$, if the j^{th} position of $v_{i_1} \cdots v_{i_m}$ occurs in v_{i_k} for some k, then the j^{th} position of $u_{i_1} \cdots u_{i_m}$ occurs in $u_{i_{k'}}$ for some $k' > k$?

Lemma 34. The MPCP^{dir} problem is undecidable.

This is a corollary of the undecidability proof for PCP as done in [14]. By reducing MPCP^{dir} to satisfiability for LRV$_{vec}$, we get undecidability.

Proof: In [14], the halting problem for Turing machines is first reduced to a variation of PCP called modified PCP. In modified PCP, a requirement is that the solution should begin with the pair u_1, v_1 (this requirement can be easily eliminated by encoding it into the standard PCP, but here we find it convenient to work in the presence of a generalization of this requirement). To ensure that there is a solution of even length whenever there is a solution, we let $u_2 = \$u_1$ and $v_2 = \$v_1$, where \$ is a new symbol. Now $u_1 u_{i_2} \cdots u_{i_m} = v_1 v_{i_2} \cdots v_{i_m}$ is a solution iff $u_2 u_{i_2} \cdots u_{i_m} = v_2 v_{i_2} \cdots v_{i_m}$ is a solution. In modified PCP, $|u_1| = 1$ and $|v_1| \geq 3$. Hence, $|u_2| = 2$ and $|v_2| \geq 3$. The resulting set of strings $u_1, \ldots, u_n, v_1, \ldots, v_n$ make up our instance of MPCP^{dir}.

In the encoding of the halting problem from the cited work, $|u_i| \leq 3$ for any i between 1 and n (this continues to hold even after we add \$ to the first pair as above). A close examination of the proof in the cited work reveals that if the modified PCP instance $u_1, \ldots, u_n, v_1, \ldots, v_n \in \Sigma^*$ has a solution $u_{i_1} \cdots u_{i_m} = v_{i_1} \cdots v_{i_m}$, then for every $|u_{i_1}| < j < |u_{i_1} \cdots u_{i_m}|$, if the j^{th} position of $v_{i_1} \cdots v_{i_m}$ occurs in v_{i_k} for some k, then the j^{th} position of $u_{i_1} \cdots u_{i_m}$ occurs in $u_{i_{k'}}$ for some $k' > k$ (we call this the directedness property). In short, the reason for this is that for any $k \in \mathbb{N}$, if $v_{i_1} \cdots v_{i_k}$ encodes the first $\ell + 1$ consecutive configurations of a Turing machine, then $u_{i_1} \cdots u_{i_k}$ encodes the first ℓ configurations. Hence, for any letter in $v_{i_{k+1}}$ (which starts encoding $(\ell+2)^{\text{th}}$ configuration), the corresponding letter cannot occur in $u_{i_1} \cdots u_{i_k+1}$ (unless the single string $u_{i_{k+1}}$ encodes the entire $(\ell+1)^{\text{st}}$ configuration and starts encoding $(\ell+2)^{\text{th}}$ configuration; this however is not possible since there are at most 3 letters in $u_{i_{k+1}}$ and encoding a configuration requires at least 4 letters). After the last configuration has been encoded, the length of $u_{i_1} \cdots u_{i_k}$ starts catching up with the length of $v_{i_1} \cdots v_{i_k}$ with the help of pairs (u, v) where $|u| = 2$ and $|v| = 1$. However, as long as the lengths do not catch up, the directedness property continues to hold. As soon as the lengths do catch up, it is a solution to the PCP. Only the positions of u_{i_1} and the last position of the solution violate the directedness property. ∎

Given an instance *pcp* of MPCP^{dir}, we construct a LRV$_{vec}(\mathsf{X}, \mathsf{U})$ formula ϕ_{pcp} such that *pcp* has a solution iff ϕ_{pcp} is satisfiable. To do so, we adapt a proof technique from [1], [17] but we need to provide substantial changes in order to fit our logic. Moreover, none of the results in [1], [17] allow to derive our main undecidability result since we use neither past-time temporal operators nor past obligations of the form $\langle \mathtt{x}, \mathtt{x}' \rangle \approx \langle \varphi ? \rangle^{-1} \langle \mathtt{y}, \mathtt{y}' \rangle$

Let $\bar{\Sigma} = \{\bar{a} \mid a \in \Sigma\}$ be a disjoint copy of Σ. For convenience, we assume that each position of a LRV model is labelled by a letter from $\Sigma \cup \bar{\Sigma}$ (these labels can be easily encoded as equivalence classes

of some extra variables). For such a model σ, let σ_Σ (resp. $\sigma_{\bar\Sigma}$) be the model obtained by restricting σ to positions labelled by Σ (resp. $\bar\Sigma$). If $u_{i_1}\cdots u_{i_m} = v_{i_1}\cdots v_{i_m}$ is a solution to *pcp*, the idea is to construct a LRV model whose projection to $\Sigma \cup \bar\Sigma$ is $u_{i_1}\overline{v_{i_1}}\cdots u_{i_m}\overline{v_{i_m}}$. To check that such a model σ actually represents a solution, we will write $\mathrm{LRV}_{vec}(\mathsf{X},\mathsf{U})$ formulas to say that "for all j, if the j^{th} position of σ_Σ is labelled by a, then the j^{th} position of $\sigma_{\bar\Sigma}$ is labelled by $\bar a$".

The main difficulty is to get a handle on the j^{th} position of σ_Σ. The difficulty arises since the LRV model σ is an interleaving of σ_Σ and $\sigma_{\bar\Sigma}$. This is handled by using two variables $\mathtt{x}$ and $\mathtt{y}$. Positions 1 and 2 of σ_Σ will have the same value of $\mathtt{x}$, positions 2 and 3 will have the same value of $\mathtt{y}$, positions 3 and 4 will have the same value of $\mathtt{x}$ and so on. Generalising, odd positions of σ_Σ will have the same value of $\mathtt{x}$ as in the next position of σ_Σ and the same value of $\mathtt{y}$ as in the previous position of σ_Σ. Even positions of σ_Σ will have the same value of $\mathtt{x}$ as in the previous position of σ_Σ and the same value of $\mathtt{y}$ as in the next position of σ_Σ. To easily identify odd and even positions, an additional label is introduced at each position, which can be O or E. The sequence σ_Σ looks as follows.

$$\begin{bmatrix} \mathtt{x} \\ \text{Odd/Even} \\ \text{Letter} \\ \mathtt{y} \end{bmatrix} : \begin{pmatrix} d_1 \\ O_{in} \\ a_1 \\ d'_1 \end{pmatrix}\begin{pmatrix} d_1 \\ E \\ a_2 \\ d'_2 \end{pmatrix}\begin{pmatrix} d_3 \\ O \\ a_3 \\ d'_2 \end{pmatrix}\begin{pmatrix} d_3 \\ E \\ a_4 \\ d'_4 \end{pmatrix}\cdots\begin{pmatrix} d_{m-1} \\ O \\ a_{m-1} \\ d'_{m-2} \end{pmatrix}\begin{pmatrix} d_{m-1} \\ E_{fi} \\ a_m \\ d'_m \end{pmatrix} \qquad (\star)$$

We assume that the atomic formula a ($a \in \Sigma$) is true at some position j of a model σ iff the position j is labelled by the letter a. We denote $\bigvee_{a\in\Sigma} a$ by Σ. For a word $u \in \Sigma^*$, we denote by ϕ^i_u the LRV formula that ensures that starting from i positions to the right of the current position, the next $|u|$ positions are labelled by the respective letters of u (e.g., $\phi^3_{abb} \stackrel{\text{def}}{\equiv} \mathsf{X}^3 a \wedge \mathsf{X}^4 b \wedge \mathsf{X}^5 b$). We enforce a model of the form ($\star$) above through the following formulas.

(1) The projection of the labeling of σ on to $\Sigma \cup \bar\Sigma$ is in $(u_1\overline{v_1} + u_2\overline{v_2})\{u_i\bar v_i \mid i \in [n]\}^*$: to facilitate writing this condition in LRV, we introduce two new variables $\mathtt{z}^1_b, \mathtt{z}^2_b$ such that at any position, they have the same value only if that position is not the starting point of some pair $u_i\overline{v_i}$.

$$\mathtt{z}^1_b \not\approx \mathtt{z}^2_b \wedge \bigvee_{i\in\{1,2\}} (\phi^0_{u_i} \wedge \phi^{|u_i|+1}_{\overline{v_i}} \wedge (\mathsf{X}^{|u_iv_i|+1}\top \Rightarrow \mathsf{X}^{|u_iv_i|+1}\mathtt{z}^1_b \not\approx \mathtt{z}^2_b))$$
$$\wedge\, \mathsf{G}\left(\mathtt{z}^1_b \approx \mathtt{z}^2_b \vee \bigvee_{i\in[1,n]} (\phi^0_{u_i} \wedge \phi^{|u_i|+1}_{\overline{v_i}}) \wedge (\mathsf{X}^{|u_iv_i|+1}\top \Rightarrow \mathsf{X}^{|u_iv_i|+1}\mathtt{z}^1_b \not\approx \mathtt{z}^2_b)\right)$$

(2) a) For every data value d, there are at most two positions i, j in σ_Σ with $\sigma_\Sigma(i)(\mathtt{x}) = \sigma_\Sigma(j)(\mathtt{x}) = d$ or $\sigma_\Sigma(i)(\mathtt{y}) = \sigma_\Sigma(j)(\mathtt{y}) = d$.

$$\mathsf{G}\,(\Sigma \Rightarrow (\neg\mathtt{x} \approx \langle\Sigma?\rangle\mathtt{x}) \vee (\mathtt{x} \approx \langle\Sigma \wedge (\neg\mathtt{x} \approx \langle\Sigma?\rangle\mathtt{x}?\rangle\mathtt{x})))$$
$$\wedge\mathsf{G}\,(\Sigma \Rightarrow (\neg\mathtt{y} \approx \langle\Sigma?\rangle\mathtt{y}) \vee (\mathtt{y} \approx \langle\Sigma \wedge (\neg\mathtt{y} \approx \langle\Sigma?\rangle\mathtt{y}?\rangle\mathtt{y})))$$

The same condition for $\sigma_{\bar\Sigma}$, enforced with a formula similar to the one above.

b) σ_Σ projected onto $\{O, O_{in}, E, E_{fi}\}$ is in $O_{in}(EO)^*E_{fi}$.

$$O_{in} \wedge \mathsf{G}\left((O \vee O_{in}) \Rightarrow \bar\Sigma\mathsf{U}(\Sigma \wedge (E \vee E_{fi}))\right) \wedge \mathsf{G}\left(E \Rightarrow \bar\Sigma\mathsf{U}(\Sigma \wedge O)\right)$$
$$\wedge\, \mathsf{G}(E_{fi} \Rightarrow \neg\mathsf{X}\mathsf{F}\Sigma)$$

$\sigma_{\bar\Sigma}$ projected onto $\{\bar O, \overline{O_{in}}, \bar E, \overline{E_{fi}}\}$ is in $\overline{O_{in}}(\bar E\bar O)^*\overline{E_{fi}}$.

$$\Sigma\mathsf{U}\overline{O_{in}} \wedge \mathsf{G}\left((\bar O \vee \overline{O_{in}}) \Rightarrow \Sigma\mathsf{U}(\bar\Sigma \wedge (\bar E \vee \overline{E_{fi}}))\right) \wedge \mathsf{G}\left(\bar E \Rightarrow \Sigma\mathsf{U}(\bar\Sigma \wedge \bar O)\right)$$
$$\wedge\, \mathsf{G}(\overline{E_{fi}} \Rightarrow \neg\mathsf{X}\top)$$

c) For every position i of σ_Σ labelled by O or O_{in}, there exists a future position $j > i$ of σ_Σ labelled by E or E_{fi} so that $\sigma_\Sigma(i)(\mathtt{x}) = \sigma_\Sigma(j)(\mathtt{x})$.

$$\mathsf{G}\,((O \vee O_{in}) \Rightarrow \mathtt{x} \approx \langle\Sigma \wedge (E \vee E_{fi})?\rangle\mathtt{x})$$

For every position i of $\sigma_{\bar\Sigma}$ labelled by $\bar O$ or $\overline{O_{in}}$, there exists a future position $j > i$ of $\sigma_{\bar\Sigma}$ labelled by $\bar E$ or $\overline{E_{fi}}$ so that $\sigma_{\bar\Sigma}(i)(\mathtt{x}) = \sigma_{\bar\Sigma}(j)(\mathtt{x})$ (enforced with a formula similar to the one above).

d) For every position i of σ_Σ labelled by E, there exists a future position $j > i$ of σ_Σ labelled by O so that $\sigma_\Sigma(i)(\mathtt{y}) = \sigma_\Sigma(j)(\mathtt{y})$.

$$\mathsf{G}\,(E \Rightarrow \mathtt{y} \approx \langle \Sigma \wedge O? \rangle \mathtt{y})$$

For every position i of $\sigma_{\bar\Sigma}$ labelled by $\bar E$, there exists a future position $j > i$ of $\sigma_{\bar\Sigma}$ labelled by $\bar O$ so that $\sigma_{\bar\Sigma}(i)(\mathtt{y}) = \sigma_{\bar\Sigma}(j)(\mathtt{y})$ (enforced with a formula similar to the one above).

(3) For any position i of u_{i_1}, the corresponding position j in $\overline{v_{i_1}} \cdots \overline{v_{i_m}}$ (which always happens to be in v_{i_1} MPCP^{dir}, since $|u_{i_1}| \leq 2$ and $|v_{i_1}| \geq 3$) should satisfy $\sigma(i)(\mathtt{x}) = \sigma(j)(\mathtt{x})$ and $\sigma(i)(\mathtt{y}) = \sigma(j)(\mathtt{y})$. In addition, position i is labelled with $a \in \Sigma$ iff position j is labelled with $\bar a \in \bar\Sigma$.

$$\left(\bigvee_{a\in\Sigma} (O_{in} \wedge a) \Rightarrow \langle \mathtt{x}, \mathtt{y}\rangle \approx \langle \overline{O_{in}} \wedge \overline{a}? \rangle \langle \mathtt{x}, \mathtt{y}\rangle\right)$$
$$\wedge\, \mathsf{X}\Sigma \Rightarrow \bigvee_{a\in\Sigma} \mathsf{X}a \wedge \mathsf{X}^3\overline{a} \wedge \mathsf{X}(\mathtt{x} \approx \mathsf{X}^2\mathtt{x} \wedge \mathtt{y} \approx \mathsf{X}^2\mathtt{y})$$

(4) For any position i of σ with $2|u_{i_1}| < i < |u_{i_1} \cdots u_{i_m}|$, if it is labeled with $\bar a \in \bar\Sigma$, there is a future position $j > i$ labeled with $a \in \Sigma$ such that $\sigma(i)(\mathtt{x}) = \sigma(j)(\mathtt{x})$ and $\sigma(i)(\mathtt{y}) = \sigma(j)(\mathtt{y})$. The following formula assumes that $|u_{i_1}| \leq 2$, as it is in MPCP^{dir}.

$$\mathsf{X}\Sigma \Rightarrow \Sigma\mathsf{U}\left(\mathsf{X}^2\mathsf{G}\bigvee_{\bar a\in\bar\Sigma} \left((\bar a \wedge \neg\overline{E_{fi}}) \Rightarrow (\mathtt{x}, \mathtt{y}) \approx \langle a? \rangle (\mathtt{x}, \mathtt{y})\right)\right) \quad (\mathsf{X}\Sigma \text{ is true when } |u_{i_1}| = 2)$$

$$\wedge\, \mathsf{X}\bar\Sigma \Rightarrow \Sigma\mathsf{U}\left(\mathsf{X}\mathsf{G}\bigvee_{\bar a\in\bar\Sigma} \left((\bar a \wedge \neg\overline{E_{fi}}) \Rightarrow (\mathtt{x}, \mathtt{y}) \approx \langle a? \rangle (\mathtt{x}, \mathtt{y})\right)\right) \quad (\mathsf{X}\bar\Sigma \text{ is true when } |u_{i_1}| = 1)$$

(5) If i and j are the last positions labeled with Σ and $\bar\Sigma$ respectively, then $\sigma(i)(\mathtt{x}) = \sigma(j)(\mathtt{x})$ and $\sigma(i)(\mathtt{y}) = \sigma(j)(\mathtt{y})$. In addition, position i is labelled with $a \in \Sigma$ iff position j is labelled with $\bar a \in \bar\Sigma$.

$$\mathsf{G}\left(\bigvee_{a\in\Sigma} (E_{fi} \wedge a) \Rightarrow \langle \mathtt{x}, \mathtt{y}\rangle \approx \langle \overline{E_{fi}} \wedge \overline{a}? \rangle \langle \mathtt{x}, \mathtt{y}\rangle\right)$$

Given an instance pcp of MPCP^{dir}, the required formula ϕ_{pcp} is the conjunction of all the formulas above.

Lemma 35. Given an instance pcp of MPCP^{dir}, ϕ_{pcp} is satisfiable iff pcp has a solution.

Proof: Suppose $u_{i_1} \cdots u_{i_m} = v_{i_1} \cdots v_{i_m}$ is a solution of pcp. It is routine to check that, with this solution, a model satisfying ϕ_{pcp} can be built.

Now suppose that ϕ_{pcp} has a satisfying model σ. From condition (1), we get a sequence $u_{i_1}\overline{v_{i_1}} \cdots u_{i_m}\overline{v_{i_m}}$. It is left to prove that $u_{i_1} \cdots u_{i_m} = v_{i_1} \cdots v_{i_m}$.

Let σ_Σ (resp. $\sigma_{\bar\Sigma}$) be the model obtained from σ by restricting it to positions labeled with Σ (resp. $\bar\Sigma$). Construct a directed graph whose vertices are the positions of σ_Σ and there is an edge from i to j iff $i < j$ and $\sigma_\Sigma(i)(\mathtt{x}) = \sigma_\Sigma(j)(\mathtt{x})$ or $\sigma_\Sigma(i)(\mathtt{y}) = \sigma_\Sigma(j)(\mathtt{y})$. We claim that the set of edges of this directed graph represents the successor relation induced on σ_Σ by σ. To prove this claim, we first show that all positions have in degree 1 (except the first one, which has in degree 0) and that all positions have out degree 1 (except the last one, which has out degree 0). Indeed, condition (2a) ensures that for any position, both the in degree and out degree are at most 1. From conditions (2b), (2c) and (2d), each position (except the last one) has out degree at least 1. Hence, all positions except the last one have out degree exactly 1. By the definition of the set of edges, the last position has out degree 0. If more than one position has in degree 0, it will force some other position to have in degree more than 1, which is not possible. By the definition of the set of edges, the first position has in degree 0 and hence, all other positions have in degree exactly 1. To finish proving the claim (that the set of edges of our directed graph represents the successor relation induced on σ_Σ by σ), we will now prove that at any position of σ_Σ except the last one, the outgoing edge goes to the successor position in σ_Σ. If this is not the case, let i be the last position where this condition is violated. The outgoing edge from i then goes to some position $j > i + 1$. Since the outgoing edges from each position between $i + 1$ and $j - 1$ go to the respective successors, position j will have in degree 2, which is not possible.

Next we will prove that there cannot be two positions of σ_Σ with the same value for variables x and y. Suppose there were two such positions and the first one is labeled O (the argument for E is similar). If the second position is also labeled O, then by condition (2c), there is at least one position labeled E with the same value for variable x, so there are three positions with the same value for variable x, violating condition (2a). Hence, the second position must be labelled E. Then by condition (2d), there is a position after the second position with the same value for variable y. This implies there are three positions with the same value for variable y, again violating condition (2a). Therefore, there cannot be two positions of σ_Σ with the same value for variables x and y.

Finally, we prove that for every position i of $\sigma_{\bar{\Sigma}}$, if $\overline{a_i}$ is its label, then the unique position in σ_Σ with the same value of x and y is position i of σ_Σ and carries the label a_i. For $1 \leq i \leq |u_{i_1}|$, this follows from condition (3). For $i = |u_{i_1} \cdots u_{i_m}|$, this follows from condition (5). The rest of the proof is by induction on i. The base case is already proved since $|u_{i_1}| \geq 1$. For the induction step, assume the result is true for all positions of $\sigma_{\bar{\Sigma}}$ up to position i. Supppose position i of $\sigma_{\bar{\Sigma}}$ is labeled by $\bar{O}$ (the case of $\bar{E}$ is symmetric). Then by induction hypothesis and (2b), position i of σ_Σ is labeled by O (or O_{in}, if $i = 1$). By condition (2c) and the definition of edges in the directed graph that we built, position $i+1$ of $\sigma_{\bar{\Sigma}}$ (resp. σ_Σ) has same value of x as that of position i. We know from the previous paragraph that there is exactly one position of σ_Σ with the same value of x and y as that of position $i+1$ in $\sigma_{\bar{\Sigma}}$. This position in σ_Σ cannot be i or before due to induction hypothesis. If it is not $i+1$ either, then there will be three positions of σ_Σ with the same value of x, which violates condition (2a). Hence, the position of σ_Σ with the same value of x and y as that of position $i+1$ in $\sigma_{\bar{\Sigma}}$ is indeed $i+1$ and it carries the label a_i by condition (4). ■

As a conclusion, the satisfiability problem for $\mathrm{LRV}_{vec}(\mathsf{X}, \mathsf{U})$ is undecidable.

M. Implications for Logics on Data Words

1) **First Order Logic:**

Proof of Proposition 24: Through a standard translation we can bring any formula of *forward*-EMSO$^2(+1, <, \sim)$ into a formula of the form

$$\varphi = \exists X_1, \ldots, X_n \ \left(\forall x \forall y \ \chi \ \wedge \ \bigwedge_k \forall x \exists y \ (x \leq y \wedge \psi_k) \right)$$

that preserves satisfiability, where χ and all ψ_k's are quantifier-free formulas, and there are no tests for labels. Furthermore, this is a polynomial-time translation. This translation is just the Scott normal form of EMSO$^2(+1, <, \sim)$ [1] adapted to *forward*-EMSO$^2(+1, <, \sim)$, and can be done in the same way.

We now give an exponential-time translation tr : *forward*-EMSO$^2(+1, <, \sim) \to$ LRV. For any formula φ of *forward*-EMSO$^2(+1, <, \sim)$, $tr(\varphi)$ is an equivalent (in the sense of satisfiability) LRV formula, whose satisfiability can be tested in 2EXPSPACE (Corollary 10). This yields an upper bound of 3EXPSPACE for *forward*-EMSO$^2(+1, <, \sim)$.

The translation makes use of: a distinguished variable x that encodes the data values of any data word satisfying φ; variables $\mathtt{x}_0, \ldots, \mathtt{x}_n$ that are used to encode the monadic relations $X_1, \ldots, X_n$; and a variable $\mathtt{x}_{prev}$ whose purpose will be explained later on. We give now the translation. To translate $\forall x \forall y \ \chi$, we first bring the formula to a form

$$\bigwedge_{m \in M} \ \neg \exists x \exists y \ \big(\ x \leq y \ \wedge \ \chi_m \ \wedge \ \chi_m^x \ \wedge \ \chi_m^y \ \big), \qquad (\star)$$

where every χ_m^x (resp. χ_m^y) is a conjunction of (negations of) atoms of monadic relations on x (resp. y); and $\chi_m = \mu \wedge \nu$ where $\mu \in \{x{=}y, +1(x,y), \neg(+1(x,y) \vee x{=}y)\}$ and $\nu \in \{x \sim y, \neg(x \sim y)\}$.

Claim 36. $\forall x \forall y \ \chi$ can be translated into an equivalent formula of the form ($\star$) in exponential time.

Proof: As an example, if

$$\chi = \neg(x \sim y) \vee X(x) \vee X(y),$$

then the corresponding formula would be

$$\bigwedge_{\mu, \chi^x, \chi^y} \big((\neg \exists x \exists y \ x \leq y \wedge \mu \wedge \neg X(x) \wedge \chi^y) \ \wedge \\ (\neg \exists x \exists y \ x \leq y \wedge \mu \wedge \chi^x \wedge \neg X(y)) \big)$$

for all $\mu \in \{x{=}y, {+}1(x,y), \neg({+}1(x,y) \vee x{=}y)\}$, $\chi^x \in \{X(x), \neg X(x)\}$, $\chi^y \in \{X(y), \neg X(y)\}$. We can bring the formula into this normal form in exponential time. To this end, we can first bring χ into CNF, $\chi = \bigwedge_{i\in I} \bigvee_{j\in J_i} \nu_{ij}$, where every ν_{ij} is an atom or a negation of an atom. Then,

$$\forall x \forall y\ \chi \ \equiv\ \forall x \forall y \bigwedge_{i\in I} \bigvee_{j\in J_i} \nu_{ij} \ \equiv\ \bigwedge_{i\in I} \forall x \forall y \bigvee_{j\in J_i} \nu_{ij} \ \equiv\ \bigwedge_{i\in I} \neg\exists x \exists y \bigwedge_{j\in J_i} \neg\nu_{ij}$$

Let $\nu_{ij}^{x\leftrightarrow y}$ be ν_{ij} where x and y are swapped. Note that $\exists x \exists y\ \bigwedge_{j\in J_i} \neg\nu_{ij}$ is equivalent to $\exists x \exists y\ \bigwedge_{j\in J_i} \neg\nu_{ij}^{x\leftrightarrow y}$. Now for every $i \in I$, let

$$\mu_{i,1} = x \leq y \wedge \bigwedge_{j\in J_i} \neg\nu_{ij} \qquad\qquad \mu_{i,2} = x \leq y \wedge \bigwedge_{j\in J_i} \neg\nu_{ij}^{x\leftrightarrow y}.$$

Note that $\exists x \exists y\ \mu_{i,1} \vee \exists x \exists y\ \mu_{i,2}$ is equivalent to $\exists x \exists y\ \bigwedge_{j\in J_i} \neg\nu_{ij}$. Hence,

$$\bigwedge_{i\in I} \neg \bigvee_{j\in\{1,2\}} \exists x \exists y\ (x \leq y \wedge \mu_{i,j}) \ \equiv\ \bigwedge_{i\in I} \bigwedge_{j\in\{1,2\}} \neg\exists x \exists y\ (x \leq y \wedge \mu_{i,j})$$

is equivalent to $\forall x \forall y\ \chi$. Finally, every $\mu_{i,j}$ can be easily split into a conjunction of three formulas (one of binary relations, one of unary relations on x, and one of unary relations on y), thus obtaining a formula of the form ($\star$). This procedure takes polynomial time once the CNF normal form is obtained, and it then takes exponential time in the worst case. ∎

We define $tr(\chi_m^x)$ as the conjunction of all the formulas $\mathtt{x}_0 \approx \mathtt{x}_i$ so that $X_i(x)$ is a conjunct of χ_m^x, and all the formulas $\neg(\mathtt{x}_0 \approx \mathtt{x}_i)$ so that $\neg X_i(x)$ is a conjunct of χ_m^x; we do similarly for $tr(\chi_m^y)$. If $\mu = {+}1(x,y)$ and $\nu = x \sim y$ we translate

$$tr\big(\exists y\ (\ x \leq y\ \wedge\ \chi_m\ \wedge\ \chi_m^x\ \wedge\ \chi_m^y\ )\big)\ =\ \mathtt{x} \approx \mathsf{X}\mathtt{x}\ \wedge\ tr(\chi_m^x)\ \wedge\ \mathsf{X}tr(\chi_m^y).$$

If $\mu = (x = y)$ and $\nu = x \sim y$ we translate

$$tr\big(\exists y\ (\ x \leq y\ \wedge\ \chi_m\ \wedge\ \chi_m^x\ \wedge\ \chi_m^y\ )\big)\ =\ tr(\chi_m^x)\ \wedge\ tr(\chi_m^y).$$

We proceed similarly for $\mu = {+}1(x,y)$, $\nu = \neg(x \sim y)$; and the translation is of course $\bot$ (false) if $\mu = (x{=}y)$, $\nu = \neg(x{\sim}y)$. The difficult cases are the remaining ones. Suppose $\mu = \neg({+}1(x,y) \vee x{=}y)$, $\nu = x{\sim}y$. In other words, x is at least two positions before y, and they have the same data value. Observe that the formula $tr(\chi_m^x) \wedge \mathtt{x} \approx \langle tr(\chi_m^y)?\rangle \mathtt{x}$ does not encode precisely this case, as it would correspond to a weaker condition $x < y \wedge x \sim y$. In order to properly translate this case we make use of the variable $\mathtt{x}_{\mathit{prev}}$, ensuring that it always has the data value of the variable $\mathtt{x}$ in the previous position

$$\mathit{prev}\ =\ \mathsf{G}\big(\ \mathsf{X}\top\ \Rightarrow\ \mathtt{x} \approx \mathsf{X}\mathtt{x}_{\mathit{prev}}\ \big).$$

We then define $tr\big(\exists y\ (\ x \leq y\ \wedge\ \chi_m\ \wedge\ \chi_m^x\ \wedge\ \chi_m^y\ )\big)$ as

$$tr(\chi_m^x)\ \wedge\ \mathtt{x} \approx \langle \mathtt{x}_{\mathit{prev}} \approx \langle tr(\chi_m^y)?\rangle \mathtt{x}?\rangle\, \mathtt{x}_{\mathit{prev}}.$$

Note that by nesting twice the future obligation we ensure that the target position where $tr(\chi_m^y)$ must hold is at a distance of at least two positions. For $\nu = \neg(x{\sim}y)$ we produce a similar formula, replacing the innermost appearance of $\approx$ with $\not\approx$ in the formula above. We then define $tr(\forall x \forall y\ \chi)$ as

$$\mathit{prev}\ \wedge\ \bigwedge_{m\in M}\ \neg\ \mathsf{F}\ tr\big(\exists y\ (x \leq y\ \wedge\ \chi_m \wedge \chi_m^x \wedge \chi_m^y)\big).$$

To translate $\forall x \exists y\ (x \leq y \wedge \psi_k)$ we proceed in a similar way. As before, we bring $x \leq y \wedge \psi_k$ into the form $\bigvee_{m\in M}\ x \leq y \wedge \chi_m \wedge \chi_m^x \wedge \chi_m^y$, in exponential time. We then define $tr(\forall x \exists y\ (x \leq y \wedge \psi_k))$ as

$$\mathit{prev}\ \wedge\ \mathsf{G}\ \bigvee_{m\in M}\ tr\big(\exists y\ (x \leq y \wedge \chi_m \wedge \chi_m^x \wedge \chi_m^y)\big).$$

Thus,

$$tr(\varphi)\ =\ tr\big(\forall x \forall y\ \chi\big)\ \wedge\ \bigwedge_k tr\big(\forall x \exists y\ (x \leq y \wedge \psi_k)\big).$$

One can show that the translation tr defined above preserves satisfiability. More precisely, it can be seen that:

(1) Any data word whose data values are the $\mathtt{x}$-projection from a model satisfying $tr(\varphi)$, satisfies φ; and, conversely,
(2) for any data word satisfying φ with a given assignment for $X_1, \ldots, X_n$ to the word positions, and for any model σ such that
   - σ has the same length as the data word,
   - for every position i, $\sigma(i)(\mathtt{x})$ is the data value of position i from the data word, and $\sigma(i)(\mathtt{x}_0) = \sigma(i)(\mathtt{x}_j)$ iff X_j holds at position i of the data word, and
   - for every position $i > 0$, $\sigma(i)(\mathtt{x}_{\mathit{prev}}) = \sigma(i-1)(\mathtt{x})$,

   we have that $\sigma \models tr(\varphi)$.

By Corollary 10 we can decide the satisfiability of the translation in 2EXPSPACE, and since the translation is exponential, this gives us a 3EXPSPACE upper bound for the satisfiability of *forward*-EMSO$^2(+1, <, \sim)$. ∎

Remark 37. The proof above can be also extended to work with a similar fragment of EMSO$^2(+1, \ldots, +k, <, \sim)$, that is, EMSO$^2(+1, <, \sim)$ extended with all binary relations of the kind $+i(x, y)$ for every $i \leq k$, with the semantics that x is i positions after y. Hence, we also obtain the decidability of the satisfiability problem for this logic in 3EXPSPACE. We do not know if the upper bounds we give for *forward*-EMSO$^2(+1, <, \sim)$ and *forward*-EMSO$^2(+1, \ldots, +k, <, \sim)$ can be improved.

Our result stating that PLRV$_1$ is equivalent to reachability in VASS (Corollary 22), can also be seen as a hardness result for FO$^2(<, \sim, \{+k\}_{k\in\mathbb{N}})$, that is, first-order logic with two variables on data words, extended with all binary relations $+k(x, y)$ denoting that two elements are at distance k. It is easy to see that this logic captures PLRV$_1$ and hence that it is equivalent to reachability in VASS, even in the absence of an alphabet.

Corollary 38. The satisfiability problem for FO$^2(<, \sim, \{+k\}_{k\in\mathbb{N}})$ is as hard as the reachability problem in VASS, even when restricted to having an alphabet $\Sigma = \emptyset$.

2) **Temporal Logics:**

Consider a temporal logic with future operators $\mathsf{F}_=$ and $\mathsf{F}_{\neq}$, so that $\mathsf{F}_= \varphi$ (resp. $\mathsf{F}_{\neq} \varphi$) holds at some position i of the finite data word if there is some future position $j > i$ where φ is true, and the data values of positions i and j are equal (resp. distinct). We also count with "next" operators $\mathsf{X}^k_=$ and $\mathsf{X}^k_{\neq}$ for any $k \in \mathbb{N}$, where $\mathsf{X}^k_= \varphi$ (resp. $\mathsf{X}^k_{\neq} \varphi$) holds at position i if φ holds at position $i + k$, and the data values of position $i + k$ and i are equal (resp. distinct). Finally, the logic also features a standard until operator U, tests for the labels of positions, and it is closed under Boolean operators. We call this logic $\mathsf{LTL}(\mathsf{U}, \mathsf{F}_=, \mathsf{F}_{\neq}, \{\mathsf{X}^k_=, \mathsf{X}^k_{\neq}\}_{k\in\mathbb{N}})$. There is an efficient satisfiability-preserving translation from $\mathsf{LTL}(\mathsf{U}, \mathsf{F}_=, \mathsf{F}_{\neq}, \{\mathsf{X}^k_=, \mathsf{X}^k_{\neq}\}_{k\in\mathbb{N}})$ into LRV and back and hence we have the following.

Proposition 39. The satisfiability problem for $\mathsf{LTL}(\mathsf{U}, \mathsf{F}_=, \mathsf{F}_{\neq}, \{\mathsf{X}^k_=, \mathsf{X}^k_{\neq}\}_{k\in\mathbb{N}})$ is 2EXPSPACE-complete.

Proof sketch: There is a straightforward polynomial-time translation into LRV that preserves satisfiability, where $\mathsf{F}_=\varphi$ is translated as $\mathtt{x} \approx \langle\varphi'?\rangle\mathtt{x}$ and $\mathsf{F}_{\neq}\varphi$ is translated as $\mathtt{x} \not\approx \langle\varphi'?\rangle\mathtt{x}$; $\mathsf{X}^k\varphi$ as $\mathsf{X}^k\varphi' \wedge \mathtt{x} \approx \mathsf{X}^k\mathtt{x}$; and any test for label a_i by $\mathtt{x}_0 \approx \mathtt{x}_i$. ∎

In fact, $\mathsf{LTL}(\mathsf{U}, \mathsf{F}_=, \mathsf{F}_{\neq}, \{\mathsf{X}^k_=, \mathsf{X}^k_{\neq}\}_{k\in\mathbb{N}})$ corresponds to a fragment of the linear-time temporal logic LTL extended with one register for storing and comparing data values. We denote it by $\mathsf{LTL}^{\downarrow}_1$, and it was studied in [7]. This logic contains one operator to store the current datum, one operator to test the whether the current datum is equal to the one stored. The *freeze* operator $\downarrow \varphi$ permits to *store* the current datum in the register and continue the evaluation of the formula φ. The operator $\uparrow$ *tests* whether the current data value is equal to the one stored in the register. When the temporal operators are limited to F, U and X, this logic is decidable with non-primitive-recursive complexity [7].

Indeed $\mathsf{LTL}(\mathsf{U}, \mathsf{F}_=, \mathsf{F}_{\neq}, \{\mathsf{X}^k_=, \mathsf{X}^k_{\neq}\}_{k\in\mathbb{N}})$ is the fragment where we only allow $\downarrow$ and $\uparrow$ to appear in the form of $\downarrow \mathsf{F}(\uparrow \wedge \varphi)$ and $\downarrow \mathsf{X}^k(\uparrow \wedge \varphi)$ —or with $\neg\uparrow$ instead of $\uparrow$. Markedly, this restriction allows us to jump from a non-primitive-recursive complexity of the satisfiability problem, to an elementary 2EXPSPACE complexity.