

REASONING ABOUT DATA REPETITIONS WITH COUNTER SYSTEMS*

STÉPHANE DEMRI^a, DIEGO FIGUEIRA^b, AND M. PRAVEEN^c

^a LSV, ENS Cachan, CNRS, Université Paris-Saclay, 94235 Cachan, France

^b LaBRI, CNRS, 33405 Talence, France

^c CMI, Chennai, India

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.

1. INTRODUCTION

Words with multiple data. Finite data words [Bou02] 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 arrays are finite data words [AvW12], finite data trees model XML

2012 ACM CCS: [Theory of computation]: Logic; Formal languages and automata theory; Theory and algorithms for application domains.

Key words and phrases: data-word, counter systems, LTL.

* Complete version of [DFP13].

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.

documents with attribute values [Fig10] and so on. A wealth of specification formalisms for data words (or slight variants) has been introduced stemming from automata, see e.g. [NSV04, Seg06], to adequate logical languages such as first-order logic [BDM⁺11, Dav09] or temporal logics [LP05, KV06, Laz06, Fig10, KSZ10, Fig11] (see also a related formalism in [Fit02]). Depending on the type of structures, other formalisms have been considered such as XPath [Fig10] or monadic second-order logic [BCGK12]. 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. [BDM⁺11], limiting the set of the temporal operators or the use of the data manipulating operator, see e.g. [FS09, DDG12]. As far as classes of automata for data languages are concerned, other questions arise related to closure properties or to logical characterisations, see e.g. [BDM⁺11, BL10, KST12]. Moreover, interesting and surprising results have been exhibited about relationships between logics for data words and counter automata (including vector addition systems with states) [BDM⁺11, DL09, BL10], leading to a first classification of automata on data words [BL10, Bol11]. 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.

Motivations. In [DDG12], a decidable linear-time temporal 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 $x \approx X^i y$ or $x \approx \langle \top? \rangle y$. The formula $x \approx X^i y$ states that the current value of variable x is the same as the value of y i steps ahead (local constraint) whereas $x \approx \langle \top? \rangle y$ states that the current value of x is repeated in a future value of y (future obligation). Such atomic properties can be naturally expressed with a *freeze* operator that stores a data value for later comparison, and in [DDG12], it is shown that the satisfiability problem is decidable with the temporal operators in $\{X, X^{-1}, U, S\}$. 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 [DDG12] 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 it holds for finite or infinite sequences, for any set of MSO-definable temporal operators and with the addition of atomic formulas of the form $x \approx \langle \top? \rangle^{-1} y$ stating that the current value of x is repeated in a past value of y (past obligation). Decidability can be shown either by reduction into $FO^2(\sim, <, +\omega)$, a first-order logic over data words introduced in [BDM⁺11] or by reduction into the verification of fairness properties in Petri nets, shown decidable in [Jan95]. 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. [Ler11] (see also a first upper bound established recently in [LS15]). Hence, even though the logics shown decidable in [DDG12] poorly use the freeze operator (or equivalently, 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 [KSZ10], a richer logic BD-LTL has been introduced and it has been shown that satisfiability is equivalent to the reachability problem for Petri nets. Moreover, in [DHLT14],

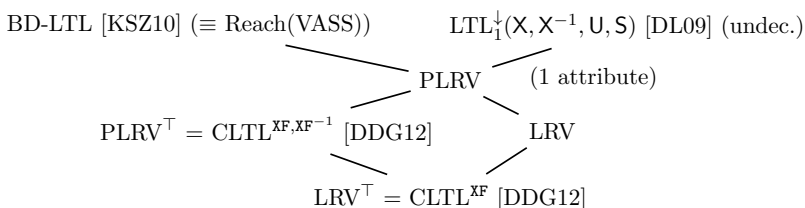


Figure 1: Placing LRV and variants in the family of data logics

two fragments BD-LTL^- and BD-LTL^+ of that richer logic have been introduced and shown to admit 2EXPSPACE -complete satisfiability problems. Forthcoming logic LRV^\top is shown in [DHLLT14] to be strictly less expressive than BD-LTL^+ .

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 [DDG12, BDM⁺11, DL09, FS09, KSZ10, Fig10, Fig11]: 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 linear-time temporal logic LRV (“Logic of Repeating Values”) interpreted over finite words with multiple data, equipped with atomic formulas of the form either $x \approx X^i y$ or $x \approx \langle \phi? \rangle y$ [resp. $x \not\approx \langle \phi? \rangle y$], where $x \approx \langle \phi? \rangle y$ [resp. $x \not\approx \langle \phi? \rangle y$] states that the current value of x is repeated [resp. is not repeated] in some future value of y in a position where ϕ holds true. When we impose $\phi = \top$, the logic introduced in [DDG12] is obtained and it is denoted by LRV^\top (a different name is used in [DDG12], namely CLTL^{XF}). Note that the syntax for future obligations is freely inspired from propositional dynamic logic 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 $x \approx \langle \phi? \rangle^{-1} y$ or $x \not\approx \langle \phi? \rangle^{-1} y$. Figure 1 illustrates how LRV and variants are compared to existing data logics.

Our main results are listed below.

- i. We begin where [DDG12] stopped: the reachability problem for Petri nets is reduced to the satisfiability problem of PLRV (i.e., the logic with past obligations).
- ii. Without past obligations, the satisfiability problem is much easier: we reduce the satisfiability problem of 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.
- iii. 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.
- iv. 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.
 - v. The power of nested testing formulas: we show that the complexity of the satisfiability problem for 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 LRV^\top and LRV is that the later allows any ϕ in $\mathbf{x} \approx \langle \phi? \rangle \mathbf{y}$ while the former restricts ϕ to just \top .
 - vi. The power of pairs of repeating values: we show that the satisfiability problem of LRV^\top augmented with $\langle \mathbf{x}, \mathbf{y} \rangle \approx \langle \top? \rangle \langle \mathbf{x}', \mathbf{y}' \rangle$ (repetitions of pairs of data values) is undecidable, even when $\langle \mathbf{x}, \mathbf{y} \rangle \approx \langle \top? \rangle^{-1} \langle \mathbf{x}', \mathbf{y}' \rangle$ is not allowed (i.e., even when past obligations are not allowed).
 - vii. Implications for classical logics: we show a 3EXPSPACE upper bound for the satisfiability problem for *forward-EMSO*²(+1, <, \sim) over data words, using results on LRV .

For proving the result mentioned in point iii. 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 $(k+1)\text{EXPSPACE}$ -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 $(k+1)\text{EXPSPACE}$ lower bound is obtained by a non-trivial extension of the EXPSPACE -hardness result from [Lip76, Esp98]. 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 5). It was known that data logics are strongly related to classes of counter automata, see e.g. [BDM⁺11, DL09, BL10] but herein, we show how varying the expressive power of logics leads to correspondence with different reachability problems for counter machines.

2. PRELIMINARIES

We write \mathbb{N} [resp. \mathbb{Z}] to denote the set of non-negative integers [resp. integers] and $[i, j]$ to denote the set $\{k \in \mathbb{Z} : i \leq k \text{ and } k \leq j\}$. For every $\mathbf{v} \in \mathbb{Z}^n$, $\mathbf{v}(i)$ 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 $u = a_1 \dots a_k$ over Σ , we write $|u|$ to denote its *length* k . For every $0 \leq i < |u|$, $u(i)$ represents the $(i+1)$ -th letter of the word, here a_{i+1} . We use $\text{card}(X)$ to denote the number of elements of a finite set X .

2.1. Logics of Repeating Values. Let $\text{VAR} = \{\mathbf{x}_1, \mathbf{x}_2, \dots\}$ be a countably infinite set of *variables*. We denote by LRV the logic whose formulas are defined as follows:

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

where $\mathbf{x}, \mathbf{y} \in \text{VAR}$ and $i \in \mathbb{N}$. Formulas of one of the forms $\mathbf{x} \approx \mathbf{X}^i \mathbf{y}$, $\mathbf{x} \approx \langle \phi? \rangle \mathbf{y}$ or $\mathbf{x} \not\approx \langle \phi? \rangle \mathbf{y}$ are said to be *atomic* and an expression of the form $\mathbf{X}^i \mathbf{x}$ (abbreviation for i next symbols followed by a variable) is called a *term*.

A *valuation* is a map from VAR to \mathbb{N} , and a *model* is a finite non-empty sequence σ of valuations. All the subsequent developments can be equivalently done with the domain \mathbb{N} replaced by an infinite set \mathbb{D} since only equality tests are performed in the logics.

We write $|\sigma|$ to denote the length of σ . For every model σ and $0 \leq i < |\sigma|$, the satisfaction relation \models is defined inductively as follows. Note that the temporal operators next (\mathbf{X}), previous (\mathbf{X}^{-1}), until (\mathbf{U}) and since (\mathbf{S}) and Boolean connectives are defined in the usual way.

$$\begin{array}{ll}
\sigma, i \models \mathbf{x} \approx \mathbf{X}^j \mathbf{y} & \text{iff } i + j < |\sigma| \text{ and } \sigma(i)(\mathbf{x}) = \sigma(i + j)(\mathbf{y}) \\
\sigma, i \models \mathbf{x} \approx \langle \phi? \rangle \mathbf{y} & \text{iff there exists } j \text{ such that } i < j < |\sigma|, \\
& \sigma(i)(\mathbf{x}) = \sigma(j)(\mathbf{y}) \text{ and } \sigma, j \models \phi \\
\sigma, i \models \mathbf{x} \not\approx \langle \phi? \rangle \mathbf{y} & \text{iff there exists } j \text{ such that } i < j < |\sigma|, \\
& \sigma(i)(\mathbf{x}) \neq \sigma(j)(\mathbf{y}) \text{ and } \sigma, j \models \phi \\
\sigma, i \models \phi \wedge \phi' & \text{iff } \sigma, i \models \phi \text{ and } \sigma, i \models \phi' \\
\sigma, i \models \neg \phi & \text{iff } \sigma, i \not\models \phi \\
\sigma, i \models \mathbf{X} \phi & \text{iff } i + 1 < |\sigma| \text{ and } \sigma, i + 1 \models \phi \\
\sigma, i \models \mathbf{X}^{-1} \phi & \text{iff } i > 0 \text{ and } \sigma, i - 1 \models \phi \\
\sigma, i \models \phi \mathbf{U} \phi' & \text{iff there is } i \leq j < |\sigma| \text{ such that } \sigma, j \models \phi' \text{ and} \\
& \text{for every } i \leq l < j, \text{ we have } \sigma, l \models \phi \\
\sigma, i \models \phi \mathbf{S} \phi' & \text{iff there is } 0 \leq j \leq i \text{ such that } \sigma, j \models \phi' \text{ and} \\
& \text{for every } j < l \leq i \text{ we have } \sigma, l \models \phi.
\end{array}$$

We write $\sigma \models \phi$ if $\sigma, 0 \models \phi$. We use the standard derived temporal operators (\mathbf{G} , \mathbf{F} , \mathbf{F}^{-1} , \dots), and derived Boolean operators (\vee , \Rightarrow , \dots) and constants \top , \perp .

We also use the notation $\mathbf{X}^i \mathbf{x} \approx \mathbf{X}^j \mathbf{y}$ as an abbreviation for the formula $\mathbf{X}^i (\mathbf{x} \approx \mathbf{X}^{j-i} \mathbf{y})$ (assuming without any loss of generality that $i \leq j$). Similarly, $\mathbf{X}^j \mathbf{y} \approx \mathbf{x}$ is an abbreviation for $\mathbf{x} \approx \mathbf{X}^j \mathbf{y}$.

Given a set of temporal operators \mathcal{O} definable from those in $\{\mathbf{X}, \mathbf{X}^{-1}, \mathbf{S}, \mathbf{U}\}$ and a natural number $k \geq 0$, we write $\text{LRV}_k(\mathcal{O})$ to denote the fragment of LRV restricted to formulas with temporal operators from \mathcal{O} and with at most k variables. The *satisfiability problem* for LRV (written $\text{SAT}(\text{LRV})$) is to check for a given LRV formula ϕ , whether there exists a model σ such that $\sigma \models \phi$. Note that there is a logarithmic-space reduction from the satisfiability problem for LRV to its restriction where atomic formulas of the form $\mathbf{x} \approx \mathbf{X}^i \mathbf{y}$ satisfy $i \in \{0, 1\}$ (at the cost of introducing new variables).

Let PLRV be the extension of LRV with additional atomic formulas of the form $\mathbf{x} \approx \langle \phi? \rangle^{-1} \mathbf{y}$ and $\mathbf{x} \not\approx \langle \phi? \rangle^{-1} \mathbf{y}$. The satisfaction relation is extended as follows:

$$\begin{array}{ll}
\sigma, i \models \mathbf{x} \approx \langle \phi? \rangle^{-1} \mathbf{y} & \text{iff there is } 0 \leq j < i \text{ such that } \sigma(i)(\mathbf{x}) = \sigma(j)(\mathbf{y}) \text{ and } \sigma, j \models \phi \\
\sigma, i \models \mathbf{x} \not\approx \langle \phi? \rangle^{-1} \mathbf{y} & \text{iff there is } 0 \leq j < i \text{ such that } \sigma(i)(\mathbf{x}) \neq \sigma(j)(\mathbf{y}) \text{ and } \sigma, j \models \phi.
\end{array}$$

We write LRV^\top [resp. PLRV^\top] to denote the fragment of LRV [resp. PLRV] in which atomic formulas are restricted to $\mathbf{x} \approx \mathbf{X}^i \mathbf{y}$ and $\mathbf{x} \approx \langle \top? \rangle \mathbf{y}$ [resp. $\mathbf{x} \approx \mathbf{X}^i \mathbf{y}$, $\mathbf{x} \approx \langle \top? \rangle \mathbf{y}$ and $\mathbf{x} \approx \langle \top? \rangle^{-1} \mathbf{y}$]. These are precisely the fragments considered in [DDG12] and shown decidable by reduction into the reachability problem for Petri nets.

Proposition 2.1. [DDG12]

- (I) *Satisfiability problem for LRV^\top is decidable (by reduction to the reachability problem for Petri nets).*
- (II) *Satisfiability problem for LRV^\top restricted to a single variable is PSPACE-complete.*
- (III) *Satisfiability problem for PLRV^\top is decidable (by reduction to the reachability problem for Petri nets).*

In [DDG12], there are no reductions in the directions opposite to (I) and (III). The characterisation of the computational complexity for the satisfiability problems for LRV and PLRV remained unknown so far and this will be a contribution of the paper.

2.2. 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 $\mathbf{x} \approx \langle \phi? \rangle \mathbf{y}$ and $\mathbf{x} \not\approx \langle \phi? \rangle \mathbf{y}$ when ϕ is an arbitrary formula since there is no obvious way to express one with the other.

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

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 \mathbf{x} . Let us express that whenever there are $i < j$ such that i and j [resp. $i+1$ and $j+1$, $i+2$ and $j+2$] are labelled by a [resp. a' , a''], $\sigma(i+1)(\mathbf{x}) \neq \sigma(j+1)(\mathbf{x})$. This can be stated in LRV by:

$$\mathbf{G}(a \wedge \mathbf{X}(a' \wedge \mathbf{X}a'') \Rightarrow \mathbf{X}\neg(\mathbf{x} \approx \langle \mathbf{X}^{-1}a \wedge a' \wedge \mathbf{X}a''? \rangle \mathbf{x})).$$

This is an example of key constraints, see e.g. [NS11, Definition 2.1] and the current paper contains also numerous examples of properties that can be captured by LRV.

2.3. Basics on VASS. A *vector addition system with states* 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 $\langle q, \mathbf{v} \rangle \rightarrow \langle q', \mathbf{v}' \rangle$ if there is a transition $(q, \mathbf{u}, q') \in \delta$ such that $\mathbf{v}' = \mathbf{v} + \mathbf{u}$. Let $\xrightarrow{*}$ be the reflexive and transitive closure of \rightarrow . The reachability problem for VASS (written $\text{Reach}(\text{VASS})$) consists of 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 but all known algorithms [May84, Kos82, Lam92, Ler11] take non-primitive recursive space in the worst case. The best known lower bound is EXPSPACE [Lip76, Esp98] whereas a first upper bound has been recently established in [LS15]. The control state reachability problem consists in checking whether $\langle q_0, \mathbf{v}_0 \rangle \xrightarrow{*} \langle q_f, \mathbf{v} \rangle$ for some

$\mathbf{v} \in \mathbb{N}^C$, given a configuration $\langle q_0, \mathbf{v}_0 \rangle$ and a control state q_f . This problem is known to be EXPSpace-complete [Lip76, Rac78]. The relation \rightarrow denotes the one-step transition in a perfect computation. In the paper, we need to introduce computations with gains or with losses. We define below the variant relations $\rightarrow_{\text{gainy}}$ and $\rightarrow_{\text{lossy}}$. We write $\langle q, \mathbf{v} \rangle \rightarrow_{\text{gainy}} \langle q', \mathbf{v}' \rangle$ if there is a transition $(q, \mathbf{u}, q') \in \delta$ and $\mathbf{w}, \mathbf{w}' \in \mathbb{N}^C$ such that $\mathbf{v} \preceq \mathbf{w}$, $\mathbf{w}' = \mathbf{w} + \mathbf{u}$ and $\mathbf{w}' \preceq \mathbf{v}'$. Let $\xrightarrow{*}_{\text{gainy}}$ be the reflexive and transitive closure of $\rightarrow_{\text{gainy}}$. Similarly, we write $\langle q, \mathbf{v} \rangle \rightarrow_{\text{lossy}} \langle q', \mathbf{v}' \rangle$ if there is a transition $(q, \mathbf{u}, q') \in \delta$ and $\mathbf{w}, \mathbf{w}' \in \mathbb{N}^C$ such that $\mathbf{w} \preceq \mathbf{v}$, $\mathbf{w}' = \mathbf{w} + \mathbf{u}$ and $\mathbf{v}' \preceq \mathbf{w}'$. Let $\xrightarrow{*}_{\text{lossy}}$ be the reflexive and transitive closure of $\rightarrow_{\text{lossy}}$. Counter automata with imperfect computations such as lossy channel systems [AJ96, FS01], lossy counter automata [May03] or gainy counter automata [Sch10b] have been intensively studied (see also [Sch10a]). In the paper, imperfect computations are used with VASS in Section 4.

3. THE POWER OF PAST: FROM REACH(VASS) TO SAT(PLRV)

While [DDG12] 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 (see recent developments in [LS15]). 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 probably much more complicated.

Theorem 3.1. *There is a polynomial-space reduction from Reach(VASS) into SAT(PLRV).*

The proof of Theorem 3.1 is analogous to the proof of [BDM⁺11, Theorem 16] except that properties are expressed in PLRV instead of being expressed in FO²($\sim, <, +1$).

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 [BDM⁺11, Theorem 16] except that the properties are expressed in PLRV instead of FO²($\sim, <, +1$). Letters from the finite alphabet are also encoded by equalities of the form $\mathbf{x}_0 = \mathbf{x}_i$.

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 $\mathbf{x}_0, \dots, \mathbf{x}_N$, with $N = \lceil \log(\text{card}(\delta)) \rceil$. For any $t \in \delta$, let $\langle t \rangle \in \text{PLRV}$ be the formula that tests for label t at the current position. More precisely, for any fixed injective function $\lambda : \delta \rightarrow 2^{[1, N]}$ we define

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

Note that $\langle t \rangle$ uses exclusively variables $\mathbf{x}_0, \dots, \mathbf{x}_N$, that is of size logarithmic in $\text{card}(\delta)$, 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 $\langle \langle q_i, \text{instr}, q \rangle \rangle$ for some $q \in Q$.
- (3) The last position satisfies $\langle \langle q, \text{instr}, q_f \rangle \rangle$ for some $q \in Q$.
- (4) There are no two consecutive positions i and $i + 1$ satisfying $\langle \langle q_1, \text{instr}, q_2 \rangle \rangle$ and $\langle \langle q'_1, \text{instr}', q'_2 \rangle \rangle$ respectively, with $q_2 \neq q'_1$.

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

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

$$\mathbf{G}(\text{inc}(\mathbf{c}) \Rightarrow \neg(\mathbf{x} \approx \langle \text{inc}(\mathbf{c})? \rangle \mathbf{x}))$$

where $\text{inc}(\mathbf{c})$ is a shortcut for $\bigvee_{\langle q, \text{inc}(\mathbf{c}), q' \rangle \in \delta} \langle t \rangle$. A similar constraint can be expressed with $\text{dec}(\mathbf{c})$.

- (2) For every counter $\mathbf{c} \in C$, for every position labelled by a transition with instruction $\text{inc}(\mathbf{c})$, there is a future position labelled by a transition with instruction $\text{dec}(\mathbf{c})$ with the same value for \mathbf{x} :

$$\mathbf{G}(\text{inc}(\mathbf{c}) \Rightarrow \mathbf{x} \approx \langle \text{dec}(\mathbf{c})? \rangle \mathbf{x})$$

where $\text{dec}(\mathbf{c})$ is a shortcut for $\bigvee_{\langle q, \text{dec}(\mathbf{c}), q' \rangle \in \delta} \langle t \rangle$. This guarantees that the final configuration ends with all counters equal to zero.

- (3) Similarly, for every counter $\mathbf{c} \in C$, for every position labelled by a transition with instruction $\text{dec}(\mathbf{c})$, there is a past position labelled by a transition with instruction $\text{inc}(\mathbf{c})$ with the same value for \mathbf{x} :

$$\mathbf{G}(\text{dec}(\mathbf{c}) \Rightarrow \mathbf{x} \approx \langle \text{inc}(\mathbf{c})? \rangle^{-1} \mathbf{x}).$$

This guarantees 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 [BDM⁺11, 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 $\langle q_f, \vec{0} \rangle$. \square

4. 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 $\mathbf{x} \approx \langle \phi? \rangle \mathbf{y}$ to remove the test formula ϕ (i.e., a reduction from SAT(LRV) into SAT(LRV^T)); and
- (2) reducing from SAT(LRV^T) into the control state reachability problem in VASS.

4.1. Elimination of Test Formulas. We give a polynomial-time algorithm such that given $\varphi \in \text{LRV}$, it computes a formula $\varphi' \in \text{LRV}^1$ that preserves satisfiability: there is a model σ such that $\sigma \models \varphi$ iff there is a model σ' such that $\sigma' \models \varphi'$. We give the reduction in two steps. First, we eliminate formulas with inequality tests of the form $\mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y}$ using only positive tests of the form $\mathbf{x} \approx \langle \psi? \rangle \mathbf{y}$. We then eliminate formulas of the form $\mathbf{x} \approx \langle \psi? \rangle \mathbf{y}$, using only formulas of the form $\mathbf{x} \approx \langle \top? \rangle \mathbf{y}$. Although both reductions share some common structure, they use independent coding strategies, and exploit different features of the logic; we therefore present them separately.

We first show how to eliminate all formulas with inequality tests of the form $\mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y}$.

Let LRV^\approx be the logic LRV where there are no appearances of formulas of the form $\mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y}$; and let PLRV^\approx be PLRV without $\mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y}$ or $\mathbf{x} \not\approx \langle \psi? \rangle^{-1} \mathbf{y}$.

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

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

Lemma 4.1. *There is a polynomial-time satisfiability-preserving translation $t : \text{LRV} \rightarrow \text{LRV}$ [resp. $t : \text{LRV}^\approx \rightarrow \text{LRV}^\approx$] such that for every φ and $\psi \in \text{sub}_\Omega(t(\varphi))$, we have $\text{sub}_\Omega(\psi) = \emptyset$.*

Proof. This is a standard reduction. Indeed, given a formula φ with subformula $\mathbf{x} \approx \langle \psi? \rangle \mathbf{y}$, φ is satisfiable iff $\mathbf{G}(\psi \Leftrightarrow \mathbf{x}_{\text{new}} \approx \mathbf{y}_{\text{new}}) \wedge \varphi[\mathbf{x} \approx \langle \psi? \rangle \mathbf{y} \leftarrow \mathbf{x} \approx \langle \mathbf{x}_{\text{new}} \approx \mathbf{y}_{\text{new}}? \rangle \mathbf{y}]$ is satisfiable, where $\mathbf{x}_{\text{new}}, \mathbf{y}_{\text{new}} \notin \text{vars}(\varphi)$, and $\varphi[\mathbf{x} \approx \langle \psi? \rangle \mathbf{y} \leftarrow \mathbf{x} \approx \langle \mathbf{x}_{\text{new}} \approx \mathbf{y}_{\text{new}}? \rangle \mathbf{y}]$ is the result of replacing every occurrence of $\mathbf{x} \approx \langle \psi? \rangle \mathbf{y}$ by $\mathbf{x} \approx \langle \mathbf{x}_{\text{new}} \approx \mathbf{y}_{\text{new}}? \rangle \mathbf{y}$ in φ . Similarly, given a formula φ with subformula $\mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y}$, φ is satisfiable iff $\mathbf{G}(\psi \Leftrightarrow \mathbf{x}_{\text{new}} \approx \mathbf{y}_{\text{new}}) \wedge \varphi[\mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y} \leftarrow \mathbf{x} \not\approx \langle \mathbf{x}_{\text{new}} \approx \mathbf{y}_{\text{new}}? \rangle \mathbf{y}]$ is satisfiable. We need to apply these replacements repeatedly, at most a polynomial number of times if we apply it to the innermost occurrences. \square

Proposition 4.2 (from LRV to LRV^\approx). *There is a polynomial-time reduction from $\text{SAT}(\text{LRV})$ into $\text{SAT}(\text{LRV}^\approx)$; and from $\text{SAT}(\text{PLRV})$ into $\text{SAT}(\text{PLRV}^\approx)$.*

Proof. For every $\varphi \in \text{LRV}$, we compute $\varphi' \in \text{LRV}^\approx$ in polynomial time, which preserves satisfiability. The variables of φ' consist of all the variables of φ , plus: a distinguished variable \mathbf{k} , and variables $\mathbf{v}_{\mathbf{y}, \psi}^\approx, \mathbf{v}_{\mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y}}$ for every subformula ψ of φ and variables \mathbf{x}, \mathbf{y} of φ . The variables $\mathbf{v}_{\mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y}}$'s will be used to get rid of $\not\approx$ in formulas of the form $\mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y}$, and the variables $\mathbf{v}_{\mathbf{y}, \psi}^\approx$'s to treat formulas of the form $\neg(\mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y})$. Finally, \mathbf{k} is a special variable, which has a constant value, different from all the values of the variables of φ .

Assume φ is in negation normal form. Note that each positive occurrence of $\mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y}$ can be safely replaced with $\mathbf{x} \not\approx \mathbf{v}_{\mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y}} \wedge \mathbf{v}_{\mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y}} \approx \langle \psi? \rangle \mathbf{y}$. Indeed, the latter formula implies the former, and it is not difficult to see that whenever there is a model for the former formula, there is also one for the latter. On the other hand, translating formulas of the form $\neg(\mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y})$ is more involved as these implicate some form of universal quantification. For treating these formulas, we use the variables $\mathbf{v}_{\mathbf{y}, \psi}^\approx$ and \mathbf{k} as explained next. Let i be the first position of the model so that all future positions $j > i$ verifying ψ have the same value on variable \mathbf{y} , say value n . As we will see, with a formula of LRV^\approx , one can ensure that $\mathbf{v}_{\mathbf{y}, \psi}^\approx$ has the same value as \mathbf{k} for all $j \leq i$ and value n for all other positions. The enforced values are illustrated below, with an initial prefix where variable $\mathbf{v}_{\mathbf{y}, \psi}^\approx$ is equal to \mathbf{k} until we reach position i , from which point all values of $\mathbf{v}_{\mathbf{y}, \psi}^\approx$ coincide with value n —represented as a dashed area.

- First, the variable \mathbf{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 .

$$const = \mathbf{G} \left((\mathbf{k} \approx \mathbf{Xk} \vee \neg \mathbf{XT}) \wedge \bigwedge_{\mathbf{x} \in \text{vars}(\varphi)} (\mathbf{k} \not\approx \mathbf{x}) \right).$$

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

$$val\text{-}\mathbf{v}_{\mathbf{x},\psi}^{\approx} = \mathbf{G} \left(\begin{array}{l} \mathbf{k} \not\approx \mathbf{v}_{\mathbf{x},\psi}^{\approx} \Rightarrow \mathbf{v}_{\mathbf{x},\psi}^{\approx} \approx \mathbf{Xv}_{\mathbf{x},\psi}^{\approx} \vee \neg \mathbf{XF}\psi \wedge \\ \mathbf{k} \not\approx \mathbf{v}_{\mathbf{x},\psi}^{\approx} \wedge \psi \Rightarrow \mathbf{v}_{\mathbf{x},\psi}^{\approx} \approx \mathbf{x} \end{array} \right).$$

- Finally, let φ^{\approx} be the result of replacing
 - (f) every appearance of $\neg(\mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y})$ by $\neg \mathbf{XF}\psi \vee \mathbf{x} \approx \mathbf{Xv}_{\mathbf{y},\psi}^{\approx}$, and
 - (g) every positive appearance of $\mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y}$ by $\mathbf{x} \not\approx \mathbf{v}_{\mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y}} \wedge \mathbf{v}_{\mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y}} \approx \langle \psi? \rangle \mathbf{y}$ in φ .

The formula φ' is defined as follows:

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

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

Claim 4.3. φ is satisfiable if, and only if, φ' is satisfiable.

Proof. [\Rightarrow] Note that one direction would follow from (4.1): if φ is satisfiable by some σ , then φ' is satisfiable in σ_{φ} . In order to establish (4.1), we show that

$$\text{for every subformula } \gamma \text{ of } \varphi \text{ and } 0 \leq i < |\sigma_{\varphi}|, \text{ we have } \sigma, i \models \gamma \text{ iff } \sigma_{\varphi}, i \models \gamma^{\approx}. \quad (4.2)$$

We further assume that γ is not an atomic formula that is dominated by a negation in φ .

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

We show (4.2) by structural induction. If $\gamma = \mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y}$, then $\sigma, i \models \gamma$ if there is some $j > i$ where $\sigma(j)(\mathbf{y}) \neq \sigma(i)(\mathbf{x})$ and $\sigma, j \models \psi$. Let j_0 be the first such j . Then, by Condition ((e)), $\sigma(i)(\mathbf{x}) \neq \sigma_{\varphi}(i)(\mathbf{v}_{\mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y}}) = \sigma(j_0)(\mathbf{y})$. Hence, $\sigma_{\varphi}, i \models \mathbf{v}_{\mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y}} \approx \langle \psi? \rangle \mathbf{y}$ and $\sigma_{\varphi}, i \models \mathbf{x} \not\approx \mathbf{v}_{\mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y}}$ and thus $\sigma_{\varphi}, i \models \gamma^{\approx}$.

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

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

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

Finally, the proof for the base case of the form $\gamma = \mathbf{x} \approx \mathbf{X}^{\ell} \mathbf{y}$ [resp. $\mathbf{x} \not\approx \mathbf{X}^{\ell} \mathbf{y}$] and for all Boolean and temporal operators are by an easy verification, since $(\cdot)^{\approx}$ is homomorphic for these. Hence, (4.2) holds.

[\Leftarrow] Now suppose that $\sigma \models \varphi'$. Since $\sigma \models const$, we have that σ verifies Condition ((c)). And since $\sigma \models val\text{-}\mathbf{v}_{\mathbf{x},\psi}^{\approx}$ for every $\psi \in \text{sub}_{\langle \rangle}(\varphi)$, $\mathbf{x} \in \text{vars}(\varphi)$, we have that σ verifies

Condition ((d)). We prove by structural induction that for every subformula γ of φ , if $\sigma, i \models \gamma^{\approx}$ then $\sigma, i \models \gamma$ (γ is not an atomic formula that is dominated by a negation in φ).

- If $\gamma = \mathbf{x} \approx \mathbf{y}$ [resp. $\mathbf{x} \not\approx \mathbf{y}$, $\mathbf{x} \approx \mathbf{X}^\ell \mathbf{y}$, $\mathbf{x} \not\approx \mathbf{X}^\ell \mathbf{y}$] it is immediate since $\gamma^{\approx} = \gamma$.
- If $\gamma = \mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y}$ and thus $\gamma^{\approx} = \mathbf{x} \not\approx \mathbf{v}_{\mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y}} \wedge \mathbf{x}_{\mathbf{y}, \psi}^{\approx} \approx \langle \psi? \rangle \mathbf{y}$. Then, $\sigma(i)(\mathbf{x}) \neq \sigma(i)(\mathbf{v}_{\mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y}}) = \sigma(j)(\mathbf{y})$ for some $j > i$ where $\sigma, j \models \psi$. Hence, $\sigma, i \models \mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y}$.
- If $\gamma = \neg(\mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y})$, and thus $\gamma^{\approx} = \neg \mathbf{X} \mathbf{F} \psi \vee \mathbf{x} \approx \mathbf{X} \mathbf{v}_{\mathbf{y}, \psi}^{\approx}$. This means that either
 - there is no $j > i$ so that $\sigma, j \models \psi$ and hence $\sigma, i \models \neg(\mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y})$, or
 - $\sigma, j \models \psi$ for some $j > i$ and $\sigma(i)(\mathbf{x}) = \sigma(i+1)(\mathbf{v}_{\mathbf{y}, \psi}^{\approx})$. By Condition ((d)), we have that for all $i < j'$, so that $\sigma, j' \models \psi$, we have $\sigma(j')(\mathbf{v}_{\mathbf{y}, \psi}^{\approx}) = \sigma(j')(\mathbf{y})$. Then, $\sigma, i \models \neg(\mathbf{x} \not\approx \langle \psi? \rangle \mathbf{y})$.
- If $\gamma = \mathbf{F} \psi$ and $\gamma^{\approx} = \mathbf{F} \psi^{\approx}$, there must be some position $i' \geq i$ so that $\sigma, i' \models \psi^{\approx}$. By inductive hypothesis, $\sigma, i' \models \psi$ and hence $\sigma, i \models \mathbf{F} \psi$. 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. \square

We can easily extend this coding allowing for past obligations. We only need to use some extra variables $\mathbf{v}_{\mathbf{x}, \psi}^{-1, \approx}$, $\mathbf{v}_{\mathbf{x} \not\approx \langle \psi? \rangle^{-1} \mathbf{y}}^{-1}$ that behave in the same way as the previously defined, but with past obligations. That is, we also define a $val\text{-}\mathbf{v}_{\mathbf{x}, \psi}^{-1, \approx}$ as $val\text{-}\mathbf{v}_{\mathbf{x}, \psi}^{\approx}$, but making use of $\mathbf{v}_{\mathbf{x}, \psi}^{-1, \approx}$, \mathbf{X}^{-1} and \mathbf{F}^{-1} . And finally, we have to further replace

- (c) every appearance of $\neg(\mathbf{x} \not\approx \langle \psi? \rangle^{-1} \mathbf{y})$ by $\neg \mathbf{X}^{-1} \mathbf{F}^{-1} \psi \vee \mathbf{x} \approx \mathbf{X} \mathbf{v}_{\mathbf{y}, \psi}^{-1, \approx}$, and
 - (d) every positive appearance of $\mathbf{x} \not\approx \langle \psi? \rangle^{-1} \mathbf{y}$ by $\mathbf{x} \not\approx \mathbf{v}_{\mathbf{x} \not\approx \langle \psi? \rangle^{-1} \mathbf{y}}^{-1} \wedge \mathbf{v}_{\mathbf{x} \not\approx \langle \psi? \rangle^{-1} \mathbf{y}}^{-1} \approx \langle \psi? \rangle^{-1} \mathbf{y}$
- in φ to obtain φ^{\approx} . \square

Proposition 4.4 (from LRV^{\approx} to LRV^{\top}). *There is a polynomial-time reduction from $\text{SAT}(\text{LRV}^{\approx})$ to $\text{SAT}(\text{LRV}^{\top})$; and from $\text{SAT}(\text{PLRV}^{\approx})$ into $\text{SAT}(\text{PLRV}^{\top})$.*

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

	\mathbf{k}	\mathbf{k}		\mathbf{k}	\mathbf{k}
		\approx			\approx
	$\mathbf{v}_{\mathbf{y}, \psi}$	$\mathbf{v}_{\mathbf{y}, \psi}$	$\mathbf{v}_{\mathbf{y}, \psi}$	$\mathbf{v}_{\mathbf{y}, \psi}$
	\approx			\approx	
	\mathbf{y}	\mathbf{y}		\mathbf{y}	\mathbf{y}
	ψ	$\neg \psi$		ψ	$\neg \psi$

Next, we formalise these ideas.

Let $\varphi \in \text{LRV}^{\approx}$. For any model σ , let σ_φ be so that

- (a) $|\sigma| = |\sigma_\varphi|$,
- (b) for every $0 \leq i < |\sigma|$ and $\mathbf{x} \in \text{vars}(\varphi)$, $\sigma(i)(\mathbf{x}) = \sigma_\varphi(i)(\mathbf{x})$,
- (c) there is some data value $\mathfrak{d} \notin \{\sigma_\varphi(i)(\mathbf{x}) \mid \mathbf{x} \in \text{vars}(\varphi), 0 \leq i < |\sigma|\}$ such that for every $0 \leq i < |\sigma|$, $\sigma_\varphi(i)(\mathbf{k}) = \mathfrak{d}$, and
- (d) for every $0 \leq i < |\sigma|$, $\sigma_\varphi(i)(\mathbf{v}_{\mathbf{x}, \psi}) = \sigma_\varphi(i)(\mathbf{x})$ if $\sigma, i \models \psi$, and $\sigma_\varphi(i)(\mathbf{v}_{\mathbf{x}, \psi}) = \sigma_\varphi(i)(\mathbf{k})$ 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 $\varphi' \in \text{LRV}^\top$ so that

$$\sigma \models \varphi \text{ if, and only if, } \sigma_\varphi \models \varphi'. \quad (4.3)$$

We assume that for every $\psi \in \text{sub}_{\langle \rangle}(\varphi)$, we have that $\text{sub}_{\langle \rangle}(\psi) = \emptyset$. This is without any loss of generality by Lemma 4.1.

- First, the variable \mathbf{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 .

$$\text{const} = \mathbf{G} \left(\left(\mathbf{k} \approx \mathbf{Xk} \vee \neg \mathbf{X}\top \right) \wedge \bigwedge_{\mathbf{x} \in \text{vars}(\varphi)} (\mathbf{k} \not\approx \mathbf{x}) \right).$$

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

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

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

We then define φ' as follows.

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

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

Claim 4.5. φ is satisfiable if, and only if, φ' is satisfiable.

Proof. [\Rightarrow] Note that one direction would follow from (4.3): if φ is satisfiable by some σ , then φ' is satisfiable in σ_φ . In order to establish (4.3), we show that

$$\text{for every subformula } \gamma \text{ of } \varphi \text{ and } 0 \leq i < |\sigma_\varphi|, \text{ we have } \sigma, i \models \gamma \text{ iff } \sigma_\varphi, i \models \gamma^\top. \quad (4.4)$$

Since $\sigma_\varphi \models \text{const}$ by condition ((c)) and $\sigma_\varphi \models \text{val-}\mathbf{v}_{\mathbf{x},\psi}$ 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 $\sigma, i \models \mathbf{x} \approx \langle \psi? \rangle \mathbf{y}$ then there is some $j > i$ where $\sigma(j)(\mathbf{y}) = \sigma(i)(\mathbf{x})$ and $\sigma, j \models \psi$. Then, by condition ((d)), $\sigma_\varphi(j)(\mathbf{v}_{\mathbf{y},\psi}) = \sigma_\varphi(j)(\mathbf{y})$ and thus $\sigma_\varphi, i \models \mathbf{x} \approx \langle \top? \rangle \mathbf{v}_{\mathbf{y},\psi}$. Conversely, if $\sigma_\varphi, i \models \mathbf{x} \approx \langle \top? \rangle \mathbf{v}_{\mathbf{y},\psi}$ this means that there is some $j > i$ such that $\sigma_\varphi(i)(\mathbf{x}) = \sigma_\varphi(j)(\mathbf{v}_{\mathbf{y},\psi})$. By ((c)), we have that $\sigma_\varphi(j)(\mathbf{v}_{\mathbf{y},\psi}) \neq \sigma_\varphi(j)(\mathbf{k})$, and by ((d)) this means that $\sigma_\varphi(j)(\mathbf{v}_{\mathbf{y},\psi}) = \sigma_\varphi(j)(\mathbf{y})$ and $\sigma_\varphi, j \models \psi$. Therefore, $\sigma, i \models \mathbf{x} \approx \langle \psi? \rangle \mathbf{y}$. The proof for the base case of the form $\psi = \mathbf{x} \approx \mathbf{X}^i \mathbf{y}$ and for all the cases of the induction step are by an easy verification since $(\cdot)^\top$ is homomorphic. Hence, (4.4) holds.

[\Leftarrow] Suppose that $\sigma \models \varphi'$. Note that due to const condition ((c)) holds in σ , and due to $\text{val-}\mathbf{v}_{\mathbf{x},\psi}$, 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 $\gamma = \mathbf{x} \approx \langle \psi? \rangle \mathbf{y}$, and $\gamma^\top = \mathbf{x} \approx \langle \top? \rangle \mathbf{v}_{\mathbf{y},\psi}$, since for all boolean and temporal operators $(\cdot)^\top$ is homomorphic. If $\sigma, i \models \mathbf{x} \approx \langle \top? \rangle \mathbf{v}_{\mathbf{y},\psi}$ there must be some $j > i$ so that $\sigma(j)(\mathbf{v}_{\mathbf{y},\psi}) = \sigma(i)(\mathbf{x})$. By condition ((c)), $\sigma(j)(\mathbf{k}) \neq \sigma(j)(\mathbf{v}_{\mathbf{y},\psi}) = \sigma(i)(\mathbf{x})$, and by condition ((d)), this means that $\sigma, j \models \psi$ and $\sigma(j)(\mathbf{v}_{\mathbf{y},\psi}) = \sigma(j)(\mathbf{y})$. Hence, $\sigma, i \models$

$\mathbf{x} \approx \langle \psi? \rangle \mathbf{y}$. The remaining cases are straightforward since $(\cdot)^\top$ is homomorphic on temporal and boolean operators. \square

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

By combining Proposition 4.2 and Proposition 4.4, we get the following result.

Corollary 4.6. *There is a polynomial-time reduction from $\text{SAT}(\text{LRV})$ into $\text{SAT}(\text{LRV}^\top)$ [resp. from $\text{SAT}(\text{PLRV})$ into $\text{SAT}(\text{PLRV}^\top)$].*

Since $\text{SAT}(\text{PLRV}^\top)$ is decidable [DDG12], we obtain the decidability of $\text{SAT}(\text{PLRV})$.

Corollary 4.7. *$\text{SAT}(\text{PLRV})$ is decidable.*

We have seen how to eliminate test formulas ϕ from $\mathbf{x} \approx \langle \phi? \rangle \mathbf{y}$ and $\mathbf{x} \approx \langle \phi? \rangle^{-1} \mathbf{y}$. Combining this with the decidability proof for PLRV^\top satisfiability from [DDG12], we get that both

Corollary 4.8. *$\text{SAT}(\text{PLRV})$ and $\text{SAT}(\text{PLRV}^\top)$ are equivalent to $\text{Reach}(\text{VASS})$ modulo polynomial-space reductions.*

4.2. From LRV^\top Satisfiability to Control State Reachability. Recall that in [DDG12], $\text{SAT}(\text{LRV}^\top)$ 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 [DDG12], the standard concept of atoms from the Vardi-Wolper construction of automaton for LTL is used (see also Appendix A). Refer to the diagram at the top of Figure 2.

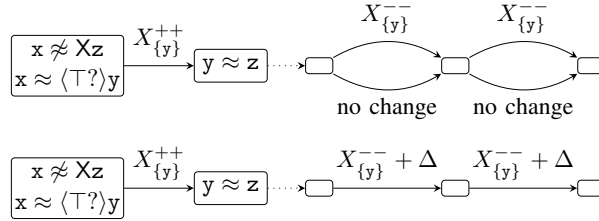


Figure 2: Automaton constructions from [DDG12] (top) and from this paper (bottom).

The formula $\mathbf{x} \approx \langle T? \rangle \mathbf{y}$ in the left atom creates an obligation for the current value of \mathbf{x} to appear some time in the future in \mathbf{y} . This obligation cannot be satisfied in the second atom, since \mathbf{y} has to satisfy some other constraint there ($\mathbf{y} \approx \mathbf{z}$). To remember this unsatisfied obligation about \mathbf{y} while taking the transition from the first atom to the second, the counter $X_{\{y\}}$ is incremented. The counter can be decremented later in transitions that allow the repetition in \mathbf{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 2.

The idea here is to replace the combination of the decrementing transition and the “no change” transition in the top of Figure 2 with a single transition with incremental errors as

shown in the bottom. After Lemma 4.9 below that formalises ideas from [DDG12, Section 7] (see also Appendix A), we prove that the transition with incremental errors is sufficient.

Lemma 4.9 ([DDG12]). *For a LRV[⊤] formula ϕ that uses the variables $\{\mathbf{x}_1, \dots, \mathbf{x}_k\}$, 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., such that*

- *the set of counters C consists of all nonempty subsets of $\{\mathbf{x}_1, \dots, \mathbf{x}_k\}$.*
- *For all $q, q' \in Q$, either $\{\mathbf{u} \mid (q, \mathbf{u}, q') \in \delta\} = [n_1, m_1] \times \dots \times [n_{|C|}, m_{|C|}]$ for some $n_1, m_1, \dots, n_{|C|}, m_{|C|} \in [-k, k]$, 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 2, only one counter $X_{\{y\}}$ 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 $k_1 - 1, \dots, 1, 0, -1, \dots, -(k_2 - 1)$. This corresponds to the closure under component-wise interpolation mentioned in Lemma 4.9. 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 $\mathcal{A}_{\text{inc}} = \langle Q, C, \delta^{\text{min}} \rangle$ and $Q_0, Q_f \subseteq Q$, where Q, Q_0, Q_f and C are same as those of \mathcal{A}_ϕ and δ^{min} is defined as follows: $(q, \text{minup}_{q,q'}, q') \in \delta^{\text{min}}$ iff $\delta \cap (\{q\} \times [-k, k]^C \times \{q'\})$ is not empty and $\text{minup}_{q,q'}(X) \stackrel{\text{def}}{=} \min_{\mathbf{u}: (q, \mathbf{u}, q') \in \delta} \{\mathbf{u}(X)\}$ for all $X \in C$. Similarly, $\text{maxup}_{q,q'}(X) \stackrel{\text{def}}{=} \max_{\mathbf{u}: (q, \mathbf{u}, q') \in \delta} \{\mathbf{u}(X)\}$ for all $X \in C$

Lemma 4.10. *If $\langle q, \mathbf{v} \rangle \rightarrow \langle q', \mathbf{v}' \rangle$ in \mathcal{A}_ϕ , then $\langle q, \mathbf{v} \rangle \rightarrow_{\text{gainy}} \langle q', \mathbf{v}' \rangle$ in \mathcal{A}_{inc} .*

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

Lemma 4.11. *If $\langle q_1, \mathbf{v}_1 \rangle \xrightarrow{*}_{\text{gainy}} \langle q_2, \mathbf{0} \rangle$ in \mathcal{A}_{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}_ϕ .*

Proof. 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 $\langle q_1, \mathbf{v}_1 \rangle \rightarrow_{\text{gainy}} \langle q_3, \mathbf{v}_3 \rangle \xrightarrow{*}_{\text{gainy}} \langle q_2, \mathbf{0} \rangle$ and $\mathbf{v}'_1 \preceq \mathbf{v}_1$. We will define an update function \mathbf{u}'

such that $\langle q_1, \mathbf{v}'_1 \rangle \rightarrow \langle q_3, \mathbf{v}'_3 \rangle$, $\mathbf{v}'_3(X) = \mathbf{v}'_1(X) + \mathbf{u}'(X)$ 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: $\mathbf{v}'_1(X) + \text{minup}_{q_1, q_3}(X) \geq 0$. Let $\mathbf{u}'(X) = \text{minup}_{q_1, q_3}(X)$.

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

Case 2: $\mathbf{v}'_1(X) + \text{minup}_{q_1, q_3}(X) < 0$. Therefore, $\text{minup}_{q_1, q_3}(X) < -\mathbf{v}'_1(X)$. Moreover, since $\text{maxup}_{q_1, q_3}(X) \geq 0$ (due to the optional decrement property) and $\mathbf{v}'_1(X) \geq 0$, $-\mathbf{v}'_1(X) \leq \text{maxup}_{q_1, q_3}(X)$. 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}_ϕ under component-wise interpolation, we have $(q_1, \mathbf{u}', q_3) \in \delta$ and hence $\langle q_1, \mathbf{v}'_1 \rangle \rightarrow \langle q_3, \mathbf{v}'_3 \rangle$. 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 $\langle q_1, \mathbf{v}'_1 \rangle \rightarrow \langle q_3, \mathbf{v}'_3 \rangle \xrightarrow{*} \langle q_2, \mathbf{0} \rangle$. \square

Theorem 4.12. $\text{SAT}(\text{LRV}^\top)$ is in 2EXSPACE .

Proof. The proof is in four steps.

Step 1: From [DDG12], a LRV^\top formula ϕ is satisfiable iff $\langle q_0, \mathbf{0} \rangle \xrightarrow{*} \langle q_f, \mathbf{0} \rangle$ in \mathcal{A}_ϕ 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 4.10 and 4.11, $\langle q_0, \mathbf{0} \rangle \xrightarrow{*} \langle q_f, \mathbf{0} \rangle$ in \mathcal{A}_ϕ iff $\langle q_0, \mathbf{0} \rangle \xrightarrow{*}_{\text{gainy}} \langle q_f, \mathbf{0} \rangle$ in \mathcal{A}_{inc} .

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

Step 4: This is another standard trick. Since \mathcal{A}_{dec} does not have zero-tests, we can remove all decrementing errors from a run of \mathcal{A}_{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}_{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's proof gives a bound on the length of a shortest run reaching it [Rac78] (see also [DJLL09]). 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 [Sav70] gives us the required 2EXPSpace upper bound for the satisfiability problem of LRV^\top . \square

Corollary 4.13. $\text{SAT}(\text{LRV})$ is in 2EXPSpace.

5. SIMULATING EXPONENTIALLY MANY COUNTERS

In Section 4, the reduction from $\text{SAT}(\text{LRV})$ to control state reachability for VASS involves an exponential blow up, since we use one counter for each nonempty subset of variables. The question 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 [MR09, MR12].

Here we prove that LRV is indeed powerful enough to reason about subsets of variables. We establish a 2EXPSpace lower bound. The main idea behind this lower bound proof is that the power of LRV to reason about subsets of variables can be used to simulate exponentially many counters. The lower bound is developed in three parts, with each part explained in a sub-section of this section. The first part defines chain systems, which are like VASS, except that transitions can not access counters directly, but can access a pointer to a chain of counters. 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.

5.1. Chain Systems. We introduce a new class of counter systems that is instrumental to show that $\text{SAT}(\text{LRV}^\top)$ 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 in some part of the input (see details below). Let $\text{exp}(0, n) \stackrel{\text{def}}{=} n$ and $\text{exp}(k + 1, n) \stackrel{\text{def}}{=} 2^{\text{exp}(k, n)}$ for every $k \geq 0$.

Definition 5.1. 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) $f : [1, n] \rightarrow \mathbb{N}$ where $n \geq 1$ is the *number of chains* and $\text{exp}(k, f(\alpha))$ 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

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

By convention, sometimes, we write $q \xrightarrow{\text{instr}} q'$ instead of $\langle q, \text{instr}, q' \rangle \in \delta$.

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 $\text{inc}(\alpha)$ is α -*incrementing*, and a transition containing $\text{dec}(\alpha)$ is α -*decrementing*. The idea is that for each chain $\alpha \in [1, n]$, we have $\text{exp}(k, f(\alpha))$ counters, but we cannot access them directly in the transitions as 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 ($\overline{\text{first}(\alpha)?}$) or not ($\overline{\text{first}(\alpha)?}$), or to the last counter ($\overline{\text{last}(\alpha)?}$) or not ($\overline{\text{last}(\alpha)?}$), and we can change the pointer to the next ($\overline{\text{next}(\alpha)}$) or previous ($\overline{\text{prev}(\alpha)}$) counter.

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

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

$$c_i^\alpha = \text{card}(\{i' < i \mid \rho(i') = q \xrightarrow{\text{next}(\alpha)} r\}) - \text{card}(\{i' < i \mid \rho(i') = q \xrightarrow{\text{prev}(\alpha)} r\}), \quad (5.1)$$

- (3) for every $i \in [1, |\rho|]$ and for every chain $\alpha \in [1, n]$,

- (a) if $\rho(i) = q \xrightarrow{\overline{\text{first}(\alpha)?}} q'$, then $c_i^\alpha = 0$;
(b) if $\rho(i) = q \xrightarrow{\overline{\text{first}(\alpha)?}} q'$, then $c_i^\alpha \neq 0$;
(c) if $\rho(i) = q \xrightarrow{\overline{\text{last}(\alpha)?}} q'$, then $c_i^\alpha = \mathbf{exp}(k, f(\alpha)) - 1$;
(d) if $\rho(i) = q \xrightarrow{\overline{\text{last}(\alpha)?}} q'$, then $c_i^\alpha \neq \mathbf{exp}(k, f(\alpha)) - 1$.

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

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

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* (different from ‘ends in zero’ defined below) iff for every chain $\alpha \in [1, n]$, there is some injective function

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

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 $k \geq 0$ ($\text{Per}(k)$)
INPUT: A chain system \mathcal{A} of level k .
QUESTION: Does \mathcal{A} have a <i>perfect</i> accepting run?
PROBLEM: Existence of a gainy accepting run of level $k \geq 0$ ($\text{Gainy}(k)$)
INPUT: A chain system \mathcal{A} of level k .
QUESTION: Does \mathcal{A} have a <i>gainy</i> accepting run?

$\text{Per}(k)$ is actually a control state reachability problem in VASS where the counters are encoded succinctly. $\text{Gainy}(k)$ 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.

Let $\mathcal{A} = \langle Q, f, k, Q_0, Q_F, \delta \rangle$ be a chain system 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 $\text{Per}^{\text{zero}}(k)$ and $\text{Gainy}^{\text{zero}}(k)$ to denote the variant problems of $\text{Per}(k)$ and $\text{Gainy}(k)$, respectively, in which runs that end in zero are considered. First, note that $\text{Per}^{\text{zero}}(k)$ and $\text{Per}(k)$ 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, $\text{Gainy}^{\text{zero}}(k)$ and $\text{Gainy}(k)$ are interreducible in logarithmic space.

Lemma 5.2. *For every $k \geq 0$, $\text{Per}(k)$ and $\text{Gainy}(k)$ are interreducible in logarithmic space.*

Proof. Below, we show that $\text{Per}^{\text{zero}}(k)$ and $\text{Gainy}^{\text{zero}}(k)$ 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 chain system $\widetilde{\mathcal{A}}$ of level k , where the reverse operation $\widetilde{\cdot}$ is defined on instructions, transitions, sets of transitions and systems as follows:

- $\widetilde{\text{inc}(\alpha)} \stackrel{\text{def}}{=} \text{dec}(\alpha)$; $\widetilde{\text{dec}(\alpha)} \stackrel{\text{def}}{=} \text{inc}(\alpha)$; $\widetilde{\text{next}(\alpha)} \stackrel{\text{def}}{=} \text{prev}(\alpha)$; $\widetilde{\text{prev}(\alpha)} \stackrel{\text{def}}{=} \text{next}(\alpha)$,
- $\widetilde{\text{first}(\alpha)?} \stackrel{\text{def}}{=} \text{last}(\alpha)?$; $\widetilde{\text{last}(\alpha)?} \stackrel{\text{def}}{=} \text{first}(\alpha)?$; $\widetilde{\text{first}(\alpha)?} \stackrel{\text{def}}{=} \overline{\text{last}(\alpha)?}$; $\widetilde{\text{last}(\alpha)?} \stackrel{\text{def}}{=} \overline{\text{first}(\alpha)?}$,
- $q \xrightarrow{\text{instr}} q' \stackrel{\text{def}}{=} q' \xrightarrow{\widetilde{\text{instr}}} q$,
- $\widetilde{\mathcal{A}} \stackrel{\text{def}}{=} \langle Q, f, k, Q_F, Q_0, \widetilde{\delta} \rangle$ with $\widetilde{\delta} \stackrel{\text{def}}{=} \{q \xrightarrow{\text{instr}} q' : q \xrightarrow{\text{instr}} q' \in \delta\}$. 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 4.12 (step 3).

One can show the following implications, for any run ρ for \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 $\text{Per}^{\text{zero}}(k)$ and $\text{Gainy}^{\text{zero}}(k)$ are interreducible in logarithmic space. \square

Lemma 5.3. *$\text{Per}(k)$ is in $(k + 1)\text{EXPSpace}$.*

The proof of Lemma 5.3 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.

Proof. It is sufficient to show that $\text{Per}^{\text{zero}}(k)$ is in $(k + 1)\text{EXPSpace}$.

Let $\mathcal{A} = \langle Q, f, k, Q_0, Q_F, \delta \rangle$ be a chain system with $f : [1, n] \rightarrow \mathbb{N}$. We reduce this instance of $\text{Per}^{\text{zero}}(k)$ 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 $\mathcal{O}(\exp(k, |\mathcal{A}|)^{|\mathcal{A}|})$, which provides the $(k + 1)\text{EXPSpace}$ upper bound by [Rac78]. 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

$$C' = \{c_0^1, \dots, c_{\exp(k, f(1))-1}^1, \dots, c_0^n, \dots, c_{\exp(k, f(n))-1}^n\}.$$

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

to define the transition relation δ' ($\langle \beta_1, \dots, \beta_n \rangle$ below is any element in $[0, \mathbf{exp}(k, f(1)) - 1] \times \dots \times [0, \mathbf{exp}(k, f(n)) - 1]$, unless conditions apply):

- Whenever $q \xrightarrow{\text{inc}(\alpha)} q' \in \delta$, we have $\langle q, \beta_1, \dots, \beta_n \rangle \xrightarrow{\text{inc}(c_{\beta_\alpha}^\alpha)} \langle q', \beta_1, \dots, \beta_n \rangle \in \delta'$ (and similarly with decrements),
- Whenever $q \xrightarrow{\text{next}(\alpha)} q' \in \delta$, we have

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

if $\beta_\alpha + 1 < \mathbf{exp}(k, f(\alpha))$ (and similarly with $\text{prev}(\alpha)$),

- When $q \xrightarrow{\text{first}(\alpha)?} q' \in \delta$, $\langle q, \beta_1, \dots, \beta_n \rangle \xrightarrow{\text{skip}} \langle q', \beta_1, \dots, \beta_n \rangle \in \delta'$ if $\beta_\alpha = 0$ (and similarly with $\text{first}(\alpha)?$, $\text{last}(\alpha)?$ and $\text{last}(\bar{\alpha})?$).

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 $\rho = t_1 \dots t_L$ be a run of \mathcal{A} such that for every $i \in [1, L - 1]$, $t_i = q_{i-1} \xrightarrow{\text{instr}_i} q_i$ and the values c_i^α are defined as before (5.1). One can show that $\rho' = t'_1 \dots t'_L$ with $t'_i = \langle q_{i-1}, \mathbf{x}_{i-1} \rangle \xrightarrow{\text{instr}'_i} \langle q_i, \mathbf{x}_i \rangle$ for every i , is a run of \mathcal{A}' where

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

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

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

5.2. Hardness Results for Chain Systems. We show that $\text{Per}(k)$ is $(k + 1)\text{EXPSpace-hard}$. 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 [Lip76] (see also its exposition in [Esp98]). 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.

Theorem 5.4. *Per(k) is (k + 1)EXPSpace-hard.*

Proof. We give a reduction from the control state reachability problem for the counter automata with zero tests whose counters are bounded by 2^{2^N} , where $N = \text{exp}(k, n^\gamma)$ for some $\gamma \geq 1$ and n is the size of counter automaton. Without any loss of generality, we can assume that the initial counter values are zero. The main challenge in showing the reduction to $\text{Per}(k)$ is to simulate the zero tests of the counter automaton in a chain system. The main idea, as in Lipton’s proof [Lip76], is to use the fact that counters are bounded by 2^{2^N} . For each counter c we keep another counter \bar{c} so that the sum of the values of c and \bar{c} is always 2^{2^N} . Testing c for zero is then equivalent to testing that \bar{c} has the value 2^{2^N} , which can be done by decrementing \bar{c} 2^{2^N} times. This involves extending ideas from [Lip76].

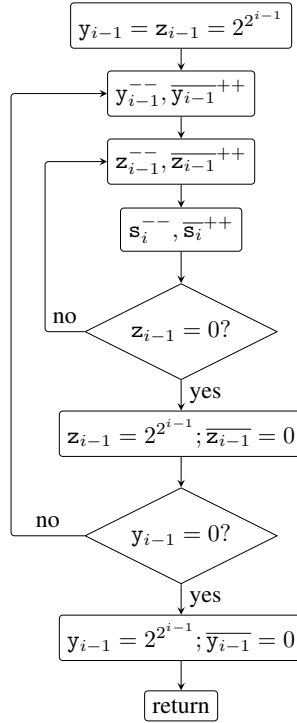
In light of this, the chain system will have two chains: **counters** and $\overline{\text{counters}}$. The value of the counter c_j of the counter automaton will be maintained in the j^{th} counter of the chain **counters**, while the value of the complement counter \bar{c}_j will be maintained in the j^{th} counter of the chain $\overline{\text{counters}}$. In the descriptions that follow, whenever we write “increment j^{th} counter of **counters**”, we implicitly assume that the increment is followed by the decrement of the j^{th} counter of $\overline{\text{counters}}$, to maintain the sum at 2^{2^N} . The transitions of the chain system are designed to ensure the following high-level structure:

- (1) Begin by setting the values of the first D counters of the chain $\overline{\text{counters}}$ to 2^{2^N} , and other initializations that will be needed for simulating zero tests. (Here, D is the number of counters in the original counter automaton.)
- (2) For every transition $\langle q, c_j \leftarrow c_j + 1, q' \rangle$ of the counter automaton, the chain system will have transitions corresponding to the following listing:
 - (a) move the pointers to j^{th} counters on **counters** and $\overline{\text{counters}}$
 : $(\text{next}(\text{counters}); \text{next}(\overline{\text{counters}}))^j$;
 (We use ‘;’ to compose transitions and ‘ $(\cdot)^j$ ’ to repeat j times a finite sequence of instructions.)
 - (b) increment the j^{th} counter on **counters**, to simulate incrementing c_j
 : $\text{inc}(\text{counters})$;
 - (c) decrement the j^{th} counter on $\overline{\text{counters}}$, to maintain the sum of j^{th} counters on **counters** and $\overline{\text{counters}}$ at 2^{2^N}
 : $\text{dec}(\overline{\text{counters}})$;
 - (d) move the pointers on **counters** and $\overline{\text{counters}}$ back to the first counter and go to q'
 : $(\text{prev}(\text{counters}); \text{prev}(\overline{\text{counters}}))^j$, goto q'
- (3) For every transition $\langle q : \text{if } c_j = 0 \text{ goto } q_1 \text{ else } c_j \leftarrow c_j - 1 \text{ goto } q_2 \rangle$ of the counter automaton, the chain system will have transitions corresponding to the following listing:
 - (a) non-deterministically guess that j^{th} counter is nonzero or zero
 q : goto nonzero or zero;
 - (b) if the guess is that j^{th} counter is nonzero; decrement it and goto q_2
nonzero: $(\text{next}(\text{counters}); \text{next}(\overline{\text{counters}}))^j$; $\text{dec}(\text{counters})$; $\text{inc}(\overline{\text{counters}})$;
 $(\text{prev}(\text{counters}); \text{prev}(\overline{\text{counters}}))^j$; goto q_2
 - (c) otherwise, the guess is that j^{th} counter is zero, hence the complement of the j^{th} counter is 2^{2^N} ; decrement the complement 2^{2^N} times, increment it back to its original value and go to q_1
zero: $(\text{next}(\text{counters}); \text{next}(\overline{\text{counters}}))^j$; [$\text{decrement } \overline{\text{counters}} \text{ } 2^{2^N} \text{ times}$; $\text{increment } \overline{\text{counters}} \text{ } 2^{2^N} \text{ times}$]; $(\text{prev}(\text{counters}); \text{prev}(\overline{\text{counters}}))^j$; goto q_1 .

In (2) above, the chain system simply imitates the counter automaton, maintaining the condition that $c_j + \bar{c}_j = 2^{2^N}$. In (3), the zero test of the counter automaton is replaced by a non-deterministic choice between **nonzero** and **zero**. The counter c_j itself can be nonzero or zero, so there are four possible cases. In the case where the nondeterministic choice coincides with the counter value, the chain system continues to simulate the counter automaton. On the other hand, if **nonzero** is chosen when $c_j = 0$, the chain system gets stuck since $(\text{next}(\mathbf{counters}); \text{next}(\overline{\mathbf{counters}}))^j; \text{dec}(\mathbf{counters})$ can not be executed (since the j^{th} counter of $\mathbf{counters}$ has value zero and can not be decremented). If **zero** is chosen when $c_j \neq 0$ (and hence $\bar{c}_j < 2^{2^N}$), the chain system also gets stuck, since (as we shall show later) the sequence of transitions $(\text{next}(\mathbf{counters}); \text{next}(\overline{\mathbf{counters}}))^j; [\text{decrement } \overline{\mathbf{counters}} 2^{2^N} \text{ times}]$ cannot be executed. In the rest of this sub-section, we will show that $[\text{decrement } \overline{\mathbf{counters}} 2^{2^N} \text{ times}; \text{increment } \overline{\mathbf{counters}} 2^{2^N} \text{ times}]$ can be implemented in such a way that there is one run that does exactly what is required and that any other run deviating from the expected behaviour gets stuck. This allows us to conclude that the final state in the given counter automaton is reachable if and only if it is reachable in the constructed chain system. \square

The basic principle for decrementing some counter 2^{2^N} times is the same one used in Lipton's proof [Lip76], which we now recall briefly. We will have two chains of counters \mathbf{s} and $\bar{\mathbf{s}}$. We will denote the counters in these chains as $\mathbf{s}_N, \mathbf{s}_{N-1}, \dots, \mathbf{s}_1$ and $\bar{\mathbf{s}}_N, \bar{\mathbf{s}}_{N-1}, \dots, \bar{\mathbf{s}}_1$ respectively. If the counter \mathbf{s}_N has the value 2^{2^N} , we describe how to decrement it 2^{2^N} times. Further, if the counter \mathbf{s}_i has the value 2^{2^i} , we describe how to decrement it 2^{2^i} times. For this, we will use four more chains $\mathbf{y}, \bar{\mathbf{y}}, \mathbf{z}$ and $\bar{\mathbf{z}}$. Our description assumes that for each i , the counters \mathbf{y}_i and \mathbf{z}_i have the value 2^{2^i} (initializing these values will be described later as part of implementing step (1) in the proof of Theorem 5.4). Decrementing $\mathbf{s}_i 2^{2^i}$ times is done by nested loops as shown in Figure 3. The outer loop is indexed by \mathbf{y}_{i-1} and the inner loop by \mathbf{z}_{i-1} . Since both \mathbf{y}_{i-1} and \mathbf{z}_{i-1} have the value $2^{2^{i-1}}$, the instruction inside the inner loop (decrementing \mathbf{s}_i) is executed $2^{2^{i-1}} \times 2^{2^{i-1}} = 2^{2^i}$ times. To implement these loops, we will need to test when \mathbf{y}_{i-1} and \mathbf{z}_{i-1} become zero. This is done the same way as above, replacing $i, i-1$ by $i-1, i-2$ respectively. These two zero tests are done recursively by the same set of transitions. After the recursive zero test, there needs to be a mechanism to determine if the recursive call was made from the inner loop or the outer loop. This mechanism is provided by a chain of counters **stack**: if the i^{th} counter in the chain **stack** has the value 1 (respectively 0), then the recursive call was made by the inner (respectively outer) loop.

We now give the implementation of instructions that follow the control state named **zero** in item (3) of page 21 above. In some intermediate configurations when the implementation is executing, the four pointers of the four chains $\mathbf{y}, \mathbf{z}, \mathbf{s}$ and **stack** will all be pointing to the i^{th} counters of their chains. In this case, we say that the system is *at stage i* . The N^{th} stage is the last one. We frequently need to move all four pointers to the next stage or previous stage, for which we introduce some macros. The macro **Nextstage** is short for the sequence of transitions $\text{next}(\mathbf{y}); \text{next}(\bar{\mathbf{y}}); \dots; \text{next}(\mathbf{stack}); \text{next}(\overline{\mathbf{stack}})$, which moves all the pointers one stage up. The macro **Prevstage** similarly moves all the pointers one stage down. The macro **transfer**($\bar{\mathbf{z}}, \mathbf{s}$) is intended to transfer the value of the counter in $\bar{\mathbf{z}}$ at current stage to the respective counter in \mathbf{s} . The macro consists of the following transitions.

Figure 3: Control flow of algorithm that decrements s_i 2^{2^i} times.

```

: transfer( $\bar{z}$ ,  $s$ )
: {
:   : subtract: inc( $z$ ); dec( $\bar{z}$ ); inc( $s$ ); dec( $\bar{s}$ ); /* subtract a count from  $\bar{z}$  and add it to  $s$  */
:   : goto subtract or exit macro /* non-deterministically choose to repeat another step of the count transfer
:     or to exit the macro */
: }

```

The macro **transfer**(\bar{y} , s) is similar to the above, with y and \bar{y} replacing z and \bar{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: $\text{next}(z)$; $\text{inc}(z)$; $\text{prev}(z)$. The macros **inc(next(y))** and **inc(next(\bar{s}))** are similar to **inc(next(z))** with y and \bar{s} replacing z respectively. The macro **dec(next(s))** is similarly defined to decrement the next counter in the chain s . The macro **stack == 1** tests if the counter in the current stage in the chain **stack** is greater than 0: $\text{dec}(\text{stack})$; $\text{inc}(\text{stack})$.

Following is the set of transitions at the control state **zero** in (3) in the proof of Theorem 5.4 above. 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 system. 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: $\text{dec}(z)$; $\text{inc}(z)$; goto innerloop2” actually represents the set of transitions $\{\langle \text{innernonzero2}, \text{dec}(z), q \rangle, \langle q, \text{inc}(z), \text{innerloop2} \rangle\}$. The instruction “ q : if $\text{first}(\alpha)$? then {program 1} else {program 2}” represents the set of transitions

$$\{\langle q, \text{first}(\alpha)?, q_1 \rangle, \langle q, \overline{\text{first}(\alpha)}?, q_2 \rangle\} \cup \\ \{\text{transitions for program 1 from } q_1\} \cup \{\text{transitions for program 2 from } q_2\}.$$

An instruction of the form “ $q : \text{inc}(\overline{\text{stack}}); \text{goto outernonzero2}$ or outerzero2 ” represents the set of transitions $\{\langle q, \text{inc}(\overline{\text{stack}}), \text{outernonzero2} \rangle, \langle q, \text{inc}(\overline{\text{stack}}), \text{outerzero2} \rangle\}$. 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.

The action [decrement $\overline{\text{counters}}$ 2^{2^N} times] in item (3) of page 21 is implemented in Chain system 5.1 by “ $\text{transfer}(\overline{\text{counters}}, \text{s}); \text{goto Dec}_{\text{zerorep}}$ ” in the first line. There is a non-deterministic choice for exiting “ $\text{transfer}(\overline{\text{counters}}, \text{s})$ ”, and we wish to block any run that exits before incrementing s 2^{2^N} times. This is done by “ $\text{goto Dec}_{\text{zerorep}}$ ”, which forces the chain system to decrement s 2^{2^N} times.

```

: zero: (next(counters); next( $\overline{\text{counters}}$ ))j; transfer( $\overline{\text{counters}}$ , s); goto Deczerorep
: zerorep: transfer(counters, s); goto Deczeropass
: zeropass: (prev(counters); prev( $\overline{\text{counters}}$ ))j; goto q1
: Dec(address):
: if first(stack)? then
:   : (dec(s))4; (inc( $\overline{\text{s}}$ ))4; goto DecFinished
: else
:   : Prevstage
:   : outerloop2: dec(y); inc( $\overline{\text{y}}$ ) /* y is the index for outer loop */
:   :   : innerloop2: dec(z); inc( $\overline{\text{z}}$ ) /* z is the index for inner loop */
:   :     : dec(next(s)); inc(next( $\overline{\text{s}}$ ))
:   :     : innertest2: goto innernonzero2 or innerzero2
:   :     : innernonzero2: dec(z); inc(z); goto innerloop2 /* inner loop not yet complete */
:   :     : innerzero2: transfer( $\overline{\text{z}}$ , s); inc(stack); dec(stack); goto Dec(address) /* inner loop
:   :       complete. (i - 1)th counter of stack is set to 1, so the recursive call to Dec(address) returns to outertest2
:   :       */
:   :     : outertest2: dec(stack); inc( $\overline{\text{stack}}$ ); goto outernonzero2 or outerzero2
:   :     : outernonzero2: dec(y); inc(y); goto outerloop2 /* outer loop not yet complete */
:   :     : outerzero2: transfer( $\overline{\text{y}}$ , s); goto Dec(address) /* outer loop complete. (i - 1)th counter of stack is
:   :       set to 0, so the recursive call to Dec(address) returns to outerexit2 */
:   :     : outerexit2: Nextstage; goto DecFinished
: fi
: DecFinished: goto backtrack
:
: backtrack:
: if ( $\overline{\text{last}}(\text{stack})?$ ) then /* is the pointer at some intermediate stage, which means we just completed a recursive
:   call? */
:   : if (stack == 1) then goto outertest2 /* ith counter of stack is set to 1, so return to outertest2 */
:   : else goto outerexit2 /* ith counter of stack is set to 0, so return to outerexit2 */
: else /* pointer is at the last stage, decrementation is complete */
:   : goto <address>
: fi

```

Chain system 5.1: Decrement j^{th} counter in $\overline{\text{counters}}$ 2^{2^N} times.

Suppose, as in the proof of Theorem 5.4, we want to simulate the $c_j = 0$ case of the counter automaton's transition $\langle q : \text{if } c_j = 0 \text{ goto } q_1 \text{ else } c_j \leftarrow c_j - 1 \text{ goto } q_2 \rangle$. The action [increment $\overline{\text{counters}}$ 2^{2^N} times] in the proof of Theorem 5.4 is implemented above by “transfer(counters, s); goto Dec_{zeropass}” in the second line. If Dec_{zeropass} returns successfully, the control can go to q_1 as required. The only difference between Dec_{zerorep} and Dec_{zeropass} is the return address **zerorep** and **zeropass**. A generic Dec_{\langle address \rangle} is shown above, which implements the algorithm shown in Figure 3. Formally, every instruction label (such as **outerloop2**, **innerloop2** etc.) should also be parameterized with $\langle \text{address} \rangle$, but these have been omitted for the sake of readability. The case “if first(stack)?” implements the base case $i = 1$. If $i > 1$, the else branch is taken, where the first instruction is to move all pointers one stage down, so that they now point to y_{i-1} and z_{i-1} as required by the algorithm of Figure 3. The loop between **outerloop2** and **outerzero2** implements the outer loop of Figure 3 and the loop between **innerloop2** and **innerzero2** implements the inner loop of Figure 3. The non-deterministic choice in **innertest2** decides whether to continue the inner loop or to exit. The macro transfer(\overline{z} , s) at the beginning of **innerzero2** will reset the $(i - 1)^{\text{th}}$ counter of \overline{z} back to $2^{2^{i-1}}$, at the same time setting $(i - 1)^{\text{th}}$ counter of s to $2^{2^{i-1}}$. So the run that behaves as expected increments \overline{z}_{i-1} exactly $2^{2^{i-1}}$ times. All other runs are blocked by the recursive call to Dec_{\langle address \rangle} at the end of **innerzero2**. The purpose of inc(stack); dec(stack); in the middle of **innerzero2** is to ensure that the recursive call to Dec_{\langle address \rangle} returns to the correct state. After similar tests in **outerzero2**, “Nextstage” at the beginning of **outerexit2** moves all the pointers back to stage i . Then in **backtrack**, the correct return state is figured out. Exactly how this is done will be clear in the proof of the following lemma, where we construct runs of the chain system by induction on stage.

Lemma 5.5. *In the state **zero** in the listing above, if the j^{th} counter in the chain counters has the value 0, there is a run that reaches the state q_1 without causing any changes. Any run from the state **zero** will be blocked if the j^{th} counter in the chain counters has a value other than 0.*

Proof. We first prove the existence of a run reaching q_1 when the j^{th} counter in the chain counters has the value 0. The j^{th} counter in the chain $\overline{\text{counters}}$ will have the value 2^{2^N} . Our run executes transfer($\overline{\text{counters}}$, s) to set counters_j and s_N to 2^{2^N} and set $\overline{\text{s}}_N$ and $\overline{\text{counters}}_j$ to 0. With these values in the counters, we will prove next that there is run from Dec_{zerorep} to **zerorep** that sets s_N to 0 and $\overline{\text{s}}_N$ to 2^{2^N} . From **zerorep**, our run continues to execute transfer(counters, s) to set counters_j and $\overline{\text{s}}_N$ to 0 and set $\overline{\text{counters}}_j$ and s_N to 2^{2^N} . Then there is again a run from Dec_{zeropass} to **zeropass** which sets s_N to 0 and sets $\overline{\text{s}}_N$ to 2^{2^N} . From **zeropass**, our run moves the pointers on the chains **counters** and $\overline{\text{counters}}$ back to the first position and goes to state q_1 .

Now suppose that in the state Dec_{\langle address \rangle}, s_N is set to 2^{2^N} , $\overline{\text{s}}_N$ is set to 0 and the pointer in the chain **stack** is at the last position, as mentioned in the previous paragraph. We will prove that there is a run that reaches $\langle \text{address} \rangle$, setting s_N to 0 and $\overline{\text{s}}_N$ to 2^{2^N} . We will in fact prove the following claim by induction on i .

Claim: Suppose that in the state Dec_{\langle address \rangle}, s_i is set to 2^{2^i} , $\overline{\text{s}}_i$ is set to 0 and the pointers in the chains **stack**, s, y, z are at position i . There is a run ρ_i that reaches **DecFinished**, setting s_i to 0 and $\overline{\text{s}}_i$ to 2^{2^i} , with the pointers in the same position.

When we first call Dec_{zerorep} or Dec_{zeropass}, the pointers in the chains **stack**, s, y, z are at the last position (N). Hence, when the run ρ_N given by the above claim reaches

DecFinished, it can continue to the state **backtrack** and then to **zerorep** or **zeropass** respectively.

Base case of the claim: $i = 1$. The run ρ_1 is as follows: if $\text{first}(\mathbf{s})?$ then $(\text{dec}(\mathbf{s}))^4; (\text{inc}(\overline{\mathbf{s}}))^4; \text{goto DecFinished}$.

Induction step of the claim: The run ρ_{i+1} is as follows.

```

: Prevstage
: outerloop2: dec(y); inc( $\overline{y}$ )
  : innerloop2: dec(z); inc( $\overline{z}$ )
    : dec(next(s)); inc(next( $\overline{s}$ ))
    : innertest2: goto innernonzero2  $2^{2^i} - 1$  times, then goto innerzero2
    : innernonzero2: dec(z); inc(z); goto innerloop2
  : innerzero2: transfer( $\overline{z}$ , s); inc(stack); dec( $\overline{\text{stack}}$ ); goto Dec(address) /* Follow  $\rho_i$  here.
    Since we set  $\text{stack}_i$  to 1, we can return to outertest2 at the end of  $\rho_i$  to continue our  $\rho_{i+1}$  */
  : outertest2: dec(stack); inc( $\overline{\text{stack}}$ ); goto outernonzero2  $2^{2^i} - 1$  times, then goto
    outerzero2
  : outernonzero2: dec(y); inc(y); goto outerloop2
: outerzero2: transfer( $\overline{y}$ , s); goto Dec(address) /* Follow  $\rho_i$  here. Since now  $\text{stack}_i$  is 0, we can return
  to outerexit2 at the end of  $\rho_i$  to continue our  $\rho_{i+1}$  */
: outerexit2: Nextstage; goto DecFinished

```

This completes the induction step and the proof of the claim.

Finally, we will prove that any run from **zero** will be blocked if the j^{th} counter in the chain **counters** has a value other than 0. Indeed, the j^{th} counter in the chain $\overline{\text{counters}}$ has a value less than 2^{2^N} when the state is **Dec_{zerorep}**. Hence no run can execute **innerloop2** and **outerloop2** $2^{2^{N-1}}$ times. Hence, any run will either get blocked when the pointer in the chain **s** is still at position N , or it will go to state **Dec_(address)** with \mathbf{s}_{N-1} less than $2^{2^{N-1}}$. In the latter case, we can argue in the same way to conclude that any run is blocked either when the pointer in the chain **s** is still at $N - 1$ or it will go to state **Dec_(address)** with \mathbf{s}_{N-2} less than $2^{2^{N-2}}$ and so on. Any run is blocked at some stage between N and 1. \square

Next, we explain the implementation of step (1) in the proof of Theorem 5.4. We show 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 the required initial values:

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

The 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 later 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 $\text{last}(\text{init})?$ is true if the pointer in the chain **init** is at stage N . The macros **Nextstage** and **Prevstage** moves the pointers of the chain **init**, in addition to all the other chains.

We now give the code used to initialize y , z , s and 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. The counters y_0 , z_0 , s_0 and stack_0 are initialized directly. For the counters at higher stages, we again use nested loops similar to those shown in Figure 3, assuming that counters at lower stages have been initialized. These nested loops are implemented between **innerinit**–**innerzero1** and **outerinit**–**outerzero1**. The zero tests involved in terminating these loops are performed by the same decrementing algorithm, assuming that counters at lower stages are already initialized. This will introduce some complications while backtracking from the decrementing algorithm — we have to figure out whether the call to **Dec** was from **innerzero1** or **outerzero1**, or from within **Dec** itself from a recursive call. To handle this, the “backtrack” part of the code below has been updated to check if $(\mathit{next}(\mathit{init}) == 1)$: if so, we are inside some recursive call to **Dec**. Otherwise, the call to **Dec** was from one of the loops initializing a higher level.

```

: begininit:  $(\mathit{inc}(y))^4; (\mathit{inc}(z))^4; (\mathit{inc}(\bar{s}))^4; \mathit{inc}(\mathit{init}); \mathit{inc}(\overline{\mathit{stack}})$ 
: initialize: If  $\mathit{last}(\mathit{init})?$  goto beginsim else goto outerinit
: outerinit:  $\mathit{dec}(y); \mathit{inc}(\bar{y})$  /*  $y$  is the index for outer loop */
  : innerinit:  $\mathit{dec}(z); \mathit{inc}(\bar{z})$  /*  $z$  is the index for inner loop */
    : INC:  $\mathit{inc}(\mathit{next}(y)); \mathit{inc}(\mathit{next}(z)); \mathit{inc}(\mathit{next}(\bar{s}))$ 
    : innertest1: goto innernonzero1 or innerzero1
    : innernonzero1:  $\mathit{dec}(z); \mathit{inc}(z);$  goto innerinit /* inner loop not yet complete */
  : innerzero1:  $\mathit{transfer}(\bar{z}, s); \mathit{inc}(\mathit{stack}); \mathit{dec}(\overline{\mathit{stack}});$  goto Dec /* inner loop complete.
    ( $\mathit{next}(\mathit{init}) == 0$ ) and ( $\mathit{stack} == 1$ ), so Dec returns to outertest1 */
  : outertest1:  $\mathit{dec}(\mathit{stack}); \mathit{inc}(\overline{\mathit{stack}});$  goto outernonzero1 or outerzero1
  : outernonzero1:  $\mathit{dec}(y); \mathit{inc}(y);$  goto outerinit /* outer loop not yet complete */
: outerzero1:  $\mathit{transfer}(\bar{y}, s);$  goto Dec /* outer loop complete. ( $\mathit{next}(\mathit{init}) == 0$ ) and ( $\mathit{stack} == 0$ ), so
  Dec returns to outerexit1 */
: outerexit1: Nextstage;  $\mathit{inc}(\mathit{init}); \mathit{inc}(\overline{\mathit{stack}});$  goto initialise
: beginsim: start simulating the counter machine (see part of the proof of Theorem 5.4
  related to the simulation)
:
: Dec:
: if  $\mathit{first}(\mathit{init})?$  then
  :  $(\mathit{dec}(s))^4; (\mathit{inc}(\bar{s}))^4;$  goto DecFinished
: else
  : Prevstage
  : outerloop2:  $\mathit{dec}(y); \mathit{inc}(\bar{y})$  /*  $y$  is the index for outer loop */
    : innerloop2:  $\mathit{dec}(z); \mathit{inc}(\bar{z})$  /*  $z$  is the index for inner loop */
      :  $\mathit{dec}(\mathit{next}(s)); \mathit{inc}(\mathit{next}(\bar{s}))$ 
      : innertest2: goto innernonzero2 or innerzero2
      : innernonzero2:  $\mathit{dec}(z); \mathit{inc}(z);$  goto innerloop2 /* inner loop not yet complete */
    : innerzero2:  $\mathit{transfer}(\bar{z}, s); \mathit{inc}(\mathit{stack}); \mathit{dec}(\overline{\mathit{stack}});$  goto Dec /* inner loop complete.
      ( $\mathit{next}(\mathit{init}) == 1$ ) and ( $\mathit{stack} == 1$ ), so Dec returns to outertest2 */
    : outertest2:  $\mathit{dec}(\mathit{stack}); \mathit{inc}(\overline{\mathit{stack}});$  goto outernonzero2 or outerzero2
    : outernonzero2:  $\mathit{dec}(y); \mathit{inc}(y);$  goto outerloop2 /* outer loop not yet complete */
  : outerzero2:  $\mathit{transfer}(\bar{y}, s);$  goto Dec /* outer loop complete. ( $\mathit{next}(\mathit{init}) == 1$ ) and ( $\mathit{stack} == 0$ ), so Dec returns 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 ideas involved in the above listing are similar to the ones in the code decrementing counters $_j$ 2^{2^N} times.

Lemma 5.6. *Suppose the control state is **begininit**, all the pointers in all the chains are at stage 1 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. If a run starts from **begininit** as above, it will not reach **beginsim** unless all the counters have been properly initialized.*

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 in the proof of Lemma 5.5.

```

: initialise: If last(init)? goto beginsim else goto outerinit /* pointer in chain init is at  $i \neq N$ ; go
to outerinit */
: outerinit: dec(y); inc( $\bar{y}$ )
: innerinit: dec(z); inc( $\bar{z}$ )
:   : INC: inc(next(y)); inc(next(z)); inc(next( $\bar{s}$ ))
:   : innertest1: goto innernonzero1  $2^{2^i} - 1$  times then goto innerzerol
:   : innernonzero1: dec(z); inc(z); goto innerinit /* inner loop not yet complete */
:   : innerzerol: transfer( $\bar{z}$ , s); inc(stack); dec( $\overline{\text{stack}}$ ); goto Dec /* follow  $\rho_i$  here; since
(next(init) == 0) and stack == 1, we can rerun to outertest1 to continue our  $\rho'_{i+1}$  */
:   : outertest1: dec(stack); inc( $\overline{\text{stack}}$ ); goto outernonzero1  $2^{2^i} - 1$  times then goto out-
erzerol
:   : outernonzero1: dec(y); inc(y); goto outerinit
: outerzerol: transfer( $\bar{y}$ , s); goto Dec /* follow  $\rho_i$  here; since (next(init) == 0) and stack == 0, we
can rerun to outerexit1 to continue our  $\rho'_{i+1}$  */
: outerexit1: Nextstage; inc(init); inc( $\overline{\text{stack}}$ ); goto initialise

```

This completes the induction step, proving the existence of a run ending at **initialize** as required.

Finally we argue that any run from **begininit** that does not initialize all the counters properly will get stuck. Indeed, the only way out of the initialization code is to go to **beginsim**, which is only reachable via **initialize**. From **initialize**, we can go to **beginsim** only when the pointers are at stage N . We finish the argument by showing that any run that visits **initialize** for the first time with pointers at stage i would have initialized all

counters at or below stage i . The only way to go to **initialize** with pointers at stage i is via **begininit** (for $i = 1$) or via **outerexit1** (for $i > 1$). It is clear in the case of $i = 1$ that the counters at stage 1 are properly initialized. In the case of $i > 1$, the only way to reach **outerexit1** is via **Dec**. In this case, all runs are forced to iterate the loops at **outerinit** and **innerinit** the correct number of times as we have seen in the proof of Lemma 5.5, which will again force all the counters at or below stage i to be properly initialized. \square

To complete the proof of Theorem 5.4, we finally show how to initialize the first $D \leq n$ counters from **counters** and $\overline{\text{counters}}$ that correspond to the counters of the original counter automaton (point (4) of the initialization values listed after Lemma 5.5). This is achieved by replacing the code for **INC** from the code for the initialization phase by the following one:

```

: INC: inc(next(y)); inc(next(z)); inc(next( $\bar{s}$ ))
: next(init)
  if last(init)? then
  : inc( $\overline{\text{counters}}$ )
  : (next( $\overline{\text{counters}}$ ); inc( $\overline{\text{counters}}$ ))D-1
  : (prev( $\overline{\text{counters}}$ ))D-1
  prev(init)

```

This finishes the proof of Theorem 5.4.

5.3. Reasoning About Chain Systems with LRV. Given a chain system \mathcal{A} of level 1, we construct a polynomial-size 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 5.7. *There is a polynomial-time reduction from Gainy(1) into SAT(LRV(X, F)).*

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

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 $\delta = \{t_1, \dots, t_m\}$ with the variables $\mathfrak{t}_0, \dots, \mathfrak{t}_m$, where a node has an encoding of the label t_i iff the formula $\langle t_i \rangle \stackrel{\text{def}}{=} \mathfrak{t}_0 \approx \mathfrak{t}_i$ 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 $\langle t \rangle$ for some unique $t \in \delta$.
- (b) The first position satisfies $\langle (q_0, \text{instr}, q) \rangle$ for some $q_0 \in Q_0$, $\text{instr} \in I$, $q \in Q$.
- (c) The last position satisfies $\langle (q, \text{instr}, q') \rangle$ for some $q \in Q$, $\text{instr} \in I$, $q' \in Q_F$.
- (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 \mathbf{x} , and variables $\mathbf{x}_{inc}^\alpha, \mathbf{x}_{dec}^\alpha, \mathbf{x}_i^\alpha$ for every chain α and for every $i \in [1, f(\alpha)]$. Let us fix the bijections $\chi^\alpha : [0, 2^{f(\alpha)} - 1] \rightarrow 2^{[1, f(\alpha)]}$ 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 in a run is α -*incrementing* [resp. α -*decrementing*] if it satisfies $\langle (q, u, q') \rangle$ for some $q, q' \in Q$ and $u = \text{inc}(\alpha)$ [resp. $u = \text{dec}(\alpha)$].

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

$$X_i = \{b \in [1, f(\alpha)] \mid \sigma(i)(\mathbf{x}) = \sigma(i)(\mathbf{x}_b^\alpha)\}. \quad (5.2)$$

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

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

- (1) Any two positions of σ have different values of \mathbf{x}_{inc}^α [resp. of \mathbf{x}_{dec}^α].
- (2) For every position i of σ operating on an α -counter \mathbf{c} containing an instruction ‘first(α)?’ [resp. ‘first(α)?’], ‘last(α)?’], ‘last(α)?’], we have $\mathbf{c} = 0$ [resp. $\mathbf{c} \neq 0$, $\mathbf{c} = 2^{f(\alpha)} - 1$, $\mathbf{c} \neq 2^{f(\alpha)} - 1$].
- (3) For every position i of σ operating on an α -counter \mathbf{c} ,
 - if the position contains an instruction ‘next(α)’ [resp. ‘prev(α)’], then the next position $i + 1$ operates on the α -counter $\mathbf{c} + 1$ [resp. $\mathbf{c} - 1$],
 - otherwise, the position $i + 1$ operates on the α -counter \mathbf{c} .
- (4) For every α -incrementing position i of σ operating on an α -counter \mathbf{c} there is a future α -decrementing position $j > i$ operating on the same α -counter \mathbf{c} , such that $\sigma(i)(\mathbf{x}_{inc}^\alpha) = \sigma(j)(\mathbf{x}_{dec}^\alpha)$.

Claim 5.8. 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 \mathbf{x}_{dec}^α , 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 \mathbf{x}_{inc}^α , 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. (5.2)). 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

$$\gamma_\sigma^\alpha : \{0 \leq i < |\sigma| : i \text{ is } \alpha\text{-incrementing}\} \rightarrow \{0 \leq i < |\sigma| : i \text{ is } \alpha\text{-decrementing}\}$$

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

Consider $\rho \in \delta^*$ as having the same length as σ and such that $\rho(i) = (q, \text{instr}, q')$ whenever $\sigma, i \models \langle (q, \text{instr}, q') \rangle$. From conditions (2) and (3), it follows that the counter c_i^α corresponding to position i from ρ , is equal to $\chi^{-1}(X_i)$. Therefore, the functions $\{\gamma_\sigma^\alpha\}_{\alpha \in [1, n]}$ 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

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

where for every $\gamma^\alpha(i) = j$ we have that $j > i$ and $c_i^\alpha = c_j^\alpha$.

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 $\mathfrak{d}, \mathfrak{e}$. For any position i , we have $\sigma(i)(\mathfrak{t}_0) = \mathfrak{d}$; and $\sigma(i)(\mathfrak{t}_j) = \mathfrak{d}$ if the i th transition in ρ is t_j and $\sigma(i)(\mathfrak{t}_j) = \mathfrak{e}$ otherwise. In this way we make sure that the properties ((a))–((d)) hold.

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

- For every α , $\sigma(i)(\mathbf{x}_{inc}^\alpha) = \mathfrak{d}_i$.
- For every α ,
 - if i is α -decrementing, we define $\sigma(i)(\mathbf{x}_{dec}^\alpha) = \sigma(i')(\mathbf{x}_{inc}^\alpha)$ where $i' = (\gamma^\alpha)^{-1}(i)$; if such $(\gamma^\alpha)^{-1}(i)$ does not exist, then $\sigma(i)(\mathbf{x}_{dec}^\alpha) = \mathfrak{d}'_i$;
 - otherwise, if i is not α -decrementing, $\sigma(i)(\mathbf{x}_{dec}^\alpha) = \mathfrak{d}'_i$.
- $\sigma(i)(\mathbf{x}) = \mathfrak{d}$.
- For every α , $\sigma(i)(\mathbf{x}_l^\alpha) = \sigma(i)(\mathbf{x}) = \mathfrak{d}$ if $l \in \chi(c_i^\alpha)$, otherwise $\sigma(i)(\mathbf{x}_l^\alpha) = \mathfrak{e}$.

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 \mathbf{x}_{inc}^α . Let us show that the same holds for \mathbf{x}_{dec}^α . Note that at any position i , \mathbf{x}_{dec}^α has: the data value of $\sigma(i')(\mathbf{x}_{inc}^\alpha) = \mathfrak{d}_{i'}$ for some $i' < i$; or the data value \mathfrak{d}'_i . If there were two positions $i < j$ with $\sigma(i)(\mathbf{x}_{dec}^\alpha) = \sigma(j)(\mathbf{x}_{dec}^\alpha)$ it would then be because: (i) $\mathfrak{d}'_i = \mathfrak{d}'_j$; (ii) $\mathfrak{d}_{i'} = \mathfrak{d}_{j'}$ for some $i' < i$; (iii) $\mathfrak{d}'_i = \mathfrak{d}_{j'}$ for some $j' < j$; or (iv) $\mathfrak{d}_{i'} = \mathfrak{d}_{j'}$ for some $i' < i$, $j' < j$. It is evident that none of (i), (ii), (iii) can hold since all $\mathfrak{d}_0, \dots, \mathfrak{d}_{|\sigma|-1}, \mathfrak{d}'_0, \dots, \mathfrak{d}'_{|\sigma|-1}$ 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 \mathbf{x}_{dec}^α . Therefore, σ has the property (1).

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

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

Claim 5.9. 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 for any $i \in [1, f(\alpha)]$,

$$bit_i^\alpha \stackrel{\text{def}}{=} \mathbf{x} \approx \mathbf{x}_i^\alpha,$$

where bit_1^α represents the most significant bit. Now, we 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 \mathbf{x}_{inc}^α and \mathbf{x}_{dec}^α to have a different data value for every position of the model with the formula

$$\neg F \mathbf{x}_{inc}^\alpha \approx \langle \top? \rangle \mathbf{x}_{inc}^\alpha \wedge \neg F \mathbf{x}_{dec}^\alpha \approx \langle \top? \rangle \mathbf{x}_{dec}^\alpha.$$

- 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 (the least significant bit is the $f(\alpha)$ th one).

$$flip_i^\alpha \stackrel{\text{def}}{=} \neg bit_i^\alpha \wedge \bigwedge_{j>i} bit_j^\alpha$$

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

$$copy_i^\alpha \stackrel{\text{def}}{=} \bigwedge_{j < i} (bit_j^\alpha \Leftrightarrow X(bit_j^\alpha))$$

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

$$zero_i^\alpha \stackrel{\text{def}}{=} \bigwedge_{j > i} X(\neg bit_j^\alpha)$$

And finally by swapping the bit i .

$$swap_i^\alpha \stackrel{\text{def}}{=} X bit_i^\alpha$$

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

$$\bigwedge_{(q, \text{next}(\alpha), q') \in \delta} \left(\langle (q, \text{next}(\alpha), q') \rangle \Rightarrow \bigwedge_{i \in [1, n]} (flip_i^\alpha \Rightarrow copy_i^\alpha \wedge zero_i^\alpha \wedge swap_i^\alpha) \right).$$

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

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

$$\mathbf{x}_{inc}^\alpha \approx \langle \alpha\text{-dec?} \rangle \mathbf{x}_{dec}^\alpha \quad \text{where } \alpha\text{-dec} = \bigvee_{(q, \text{dec}(\alpha), q') \in \delta} \langle (q, \text{dec}(\alpha), q') \rangle$$

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

$$bit_i^\alpha \Rightarrow \mathbf{x}_{inc}^\alpha \approx \langle bit_i^{\alpha?} \rangle \mathbf{x}_{dec}^\alpha \wedge \neg bit_i^\alpha \Rightarrow \mathbf{x}_{inc}^\alpha \approx \langle \neg bit_i^{\alpha?} \rangle \mathbf{x}_{dec}^\alpha.$$

If the α -increment has some data value \mathfrak{d} at variable \mathbf{x}_{inc}^α , there must be only one future position j where \mathbf{x}_{dec}^α carries the data value \mathfrak{d} —since every \mathbf{x}_{dec}^α 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). \square

As a corollary of Claims 5.8 and 5.9 we obtain a polynomial-time reduction from Gainy(1) into the satisfiability problem for LRV(\mathbf{X}, \mathbf{F}). \square

6. 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. We consider infinite data words (instead of finite data words), finite sets of MSO-definable temporal operators (instead of $\mathbf{X}, \mathbf{X}^{-1}, \mathbf{S}, \mathbf{U}$) and also repetitions of pairs of values (instead of repetitions of single values).

6.1. 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 may lead to undecidability: for instance, freeze LTL with a single register is decidable over finite data words whereas it is undecidable over infinite data words [DL09]. By contrast, FO^2 over finite data words or over infinite data words is decidable [BMS⁺06]. Let $\text{SAT}_\omega(\cdot)$ be the variant of the satisfiability problem $\text{SAT}(\cdot)$ in which infinite models of length ω are taken into account instead of finite ones. The satisfaction relation for LRV, PLRV, LRV^\top , etc. on ω -models is defined accordingly.

Proposition 6.1.**(I):** $\text{SAT}_\omega(\text{PLRV})$ is decidable.**(II):** $\text{SAT}_\omega(\text{LRV})$ is 2EXPSpace-complete.

Proof. (I) The developments of Section 4.1 apply to the infinite case, and since $\text{SAT}_\omega(\text{PLRV}^\top)$ is shown decidable in [DDG12], we get decidability of $\text{SAT}_\omega(\text{PLRV})$.

(II) First, note that $\text{SAT}_\omega(\text{LRV})$ is 2EXPSpace-hard. Indeed, there is a simple logarithmic-space reduction from $\text{SAT}(\text{LRV})$ into $\text{SAT}_\omega(\text{LRV})$, which can be performed as for standard LTL. Indeed, it is sufficient to introduce two new variables \mathbf{x}_{new} and \mathbf{y}_{new} , to state that $\mathbf{x}_{new} \approx \mathbf{y}_{new}$ is true at a finite prefix of the model (herein $\mathbf{x}_{new} \approx \mathbf{y}_{new}$ plays the role of a new propositional variable) and to relativize all the temporal operators and obligations to positions on which $\mathbf{x}_{new} \approx \mathbf{y}_{new}$ holds true.

Concerning the complexity upper bound, from Section 4.1, we can conclude that there is a polynomial-time reduction from $\text{SAT}_\omega(\text{LRV})$ into $\text{SAT}_\omega(\text{LRV}^\top)$. The satisfiability problem $\text{SAT}_\omega(\text{LRV}^\top)$ is shown decidable in [DDG12] and we can adapt developments from Section 4.2 to get also a 2EXPSpace upper bound for $\text{SAT}_\omega(\text{LRV}^\top)$. This is the purpose of the rest of the proof.

By analyzing the constructions from [DDG12, Section 7], one can show that ϕ of LRV^\top built over the variables $\{\mathbf{x}_1, \dots, \mathbf{x}_k\}$ is ω -satisfiable iff there are $Z \subseteq \mathcal{P}^+(\{\mathbf{x}_1, \dots, \mathbf{x}_k\})$, 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 respectively and a Büchi automaton \mathcal{B}^Z such that:

- (1) \mathcal{A}_ϕ^Z is the restriction of \mathcal{A}_ϕ defined in Section 4.2 and $\langle q_0, \mathbf{0} \rangle \xrightarrow{*} \langle q_f, \mathbf{0} \rangle$ in \mathcal{A}_ϕ^Z for some $q_0 \in Q_0$ and $q_f \in Q_f$.
- (2) \mathcal{B}^Z accepts a non-empty language.

By arguments similar to those from Section 4.2, 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 [DDG12, Section 7], counters in $\mathcal{P}^+(\{\mathbf{x}_1, \dots, \mathbf{x}_k\}) \setminus Z$ are also updated in \mathcal{A}_ϕ^Z (providing the VASS \mathcal{A}_ϕ) 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 $\mathcal{P}^+(\{\mathbf{x}_1, \dots, \mathbf{x}_k\}) \setminus Z$. \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 subsets 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 [Sav70]. \square

6.2. Adding MSO-Definable Temporal Operators. It is standard to extend the linear-time temporal logic LTL with MSO-definable temporal operators (see e.g. [Wol83, GK03]), and the same can be done with LRV. For this, one considers MSO formulas over a linear order $<$; that is, a structure $\mathbb{A} = (A, <)$ contains only one (binary) relational symbol $<$ and it is determined (modulo isomorphism) by the size of A . Let $\mathbb{A}_n = (\{0, \dots, n-1\}, <)$ [resp. $\mathbb{A}_\omega = (\mathbb{N}, <)$] be the linear order $0 < \dots < n-1$ [resp. $0 < 1 < \dots$]. A temporal operator \oplus of arity n is *MSO-definable* whenever there is an MSO($<$) formula $\phi(\mathbf{x}, P_1, \dots, P_n)$ with a unique free position variable \mathbf{x} and with n free unary predicates P_1, \dots, P_n such that

$$\sigma, i \models \oplus(\psi_1, \dots, \psi_n) \text{ iff } \mathbb{A}_{|\sigma|} \models \phi(i, X_1, \dots, X_n),$$

where ψ_j is a formula of LRV extended with \oplus and $X_j = \{i \mid \sigma, i \models \psi_j\}$ for every j , see e.g. [GK03]. The 2EXPSPACE upper bound is preserved with a fixed finite set of MSO-definable temporal operators.

Theorem 6.2. *Let $\{\oplus_1, \dots, \oplus_N\}$ be a finite set of MSO-definable temporal operators. Satisfiability problem for LRV extended with $\{\oplus_1, \dots, \oplus_N\}$ is 2EXPSPACE-complete.*

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 of LTL (see e.g. [GK03]):

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

All these 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 4.1, it is straightforward to show that there is a logarithmic-space reduction from the satisfiability problem for $\text{LRV} + \{\oplus_1, \dots, \oplus_N\}$ into the satisfiability problem for $\text{LRV}^\top + \{\oplus_1, \dots, \oplus_N\}$.
- (2) By combining [GK03] and [DDG12, Theorem 4] (see also Appendix A), a formula ϕ in $\text{LRV}^\top + \{\oplus_1, \dots, \oplus_N\}$ 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}_ϕ such that the number of states is exponential in $|\phi|$. The relatively small size for \mathcal{A}_ϕ is due to the fact that \mathcal{A}_ϕ 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 [GK03] (see also more details in Section 4.2).

By using arguments from Section 4.2, 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}_ϕ , whence the 2EXPSPACE upper bound. \square

Note that PLRV augmented with MSO-definable temporal operators is decidable too. Indeed, it can be translated into PLRV^\top augmented with MSO-definable temporal operators by adapting the developments from Section 4.1. Then, the satisfiability problem of this latter logic can be translated in the reachability problem for VASS as done in [DDG12, Theorem 4] except that finite-state automata are built according to [GK03].

6.3. 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 [LMS02]. The satisfaction relation is parameterised by the current position of the origin and past-time temporal operators use that position. For instance, given $i, o \in \mathbb{N}$ with $o \leq i$ (o is the position of the origin),

$$\begin{aligned} \sigma, i \models_o \text{Now } \phi &\stackrel{\text{def}}{\iff} \sigma, i \models_i \phi \\ \sigma, i \models_o \phi_1 \mathbf{S} \phi_2 &\stackrel{\text{def}}{\iff} \text{there is } j \in [o, i] \text{ such that } \sigma, j \models_o \phi_2 \text{ and} \\ &\text{for all } j' \in [j - 1, i], \text{ we have } \sigma, j' \models_o \phi_1. \end{aligned}$$

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 and past-time operators are interpreted relatively to that position.

Theorem 6.3. *SAT(LRV + Now) is 2EXPSpace-complete.*

Proof. Again, 2EXPSpace-hardness is inherited from LRV. In order to get the 2EXPSpace, we use arguments similar to those from forthcoming Proposition 6.4. Indeed, the decidability proof from [DDG12, Theorem 4] (see also Appendix A) 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 $\mathcal{A}_{\text{dec}} = \langle Q, C, \delta \rangle$ 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}_{dec} is bounded by 2^k , $\text{card}(Q)$ is double exponential in $|\phi|$ and the maximal value for an update in a transition of \mathcal{A}_{dec} is k . Indeed, a formula from Past LTL+Now is equivalent to a Büchi automaton of double exponential size in its size [LMS02]. 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 [Rac78] (see also [DJLL09]), 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 $p(|\mathcal{A}_{\text{dec}}| + \max(\mathcal{A}_{\text{dec}}))^{f(k)}$ where $p(\cdot)$ is a polynomial, f is a map of double exponential growth and $\max(\mathcal{A}_{\text{dec}})$ denotes the maximal absolute value in an update (bounded by k presently). In order to take advantage of the results on VAS (vector addition system –without states–), we use the translation from VASS to VAS introduced in [HP79]: 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}_{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 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. \square

This contrasts with the undecidability results from [KSZ10, Theorem 5] in presence of the operator **Now**. In [KSZ10, Theorem 5], the logic has two sorts of formulae: position formulae from class formulae (a class being a sequence of positions with the same data value). It is a more expressive formalism that can navigate both the word in the usual way, as well as its data classes.

6.4. 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 LRV^\top , i.e., when test formulas ϕ are restricted to \top in $\mathbf{x} \approx \langle \phi? \rangle \mathbf{y}$. Let LRV_k [resp. LRV_k^\top , PLRV_k^\top] be the restriction to formulas with at most k variables. In [DDG12, 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 4.12 and the fact that the control state reachability problem for VASS with at most k counters (where k is a constant) is in PSPACE.

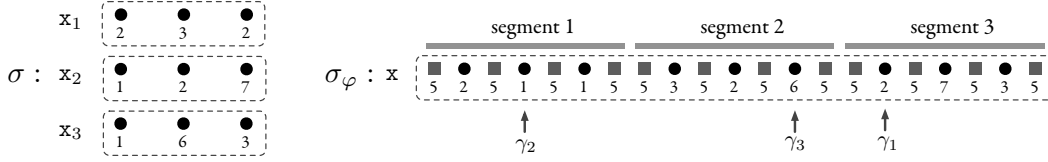


Figure 4: Example of the reduction from $\text{SAT}(\text{LRV}^\top)$ into $\text{SAT}(\text{LRV}_1)$, for $k = 3$, $N = 3$ and $\mathfrak{d} = 5$

Proposition 6.4. *For every $k \geq 1$, $\text{SAT}(\text{LRV}_k^\top)$ is PSPACE-complete.*

Proof. PSPACE-hardness is due to the fact that LTL with a single propositional variable is PSPACE-hard [DS02], which can be easily simulated with the atomic formula $\mathbf{x} \approx X\mathbf{x}$. Let $k \geq 1$ be some fixed value and $\phi \in \text{LRV}_k^\top$. In the proof of Theorem 4.12, we have seen that there exist a VASS $\mathcal{A}_{\text{dec}} = \langle Q, C, \delta \rangle$ 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}_{dec} is bounded by 2^k , $\text{card}(Q)$ is exponential in $|\phi|$ and the maximal value for an update in a transition of \mathcal{A}_{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 [Rac78] (see also [DJLL09]), 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 $p(|\mathcal{A}_{\text{dec}}| + \max(\mathcal{A}_{\text{dec}}))^{f(k)}$ where $p(\cdot)$ is a polynomial, f is a map of double exponential growth and $\max(\mathcal{A}_{\text{dec}})$ 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 $p(|\mathcal{A}_{\text{dec}}| + \max(\mathcal{A}_{\text{dec}}))^{f(k)}$ from $\langle q_f, \mathbf{0} \rangle$ to some $\langle q_0, \mathbf{v}' \rangle$. Counter valuations can be represented in polynomial space too. By Savitch's Theorem [Sav70], we conclude that the satisfiability problem for LRV_k^\top is in PSPACE. \square

This does not imply that LRV_k is in PSPACE, since the reduction from LRV into LRV^\top in Section 4.1 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 2EXSPACE -hard, by the following reduction.

Lemma 6.5. *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. The idea of the coding is the following. Suppose we have a formula $\varphi \in \text{LRV}^\top$ using k variables $\mathbf{x}_1, \dots, \mathbf{x}_k$ so that $\sigma \models \varphi$.

We will encode σ in a model σ_φ that encodes in only one variable, say \mathbf{x} the whole model of σ restricted to $\mathbf{x}_1, \dots, \mathbf{x}_k$. 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 \mathfrak{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 “ $\mathfrak{d} \mathfrak{d}_1 \mathfrak{d} \mathfrak{d}_2 \dots \mathfrak{d} \mathfrak{d}_k \mathfrak{d}$ ”, where $\mathfrak{d}_j = \sigma(i)(\mathbf{x}_j)$. Figure 4 contains an example for $k = 3$ and $N = 3$. In fact, we can force that 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 \mathbf{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

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

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

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

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 $j \in [0, k' - 1]$, $\sigma_\varphi(i + j)(\mathbf{x}) = \sigma_\varphi(i)(\mathbf{x})$ 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

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

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

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

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 4.

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

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

Now we have to translate $\mathbf{x}_i \approx \langle \top? \rangle \mathbf{x}_j$. 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

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

does not exactly encode this property. For example, consider that we would like to test $\mathbf{x}_2 \approx \langle \top? \rangle \mathbf{x}_3$ at the first element of the model σ_φ depicted in Figure 4. Although $\xi_{2,3}$ holds, the property is not true, there is no *future* segment with the data value 1 in the position encoding \mathbf{x}_3 . In fact, the formula ξ encodes correctly the property only when $\mathbf{x}_i \not\approx \mathbf{x}_j$.

However, this is not a problem since when $\mathbf{x}_i \approx \mathbf{x}_j$ the formula $\mathbf{x}_i \approx \langle \top? \rangle \mathbf{x}_j$ is equivalent to $\mathbf{x}_j \approx \langle \top? \rangle \mathbf{x}_i$. We can then translate the formula as follows.

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

Recall that $\mathbf{x}_i \not\approx \langle \top? \rangle \mathbf{x}_j$ is not translated since such formulae can be eliminated. We then define $\varphi' = \text{many-segments} \wedge tr(\varphi)$.

Claim 6.6. 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 $\sigma' \models \text{many-segments}$ 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 $\sigma(i)(\mathbf{x}_j)$ is the data value of the $2j$ -th position of the i -th segment of σ' . In this model, we have that $\sigma \models \varphi$. \square

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

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

Therefore, there is also a reduction from PLRV^\top into PLRV_1 . \square

Corollary 6.7. *For all $k \geq 1$, $\text{SAT}(\text{LRV}_k)$ is 2EXPSPACE -complete.*

Corollary 6.8. *For all $k \geq 1$, $\text{SAT}(\text{PLRV}_k)$ is as hard as $\text{Reach}(\text{VASS})$.*

6.5. The Power of Pairs of Repeating Values. Let us consider the last variant of LRV in which the repetition of tuples of values is possible. Such an extension amounts to introducing additional variables in a first-order setting. This may lead to undecidability since 3 variables are enough for undecidability, see e.g. [BDM⁺11]. However, LRV makes a restricted use of variables, leading to 2EXPSPACE -completeness. There might be hope that LRV augmented with repetitions of pairs of values have a reasonable computational cost. Consider an extension to LRV with atomic formulas of the form $(\mathbf{x}_1, \dots, \mathbf{x}_k) \approx \langle \phi? \rangle (\mathbf{y}_1, \dots, \mathbf{y}_k)$ where $\mathbf{x}_1, \dots, \mathbf{x}_k, \mathbf{y}_1, \dots, \mathbf{y}_k \in \text{VAR}$. This extends $\mathbf{x} \approx \langle \varphi? \rangle \mathbf{y}$ by testing whether the vector of data values from the variables $(\mathbf{x}_1, \dots, \mathbf{x}_k)$ of the current position coincides with that of $(\mathbf{y}_1, \dots, \mathbf{y}_k)$ in a future position.

The semantics are extended accordingly:

$$\sigma, i \models (\mathbf{x}_1, \dots, \mathbf{x}_k) \approx \langle \varphi? \rangle (\mathbf{y}_1, \dots, \mathbf{y}_k) \quad \text{iff} \quad \text{there exists } j \text{ such that } i < j < |\sigma|, \sigma, j \models \varphi, \\ \text{and } \sigma(i)(\mathbf{x}_l) = \sigma(j)(\mathbf{y}_l) \text{ for every } l \in [1, k].$$

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), see below. In order to code solutions of PCP instances, we adapt a proof technique used in [BDM⁺11] for first-order logic with two variables and two equivalence relations on words. However, our proof uses only *future* modalities (unlike the proof of [BDM⁺11, Proposition 27]) and no past obligations (unlike the proof of [KSZ10, 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 .

Theorem 6.9. $\text{SAT}(\text{LRV}_{vec}(\mathbf{X}, \mathbf{U}))$ is undecidable.

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 , $n \in \mathbb{N}$, $u_1, \dots, u_n, v_1, \dots, v_n \in \Sigma^+$, $ u_1 , u_2 \leq 2$ and $ v_1 , v_2 \geq 3$.
QUESTION:	Are there indices $1 \leq i_1, \dots, i_m \leq n$ so that $u_{i_1} \cdots u_{i_m} = v_{i_1} \cdots v_{i_m}$, $i_1 \in \{1, 2\}$, $ u_{i_1} \cdots u_{i_m} $ is even and 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 $k' > k$?

Lemma 6.10. *The MPCP^{dir} problem is undecidable.*

This is a corollary of the undecidability proof for PCP as done in [HMU01]. By reducing MPCP^{dir} to satisfiability for LRV_{vec}, we get undecidability.

Proof. In [HMU01], 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 generalisation 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 $u_1, \dots, u_n, v_1, \dots, v_n$ make up our instance of MPCP^{dir}.

In the encoding of the halting problem from the cited work, $|u_i| \leq 2$ 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 $u_1, \dots, u_n, v_1, \dots, v_n \in \Sigma^*$ 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. \square

Given an instance pcp of MPCP^{dir}, we construct an LRV_{vec}(X, U) formula ϕ_{pcp} such that pcp has a solution iff ϕ_{pcp} is satisfiable. To do so, we adapt a proof technique from [BDM⁺11, KSZ10] but we need to provide substantial changes in order to fit our logic. Moreover, none of the results in [BDM⁺11, KSZ10] allow to derive our main undecidability result since we use neither past-time temporal operators nor past obligations of the form $\langle \mathbf{x}, \mathbf{x}' \rangle \approx \langle \varphi? \rangle^{-1} \langle \mathbf{y}, \mathbf{y}' \rangle$.

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 $u_{i_1} \bar{v}_{i_1} \cdots u_{i_m} \bar{v}_{i_m}$. To check that such a model σ actually represents a solution, we will write $\text{LRV}_{vec}(\mathbf{X}, \mathbf{U})$ 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 \mathbf{x} and \mathbf{y} . Positions 1 and 2 of σ_Σ will have the same value of \mathbf{x} , positions 2 and 3 will have the same value of \mathbf{y} , positions 3 and 4 will have the same value of \mathbf{x} and so on. Generalising, odd positions of σ_Σ will have the same value of \mathbf{x} as in the next position of σ_Σ and the same value of \mathbf{y} as in the previous position of σ_Σ . Even positions of σ_Σ will have the same value of \mathbf{x} as in the previous position of σ_Σ and the same value of \mathbf{y} as in the next position of σ_Σ . These constraints allow us to chain the positions of σ corresponding to σ_Σ . To easily identify odd and even positions, an additional label is introduced at each position, which can be O or E . These labels encoding the parity status of the positions in σ_Σ will be helpful to write formulae. The sequence σ_Σ looks as follows.

$$\left[\begin{array}{c} \mathbf{x} \\ \text{Odd/Even} \\ \text{Letter} \\ \mathbf{y} \end{array} \right] : \left(\begin{array}{c} \mathfrak{d}_1 \\ O_{in} \\ a_1 \\ \mathfrak{d}'_1 \end{array} \right) \left(\begin{array}{c} \mathfrak{d}_1 \\ E \\ a_2 \\ \mathfrak{d}'_2 \end{array} \right) \left(\begin{array}{c} \mathfrak{d}_3 \\ O \\ a_3 \\ \mathfrak{d}'_2 \end{array} \right) \left(\begin{array}{c} \mathfrak{d}_3 \\ E \\ a_4 \\ \mathfrak{d}'_4 \end{array} \right) \cdots \left(\begin{array}{c} \mathfrak{d}_{m-1} \\ O \\ a_{m-1} \\ \mathfrak{d}'_{m-2} \end{array} \right) \left(\begin{array}{c} \mathfrak{d}_{m-1} \\ E_{fi} \\ a_m \\ \mathfrak{d}'_m \end{array} \right) \quad (\star)$$

The \mathfrak{d}_i 's and \mathfrak{d}'_i 's are data values for the variables \mathbf{x} and \mathbf{y} respectively. Each label is actually a pair in $\Sigma \times \{O, O_{in}, E, E_{fi}\}$ to record the letter from Σ and the parity status of the position. The letter O identifies an odd position (a sequence starts here from the first position) and the first position is identified by the special letter O_{in} . Similarly, the letter E identifies an even position and the last position is identified by the special letter E_{fi} . Similarly, in order to define the sequence $\sigma_{\bar{\Sigma}}$, we consider the letters \bar{O} , \bar{E} , \bar{O}_{in} and \bar{E}_{fi} with analogous intentions. So, by way of example, each position of σ_Σ is labelled by two letters: a letter in Σ and a letter specifying the parity of the position.

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_u^i 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., $\phi_{abb}^3 \stackrel{\text{def}}{=} \mathbf{X}^3 a \wedge \mathbf{X}^4 b \wedge \mathbf{X}^5 b$). 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 $(u_1 \bar{v}_1 + u_2 \bar{v}_2) \{u_i \bar{v}_i \mid i \in [1, n]\}^*$: to facilitate writing this condition in LRV, we introduce two new variables $\mathbf{z}_b^1, \mathbf{z}_b^2$ such that at any position, they have the same value only if that position is not the starting point of some pair $u_i \bar{v}_i$.

$$\mathbf{z}_b^1 \not\approx \mathbf{z}_b^2 \wedge \bigvee_{i \in \{1,2\}} (\phi_{u_i}^0 \wedge \phi_{\bar{v}_i}^{|u_i|+1} \wedge (\mathbf{X}^{|u_i v_i|+1} \top \Rightarrow \mathbf{X}^{|u_i v_i|+1} \mathbf{z}_b^1 \not\approx \mathbf{z}_b^2))$$

$$\wedge \mathbf{G} \left(\mathbf{z}_b^1 \approx \mathbf{z}_b^2 \vee \bigvee_{i \in [1, n]} (\phi_{u_i}^0 \wedge \phi_{\bar{v}_i}^{|u_i|+1}) \wedge (\mathbf{X}^{|u_i v_i|+1} \top \Rightarrow \mathbf{X}^{|u_i v_i|+1} \mathbf{z}_b^1 \not\approx \mathbf{z}_b^2) \right).$$

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

$$\begin{aligned} & \mathbf{G}(\Sigma \Rightarrow (\neg \mathbf{x} \approx \langle \Sigma? \rangle \mathbf{x}) \vee (\mathbf{x} \approx \langle \Sigma \wedge (\neg \mathbf{x} \approx \langle \Sigma? \rangle \mathbf{x}?) \rangle \mathbf{x})) \\ & \wedge \mathbf{G}(\Sigma \Rightarrow (\neg \mathbf{y} \approx \langle \Sigma? \rangle \mathbf{y}) \vee (\mathbf{y} \approx \langle \Sigma \wedge (\neg \mathbf{y} \approx \langle \Sigma? \rangle \mathbf{y}?) \rangle \mathbf{y})). \end{aligned}$$

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 $O_{in}(EO)^*E_{fi}$.

$$\begin{aligned} & O_{in} \wedge \mathbf{G}((O \vee O_{in}) \Rightarrow \bar{\Sigma}\mathbf{U}(\Sigma \wedge (E \vee E_{fi}))) \wedge \mathbf{G}(E \Rightarrow \bar{\Sigma}\mathbf{U}(\Sigma \wedge O)) \\ & \wedge \mathbf{G}(E_{fi} \Rightarrow \neg \mathbf{X}\mathbf{F}\Sigma). \end{aligned}$$

Note that O_{in} [resp. E_{fi}] is useful to identify the first odd position [resp. last even position]. $\sigma_{\bar{\Sigma}}$ projected onto $\{\bar{O}, \bar{O}_{in}, \bar{E}, \bar{E}_{fi}\}$ is in $\bar{O}_{in}(\bar{E}\bar{O})^*\bar{E}_{fi}$.

$$\begin{aligned} & \Sigma\mathbf{U}\bar{O}_{in} \wedge \mathbf{G}((\bar{O} \vee \bar{O}_{in}) \Rightarrow \Sigma\mathbf{U}(\bar{\Sigma} \wedge (\bar{E} \vee \bar{E}_{fi}))) \wedge \mathbf{G}(\bar{E} \Rightarrow \Sigma\mathbf{U}(\bar{\Sigma} \wedge \bar{O})) \\ & \wedge \mathbf{G}(\bar{E}_{fi} \Rightarrow \neg \mathbf{X}\mathbf{T}). \end{aligned}$$

- (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 $\sigma_\Sigma(i)(\mathbf{x}) = \sigma_\Sigma(j)(\mathbf{x})$.

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

For every position i of $\sigma_{\bar{\Sigma}}$ labelled by \bar{O} or \bar{O}_{in} , there exists a future position $j > i$ of $\sigma_{\bar{\Sigma}}$ labelled by \bar{E} or \bar{E}_{fi} so that $\sigma_{\bar{\Sigma}}(i)(\mathbf{x}) = \sigma_{\bar{\Sigma}}(j)(\mathbf{x})$ (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 $\sigma_\Sigma(i)(\mathbf{y}) = \sigma_\Sigma(j)(\mathbf{y})$.

$$\mathbf{G}(E \Rightarrow \mathbf{y} \approx \langle \Sigma \wedge O? \rangle \mathbf{y}).$$

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 $\sigma_{\bar{\Sigma}}(i)(\mathbf{y}) = \sigma_{\bar{\Sigma}}(j)(\mathbf{y})$ (enforced with a formula similar to the one above).

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

$$\begin{aligned} & \left(\bigvee_{a \in \Sigma} (O_{in} \wedge a) \Rightarrow \langle \mathbf{x}, \mathbf{y} \rangle \approx \langle \bar{O}_{in} \wedge \bar{a}? \rangle \langle \mathbf{x}, \mathbf{y} \rangle \right) \\ & \wedge \mathbf{X}\Sigma \Rightarrow \bigvee_{a \in \Sigma} \mathbf{X}a \wedge \mathbf{X}^3\bar{a} \wedge \mathbf{X}(\mathbf{x} \approx \mathbf{X}^2\mathbf{x} \wedge \mathbf{y} \approx \mathbf{X}^2\mathbf{y}). \end{aligned}$$

- (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 $\sigma(i)(\mathbf{x}) = \sigma(j)(\mathbf{x})$ and

$\sigma(i)(y) = \sigma(j)(y)$. The following formula assumes that $|u_{i_1}| \leq 2$, as it is in MPCP^{dir}.

$$\begin{aligned} \mathbf{X}\Sigma &\Rightarrow \Sigma\mathbf{U} \left(\mathbf{X}^2\mathbf{G} \bigvee_{\bar{a} \in \bar{\Sigma}} ((\bar{a} \wedge \neg \overline{E_{f_i}}) \Rightarrow (\mathbf{x}, y) \approx \langle a? \rangle(\mathbf{x}, y)) \right) && (\mathbf{X}\Sigma \text{ is true when } |u_{i_1}| = 2) \\ \wedge \mathbf{X}\bar{\Sigma} &\Rightarrow \Sigma\mathbf{U} \left(\mathbf{X}\mathbf{G} \bigvee_{\bar{a} \in \bar{\Sigma}} ((\bar{a} \wedge \neg \overline{E_{f_i}}) \Rightarrow (\mathbf{x}, y) \approx \langle a? \rangle(\mathbf{x}, y)) \right) && (\mathbf{X}\bar{\Sigma} \text{ is true when } |u_{i_1}| = 1) \end{aligned}$$

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

$$\mathbf{G} \left(\bigvee_{a \in \Sigma} (E_{f_i} \wedge a) \Rightarrow \langle \mathbf{x}, y \rangle \approx \langle \overline{E_{f_i}} \wedge \bar{a}? \rangle \langle \mathbf{x}, y \rangle \right).$$

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

Lemma 6.11. *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 $\sigma_\Sigma(i)(\mathbf{x}) = \sigma_\Sigma(j)(\mathbf{x})$ or $\sigma_\Sigma(i)(y) = \sigma_\Sigma(j)(y)$. 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 indegree 1 (except the first one, which has indegree 0) and that all positions have outdegree 1 (except the last one, which has outdegree 0). Indeed, condition (2a) ensures that for any position, both the indegree and outdegree are at most 1. From conditions (2b), (2c) and (2d), each position (except the last one) has outdegree at least 1. Hence, all positions except the last one have outdegree exactly 1. By the definition of the set of edges, the last position has outdegree 0. If more than one position has indegree 0, it will force some other position to have indegree more than 1, which is not possible. By the definition of the set of edges, the first position has indegree 0 and hence, all other positions have indegree 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 indegree 2, which is not possible.

Next we will prove that there cannot be two positions of σ_Σ with the same value for variables \mathbf{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 \mathbf{x} , so there are three positions with the same value for variable \mathbf{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 \bar{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 . Suppose position i of $\sigma_{\bar{\Sigma}}$ is labeled by \bar{O} (the case of \bar{E} is symmetric). Then by the 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). \square

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

The reduction from $\text{SAT}(\text{LRV})$ to $\text{SAT}(\text{LRV}^\top)$ can be easily adapted to lead to a reduction from $\text{SAT}(\text{LRV}_{vec})$ to $\text{SAT}(\text{LRV}_{vec}^\top)$, whence $\text{SAT}(\text{LRV}_{vec}^\top)$ is undecidable too.

7. 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.

7.1. Two-variable Logics. We study a fragment of $\text{EMSO}^2(+1, <, \sim)$ 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 $\text{EMSO}^2(+1, <, \sim)$ is known to be decidable, equivalent to the reachability problem for VASS [BDM⁺11], with no known primitive-recursive algorithm. Here we show a large fragment with elementary complexity.

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

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

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, <, ~). In fact, *forward-EMSO*²(+1, <, ~) captures $\text{EMSO}^2(+1, <)$ (i.e., all

regular languages on the finite labeling of the data word).¹ However, it seems to be strictly less expressive than $\text{EMSO}^2(+1, <, \sim)$, since *forward-EMSO*²(+1, <, \sim) does not appear to be able to express the property *there are exactly two occurrences of every data value*, which can be easily expressed in $\text{EMSO}^2(+1, <, \sim)$. 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 $\text{EMSO}^2(<, \sim)$.

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

Proposition 7.1. *The satisfiability problem for forward-EMSO²(+1, <, \sim) is in 3EXPSPACE.*

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

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

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 $\text{EMSO}^2(+1, <, \sim)$ [BDM⁺11] adapted to *forward-EMSO*²(+1, <, \sim), and can be done in the same way.

We now give an exponential-time translation $tr : \text{forward-EMSO}^2(+1, <, \sim) \rightarrow \text{LRV}$. 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 4.13). This yields an upper bound of 3EXPSPACE for *forward-EMSO*²(+1, <, \sim).

The translation makes use of: a distinguished variable \mathbf{x} that encodes the data values of any data word satisfying φ ; variables $\mathbf{x}_0, \dots, \mathbf{x}_n$ that are used to encode the monadic relations X_1, \dots, X_n ; and a variable \mathbf{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

$$\bigwedge_{m \in M} \neg \exists x \exists y (x \leq y \wedge \chi_m \wedge \chi_m^x \wedge \chi_m^y), \quad (7.1)$$

where $M \subseteq \mathbb{N}$, and 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 7.2. $\forall x \forall y \chi$ can be translated into an equivalent formula of the form (7.1) in exponential time.

Proof. As an example, if

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

then the corresponding formula would be

$$\bigwedge_{\mu} \neg \exists x \exists y (x \leq y \wedge \mu \wedge x \sim y \wedge \neg X(x) \wedge \neg X(y)).$$

¹Indeed, 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 $(\exists y . X_{fst}(y)) \wedge (\forall x \forall y . x < y \rightarrow \neg X_{fst}(y))$.

for all $\mu \in \{+1(x, y), \neg(+1(x, y) \vee x=y)\}$. 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

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

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. \square

We define $tr(\chi_m^x)$ as the conjunction of all the formulas $\mathbf{x}_0 \approx \mathbf{x}_i$ so that $X_i(x)$ is a conjunct of χ_m^x , and all the formulas $\neg(\mathbf{x}_0 \approx \mathbf{x}_i)$ 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

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

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

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

We proceed similarly for $\mu = +1(x, y)$, $\nu = \neg(x \sim y)$; and the translation is of course \perp (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 $tr(\chi_m^x) \wedge \mathbf{x} \approx \langle tr(\chi_m^y) \rangle \mathbf{x}$ 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 \mathbf{x}_{prev} , ensuring that it always has the data value of the variable \mathbf{x} in the previous position

$$prev = \mathbf{G}(\mathbf{X}\top \Rightarrow \mathbf{x} \approx \mathbf{X}\mathbf{x}_{prev}).$$

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

$$tr(\chi_m^x) \wedge \mathbf{x} \approx \langle \mathbf{x}_{prev} \approx \langle tr(\chi_m^y) \rangle \mathbf{x} \rangle \mathbf{x}_{prev}.$$

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

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

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

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

Thus,

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

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 \mathbf{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, \dots, 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 , $\sigma(i)(\mathbf{x})$ is the data value of position i from the data word, and $\sigma(i)(\mathbf{x}_0) = \sigma(i)(\mathbf{x}_j)$ iff X_j holds at position i of the data word, and
 - for every position $i > 0$, $\sigma(i)(\mathbf{x}_{prev}) = \sigma(i-1)(\mathbf{x})$,
we have that $\sigma \models tr(\varphi)$.

By Corollary 4.13 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 $forward\text{-EMSO}^2(+1, <, \sim)$. \square

Remark 7.3. The proof above can be also extended to work with a similar fragment of $EMSO^2(+1, \dots, +k, <, \sim)$, 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 y is i positions after x . 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 $forward\text{-EMSO}^2(+1, <, \sim)$ and $forward\text{-EMSO}^2(+1, \dots, +k, <, \sim)$ can be improved.

Our result stating that $PLRV_1$ is equivalent to reachability in VASS (Corollary 6.8), 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 7.4. *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 $\Sigma = \emptyset$.*

Remark 7.5. The proof above can be also extended to work with a similar fragment of $EMSO^2(+1, \dots, +k, <, \sim)$, 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 y is i positions after x . 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 $forward\text{-EMSO}^2(+1, <, \sim)$ and $forward\text{-EMSO}^2(+1, \dots, +k, <, \sim)$ can be improved.

Our result stating that $PLRV_1$ is equivalent to reachability in VASS (Corollary 6.8), 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 7.6. *The satisfiability problem for $\text{FO}^2(<, \sim, \{+k\}_{k \in \mathbb{N}})$ is as hard as the reachability problem in VASS, even when restricted to having an alphabet $\Sigma = \emptyset$.*

7.2. Temporal Logics. Consider a temporal logic with (strict) future operators $F_=$ and F_{\neq} , so that $F_= \varphi$ (resp. $F_{\neq} \varphi$) 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 $X^k_=$ and X^k_{\neq} for any $k \in \mathbb{N}$, where $X^k_= \varphi$ (resp. $X^k_{\neq} \varphi$) 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 $\text{LTL}(U, F_=, F_{\neq}, \{X^k_=, X^k_{\neq}\}_{k \in \mathbb{N}})$. There is an efficient satisfiability-preserving translation from $\text{LTL}(U, F_=, F_{\neq}, \{X^k_=, X^k_{\neq}\}_{k \in \mathbb{N}})$ into LRV and back and hence we have the following.

Proposition 7.7. *The satisfiability problem for $\text{LTL}(U, F_=, F_{\neq}, \{X^k_=, X^k_{\neq}\}_{k \in \mathbb{N}})$ is 2EXPSPACE -complete.*

Proof sketch. For the 2EXPSPACE -membership, there is a straightforward polynomial-time translation from $\text{LTL}(U, F_=, F_{\neq}, \{X^k_=, X^k_{\neq}\}_{k \in \mathbb{N}})$ into LRV that preserves satisfiability, where $F_= \varphi$ is translated as $\mathbf{x} \approx \langle \varphi' ? \rangle \mathbf{x}$ and $F_{\neq} \varphi$ is translated as $\mathbf{x} \not\approx \langle \varphi' ? \rangle \mathbf{x}$; $X^k_= \varphi$ as $X^k \varphi' \wedge \mathbf{x} \approx X^k \mathbf{x}$; and any test for label a_i by $\mathbf{x}_0 \approx \mathbf{x}_i$ (φ' is the translation of φ).

On the other hand, given a formula ϕ of LRV^\top using k variables, consider the alphabet $\{a_1, \dots, a_k\}$ and the formula ψ_k that forces the model to be a succession of ‘blocks’ of data words of length k with labels $a_1 \dots a_k$ (hence, forcing its length to be a multiple of k). We show how to define a formula $\text{tr}(\phi)$ of $\text{LTL}(U, F_=, F_{\neq}, \{X^k_=, X^k_{\neq}\}_{k \in \mathbb{N}})$ so that $\text{tr}(\phi) \wedge \psi_k$ is satisfiable if and only if ϕ is satisfiable. We define $\text{tr}(X \phi')$ as $X^k \text{tr}(\phi')$; and $\text{tr}(F \phi')$ as $F(a_1 \wedge \text{tr}(\phi'))$. For future obligation formulas, we define $\text{tr}(\mathbf{x}_i \approx \langle \top ? \rangle \mathbf{x}_j)$ as $X^{i-1} F_= a_j$ if $j \leq i$ or $X^{i-1} (X^j_{\neq} F_= a_j \vee (F_= a_j \wedge X^j_{\neq} \top))$ otherwise —note that in the second case special care must be taken to ensure that the data value is repeated at a strictly future block. Finally, for the local obligation formulas, we define $\text{tr}(\mathbf{x}_i \approx X^t \mathbf{x}_j)$ as $X^{i-1} X^{t \cdot k - (i-1) + (j-1)} \top$. Note that this can be easily extended to treat inequalities. Thus, there is a polynomial-time reduction from the satisfiability problem for LRV^\top into that of $\text{LTL}(U, F_=, F_{\neq}, \{X^k_=, X^k_{\neq}\}_{k \in \mathbb{N}})$. \square

In fact, $\text{LTL}(U, F_=, F_{\neq}, \{X^k_=, X^k_{\neq}\}_{k \in \mathbb{N}})$ 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 LTL_1^\downarrow , and it was studied in [DL09]. This logic contains one operator to store the current datum, one operator to test 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 [DL09].

Indeed $\text{LTL}(U, F_=, F_{\neq}, \{X^k_=, X^k_{\neq}\}_{k \in \mathbb{N}})$ is the fragment where we only allow \downarrow and \uparrow to appear in the form of $\downarrow F(\uparrow \wedge \varphi)$ and $\downarrow X^k(\uparrow \wedge \varphi)$ —or with $\neg \uparrow$ instead of \uparrow . Markedly, this

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

Figure 5: Summary of results.

restriction allows us to jump from a non-primitive-recursive complexity of the satisfiability problem, to an elementary 2EXPSPACE complexity.

8. CONCLUSION

We introduced the logic LRV that significantly extends the languages in [DDG12], for instance by allowing future obligations of the general form $x \approx \langle \phi? \rangle y$. We have shown that $\text{SAT}(\text{LRV})$ can be reduced to the control state reachability problem in VASS, obtaining a 2EXPSPACE upper bound as a consequence. Since LRV can be also viewed as a fragment of a logic introduced in [KSZ10] whose satisfiability is equivalent to $\text{Reach}(\text{VASS})$, we provide also an interesting fragment with elementary complexity. The reduction into the control state reachability problem involves an exponential blow-up, which is unavoidable as demonstrated by our 2EXPSPACE lower bound. To prove this lower bound, we introduced the class of chain systems of level k and we proved the $(k + 1)\text{EXPSPACE}$ -completeness of the control state reachability problem by extending the proof from [Lip76, Esp98]. This class of systems is interesting for its own sake and could be used to establish other hardness results thanks to our results. We have also shown that the proof technique we used to reduce LRV^\top to the control state reachability problem does not work in the presence of past obligations. Indeed, the satisfiability problem for PLRV^\top (LRV^\top with past obligations) is as hard as $\text{Reach}(\text{VASS})$ which witnesses that past obligations have a computational cost. Furthermore, note that none of our lower bound proofs involve the past-time operators X^{-1} and S . Finally, a new correspondence between data logics and decision problems for VASS is provided, apart from the fact that the complexity of several data logics has been characterised, some of the logics being defined with first-order features (see Section 7). A summary of the results can be found in Figure 5.

REFERENCES

- [AJ96] P. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, 1996.
- [AvW12] R. Alur, P. Černý, and S. Weinstein. Algorithmic analysis of array-accessing programs. *ACM Transactions on Computational Logic*, 13(3), 2012.
- [BCGK12] B. Bollig, A. Cyriac, P. Gastin, and K. Narayan Kumar. Model checking languages of data words. In *FoSSaCS'12*, volume 7213 of *Lecture Notes in Computer Science*, pages 391–405. Springer, 2012.
- [BDM⁺11] M. Bojańczyk, C. David, A. Muscholl, Th. Schwentick, and L. Segoufin. Two-variable logic on data words. *ACM Transactions on Computational Logic*, 12(4):27, 2011.
- [BL10] M. Bojańczyk and S. Lasota. An extension of data automata that captures XPath. In *LICS'10*, pages 243–252. IEEE, 2010.

- [BMS⁺06] M. Bojańczyk, A. Muscholl, Th. Schwentick, L. Segoufin, and C. David. Two-variable logic on words with data. In *LICS'06*, pages 7–16. IEEE, 2006.
- [Bol11] B. Bollig. An automaton over data words that captures EMSO logic. In *CONCUR'11*, volume 6901 of *Lecture Notes in Computer Science*, pages 171–186. Springer, 2011.
- [Bou02] P. Bouyer. A logical characterization of data languages. *Information Processing Letters*, 84(2):75–85, 2002.
- [Dav09] C. David. *Analyse de XML avec données non-bornées*. PhD thesis, LIAFA, 2009.
- [DDG12] S. Demri, D. D'Souza, and R. Gascon. Temporal logics of repeating values. *Journal of Logic and Computation*, 22(5):1059–1096, 2012.
- [DFP13] S. Demri, D. Figueira, and M. Praveen. Reasoning about data repetitions with counter systems. In *LICS'13*, pages 33–42. IEEE, 2013.
- [DHLT14] N. Decker, P. Habermehl, M. Leucker, and D. Thoma. Ordered navigation on multi-attributed data words. In *CONCUR'14*, volume 8704 of *Lecture Notes in Computer Science*, pages 497–511, 2014.
- [DJLL09] S. Demri, M. Jurdziński, O. Lachish, and R. Lazić. The covering and boundedness problems for branching vector addition systems. In *FST&TCS'09*, pages 181–192. LZI, 2009.
- [DL09] S. Demri and R. Lazić. LTL with the freeze quantifier and register automata. *ACM Transactions on Computational Logic*, 10(3), 2009.
- [DS02] S. Demri and Ph. Schnoebelen. The complexity of propositional linear temporal logics in simple cases. *Information and Computation*, 174(1):84–103, 2002.
- [Esp98] J. Esparza. Decidability and complexity of Petri net problems — an introduction. In *Advances in Petri Nets 1998*, volume 1491 of *Lecture Notes in Computer Science*, pages 374–428. Springer, Berlin, 1998.
- [Fig10] D. Figueira. *Reasoning on words and trees with data*. PhD thesis, ENS Cachan, 2010.
- [Fig11] D. Figueira. A decidable two-way logic on data words. In *LICS'11*, pages 365–374. IEEE, 2011.
- [Fit02] M. Fitting. Modal logic between propositional and first-order. *Journal of Logic and Computation*, 12(6):1017–1026, 2002.
- [FS01] A. Finkel and Ph. Schnoebelen. Well-structured transitions systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001.
- [FS09] D. Figueira and L. Segoufin. Future-looking logics on data words and trees. In *MFCS'09*, volume 5734 of *Lecture Notes in Computer Science*, pages 331–343, 2009.
- [GK03] P. Gastin and D. Kuske. Satisfiability and model checking for MSO-definable temporal logics are in PSPACE. In *CONCUR'03*, volume 2761 of *Lecture Notes in Computer Science*, pages 222–236. Springer, 2003.
- [HMU01] J. Hopcroft, R. Motwani, and J. Ullman. *Introduction to Automata Theory, Languages, and Computation (2nd. ed.)*. Addison Wesley, 2001.
- [HP79] J. Hopcroft and J.J. Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8:135–159, 1979.
- [Jan95] P. Jančar. Undecidability of bisimilarity for Petri Nets and some related problems. *Theoretical Computer Science*, 148:281–301, 1995.
- [Kos82] S. Rao Kosaraju. Decidability of reachability in vector addition systems. In *STOC'82*, pages 267–281, 1982.
- [KST12] A. Kara, Th. Schwentick, and T. Tan. Feasible automata for two-variable logic with successor on data words. In *LATA'12*, volume 7183 of *Lecture Notes in Computer Science*, pages 351–362. Springer, 2012.
- [KSZ10] A. Kara, Th. Schwentick, and Th. Zeume. Temporal logics on words with multiple data values. In *FST&TCS'10*, pages 481–492. LZI, 2010.
- [KV06] O. Kupferman and M. Vardi. Memoryful Branching-Time Logic. In *LICS'06*, pages 265–274. IEEE, 2006.
- [Lam92] J.L. Lambert. A structure to decide reachability in Petri nets. *Theoretical Computer Science*, 99:79–104, 1992.
- [Laz06] R. Lazić. Safely freezing LTL. In *FST&TCS'06*, volume 4337 of *Lecture Notes in Computer Science*, pages 381–392. Springer, 2006.
- [Ler11] J. Leroux. Vector addition system reachability problem: a short self-contained proof. In *POPL'11*, pages 307–316, 2011.

- [Lip76] R.J. Lipton. The reachability problem requires exponential space. Technical Report 62, Dept. of Computer Science, Yale University, 1976.
- [LMS02] F. Laroussinie, N. Markey, and Ph. Schnoebelen. Temporal logic with forgettable past. In *LICS'02*, pages 383–392. IEEE, 2002.
- [LP05] A. Lisitsa and I. Potapov. Temporal logic with predicate λ -abstraction. In *TIME'05*, pages 147–155. IEEE, 2005.
- [LS15] J. Leroux and S. Schmitz. Demystifying reachability in vector addition systems. In *LICS'15*, pages 56–67. IEEE, 2015.
- [May84] E.W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM Journal of Computing*, 13(3):441–460, 1984.
- [May03] R. Mayr. Undecidable problems in unreliable computations. *Theoretical Computer Science*, 297(1–3):337–354, 2003.
- [MR09] A. Manuel and R. Ramanujam. Counting multiplicity over infinite alphabets. In *RP'09*, volume 5797 of *Lecture Notes in Computer Science*, pages 141–153, 2009.
- [MR12] A. Manuel and R. Ramanujan. Automata over infinite alphabets. In D. D'Souza and P. Shankar, editors, *Modern applications of automata theory*, volume 2 of *IISc Research Monographs Series*, chapter 2, pages 529–554. World Scientific, 2012.
- [NS11] M. Niewerth and Th. Schwentick. Two-variable logic and key constraints on data words. In *ICDT'11*, pages 138–149. ACM, 2011.
- [NSV04] F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Transactions on Computational Logic*, 5(3):403–435, 2004.
- [Rac78] C. Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6(2):223–231, 1978.
- [Sav70] W.J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.
- [Sch10a] Ph. Schnoebelen. Lossy counter machines undecidability cheat sheet. In *RP'10*, volume 6227 of *Lecture Notes in Computer Science*, pages 51–75. Springer, 2010.
- [Sch10b] Ph. Schnoebelen. Revisiting Ackermann-hardness for lossy counter machines and reset Petri nets. In *MFCS'10*, volume 6281 of *Lecture Notes in Computer Science*, pages 616–628. Springer, 2010.
- [Seg06] L. Segoufin. Automata and logics for words and trees over an infinite alphabet. In *CSL'06*, volume 4207 of *Lecture Notes in Computer Science*, pages 41–57. Springer, 2006.
- [Wol83] P. Wolper. Temporal logic can be more expressive. *Information and Computation*, 56:72–99, 1983.

APPENDIX A. ELEMENTS OF THE PROOF OF LEMMA 4.9 FROM [DDG12]

Below, we recall developments from [DDG12] that lead to the proof of Lemma 4.9. We provide the main definitions but leave the details to [DDG12].

Let ϕ be an LRV^T formula built over variables in $\{\mathbf{x}_1, \dots, \mathbf{x}_k\}$. Let l be the maximal i such that a term of the form $X^i x$ occurs in ϕ (without any loss of generality, we can assume that $l \geq 1$). The value l is called the X -length of ϕ . In order to define the set of atomic formulae used in the symbolic models we introduce the set of constraints Ω_k^l that contains constraints of the form either $X^i \mathbf{x} = X^j \mathbf{y}$ or $X^i(\mathbf{x} \approx \langle \top? \rangle \mathbf{y})$ with $\mathbf{x}, \mathbf{y} \in \{\mathbf{x}_1, \dots, \mathbf{x}_k\}$ and $i, j \in [0, l]$.

The set of (l, k) -frames is denoted \mathbf{FFrame}_k^l , and is made up of pairs comprising a set of atomic constraints, along with some information about the last position of the model. This latter component is an element of $[0, l] \uplus \{\text{nd}\}$ where ‘nd’ means that the model does not end before the end of the frame. We have $\alpha = \text{nd}$ and $\langle fr, \alpha \rangle \in \mathbf{FFrame}_k^l$ iff fr satisfies the conditions:

- (F1): For all $i \in [0, l]$ and $\mathbf{x} \in \{\mathbf{x}_1, \dots, \mathbf{x}_k\}$, $X^i \mathbf{x} = X^i \mathbf{x} \in fr$.
- (F2): For all $i, j \in \{0, \dots, l\}$ and $\mathbf{x}, \mathbf{y} \in \{\mathbf{x}_1, \dots, \mathbf{x}_k\}$, $X^i \mathbf{x} = X^j \mathbf{y} \in fr$ iff $X^j \mathbf{y} = X^i \mathbf{x} \in fr$.
- (F3): For all $i, j, j' \in [0, l]$ and $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \{\mathbf{x}_1, \dots, \mathbf{x}_k\}$, if $\{X^i \mathbf{x} = X^j \mathbf{y}, X^j \mathbf{y} = X^{j'} \mathbf{z}\} \subseteq fr$ then $X^i \mathbf{x} = X^{j'} \mathbf{z} \in fr$.
- (F4): For all $i, j \in [0, l]$ and $\mathbf{x}, \mathbf{y} \in \{\mathbf{x}_1, \dots, \mathbf{x}_k\}$ such that $X^i \mathbf{x} = X^j \mathbf{y} \in fr$:
 - if $i = j$, then for every $\mathbf{z} \in \{\mathbf{x}_1, \dots, \mathbf{x}_k\}$ we have $X^i(\mathbf{x} \approx \langle \top? \rangle \mathbf{z}) \in fr$ iff $X^j(\mathbf{y} \approx \langle \top? \rangle \mathbf{z}) \in fr$;
 - if $i < j$ then $X^i(\mathbf{x} \approx \langle \top? \rangle \mathbf{y}) \in fr$, and for $\mathbf{z} \in \{\mathbf{x}_1, \dots, \mathbf{x}_k\}$, $X^i(\mathbf{x} \approx \langle \top? \rangle \mathbf{z}) \in fr$ iff either $X^j(\mathbf{y} \approx \langle \top? \rangle \mathbf{z}) \in fr$ or there exists $i < j' \leq j$ such that $X^i \mathbf{x} = X^{j'} \mathbf{z} \in fr$.

Additionally, we have $\alpha \in [0, l]$ and $\langle fr, \alpha \rangle \in \mathbf{FFrame}_k^l$ iff $fr \subseteq \Omega_k^\alpha$ and

- (F1'): For all $j \in [0, \alpha]$ and $\mathbf{x} \in \{\mathbf{x}_1, \dots, \mathbf{x}_k\}$, $X^j \mathbf{x} = X^j \mathbf{x} \in fr$.
- (F2'): For all $j, j' \in [0, \alpha]$ and $\mathbf{x}, \mathbf{y} \in \{\mathbf{x}_1, \dots, \mathbf{x}_k\}$, $X^j \mathbf{x} = X^{j'} \mathbf{y} \in fr$ iff $X^{j'} \mathbf{y} = X^j \mathbf{x} \in fr$.
- (F3'): For all $j, j', j'' \in [0, \alpha]$ and $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \{\mathbf{x}_1, \dots, \mathbf{x}_k\}$, if $\{X^j \mathbf{x} = X^{j'} \mathbf{y}, X^{j'} \mathbf{y} = X^{j''} \mathbf{z}\} \subseteq fr$ then $X^j \mathbf{x} = X^{j''} \mathbf{z} \in fr$.
- (F4'): For all $j, j' \in [0, \alpha]$ and $\mathbf{x}, \mathbf{y} \in \{\mathbf{x}_1, \dots, \mathbf{x}_k\}$ such that $X^j \mathbf{x} = X^{j'} \mathbf{y} \in fr$:
 - if $j = j'$, then for every $\mathbf{z} \in \{\mathbf{x}_1, \dots, \mathbf{x}_k\}$ we have $X^j(\mathbf{x} \approx \langle \top? \rangle \mathbf{z}) \in fr$ iff $X^{j'}(\mathbf{y} \approx \langle \top? \rangle \mathbf{z}) \in fr$;
 - if $j < j'$ then $X^j(\mathbf{x} \approx \langle \top? \rangle \mathbf{z}) \in fr$, and for $\mathbf{z} \in \{\mathbf{x}_1, \dots, \mathbf{x}_k\}$, $X^j(\mathbf{x} \approx \langle \top? \rangle \mathbf{z}) \in fr$ iff either $X^{j'}(\mathbf{y} \approx \langle \top? \rangle \mathbf{z}) \in fr$ or there exists $j < j'' \leq j'$ such that $X^j \mathbf{x} = X^{j''} \mathbf{z} \in fr$.
- (F5'): For every constraint $X^j(\mathbf{x} \approx \langle \top? \rangle \mathbf{z}) \in fr$ such that $j \in [0, \alpha]$ there is $j < j' \leq \alpha$ such that $X^j \mathbf{x} = X^{j'} \mathbf{y} \in fr$.

The last condition (F5') imposes that every future obligation is satisfied before the end of the model. The conditions (F1')–(F4') are variants of (F1)–(F4) in which the value l is replaced by α (index of the last position in the model). A finite model σ satisfies a frame $\langle fr, \alpha \rangle \in \mathbf{FFrame}_k^l$ at position j iff

- either $\alpha = \text{nd}$ and $|\sigma| - (j + 1) \geq l + 1$, or $\alpha = |\sigma| - (j + 1)$,
- for every constraint φ in fr we have $\sigma, j \models \varphi$.

In this case, we write $\sigma, j \models \langle fr, \alpha \rangle$.

We set $\langle \top? \rangle_{\langle fr, \alpha \rangle}(\mathbf{x}, i) \stackrel{\text{def}}{=} [(\mathbf{x}, i)]_{\langle fr, \alpha \rangle} \stackrel{\text{def}}{=} \emptyset$ when $\alpha \neq \text{nd}$ and $i > \alpha$. If $\alpha = \text{nd}$ or $i \leq \alpha$ then the definitions are the following:

$$\langle \top? \rangle_{\langle fr, \alpha \rangle}(\mathbf{x}, i) \stackrel{\text{def}}{=} \{\mathbf{y} \mid \mathbf{X}^i(\mathbf{x} \approx \langle \top? \rangle \mathbf{y}) \in fr\}.$$

$$[(\mathbf{x}, i)]_{\langle fr, \alpha \rangle} \stackrel{\text{def}}{=} \{\mathbf{y} \mid \mathbf{X}^i \mathbf{x} = \mathbf{X}^i \mathbf{y} \in fr\}.$$

A pair of (l, k) -frames $\langle \langle fr, \alpha \rangle, \langle fr', \alpha' \rangle \rangle$ is one-step consistent iff

- $\langle \alpha, \alpha' \rangle$ belongs to $\{\langle j, j' \rangle \in [1, l] \times [0, l-1] : j' = j-1\} \cup \{\langle \text{nd}, \text{nd} \rangle, \langle \text{nd}, l \rangle\}$.
- $\langle fr, fr' \rangle$ satisfies the following conditions when $\alpha = \alpha' = \text{nd}$:
 - (OSC1): for all $\mathbf{X}^i \mathbf{x} = \mathbf{X}^j \mathbf{y} \in \Omega_k^l$ with $0 < i, j$, we have $\mathbf{X}^i \mathbf{x} = \mathbf{X}^j \mathbf{y} \in fr$ iff $\mathbf{X}^{i-1} \mathbf{x} = \mathbf{X}^{j-1} \mathbf{y} \in fr'$,
 - (OSC2): for all $\mathbf{X}^i(\mathbf{x} \approx \langle \top? \rangle \mathbf{y}) \in \Omega_k^l$ with $i > 0$, we have $\mathbf{X}^i(\mathbf{x} \approx \langle \top? \rangle \mathbf{y}) \in fr$ iff $\mathbf{X}^{i-1}(\mathbf{x} \approx \langle \top? \rangle \mathbf{y}) \in fr'$.
- $\langle fr, fr' \rangle$ satisfies the conditions below when $\alpha \in [1, l]$:
 - for every $\mathbf{X}^j \mathbf{x} = \mathbf{X}^{j'} \mathbf{y} \in \Omega_k^\alpha$ with $0 < j, j'$, we have $\mathbf{X}^j \mathbf{x} = \mathbf{X}^{j'} \mathbf{y} \in fr$ iff $\mathbf{X}^{j-1} \mathbf{x} = \mathbf{X}^{j'-1} \mathbf{y} \in fr'$,
 - for every $\mathbf{X}^j(\mathbf{x} \approx \langle \top? \rangle \mathbf{y}) \in \Omega_k^\alpha$ with $j > 0$, we have $\mathbf{X}^j(\mathbf{x} \approx \langle \top? \rangle \mathbf{y}) \in fr$ iff $\mathbf{X}^{j-1}(\mathbf{x} \approx \langle \top? \rangle \mathbf{y}) \in fr'$.

A symbolic model is a finite one-step consistent sequence of frames ending with a symbolic valuation of the form $\langle fr, 0 \rangle$. Hence, for every position i in a finite symbolic model ρ , if $|\rho| - (i+1) \geq l+1$ then $\rho(i)$ is of the form $\langle fr, \text{nd} \rangle$, otherwise $\rho(i)$ is of the form $\langle fr, |\rho| - (i+1) \rangle$.

Given a symbolic model ρ , we adopt the following notations:

$$\langle \top? \rangle_\rho(\mathbf{x}, i) \stackrel{\text{def}}{=} \langle \top? \rangle_{\rho(i)}(\mathbf{x}, 0)$$

$$[(\mathbf{x}, i)]_\rho \stackrel{\text{def}}{=} [(\mathbf{x}, 0)]_{\rho(i)}.$$

The symbolic satisfaction relation $\rho, i \models_{\text{symp}} \phi$ is defined in the obvious way (for instance, $\rho, i \models \mathbf{X}^j \mathbf{x} = \mathbf{X}^{j'} \mathbf{y}$ iff $\mathbf{X}^j \mathbf{x} = \mathbf{X}^{j'} \mathbf{y}$ belongs to the first component of $\rho(i)$). The following correspondence between symbolic and concrete models can be shown similarly.

Lemma A.1. [DDG12] *An LRV^T formula ϕ of \mathbf{X} -length l over the variables $\{\mathbf{x}_1, \dots, \mathbf{x}_k\}$ is satisfiable over finite models iff there exists a finite (l, k) -symbolic model ρ such that $\rho \models_{\text{symp}} \phi$ and ρ is realizable (i.e., there is a finite model whose symbolic model is ρ).*

We define \mathcal{A}_ϕ as an automaton over the alphabet \mathbf{FFrame}_k^l (only used for synchronisation purposes), that accepts the intersection of the languages accepted by the three automata \mathcal{A}_{1sc} , $\mathcal{A}_{\text{symp}}$ and $\mathcal{A}_{\text{real}}$ described below:

- \mathcal{A}_{1sc} recognizes the sequences of frames such that every pair of consecutive frames is one-step consistent (easy),
- $\mathcal{A}_{\text{symp}}$ recognizes the set of symbolic models satisfying ϕ (as for LTL formulae),
- $\mathcal{A}_{\text{real}}$ recognizes the set of symbolic models that are realizable.

The automaton \mathcal{A}_{1sc} is a finite-state automaton that checks that the sequence is one-step consistent. Formally, we build $\mathcal{A}_{\text{1sc}} = \langle Q, Q_0, F, \rightarrow \rangle$ such that:

- Q is the set of (l, k) -frames and $Q_0 = Q$.
- the transition relation is defined by $\langle fr_1, \alpha_1 \rangle \xrightarrow{\langle fr, \alpha \rangle} \langle fr_2, \alpha_2 \rangle$ iff $\langle fr, \alpha \rangle = \langle fr_1, \alpha_1 \rangle$ and the pair $\langle \langle fr_1, \alpha_1 \rangle, \langle fr_2, \alpha_2 \rangle \rangle$ is one-step consistent.
- The set of final states F is equal to Q .

We define the finite-state automaton $\mathcal{A}_{\text{symb}}$ by adapting the construction for LTL. We define $cl(\phi)$ to be the standard *closure* of ϕ , namely the smallest set of formulas X that contains ϕ , is closed under subformulas, and satisfies the following conditions:

- If $\psi \in X$ and ψ is not of the form $\neg\psi_1$ for some ψ_1 , then $\neg\psi \in X$.
- If $\psi_1 \mathbf{U} \psi_2 \in X$ then $\mathbf{X}(\psi_1 \mathbf{U} \psi_2) \in X$.
- If $\psi_1 \mathbf{S} \psi_2 \in X$ then $\mathbf{X}^{-1}(\psi_1 \mathbf{S} \psi_2) \in X$.

An atom of ϕ is a subset At of $cl(\phi)$ which is maximally consistent: it satisfies the following conditions.

- For every $\neg\psi \in cl(\phi)$, we have $\neg\psi \in At$ iff $\psi \notin At$.
- For every $\psi_1 \wedge \psi_2 \in cl(\phi)$, we have $\psi_1 \wedge \psi_2 \in At$ iff ψ_1 and ψ_2 are in At .
- For every $\psi_1 \vee \psi_2 \in cl(\phi)$, we have $\psi_1 \vee \psi_2 \in At$ iff ψ_1 or ψ_2 is in At .
- For every $\psi_1 \mathbf{U} \psi_2 \in cl(\phi)$, we have $\psi_1 \mathbf{U} \psi_2 \in At$ iff either $\psi_2 \in At$ or both ψ_1 and $\mathbf{X}(\psi_1 \mathbf{U} \psi_2)$ are in At .
- For every $\psi_1 \mathbf{S} \psi_2 \in cl(\phi)$, we have $\psi_1 \mathbf{S} \psi_2 \in At$ iff either $\psi_2 \in At$ or both ψ_1 and $\mathbf{X}^{-1}(\psi_1 \mathbf{S} \psi_2)$ are in At .

We denote by $\text{Atom}(\phi)$ the set of atoms of ϕ . We now define $\mathcal{A}_{\text{symb}} = (Q, Q_0, \rightarrow, F)$ over the alphabet \mathbf{FFrame}_k^l , where:

- $Q = \text{Atom}(\phi) \times (\mathbf{FFrame}_k^l \cup \{\#\})$ and $Q_0 = \{\langle At, \# \rangle \mid \phi \in At\}$,
- $\langle At, X \rangle \xrightarrow{\langle fr, \alpha \rangle} \langle At', \langle fr, \alpha \rangle \rangle$ iff
(atomic): if $\alpha = nd$ then $At' \cap \Omega_k^l = fr$, otherwise $At' \cap \Omega_k^\alpha = fr$,
(one-step):
 (1) if $\alpha = 0$ then there is no formula of the form $\mathbf{X}\psi$ in At , otherwise for every $\mathbf{X}\psi \in cl(\phi)$, we have $\mathbf{X}\psi \in At$ iff $\psi \in At'$,

(2) for every $\mathbf{X}^{-1}\psi \in cl(\phi)$, we have $\psi \in At$ iff $\mathbf{X}^{-1}\psi \in At'$.

- The set of final states is therefore equal to $\{\langle fr, 0 \rangle \mid \langle fr, 0 \rangle \in \mathbf{FFrame}_k^l\}$.

For each $X \in \mathcal{P}^+(\{\mathbf{x}_1, \dots, \mathbf{x}_k\})$, we introduce a counter that keeps track of the number of obligations that need to be satisfied by X . We identify the counters with finite subsets of $\{\mathbf{x}_1, \dots, \mathbf{x}_k\}$.

A counter valuation is a map from $\mathcal{P}^+(\{\mathbf{x}_1, \dots, \mathbf{x}_k\})$ to \mathbb{N} . We define below a canonical sequence of counter valuations along a symbolic model. We will need to introduce some additional definitions first. For an (l, k) -frame $\langle fr, \alpha \rangle$ and a counter $X \in \mathcal{P}^+(\{\mathbf{x}_1, \dots, \mathbf{x}_k\})$, we define a point of increment for X in $\langle fr, \alpha \rangle$ to be an equivalence class of the form $[(\mathbf{x}, 0)]_{\langle fr, \alpha \rangle}$ such that $\langle \top? \rangle_{\langle fr, \alpha \rangle}(\mathbf{x}, 0) = X$ and there is no edge between $(\mathbf{x}, 0)$ and some (\mathbf{y}, j) with $j \in [1, l]$ assuming that fr is represented graphically. In a similar way, a point of decrement for X in $\langle fr, \alpha \rangle$ is defined to be an equivalence class of the form $[(\mathbf{x}, l)]_{\langle fr, \alpha \rangle}$ such that $\langle \top? \rangle_{\langle fr, \alpha \rangle}(\mathbf{x}, l) \cup [(\mathbf{x}, l)]_{\langle fr, \alpha \rangle} = X$, and there is no edge between (\mathbf{x}, l) and some (\mathbf{y}, j) with $j \in [0, l-1]$. So, when $\alpha \neq nd$, there is point of decrement in $\langle fr, \alpha \rangle$.

We denote by $u_{\langle fr, \alpha \rangle}^+$ the counter valuation which records the number of points of increment for each counter X in $\langle fr, \alpha \rangle$. Similarly $u_{\langle fr, \alpha \rangle}^-$ is the counter valuation which records the number of points of decrement for each counter X in $\langle fr, \alpha \rangle$. The codomain of both $u_{\langle fr, \alpha \rangle}^+$ and $u_{\langle fr, \alpha \rangle}^-$ is $[0, k]$.

Let ρ be an (l, k) -symbolic model. For every $X \in \mathcal{P}^+(\{\mathbf{x}_1, \dots, \mathbf{x}_k\})$, a point of increment for X in ρ is an equivalence class of the form $[(\mathbf{x}, i)]_\rho$ such that $[(\mathbf{x}, 0)]_{\rho(i)}$ is a point of

increment for X in the frame $\rho(i)$. Similarly, a point of decrement for X in ρ is an equivalence class of the form $[(\mathbf{x}, i)]_\rho$ such that $i \geq l + 1$ and $[(\mathbf{x}, l)]_{\rho(i-l)}$ is a point of decrement for X in $\rho(i-l)$.

We can now define a canonical counter valuation sequence β along ρ , called the counting sequence along ρ , which counts the number of obligations corresponding to each subset of variables X that remain unsatisfied at each position. We define β inductively: for each $X \in \mathcal{P}^+(\{\mathbf{x}_1, \dots, \mathbf{x}_k\})$ we have $\beta(0)(X) \stackrel{\text{def}}{=} 0$ and $\beta(i+1)(X) \stackrel{\text{def}}{=} \max(0, \beta(i)(X) + (u_{\rho(i)}^+(X) - u_{\rho(i+1)}^-(X)))$, for every $0 \leq i < |\rho|$.

The following result shows that the set of realizable symbolic models can be characterized using their counting sequences.

Lemma A.2. [DDG12] *A symbolic model ρ is realizable iff for every counter X the final value of the counting sequence β along ρ is equal to 0.*

It remains to define the automaton $\mathcal{A}_{\text{real}}$ that recognizes the set of realizable symbolic modelst. The automaton $\mathcal{A}_{\text{real}}$ is equal to $\langle Q, Q_0, Q_f, \Sigma, C, \delta \rangle$ such that

- $Q = \text{FFrame}_k^l$, $Q_0 = Q$, $Q_f = \{\langle fr, 0 \rangle \mid \langle fr, 0 \rangle \in Q\}$.
- $\Sigma = \text{FFrame}_k^l$.
- $C = \mathcal{P}^+(\{\mathbf{x}_1, \dots, \mathbf{x}_k\})$.
- The transition relation δ is defined by

$$\langle fr, \alpha \rangle \xrightarrow{up, \langle fr, \alpha \rangle} \langle fr', \alpha' \rangle \in \delta$$

for every $\langle fr, \alpha \rangle, \langle fr', \alpha' \rangle \in \text{FFrame}_k^l$ and $up : C \rightarrow [-k, +k]$ verifying

$$u_{\langle fr, \alpha \rangle}^+(X) - u_{\langle fr', \alpha' \rangle}^-(X) \leq up(X) \leq u_{\langle fr, \alpha \rangle}^+(X)$$

for every $X \in C$ (the letter $\langle fr, \alpha \rangle$ shall be removed after the product construction below as well as the notions of initial or final states).

A symbolic model ρ is accepted by $\mathcal{A}_{\text{real}}$ iff ρ is realizable [DDG12].

The VASS defined as the intersection of the two automata \mathcal{A}_{isc} , $\mathcal{A}_{\text{symb}}$ and the VASS $\mathcal{A}_{\text{real}}$, is indeed a VASS satisfying the assumptions in Lemma 4.9 (once the alphabet is removed but used to build the product). It is worth noting that \mathcal{A}_{isc} and $\mathcal{A}_{\text{symb}}$ are finite-state automata and therefore the counter updates are those from the transitions in $\mathcal{A}_{\text{real}}$.