EPiC Series in Computing

Volume 57, 2018, Pages 343–362

LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning

LTL with Arithmetic and its Applications in Reasoning about Hierarchical Systems

Rachel Faran and Orna Kupferman

The Hebrew University, Israel
{rachelmi,orna}@cs.huji.ac.il

Abstract

The computational bottleneck in model-checking applications is the blow-up involved in the translation of systems to their mathematical model. This blow up is especially painful in systems with variables over an infinite domain, and in composite systems described by means of their underlying components. We introduce and study *linear temporal logic with arithmetic* (LTLA, for short), where formulas include variables that take values in $\mathbb{Z}$, and in which linear arithmetic over these values is supported. We develop an automata-theoretic approach for reasoning about LTLA formulas and use it in order to solve, in PSPACE, the satisfiability problem for the existential fragment of LTLA and the model-checking problem for its universal fragment. We show that these results are tight, as a single universally-quantified variable makes the satisfiability problem for LTLA undecidable.

In addition to reasoning about systems with variables over $\mathbb{Z}$, we suggest applications of LTLA in reasoning about hierarchical systems, which consist of subsystems that can call each other in a hierarchical manner. We use the values in $\mathbb{Z}$ in order to describe the nesting depth of components in the system. A naive model-checking algorithm for hierarchical systems flattens them, which involves an exponential blow up. We suggest a model-checking algorithm that avoids the flattening and avoids a blow up in the number of components.

1 Introduction

In model checking, we verify that a system meets its specification by translating the system to a mathematical model, translating the specification to a temporal-logic formula, and checking that the model satisfies the formula [16]. The computational bottleneck in model-checking applications is the blow-up involved in the translation of the system to its mathematical model. There are several sources to this blow-up. One source are systems with a finite control that handles data from an infinite or unbounded domain. This includes, for example, software with integer parameters [11], datalog systems with an infinite data domain [47, 10], or FIFO-channel systems [7, 12]. Another source is the succinct high-level description of the system, typically by means of its underlying components [26].

Our work here is related to both challenges: we introduce and study *linear temporal logic with arithmetic* (LTLA, for short), where formulas include variables that take values in $\mathbb{Z}$, and in which linear arithmetic over these values is supported, and we suggest applications of LTLA

in reasoning about hierarchical systems, where the values in $\mathbb{Z}$ describe the nesting depth of components in a composite system. Before we describe our contribution in detail, let us survey briefly existing related work.

Reasoning about systems with a finite control that handles data from an infinite domain requires an extension of specification formalisms to the setting of *infinite alphabets*, and a development of decision procedures for them. Initial work on languages over infinite alphabets study *register automata* [42, 32, 38, 33, 36], where a finite set of registers is used to store letters from the infinite alphabet; *pebble automata* [38, 45], where pebbles are placed on the input word in a stack-like manner, and letters on which pebbles are placed can be tested for equality. Such tests are the key feature also in the newer formalism of *data automata* [9, 8], where a finite auxiliary alphabet is used to mark positions in which letters from the infinite alphabet can be tested for equality. Finally, [29, 30, 28, 44] handle infinite alphabets by augmenting the specification formalism with variables over the infinite domain and with guards that refer to these variables.

More relevant to our work are formalisms where the infinite domain is a set of ordered numbers, and arithmetic operations are allowed. As surveyed in [20], many extensions of LTL [39] are based on its augmentation with *Presburger arithmetic*, namely the first-order theory of the natural numbers with addition. While Presburger arithmetic is decidable, its combination with automata and logic adds to the picture cycles and fixed-points and makes the general setting undecidable. The quest for decidable extensions of LTL that are useful in real applications include *Presburger LTL with integer periodicity constraints* [22, 21], where arithmetic is done modulo some number, *difference logic*, where the way in which variables are compared with each other is restricted [23], and LTL with *quantifier-free Presburger arithmetic* [23]. Additional related formalisms are *LTL with constraint systems* [18, 17], where the logic includes rational structures, *multiple counters automata* [18] and *capacitated automata* [35], where transitions can be taken only if guards referring to traversals so far are satisfied, and *queue-content decision diagrams*, which are used to represent queue content of FIFO-channel systems [7, 12]. A different direction is to add arithmetic on top of formalisms that handle general infinite domains. In [14], the authors add to register automata linear arithmetic over the rationals. That is, the infinite alphabet is $\mathbb{Q}$. The emphasize in [14] is on quantitative properties. Thus, as in *weighted automata* on finite alphabets [25], the automaton maps an input word to a value in $\mathbb{Q}$. The transition to the quantitative setting has a computational price, and indeed the main contribution in [14], as well as in other work on weighted automata [2, 24], is to find decidable and tractable fragments (deterministic, copyless, etc.).

As for reasoning about composite systems, their exponential succinctness has led to extensive research on compositional model checking, where the goal is to reason about a system by reasoning about its underlying components and without constructing an equivalent flat system (c.f., [19, 40]). In particular, researchers have studied *hierarchical systems*, in which some of the states of the system are boxes, which correspond to nested sub-systems. The naive approach to model checking such systems is to "flatten" them by repeatedly substituting references to sub-systems by copies of these sub-systems. This, however, results in a flat system that is exponential in the description of the hierarchical system. In [4], it is shown that for LTL model checking, one can avoid this blow-up altogether, whereas for CTL, one can trade it for an exponential blow-up in the (often much smaller) size of the formula and the maximal number of exits of sub-structures. Likewise, it is shown in [5] that hierarchical parity games can be solved in PSPACE, also leading to a PSPACE model-checking algorithm for the μ-calculus. In other words, while hierarchical system are exponentially more succinct than flat ones [3], in many cases the complexity of the model-checking problem is not exponentially higher in the

hierarchical setting.

Our main contribution is an introduction of *LTL with arithmetic* (LTLA, for short), which augments LTL with variables over $\mathbb{Z}$ and arithmetic on these variables. As far as we have searched, LTLA is different from fragments of Presburger arithmetic that have been studied. LTLA continues the "variable approach" of [29, 30, 44] and can be viewed as an extension of *variable LTL* studied there, to a logic in which the infinite alphabet is $\mathbb{Z}$, and linear arithmetic over the variables and values is supported. The extension makes LTLA much stronger than variable LTL and makes the related decision problems much more challenging. The semantics of LTLA formulas is defined with respect to computations in $(2^{AP} \times \mathbb{Z})^\omega$, for a finite set AP of atomic propositions. Thus, each position describes an assignment to the atomic propositions in AP as well as a value in $\mathbb{Z}$. [1] The syntax of LTLA includes the atomic term $\star$, which gets the $\mathbb{Z}$ value of the current position. For example, the LTLA formula $\mathbf{G}((\star \geq -5) \wedge (\star \leq 5))$ states that all the values in the computation are in the $[-5, 5]$ range. We consider two variants of LTLA: *existential* LTLA ($\exists$LTLA) and *universal* LTLA ($\forall$LTLA). In $\exists$LTLA, variables that appear in the formula are existentially quantified. For example, a computation satisfies the $\exists$LTLA formula $\exists x; \mathbf{G}(up \rightarrow (\star = x) \wedge \mathbf{X}(\star \geq x))$ if there is a number $m \in \mathbb{Z}$ such that in every position in which up holds, the value is m and it cannot be decreased in the next position. Then, a computation satisfies $\forall x; \mathbf{G}((\star = x) \rightarrow (inc \wedge \mathbf{X}(\star = x + 1)) \vee (reset \wedge \mathbf{X}(\star = 0)))$ if all positions are labeled by inc, in which case the value in its successors is increased by 1, or by $reset$, in which case the value is reset to 0. As has been the case with other variants of Presburger LTL and logics with quantified data variables [20, 44], the satisfiability problem for $\forall$LTLA is undecidable. In fact, a single universally quantified variable is sufficient in order to obtain undecidability, which also explains why we do not consider variants of LTLA with alternating quantifiers. We still study $\forall$LTLA, as its model checking involves reasoning about dual, namely $\exists$LTLA, formulas, for which we develop an automata-theoretic framework. Also, while the logics in [29, 30, 44] allow reset of variables along the computation, giving them the flavor of register automata, quantified variables in LTLA have a fixed value throughout the run. On the other hand, LTLA allows comparing values not only between the input and the variables, but also between simple arithmetic expressions on the value of the input and the values stored in the variables.

We define *nondeterministic Büchi word automata with arithmetic* (NBWA, for short), which define languages over alphabets of the form $\Sigma \times \mathbb{Z}$, for some finite set Σ. The transitions of an NBWA are labeled by both letters from Σ and guards involving values and variables over $\mathbb{Z}$. We show how by translating LTLA formulas to NBWAs, we can solve the satisfiability problem for $\exists$LTLA and the model-checking problem for $\forall$LTLA. The translation adds to the translation of LTL to NBW [46] a reference to the guards in the LTLA formula, which induces guards on the transitions of the NBWA. While the nonemptiness problem for traditional NBWs reduces to reachability, the nonemptiness problem for NBWAs reduces to a solution of a set of inequalities over the variables of the NBWA. We use results on *integer linear programming* (ILP) [48, 41, 34] in order to show that the problem is NP-complete and to point to NLOGSPACE fragments. By analyzing the different components of the NBWAs that correspond to LTLA formulas, in particular the guards that appear in the NBWA, we obtain a PSPACE upper bound for both the satisfiability and model-checking problems, which are thus not harder than these for LTL. In particular, the complexity in terms of the system is NLOGSPACE. The PSPACE upper bound applies even when the values in $\mathbb{Z}$ that appear in the formulas or the system are given

[1] We could handle also systems in which states are labeled by an assignment to the atomic propositions and a vector in $\mathbb{Z}^*$. We chose to study the case of a single value in $\mathbb{Z}$ as it includes the main challenge, namely the need to cope with the infinite domain of $\mathbb{Z}$, and it is present in our primary application, of hierarchical systems.

in binary.

We continue and present applications of LTLA in reasoning about hierarchical systems. There, we use the values in $\mathbb{Z}$ to refer to the depth of the computation in the system. For example, the ∀LTLA formula $\forall x; \mathbf{G}((\mathit{return} \wedge (\star = x)) \rightarrow \mathbf{X}(\star = x - 1))$ states that whenever the control of the system is in a return state, the nesting depth should be reduced by 1. As mentioned above, an algorithm for LTL model checking that avoids a flattening of the hierarchical system is described in [4]. The algorithm is based on an iterative compression of subsystems to small *gadgets* that maintain essential reachability information [27]. A naive application of this approach in the case of ∀LTLA model checking is infeasible. Indeed, while the different calls to a subsystem share the same AP labelling, they may differ in their nesting depth. Consequently, each nesting depth requires a different gadget, resulting in a blow-up that depends on the nesting depth. An attempt to overcome this difficulty by defining gadgets with guards results in gadgets whose size depend on the subsystem. We show that for ∀LTLA formulas in which guards cannot relate to more than one variable, we can replace subsystems by gadgets with no guards whose number is exponential in the formula, keeping the model-checking complexity of LTL model checking.

In Section 6 we discuss an extension of our contribution to the synthesis of hierarchical systems, to the branching-time formalism, and to recursive systems.

2 Preliminaries

2.1 Adding Arithmetics to LTL

For a set X of variables, an assignment $f : X \rightarrow \mathbb{Z}$, and a value $c \in \mathbb{N}$, we say that f is bounded by c if $-c \leq f(x) \leq c$ for all $x \in X$. The set of *terms over* X, denoted Θ_X, is defined inductively as follows.

- m, x, and $\star$, for $m \in \mathbb{Z}$, $x \in X$, and the symbol $\star$.
- $t_1 + t_2$ and $t_1 - t_2$ for $t_1, t_2 \in \Theta_X$.

For a number $k \in \mathbb{Z}$ and an assignment $f : X \rightarrow \mathbb{Z}$, let $f^k : X \cup \{\star\} \rightarrow \mathbb{Z}$ be an extension of f where for all $x \in X$, we have $f^k(x) = f(x)$, and for the symbol $\star$ we have $f^k(\star) = k$. Given k and f, we can extend f^k to terms over X in the expected way; that is, $f^k : \Theta_X \rightarrow \mathbb{Z}$ is such that $f^k(m) = m$, $f^k(t_1 + t_2) = f^k(t_1) + f^k(t_2)$, and $f^k(t_1 - t_2) = f^k(t_1) - f^k(t_2)$. Similarly, given $f : X \rightarrow \mathbb{Z}$, we can extend it to $f^\star : \Theta_X \rightarrow \Theta_X$ by replacing every composed term with its evaluation, and leaving the symbol $\star$.

The set of *guards over* X, denoted $\mathcal{G}_X$, is defined inductively as follows.

- $t_1 \leq t_2$, for $t_1, t_2 \in \Theta_X$
- $\neg\gamma_1$ and $\gamma_1 \wedge \gamma_2$, for $\gamma_1, \gamma_2 \in \mathcal{G}_X$.

Guards of the form $t_1 \leq t_2$ are termed *atomic guards*, and we denote the set of atomic guards in $\mathcal{G}_X$ by $\mathcal{G}_X^A$. A guard $\gamma \in \mathcal{G}_X$ is *simple* if all its atomic guards contain at most one variable in X. Thus, different variables in X cannot be related.

For $k \in \mathbb{Z}$, an assignment $f : X \rightarrow \mathbb{Z}$, and a guard $\gamma \in \mathcal{G}_X$, we define when k *satisfies* γ *under* f, denoted $k \models_f \gamma$, by induction on the structure of γ as follows.

- For atomic guards $(t_1 \leq t_2) \in \mathcal{G}_X^A$, we have that $k \models_f (t_1 \leq t_2)$ if $f^k(t_1) \leq f^k(t_2)$.

- For guards $\gamma_1, \gamma_2 \in \mathcal{G}_X$, we have that $k \models_f \neg\gamma$ if $k \not\models_f \gamma$, and $k \models_f \gamma_1 \wedge \gamma_2$ if $k \models_f \gamma_1$ and $k \models_f \gamma_2$.

Note that satisfaction of guards $\gamma \in \mathcal{G}_\emptyset$ is independent of an assignment. Therefore, we write $k \models \gamma$ (that is, with no f). Using the operators $\wedge$ and $\neg$, and keeping in mind that the terms are defined over the whole numbers, we can generate from $t_1 \leq t_2$ also the atomic guards $t_1 > t_2$, $t_1 < t_2$, $t_1 \geq t_2$, and $t_1 = t_2$, where $t_1, t_2 \in \Theta_X$.

We consider systems with Boolean variables as well as variables over $\mathbb{Z}$. Formally, a *system* is $K = \langle AP, W, R, W_0, l, v\rangle$, where AP is a set of atomic propositions, W is finite a set of states, $R \subseteq W \times W$ is a transition relation, W_0 is a set of initial states, and $l : W \to 2^{AP}$ and $v : W \to \mathbb{Z}$ are labeling functions. Note that such system is an extension of Kripke structures to structures in which states are labeled also with numbers. A *path* in K is an infinite sequence of states $w_0, w_1, \ldots$, where $w_0 \in W_0$ and for every $i \geq 0$, we have that $\langle w_i, w_{i+1}\rangle \in R$. A *computation* of K is then a word $\langle\sigma_0, k_0\rangle, \langle\sigma_1, k_1\rangle, \cdots \in (2^{AP} \times \mathbb{Z})^\omega$ such that there is a path $w_0, w_1, \ldots$ in K such that for all $i \geq 0$, we have that $l(w_i) = \sigma_i$ and $v(w_i) = k_i$. We define the *size* of a system $K = \langle AP, W, R, W_0, l, v\rangle$, denoted $|K|$, as $|W| + |R|$.

We specify properties of systems by *LTL with arithmetics* (LTLA, for short), which extends LTL by a reference to the values in $\mathbb{Z}$. An *open LTLA formula* φ over a set AP of atomic propositions and a set X of variables is defined inductively as follows.

- *true*, *false*, p, for $p \in AP$, and γ, for $\gamma \in \mathcal{G}_X$.
- $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$, $\mathbf{X}\varphi_1$, and $\varphi_1 \mathbf{U} \varphi_2$, for open LTLA formulas φ_1 and φ_2.

An LTLA formula is *simple* if all the guards in it are simple. Let $var(\varphi) \subseteq X$ be the set of variables appearing in an open LTLA formula φ. When $var(\varphi) = \emptyset$, we say that φ is *variable free*. Consider an open LTLA formula φ over X and an assignment $f : X \to \mathbb{Z}$. We denote by φ_f the variable-free formula obtained from φ by replacing every term $t \in \Theta_X$ by $f^\star(t)$. For example, if $\varphi = \mathbf{G}(p \to (\star = x_1) \wedge \mathbf{X}(\star \geq x_1))$ and $f(x_1) = 5$, then $\varphi_f = \mathbf{G}(p \to (\star = 5) \wedge \mathbf{X}(\star \geq 5))$.

The semantics of variable-free LTLA formulas is defined with respect to computations of systems. Consider a computation $\pi = \langle\sigma_0, k_0\rangle, \langle\sigma_1, k_1\rangle, \cdots \in (2^{AP} \times \mathbb{Z})^\omega$. We refer to σ_i as the *proposition component* of the i-th position, and refer to k_i as its *value*. For a computation π, a variable free LTLA formula φ, and a position $i \geq 0$, we use $(\pi, i) \models \varphi$ to indicate that φ is *satisfied* in the i-th position of φ. The relation $\models$ is inductively defined, for all computations π and positions $i \geq 0$, as follows.

- $(\pi, i) \models$ *true* and $(\pi, i) \not\models$ *false*,
- for $p \in AP$, we have that $(\pi, i) \models p$ if $p \in \sigma_i$,
- for $\gamma \in \mathcal{G}_\emptyset$, we have that $(\pi, i) \models \gamma$ if $k_i \models \gamma$,
- $(\pi, i) \models \neg\varphi_1$ if $(\pi, i) \not\models \varphi_1$,
- $(\pi, i) \models \varphi_1 \wedge \varphi_2$ if $(\pi, i) \models \varphi_1$ and $(\pi, i) \models \varphi_2$,
- $(\pi, i) \models \mathbf{X}\varphi$ if $(\pi, i+1) \models \varphi$, and
- $(\pi, i) \models \varphi_1 \mathbf{U} \varphi_2$ if there exists $k \geq 0$ such that $(\pi, i+k) \models \varphi_2$ and $(\pi, i+j) \models \varphi_1$ for all $0 \leq j < k$.

The computation π then satisfies φ, denoted $\pi \models \varphi$, iff $(\pi, 0) \models \varphi$. We use the abbreviations $\mathbf{F}\varphi$ and $\mathbf{G}\varphi$ for $\mathit{true}\mathbf{U}\varphi$ and $\neg\mathbf{F}\neg\varphi$, respectively.

We continue to LTLA formulas that are not open. We distinguish between the case the variables in the formula are quantified existentially and cases where they are quantified universally. An $\exists LTLA$ formula is $\psi = \exists x_1; \exists x_2; \ldots; \exists x_k; \varphi$, for an open LTLA formula φ with $var(\varphi) = \{x_1, \ldots, x_k\}$. For clarity, we also write $\psi = \exists\varphi$. A $\forall LTLA$ formula is then $\psi = \forall x_1; \forall x_2; \ldots; \forall x_k; \varphi$ (also written $\forall\varphi$). An LTLA formula is an $\exists$LTLA or $\forall$LTLA formula. In Section 2.3, we show that a single universally quantified variable makes satisfiability of LTLA undecidable, which justifies our focus in the existential and universal fragments. Indeed, while we can negate a $\forall$LTLA formula and obtain an $\exists$LTLA formula, on which we can reason, manipulations with fragments in which there is alternation between universal and existential variables would leave us with at least one universally quantified variables, leading to undecidability.

Note that quantification is allowed only outside the open formula, and thus it cannot appear in the scope of Boolean or temporal operators. We conjecture that a formalism with nested quantification is strictly stronger. Consider, for example, the formula $\psi = G(\mathit{even} \rightarrow \exists x : \star = x + x)$, which states that whenever the proposition *even* holds, the value is even. The existential quantification on x is in the scope of the G temporal operator, allowing us to consider a different assignment to x in each state. We conjecture that an LTLA formula equivalent to ψ requires a different variable for each state in the computation, and thus ψ has no finite equivalent LTLA formula. We leave the study of a temporal logic with a richer quantification for future research.

The semantics of open LTLA formulas is defined with respect to a computation π and an assignment to the variables. For an open LTLA formula φ, a computation π, and an assignment $f : X \rightarrow \mathbb{Z}$, we say that π *satisfies* φ *under* f, denoted $\pi \models_f \varphi$, if $\pi \models \varphi_f$. For the LTLA formulas $\exists\varphi$ and $\forall\varphi$, satisfaction is defined with respect to a computation:

- $\pi \models \exists\varphi$ if there is an assignment $f : X \rightarrow \mathbb{Z}$ such that $\pi \models_f \varphi$.
- $\pi \models \forall\varphi$ if for all assignments $f : X \rightarrow \mathbb{Z}$, we have $\pi \models_f \varphi$.

For example, a computation satisfies $\exists x; \mathbf{G}(\mathit{up} \rightarrow (\star = x) \wedge \mathbf{X}(\star \geq x))$ if there is $m \in \mathbb{Z}$ such that in every position in which *up* holds, the value is m, and it cannot be decreased in the next position. Then, a computation satisfies $\forall x; (x \geq 10) \rightarrow \mathbf{F}(\star = x)$, if every value greater than 10 eventually appears in it. For an LTLA formula ψ, we define the *language of* ψ, denoted $L(\psi)$, as the set of all computations that satisfy ψ. We denote by $|\psi|$ the length of ψ.

The two fundamental problems for LTLA are satisfiability and model checking. In the *satisfiability* problem, we are given an LTLA formula ψ and decide whether there is a computation that satisfies it, namely whether $L(\psi) \neq \emptyset$. In the model-checking problem, we are given a system K and an LTLA formula ψ and decide whether all the computations of K satisfy ψ. When analyzing the complexity of the problems, we assume that the values in ψ and K are given in unary.

2.2 Adding Arithmetics to NBWs

A *nondeterministic Büchi word automaton with arithmetic* (NBWA, for short) is a tuple $\mathcal{A} = \langle \Sigma, X, Q, Q_0, \Delta, \alpha \rangle$, where Σ is an alphabet, X is a set of variables, Q is a set of states, $Q_0 \subseteq Q$ is a set of initial states, $\Delta \subseteq Q \times \Sigma \times \mathcal{G}_X \times Q$ is a transition relation, and $\alpha \subseteq Q$ is an acceptance condition. We assume that for every two states $q_1, q_2 \in Q$ and letter $\sigma \in \Sigma$, there is at most one $\gamma \in \mathcal{G}_X$ such that $\langle q_1, \sigma, \gamma, q_2 \rangle \in \Delta$. Note that this can be guaranteed, since multiple transitions

from q_1 to q_2 with the same σ-label can be unified by applying a disjunction on the guards they are labeled by.

A *run* of $\mathcal{A}$ on an infinite word $\langle\sigma_0, k_0\rangle, \langle\sigma_1, k_1\rangle, \ldots$ is a sequence of states $q_0, q_1, \ldots$, where $q_0 \in Q_0$, and there is an assignment $f : X \to \mathbb{Z}$ such that for every position $i \geq 0$, there is a transition $\langle q_i, \sigma_i, \gamma, q_{i+1}\rangle \in \Delta$ such that $k_i \models_f \gamma$. Note that the assignment f is fixed throughout the run. This is contrast to variants of register automata, in particular these used in [44], which are needed to handle resets of variables during the computation. For a run r, let $inf(r) = \{q : q_i = q \text{ for infinitely many } i\text{'s}\}$. That is, $inf(r)$ is the set of states that r visits infinitely often. Then, r is *accepting* iff $inf(r) \cap \alpha \neq \emptyset$. The *language* of $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words $w \in (\Sigma \times \mathbb{Z})^\omega$ such that $\mathcal{A}$ has an accepting run on w. The nonemptiness problem for NBWAs is to decide, given an NBWA $\mathcal{A}$, whether $L(\mathcal{A})$ is empty.

Example 1. The LTLA formula $\psi_1 = (\star \leq 3)\mathbf{U}(p \wedge (\star = 4))$ states that a computation eventually reaches a position in which p holds and the value is exactly 4, and until this position the value is at most 3. The language $L(\psi_1)$ is recognized by the NBWA $\mathcal{A}_1$ appearing in Figure 1. The self loop in q_0 is labeled by $\langle true, \star \leq 3\rangle$, meaning that a run can stay in q_0, reading either p or $\neg p$, as long the value it reads is at most 3. Further, it moves from q_0 to q_1 only when p holds and the value is 4. After the run moves to q_1, it has no further restrictions.

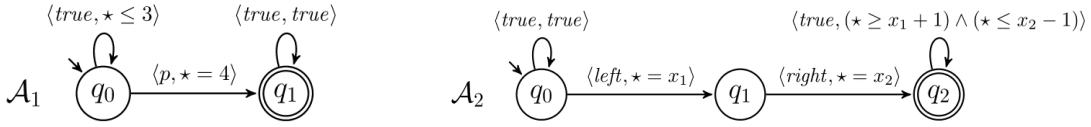

Figure 1: The NBWAs $\mathcal{A}_1$ and $\mathcal{A}_2$

The $\exists$LTLA formula $\psi_2 = \exists x_1; \exists x_2; \mathbf{F}((\mathit{left} \wedge (\star = x_1)) \wedge \mathbf{X}(\mathit{right} \wedge (\star = x_2)) \wedge \mathbf{XXG}((\star \geq x_1 + 1) \wedge (\star \leq x_2 - 1)))$ states that there are two values x_1 and x_2, such that eventually we reach a position in which *left* holds and the value is x_1, in the successive position *right* holds and the value is x_2, and all the values to follow are between $x_1 + 1$ and $x_2 - 1$. Note that ψ_2 is simple. The language $L(\psi_2)$ is recognized by the NBWA $\mathcal{A}_2$ appearing in Figure 1. An accepting run of $\mathcal{A}_2$ moves from q_0 to q_1 when it reads *left* and some value x_1, and in the next transition it moves from q_1 to q_2, reading *right* and some value x_2. From that position and on, it reads only values in $[x_1 + 1, x_2 - 1]$.

Note that while in LTLA we consider both an existential and a universal interpretation to the quantified variables, in the case of NBWA we only consider an existential one. As we show in Section 2.3 below, the reachability problem would be undecidable in an NBWA with a universal interpretation to the quantified variables. For LTLA, where negation is easy, we keep both interpretations.

2.3 Undecidability

It is well known that going to an infinite data domain in specification formalisms with variable quantification leads to undecidability of the satisfiability problem. We prove that similar considerations apply to the universal fragment of LTLA. The proof is by a reduction from the halting problem for two-counter (Minsky) machines, as is the case with other variants of Presburger LTL [20], and we bring it here for completeness. As is the case with the variable LTL of [44], a single universally quantified variable is sufficient in order to obtain undecidability. The proof there, however, is from the PCP problem.

Theorem 1. *The satisfiability problem for ∀LTLA is undecidable.*

Proof. We describe a reduction from the halting problem for two-counter (Minsky) machines. A two-counter machine $\mathcal{M}$ is a sequence $(l_1, \ldots, l_n)$ of commands involving two counters x and y. We refer to $L = \{1, \ldots, n\}$ as the *locations* of the machine. There are five possible forms of commands:

$$\text{INC}(c),\ \text{DEC}(c),\ \text{GOTO } l_i,\ \text{IF } c{=}0 \text{ GOTO } l_i \text{ ELSE GOTO } l_j,\ \text{HALT},$$

where $c \in \{x, y\}$ is a counter and l_i and l_j are locations. Since we can always check whether $c = 0$ before a DEC(c) command, we assume that the machine never reaches DEC(c) with $c = 0$. That is, the counters never have negative values. A *halting run* of M is a sequence $\rho = \rho_1, \ldots, \rho_m \in (L \times \mathbb{N} \times \mathbb{N})^*$ such that the following hold.

1. $\rho_1 = \langle l_1, 0, 0\rangle$.
2. For all $1 < i \leq m$, let $\rho_{i-1} = (l_k, \alpha, \beta)$ and $\rho_i = (l', \alpha', \beta')$. Then, the following hold.
   - If l_k is a INC(x) command (resp. INC(y)), then $\alpha' = \alpha + 1$, $\beta' = \beta$ (resp. $\beta = \beta + 1$, $\alpha' = \alpha$), and $l' = l_{k+1}$.
   - If l_k is a DEC(x) command (resp. DEC(y)), then $\alpha' = \alpha - 1$, $\beta' = \beta$ (resp. $\beta = \beta - 1$, $\alpha' = \alpha$), and $l' = l_{k+1}$.
   - If l_k is a GOTO l_s command, then $\alpha' = \alpha$, $\beta' = \beta$, and $l' = l_s$.
   - If l_k is an IF $x{=}0$ GOTO l_s ELSE GOTO l_t command, then $\alpha' = \alpha$, $\beta' = \beta$, and $l' = l_s$ if $\alpha = 0$, and $l' = l_t$ otherwise.
   - If l_k is a IF $y{=}0$ GOTO l_s ELSE GOTO l_t command, then $\alpha' = \alpha, \beta' = \beta$, and $l' = l_s$ if $\beta = 0$, and $l' = l_t$ otherwise.
   - If l' is a HALT command, then $i = m$. That is, a run does not continue after HALT.
3. $\rho_m = \langle l_k, \alpha, \beta\rangle$ such that l_k is a HALT command.

Given a counter machine $\mathcal{M}$, deciding whether $\mathcal{M}$ has a halting run is known to be undecidable [37]. Given $\mathcal{M}$, we construct an ∀LTLA formula $\psi_{\mathcal{M}} = \forall z; \varphi(z)$ such that $\psi_{\mathcal{M}}$ is satisfiable iff $\mathcal{M}$ has a halting run. Note that we use a fragment of ∀LTLA with a single variable.

Let AP be a set of atomic propositions that encode L, and, for $1 \leq k \leq n$, let $\theta(l_k)$ be an assertion over AP that is satisfied by the assignment in which the atomic proposition encode the location l_k. We associate each position in a run of $\mathcal{M}$ with a block of three successive positions in computations over $2^{AP \cup \{\#\}} \times \mathbb{Z}$. The proposition component of the first position in each block encodes the location, and the value components of the second and third positions maintain the values of the counters x and y, respectively, namely α and β in the above description. A special atomic proposition # labels the first position in each block. Thus, φ includes the conjunct $\# \wedge (\mathbf{X}\neg\#) \wedge (\mathbf{XX}\neg\#) \wedge \mathbf{G}(\# \leftrightarrow \mathbf{XXX}\#)$.

Each location l_k contributes to φ a conjunction φ_k defined as follows.

- If l_k is an INC(x) command, then $\varphi_k = \mathbf{G}((\# \wedge \theta(l_k)) \rightarrow (\mathbf{XXX}\theta(l_{k+1}) \wedge \mathbf{X}((\star = z) \rightarrow \mathbf{XXX}(\star = z + 1)) \wedge \mathbf{XX}((\star = z) \rightarrow \mathbf{XXX}(\star = z))))$. Thus, the next location is l_{k+1}, the value of x is increased by 1, and the value of y does not change. Indeed, for all values z, if the current value of x, namely the one be read in the next position, is equal to z, then the next value of x, namely the one be read three positions later, is $z + 1$. Also, if the

current value of y, namely the one be read in the next next position, is equal to z, then the next value of y, namely the one be read three positions later, stays z.

The conjunction for DEC(x) is similar, with $z-1$. The conjunctions for increasing or decreasing of y are also similar, with the value in the next next position being changed. For example, if l_k is a DEC(y) command, then $\varphi_k = \mathbf{G}((\# \wedge \theta(l_k)) \rightarrow (\mathbf{XXX}\theta(l_{k+1}) \wedge \mathbf{X}((\star = z) \rightarrow \mathbf{XXX}(\star = z)) \wedge \mathbf{XX}((\star = z) \rightarrow \mathbf{XXX}(\star = z-1))))$.

- If l_k is a GOTO l_s command, then $\varphi_k = \mathbf{G}((\# \wedge \theta(l_k)) \rightarrow (\mathbf{XXX}\theta(l_s)) \wedge \mathbf{X}((\star = z) \rightarrow \mathbf{XXX}(\star = z)) \wedge \mathbf{XX}((\star = z) \rightarrow \mathbf{XXX}(\star = z))))$. That is, the location is updated, and the values of x and y are not changed.
- If l_k is an IF x=0 GOTO l_s ELSE GOTO l_t command, then $\varphi_k = \mathbf{G}((\# \wedge \theta(l_k)) \rightarrow (\mathbf{X}(\star = 0) \rightarrow \mathbf{XXX}\theta(l_s)) \wedge (\neg\mathbf{X}(\star = 0)) \rightarrow \mathbf{XXX}\theta(l_t)) \wedge \mathbf{X}((\star = z) \rightarrow \mathbf{XXX}(\star = z)) \wedge \mathbf{XX}((\star = z) \rightarrow \mathbf{XXX}(\star = z))))$.

  That is, the location is updated according to the values of x, and the values of x and y are not changed. The case the control depends on y is similar.

In addition, φ includes a conjunct that requires the run to start in $(l_1, 0, 0)$, namely $\theta(l_1) \wedge \mathbf{X}(\star = 0) \wedge \mathbf{XX}(\star = 0)$, and a conjunct that requires the run to halt. Thus, if $H = \{k : l_k \text{ is a HALT command}\}$, then φ includes the conjunct $\mathbf{F}(\# \wedge \bigvee_{k \in H} \theta(l_k))$.

It is not hard to prove that a computation π satisfies $\psi_{\mathcal{M}}$ iff π describes a halting run of $\mathcal{M}$. First, the single halting run of $\mathcal{M}$ satisfies all conjuncts. Indeed, the requirements about the atomic propositions are satisfied, and whatever value we assign to z, either no position in the computation that is located in the second or third position in a block has z in its value component, in which case all the assertions involving z are satisfied vacuously, or there are positions in which α or β take value z, in which case the fact the run is legal guarantees that the requirements imposed by the conjuncts are satisfied. For the other direction, if $\psi_{\mathcal{M}}$ is satisfiable, then the conjunct $\mathbf{F}(\# \wedge \bigvee_{k \in H} \theta(l_k))$ guarantees that the satisfying computation includes a finite prefix that reaches a location with a HALT command. We can prove by induction on the length of the prefix that it describes a halting run. In particular, the universal quantification on z implies that α and β proceed as required. □

2.4 Useful Results on Integer Linear Programming

The *feasibility* problem in integer linear programming (ILP) is to decide, given a set $\mathcal{E}$ of linear inequalities, whether it is satisfiable by an integer assignment to the variables. We assume that $\mathcal{E}$ consists of a set of linear inequalities (with $\leq$ and $\geq$) over a set X of variables. As we show in the sequel, many problems about LTLA and NBWA are reduced to variants of ILP. In particular, while in ILP the inequalities are over a set X of variables, in our setting the inequalities may contain the term $\star$. Some of the occurrences of $\star$ are related, as they origin from guards that refer to the same input. Also, in the application of model checking, the values for $\star$ are given, and in the application of hierarchical systems, occurrences of $\star$ that correspond to states in the same depth agree on their value. These different contexts require different treatments of $\star$ in the inequalities. For example, if we naively replace each occurrence of $\star$ by a fresh variable, as is natural to do in the case of satisfiability, we increase the number of variables, giving up complexity results that make use of this parameter. In this section we define and study the feasibility problem for the variants of ILP we are going to encounter in the context of LTLA and NBWA. We start with a general known result. Throughout this section, we consider a set $\mathcal{E}$ of linear inequalities of size M over a set X of N variables, with L being the largest number that appears in $\mathcal{E}$.

Lemma 1. [48, 41, 34] *If $\mathcal{E}$ is satisfiable, then it is satisfiable by an assignment bounded by $((M+N)\cdot N\cdot L)^N$. Deciding the satisfiability of $\mathcal{E}$ is NP-complete.*

To see how the exponential blow up in Lemma 1 may be attained, consider a system $\mathcal{E}$ over $\{x_1,\ldots,x_N\}$ that includes the set of inequalities $x_1 \geq 1$, and $x_{i+1} > x_i + x_i$, for every $1 \leq i < N$. The minimal value that x_N is assigned in a satisfying assignment is exponential in N. By bounding the length of a chain of variables whose values depend on each other, we can bound the exponent in the expression in Lemma 1. In particular, when we translate a conjunction of atomic guards into an ILP and replace occurrences of $\star$ by fresh variables in Y (possibly the same variable in Y for different occurrences of $\star$ – these that origin from the same transition of a given NBWA), then the variables in Y cannot participate in such a chain. Formally, consider an ILP $\mathcal{E}$ over X and a partition $\langle X_1, X_2\rangle$ of X. We say that $\langle X_1, X_2\rangle$ is *$\star$-consistent* if all inequalities in $\mathcal{E}$ contain at most one variable from X_2, appearing with coefficient 1. A conjunction of atomic guards with $\star$ being replaced by fresh variables in Y then results in an ILP for which the partition $\langle X, Y\rangle$ is $\star$-consistent.

Lemma 2. *Let $\langle X_1, X_2\rangle$ be a $\star$-consistent partition of X. If $\mathcal{E}$ is feasible, then it is feasible by an assignment bounded by $((M+N)\cdot N\cdot L)^{|X_1|}$.*

By setting additional constraints on $\mathcal{E}$, we can reduce the range of the values in a solution even further (see also [31], for a setting without a $\star$-consistent partition).

Lemma 3. *If there is a $\star$-consistent partition $\langle X_1, X_2\rangle$ of X such that all inequalities in $\mathcal{E}$ contain at most one variable from X_1, appearing with coefficients 1 or -1, then $\mathcal{E}$ is feasible iff $\mathcal{E}$ is feasible with an assignment bounded by $O(L\cdot\min\{|X_1|,|X_2|\})$. Accordingly, if the values in $\mathcal{E}$ are given in unary, its feasibility can be decided in NLOGSPACE.*

Lemma 4. *If there is a $\star$-consistent partition $\langle X_1, X_2\rangle$ of X, for X_1 of a constant size, then $\mathcal{E}$ is feasible iff $\mathcal{E}$ is feasible with an assignment bounded by poly$(|\mathcal{E}|)$. Accordingly, if the values in $\mathcal{E}$ are given in unary, its feasibility can be decided in NLOGSPACE.*

3 Decision Procedures for NBWAs

In the automata-theoretic approach to LTL, we translate an LTL formula ψ over AP to an NBW $\mathcal{A}_\psi$ over the alphabet 2^{AP} that accepts exactly all the computations that satisfy ψ. Then, satisfiability of ψ is reduced to checking the nonemptiness of $\mathcal{A}_\psi$ and model checking of a system K with respect to ψ is reduced to checking the emptiness of the product $\mathcal{A}_{\neg\psi}\times K$, which models all the computations of K that do not satisfy ψ. In the case of LTL, the product $\mathcal{A}_{\neg\psi}\times K$ is an *input-free* NBW: the labels of K "feed" the transitions of $\mathcal{A}_{\neg\psi}$ with alphabet letters in 2^{AP}, and thus the product is a graph with an acceptance condition induced by the acceptance condition of $\mathcal{A}_{\neg\psi}$. The input-free setting is not less complex: in both settings nonemptiness is reduced to reachability and is done in NLOGSPACE.

In the case of ∀LTLA, while K feeds the transitions of the product $\mathcal{A}_{\neg\psi}\times K$ with an element in $2^{AP}\times\mathbb{Z}$, the guards on the transitions refer also to the variables in X, and thus the edges in $\mathcal{A}_{\neg\psi}\times K$ are labeled by inequalities over the variables in X. As we show below, this makes the nonemptiness problem for NBWAs strongly related to ILP in both the general and the input-free settings.

Theorem 2. *The nonemptiness problem for NBWAs is NP-complete. NP-hardness holds already for input-free NBWAs.*

Proof. For the upper bound, consider an NBWA $\mathcal{A}$ over variables in X. As in NBWs, $\mathcal{A}$ is not empty iff there is an accepting lasso in $\mathcal{A}$, namely a path from an initial state to an accepting state reachable from itself, and an assignment to the variables in X and to the $\star$'s along the lasso such that all guards are satisfied. The algorithm guesses such an accepting lasso (it is easy to see that a search for a lasso in which each state appears at most once is sufficient), resolves disjunctions in guards by guessing the disjunct to be satisfied, and checks the feasibility of the induced set $\mathcal{E}$ of inequalities. Each state q in the lasso contributes a new variable y_q that replaces the $\star$ in the guards in the transition from q to its successor in the lasso. Let Y be the union of all variables y_q, for states q in the lasso. The partition $\langle X, Y\rangle$ is $\star$-consistent. Hence, by Lemma 2, the feasibility of $\mathcal{E}$ can be decided in NP.

NP-hardness follows from NP hardness of ILP [41]. Indeed, we can arrange a given set of inequalities along the transitions of a single path from an initial state to an accepting loop of an NBWA. Clearly, the above can be done also in an input-free NBWA. □

As discussed in Section 2.4, the exponential blow-up in the possible values in a satisfying assignment to $X = \{x_1, \ldots, x_{|N|}\}$ is due to our ability to force the accepting lasso to induce a set of inequalities in which the assigned value of x_i multiples that of x_{i-1}. In Lemmas 3 and 4, we showed two ways to disable such a chain of assignments: either by forcing the guards to be simple, preventing the different variables in X to relate to each other, or by fixing their number. Hence, the following is an immediate corollary of Lemmas 3 and 4.

Theorem 3. *The nonemptiness problem is NLOGSPACE-complete for NBWAs with a constant number of variables and for NBWAs with simple guards.*

4 Decision Procedures for LTLA

In this section we present an automata-theoretic approach for LTLA and use it in order to solve the satisfiability problem for ∃LTLA and the model-checking problem for ∀LTLA. We first translate ∃LTLA formulas to NBWAs of exponential size. The construction is similar to the LTL to NBW construction in [46], except that the closure of the formula, and hence also the states of the NBWA, refer also to the guards in the formula, which induce guards on the transitions.

Theorem 4. *Given an ∃LTLA formula $\exists\varphi$, there is an NBWA $\mathcal{A}_\varphi$ such that $L(\mathcal{A}_\varphi) = L(\varphi)$, the size of $\mathcal{A}_\varphi$ is exponential in $|\varphi|$, and the guards that label the transitions of $\mathcal{A}_\varphi$ are conjunctions of atomic guards in φ and their negations.*

Proof. Consider an ∃LTLA formula $\exists\varphi$. The *extended closure* of φ, denoted $ecl(\varphi)$, includes φ and all its subformulas and their negations. Formally, the set $ecl(\varphi)$ is the smallest set of formulas that satisfies the following (we unify φ_1 with $\neg\neg\varphi_1$):

- $\varphi \in ecl(\varphi)$,
- $\varphi_1 \in ecl(\varphi)$ iff $\neg\varphi_1 \in ecl(\varphi)$,
- if $\varphi_1 \wedge \varphi_2 \in ecl(\varphi)$ or $\varphi_1 \mathbf{U} \varphi_2 \in ecl(\varphi)$, then $\varphi_1 \in ecl(\varphi)$ and $\varphi_2 \in ecl(\varphi)$, and
- if $\mathbf{X}\varphi \in ecl(\varphi)$ then $\varphi \in ecl(\varphi)$.

Note that we keep decomposing subformulas that are Boolean assertions until we reach atomic propositions or atomic guards.

We say that a subset $T \subseteq ecl(\varphi)$ is *good in* $ecl(\varphi)$ if T is a maximal set of formulas in $ecl(\varphi)$ that does not have propositional inconsistency. Formally, we require T to satisfy the following conditions:

- for every $\varphi_1 \in ecl(\varphi)$, we have $\varphi_1 \in T$ iff $\neg\varphi_1 \notin T$,
- $\bigwedge_{\gamma \in \mathcal{G}^A_X \cap T} \gamma$ is satisfiable, and
- for every $\varphi_1 \wedge \varphi_2 \in ecl(\varphi)$, we have $\varphi_1 \wedge \varphi_2 \in T$ iff $\varphi_1 \in T$ and $\varphi_2 \in T$.

The main idea of the translation of ∃LTLA formulas to NBWAs is to reduce the question of satisfaction of an ∃LTLA formula $\exists\varphi$ in a computation π to questions about the satisfaction of formulas in $ecl(\varphi)$ in the suffixes of π. For an ∃LTLA formula $\exists\varphi$ over X, we define $\mathcal{A}_\varphi = \langle 2^{AP}, X, Q, Q_0, \Delta, \alpha\rangle$, where $Q = \{T : \ T \text{ is a good subset in } ecl(\varphi)\}$ and $Q_0 = \{T : \ \varphi \in T\}$. The transitions are defined such that for every state $T \subseteq ecl(\psi)$, all the computations that leave T satisfy all the subformulas in T. Technically, the transition relation $\Delta \subseteq Q \times 2^{AP} \times \mathcal{G}_X \times Q$ is defined such that $\langle T, \sigma, \gamma, T'\rangle \in \Delta$ iff the following hold.

- $\sigma = AP \cap T$ and $\gamma = \bigwedge_{\gamma' \in \mathcal{G}^A_X \cap T} \gamma'$
- for every $\varphi \in ecl(\varphi)$, we have that $\mathbf{X}\varphi \in T$ iff $\varphi \in T'$, and
- for every $\varphi_1, \varphi_2 \in T$, we have that $\varphi_1 \mathbf{U} \varphi_2 \in T$ iff $\varphi_2 \in T$ or both $\varphi_1 \in T$ and $\varphi_1 \mathbf{U} \varphi_2 \in T'$.

We first define the automaton with respect to generalized Büchi acceptance condition, where every formula $\varphi_1 \mathbf{U} \varphi_2$ contributes to the condition the set $\{T \in Q : \varphi_2 \in T \text{ or } \neg(\varphi_1 \mathbf{U} \varphi_2) \in T\}$. By taking at most $|\varphi|$ copies of the automaton, we can translate it to an NBWA with at most $2^{O(|\varphi|)}$ states.

Finally, note that the set of guards that appear in the transitions in $\mathcal{A}_\varphi$ are conjunctions of atomic guards that either appear in φ or their negation appear in φ. □

A naive combination of Theorem 4 with the complexity of the nonemptiness algorithm described in Section 3 leads to an NEXPTIME upper bound for satisfiability. Below we provide a careful analysis of the setting, which takes into account the fact that the atomic guards in $\mathcal{A}_\varphi$ origin from φ, thus their number is only polynomial in φ.

Theorem 5. *The satisfiability problem for ∃LTLA is PSPACE-complete.*

Proof. Given an ∃LTLA formula $\exists\varphi$ over X, its satisfiability is reduced to checking the nonemptiness of $\mathcal{A}_\varphi$. Consider an accepting lasso in $\mathcal{A}_\varphi$, and the $\langle X, Y\rangle$ $\star$-consistent partition of a set $\mathcal{E}$ of inequalities induced by resolving disjunctions in the guards along it as in the proof of Theorem 2. Since $|X| \leq |\varphi|$, then, by Lemma 2, the domain of values that have to be checked when we decide the feasibility of $\mathcal{E}$ is exponential in $|\varphi|$. Thus, we can guess and store an assignment to the variables and generate the accepting lasso on-the-fly in PSPACE. The lower bound follows from the PSPACE-hardness of LTL satisfiability [43]. □

In the context of model checking, we have to combine the analysis above with a decomposition of the product into K and $\mathcal{A}_\varphi$.

Theorem 6. *The model-checking problem for ∀LTLA is PSPACE-complete. The nondeterministic space complexity is polynomial in the specification and logarithmic in the system.*

Proof. Given a system $K = \langle AP, W, R, W_0, l, v\rangle$ and a $\forall$LTLA formula $\forall\varphi$, we need to decide whether there is a computation of K that satisfies the $\exists$LTLA formula $\exists\neg\varphi$. For an atomic guard $\gamma \in \mathcal{G}_X^A$ that includes $\star$ and a value $m \in \mathbb{Z}$, let $\gamma(m)$ be the atomic guard obtained from γ by replacing $\star$ by m. For example, if $\gamma = (\star \leq x)$, then $\gamma(m) = (m \leq x)$.

Let $\mathcal{A}_{\neg\varphi} = \langle 2^{AP}, X, Q, Q_0, \Delta, \alpha\rangle$ be the NBWA for $\neg\varphi$. Taking the product of $\mathcal{A}_{\neg\varphi}$ with K, we get an input-free NBWA $\mathcal{A}_{\neg\varphi} \times K = \langle\{a\}, X, Q \times W, Q_0 \times W_0, \Delta', \alpha \times W\rangle$, where $\Delta' \subseteq (Q \times W) \times \{a\} \times \mathcal{G}_X \times (Q \times W)$ is such that there is a transition $\langle\langle q, w\rangle, a, \gamma', \langle q', w'\rangle\rangle \in \Delta'$ iff $\langle w, w'\rangle \in R$ and there is a transition $\langle q, l(w), \gamma, q'\rangle \in \Delta$ such that $\gamma' = \gamma(v(w))$. By Lemma 1, the domain of values that have to be considered when we check the feasibility of a set of equations induced by an accepting lasso is linear in $|K|$ and exponential in $|\varphi|$. Indeed, the equations refer to $|X|$ variables, with $|X| \leq |\varphi|$, their size is $|K| \cdot 2^{O(|\varphi|)}$, and the largest value appearing in them is linear in $|K|$ and $|\varphi|$. Accordingly, we can check $\mathcal{A}_{\neg\varphi} \times K$ for emptiness in PSPACE in $|\varphi|$ and NLOGSPACE in $|K|$. Indeed, we need space polynomial in $|\varphi|$ and only logarithmic in $|K|$ in order to store a state in the product and a guess for a satisfying assignment. Hardness follows from hardness for LTL model checking. □

In Section 2.3, we proved that the satisfiability problem for $\forall$LTLA is undecidable. In the setting of finite domains, satisfiability can be reduced to model checking a system that generates all possible computations. Also, in our setting, an $\forall$LTLA formula $\forall\varphi$ is satisfiable if a system that generates all computations does not satisfy the $\exists$LTLA formula $\exists\neg\varphi$. Thus, $\forall$LTLA satisfiability can be reduced to $\exists$LTLA model checking in infinite-state systems, making the latter undecidable. In the case, however, of finite systems, $\exists$LTLA model checking is decidable. To see this, consider a system K and an $\exists$LTLA formula $\exists\varphi$ over X, and let π be a computation of K. Since $\exists\varphi$ has a bounded number of atomic guards, and K is labeled with a bounded number of values in $\mathbb{Z}$, it is not hard to see that if π satisfies $\exists\varphi$, then π satisfies $\exists\varphi$ under an assignment that is bounded by some number c, where c depends on $|\varphi|$ and on $|K|$. Thus, checking whether all the computations of K satisfy $\exists\varphi$ can be reduced to checking whether all the computations of K satisfy φ_f for some $f : X \to \mathbb{Z}$ bounded by c. Thus, given a system K and an $\exists$LTLA formula $\exists\varphi$, one can model check K with respect to $\exists\varphi$ by computing such a bound c, constructing a variable-free formula $\varphi' = \bigvee_{f:X\to[-c,c]} \varphi_f$, and solving model checking for K and φ'.

5 Applications to Hierarchical Systems

We present an application of LTLA to hierarchical systems, where the $\mathbb{Z}$-component is a natural number that refers to the depth of computations in the system. We start with defining hierarchical systems, and then study their model checking.

5.1 Hierarchical Systems

A *hierarchical system* is a tuple $K = \langle K_1, \ldots, K_n\rangle$ of subsystems. For every $1 \leq i \leq n$, we have that $K_i = \langle AP, W_i, B_i, \tau_i, R_i, W_0^i, W_{Exit}^i, l_i\rangle$, where AP is a set of atomic propositions, W_i is a set of states, B_i is a set of boxes, $\tau_i : B_i \to \{i+1, \ldots, n\}$ is a function that maps every box in B_i to a subsystem K_j, for $j > i$, $W_0^i \subseteq W_i$ is a set of initial states, $W_{Exit}^i \subseteq W_i$ is a set of exit states, $l_i : W_i \to 2^{AP}$ is a labeling function, and R_i is a transition relation. Every transition leaves a state in W_i or an exit state of a box in B_i, and enters a state in W_i or an initial state of a box in B_i. Formally, let $\tau(B_i)$ be the set of all indices j such that $\tau_i(b) = j$

for some box $b \in B_i$. Then, $R_i \subseteq ((\bigcup_{j:j\in\tau(B_i)} W^j_{Exit}) \cup W_i) \times ((\bigcup_{j:j\in\tau(B_i)} W^j_0) \cup W_i)$. The *size* of a hierarchical system $K = \langle K_1, \ldots, K_n\rangle$ is $\sum_{1\leq i\leq n} |K_i|$.

A system without boxes is *flat*. Each hierarchical system can be transformed to a flat system, referred to as its *flat expansion*, by recursively substituting each box by a copy of the suitable subsystem. Note that all the states in the flat expansion of K have nesting depth 1. In order to retain information about the internal structure of K, we add to the flat expansion a function that labels each state by its nesting depth in K. Formally, given a hierarchical system K, for each subsystem K_i we inductively define its flat expansion $K^F_i = \langle AP, W^F_i, R^F_i, W^i_0, W^i_{Exit}, l^F_i, v_i\rangle$, as follows. The state space is $W^F_i = W_i \cup (\cup_{b\in B_i}\{b\} \times W^F_{\tau_i(b)})$. Note that the same subsystem may be called by different boxes. By substituting each box b by a set of states $\{b\} \times W^F_{\tau_i(b)}$, we preserve b as an identifier in the names of the states of the called subsystem. The transition relation R^F_i includes the following transitions:

- (u_1, u_2) such that $u_1, u_2 \in W_i$ and $(u_1, u_2) \in R_i$,
- $(u_1, \langle b, u_2\rangle)$ such that $u_1 \in W_i$, $u_2 = W_0^{\tau_i(b)}$ and $(u_1, u_2) \in R_i$,
- $(\langle b, u_1\rangle, u_2)$ such that $u_1 \in W^{\tau_i(b)}_{Exit}$, $u_2 \in W_i$ and $(u_1, u_2) \in R_i$, and
- $(\langle b, u_1\rangle, \langle b, u_2\rangle)$ such that $u_1, u_2 \in W^F_{\tau_i(b)}$ and $(u_1, u_2) \in R^F_{\tau_i(b)}$.

The labeling function $l^F_i : W^F_i \to 2^{AP}$ labels every state $u \in W_i$ with $l_i(u)$, and every state $\langle b, u\rangle$, where $u \in W^F_{\tau_i(b)}$, with $l^F_i(u)$. Finally, the labeling function $v_i : W^F_i \to \{1, \ldots, n\}$ labels every state in K^F_i with its *nesting depth* in K_i. Thus, for every $u \in W_i$, we have that $v_i(u) = 1$, and for every $b \in B_i$ and $u \in W^F_{\tau_i(b)}$, we have that $v_i(\langle b, u\rangle) = v_{\tau_i(b)}(u) + 1$. Note that states in W_1 have depth 1, and all other states in K^F are of the form $\langle b_1, \langle b_2, \langle \ldots, \langle b_k, u\rangle \cdots\rangle\rangle\rangle$, and have depth $k + 1$. Note also that the domain of v_i is $\{1, \ldots, n - i + 1\}$. The system K^F_1 is the flat expansion of K, and we denote it by K^F.

As an example, consider the hierarchical system $K = \langle K_1, K_2\rangle$ in Figure 2. The subsystem K_1 includes two boxes, b_1 and b_2, with $\tau_1(b_1) = \tau_1(b_2) = 2$. The subsystem K_2 is flat. The flat system K^F is described on the right. In K^F, the states w^1_0, v_1, v_2, v_3, v_4, and v_5 have depth 1, and all other vertices have depth 2.

It is not hard to see that the hierarchical setting is exponentially more succinct, for example when K_i includes two boxes that call K_{i+1}, for all $1 \leq i < n$ [3].

5.2 ∀LTLA Model Checking for Hierarchical Systems

Consider a hierarchical system K. Recall that every state in K^F has a nesting depth in K, as defined above. When we reason about K^F and specify its properties with ∀LTLA specifications, we may refer to the nesting structure of K. For example, the ∀LTLA formula $\forall x; \mathbf{G}((send \wedge (\star = x)) \rightarrow ((\star \geq 2) \wedge (\star = x)\mathbf{U}\, ack))$ states that sending cannot be performed in the outermost component, and that whenever a sending is performed, the control stays in the current component until an ack is received. In the model-checking problem, we are given a hierarchical system K and a ∀LTLA formula ψ, and we have to decide whether all the computations of K satisfy ψ. A straightforward approach to model checking is to flatten K and then continue as described in the proof of Theorem 6. Flattening K, however, involves an exponential blow-up, which we want to avoid. An algorithm that avoids this blow-up for LTL model checking is described in [4]. The main idea is to construct the product of $\mathcal{A}_{\neg\psi}$ with the components of K in an iterative manner, starting with the innermost component and

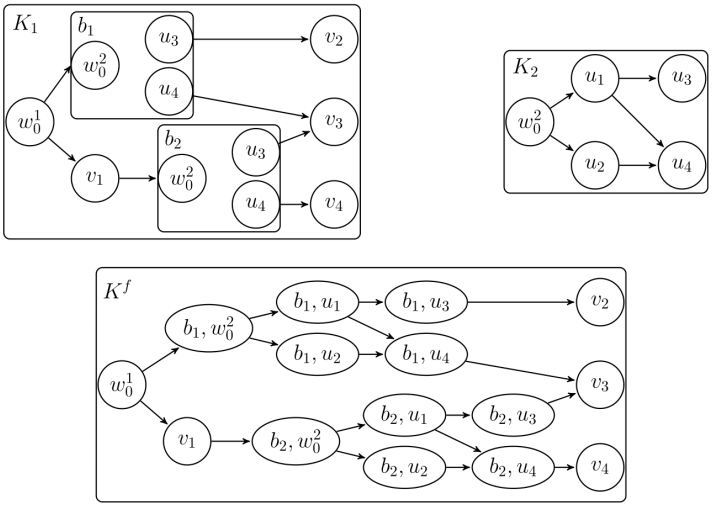

Figure 2: A hierarchical system and its flat expansion.

compressing every such product. In more detail, for every $1 \leq i \leq n$, starting with $i = n$, we construct $\mathcal{A}_{\neg\psi} \times K_i$ and generate from it a small gadget that retains only the necessary information about reachability and accepting cycles in $\mathcal{A}_{\neg\psi} \times K_i$. The gadget is computed only once, and its size is linear in $\mathcal{A}_{\neg\psi}$ and the number of exit states of K_i. If K_i has a box b that calls K_j, then, as $j > i$, the gadget corresponding to $\mathcal{A}_{\neg\psi} \times K_j$ has already been calculated, and we replace b by this gadget. Thus, the blow-up of flattening K is avoided, and indeed LTL model-checking can be solved in time polynomial in K and exponential in ψ. In terms of space complexity, the problem remains PSPACE-complete also for hierarchical systems [4].

The algorithm described above for LTL model checking follows a general technique for efficient reasoning about hierarchical systems: an iterative compression of subsystems in a way that maintains the essential information and still results in subsystems whose size depends only on the number of exit states [27]. A naive application of this approach to ∀LTLA specifications might be infeasible. To see why, consider a hierarchical system $K = \langle K_1, \ldots, K_n \rangle$ and a ∀LTLA formula $\psi = \forall\varphi$. Let $\mathcal{A}_{\neg\varphi}$ be the NBWA that corresponds to $\exists\neg\varphi$. Unlike the case of LTL, the product $\mathcal{A}_{\neg\varphi} \times K_i$, for $1 \leq i \leq n$, may depend on the nesting depth of K_i. Indeed, the guards on the transitions of $\mathcal{A}_{\neg\varphi}$ may include $\star$ and thus refer to the nesting depth. Thus, while the approach of [4] avoids the exponential blow-up in the flattening, its naive extension to ∀LTLA formulas requires i copies of the gadget for K_i – one for each possible nesting depth. We show how this upper bound can be tightened for simple ∀LTLA formulas.

Consider a hierarchical system $K = \langle K_1, \ldots, K_n \rangle$ and a ∀LTLA formula $\psi = \forall\varphi$ over X. For a subsystem K_i, a function $f : X \to \mathbb{Z}$ and a nesting depth $1 \leq d \leq i$, we define $(\mathcal{A}_{\neg\varphi} \times K_i)_{f,d}$ as the product NBWA $\mathcal{A}_{\neg\varphi} \times K_i$, where the nesting depth of K_i is d and the assignment to X is f. Lemma 5 below shows that the number of different assignments and nesting depths we should consider is exponential in $|\varphi|$ and is independent of n.

Lemma 5. *Consider a hierarchical system $K = \langle K_1, \ldots, K_n \rangle$ and a simple ∃LTLA formula $\exists\neg\varphi$ over X. There exist $c, c' \in \mathbb{N}$, linear in $|\varphi|$, such that the following hold.*

1. *If K satisfies φ, then K satisfies φ with an assignment bounded by c.*

2. *For every* $1 \leq i \leq n$*, assignment* f *bounded by* c*, and* $d \geq c'$*, we have that* $(\mathcal{A}_{\neg\varphi} \times K_i)_{f,d} = (\mathcal{A}_{\neg\varphi} \times K_i)_{f,c'}$.

Proof. The proof of the first claim follows the same considerations as in Lemma 3 (in fact, here the analysis is even tighter, as all $\star$'s from the same subgraph induce a single variable in Y). For the second claim, we define $c' = c + L + 1$, where L is the largest value. It is not hard to see that for every atomic guard γ in φ, assignment f bounded by c, and $d \geq c'$, we have that $d \models_f \gamma$ iff $c' \models_f \gamma$. Hence, reachability in $(\mathcal{A}_{\neg\varphi} \times K_i)_{f,d}$ and $(\mathcal{A}_{\neg\varphi} \times K_i)_{f,c'}$ coincides, and we are done. □

We can now define the product $(\mathcal{A}_{\neg\varphi} \times K_i)_{f,d}$. First, for a flat K_i, we define $(\mathcal{A}_{\neg\varphi} \times K_i)_{f,d} = \langle \{a\}, \emptyset, Q \times W_i, Q \times W_0^i, Q \times W_{Exit}^i, \Delta_i, \alpha \times W_i \rangle$, where $\Delta_i \subseteq (Q \times W_i) \times \{a\} \times \{true\} \times (Q \times W_i)$ is such that $\langle \langle q_1, w_1 \rangle, a, true, \langle q_2, w_2 \rangle \rangle \in \Delta_i$ iff $\langle q_1, l_i(w_1), \gamma, q_2 \rangle \in \Delta$, $R_i(w_1, w_2)$, and $d \models_f \gamma$. Essentially, $(\mathcal{A}_{\neg\varphi} \times K_i)_{f,d}$ is a product graph that preserves only transitions whose guards are satisfied by d under f.

The size of the product $(\mathcal{A}_{\neg\varphi} \times K_i)_{f,d}$ depends on the sizes of $\mathcal{A}_{\neg\varphi}$ and K_i. We now compress it, replacing paths by single edges, to retain only information about reachability to the exit vertices of K_i. We distinguish between reachability and reachability via an accepting state, and consider also the possibility of reaching an accepting cycle in K_i. Formally, we define the *compressed product* $(\mathcal{A}_{\neg\varphi} \otimes K_i)_{f,d}$ as follows. The state space of $(\mathcal{A}_{\neg\varphi} \otimes K_i)_{f,d}$ includes one copy of the initial states, two copies of the exit states of $(\mathcal{A}_{\neg\varphi} \times K_i)_{f,d}$, and a new accepting state. There are three types of transitions in $(\mathcal{A}_{\neg\varphi} \otimes K_i)_{f,d}$. All are from the initial states, and they are either to the new accepting state, in the case that there is a reachable accepting cycle in $(\mathcal{A}_{\neg\varphi} \times K_i)_{f,d}$, to the first copy of the exit states, in the case that the exit states are reachable, or to the second copy of the exit states, in the case that the exit states are reachable via an accepting state. Note that an accepting cycle in $(\mathcal{A}_{\neg\varphi} \times K^F)_{f,1}$ may include states from different subsystems, meaning that it enters and exits a box. In this case, the accepting state that makes the loop accepting may be either in the calling or in the called subsystem. The former case is captured by reachability to the first copy of an exit state, while the latter is captured by reachability to the second copy. Formally, we define $(\mathcal{A}_{\neg\varphi} \otimes K_i)_{f,d} = \langle \{a\}, \emptyset, \{q_{acc}^i\} \cup Q \times (W_0^i \cup (\{0,1\} \times W_{Exit}^i)), Q \times W_0^i, Q \times \{0,1\} \times W_{Exit}^i, \Delta_i', (\alpha \times (W_0^i \cup \{0\} \times W_{Exit}^i)) \cup \{q_{acc}^i\} \cup (Q \times \{1\} \times W_{Exit}^i) \rangle$. The transition relation $\Delta_i' \subseteq ((Q \times W_0^i) \cup \{q_{acc}^i\}) \times \{a\} \times \{true\} \times ((Q \times \{0,1\} \times W_{Exit}^i) \cup \{q_{acc}^i\})$ is such that there is a transition $\langle u_1, a, true, u_2 \rangle \in \Delta_i'$ when $u_1 = u_2 = q_{acc}^i$, or one of the following holds.

- $u_1 \in Q \times W_0^i$, and either $u_2 = \langle q, 0, w \rangle \in Q \times \{0\} \times W_{Exit}^i$ and $\langle q, w \rangle$ is reachable from u_1 in $(\mathcal{A}_{\neg\varphi} \times K_i)_{f,d}$, or $u_2 = \langle q, 1, w \rangle \in Q \times \{1\} \times W_{Exit}^i$ and $\langle q, w \rangle$ is reachable via an accepting state from u_1 in $(\mathcal{A}_{\neg\varphi} \times K_i)_{f,d}$.
- $u_1 \in Q \times W_0^i$, $u_2 = q_{acc}^i$, and there is a reachable accepting cycle from u_1 in $(\mathcal{A}_{\neg\varphi} \times K_i)_{f,d}$.

Note that Δ_i' can be computed in time polynomial in $|(\mathcal{A}_{\neg\varphi} \times K_i)_{f,d}|$. In addition, it is easy to see that $|(\mathcal{A}_{\neg\varphi} \otimes K_i)_{f,d}|$ is linear in $|Q| \cdot |W_0^i \cup W_{Exit}^i|$.

It left to show how $(\mathcal{A}_{\neg\varphi} \times K_i)_{f,d}$ is constructed in the case K_i is not flat. The idea is similar, except that we replace the product with boxes by the compressed product of the corresponding subsystem. Recall that a call to a subsystem increases the depth of the computation. Thus, we should replace boxes that call K_j with $(\mathcal{A}_{\neg\varphi} \otimes K_i)_{f,d+1}$. Here, we use the second claim in Lemma 5, and increase d only up to the bound c' form the lemma. Formally, for $1 \leq d < n$, let $inc(d) = \min\{d+1, c'\}$. Then, we construct $(\mathcal{A}_{\neg\varphi} \times K_i)_{f,d}$ by replacing boxes that call K_j with $(\mathcal{A}_{\neg\varphi} \otimes K_i)_{f,inc(d)}$, as follows. For every $1 \leq i \leq n$, let $U_{i,f,d}$ and $U_{i,f,d}^{\alpha}$ be the states and

accepting states of $(\mathcal{A}_{\neg\varphi} \otimes K_i)_{f,d}$, respectively. We define $(\mathcal{A}_{\neg\varphi} \times K_i)_{f,d} = \langle \{a\}, \emptyset, Q \times W_i \cup (\cup_{b \in B_i} \{b\} \times U_{\tau_i(b),f,inc(d)}), Q \times W_0^i, Q \times W_{Exit}^i, \Delta_i, (\alpha \times W_i) \cup (\cup_{b \in B_i} \{b\} \times U^{\alpha}_{\tau_i(b),f,inc(d)}) \rangle$. The transition relation Δ_i consists of the following transitions:

- $\langle \langle q_1, w_1 \rangle, a, true, \langle q_2, w_2 \rangle \rangle$, where $\langle q_1, l(w_1), \gamma, q_2 \rangle \in \Delta$, $R_i(w_1, w_2)$ and $d \models_f \gamma$,
- $\langle \langle q_1, w_1 \rangle, a, true, \langle b, \langle q_2, w_2 \rangle \rangle \rangle$, where $\langle q_1, l(w_1), \gamma, q_2 \rangle \in \Delta$, $b \in B_i$, $R_i(w_1, \langle b, w_2 \rangle)$ and $d \models_f \gamma$,
- $\langle \langle b, \langle q_1, w_1 \rangle \rangle, a, true, \langle q_2, w_2 \rangle \rangle$, where $\langle q_1, l(w_1), \gamma, q_2 \rangle \in \Delta$, $b \in B_i$, $R_i(\langle b, w_1 \rangle, w_2)$ and $inc(d) \models_f \gamma$, and
- $\langle \langle b, u_1 \rangle, a, true, \langle b, u_2 \rangle \rangle$ such that $b \in B_i$, $u_1, u_2 \in U_{\tau_i(b),f,inc(d)}$, and $\langle u_1, a, true, u_2 \rangle$ is a transition in $(\mathcal{A}_{\neg\varphi} \otimes K_{\tau_i(b)})_{f,inc(d)}$.

We can now conclude with the following.

Theorem 7. *$\forall$LTLA model checking for hierarchical systems is PSPACE-complete. For simple $\forall$LTLA formulas, it can be solved in time polynomial in the size of the system and exponential in the size of the formula.*

Proof. Consider an $\forall$LTLA formula $\forall\varphi$ and a hierarchical system K. Membership in PSPACE is easy, since (as is the case already for LTL model checking of hierarchical systems [4]) the polynomial space dependency is in both φ and K. Thus, we can reason about $\mathcal{A}_{\neg\varphi} \times K^F$, where each state can be stored in space polynomial in $|K| + |\varphi|$. Hardness in PSPACE follows form hardness in the flat setting.

We turn to describe an algorithm for simple $\forall$LTLA formulas. Let $K = \langle K_1, \ldots, K_n \rangle$. We proceed iteratively for every subsystem K_i, starting with $i = n$. In the i-th iteration, we compute $(\mathcal{A}_{\neg\varphi} \otimes K_i)_{f,d}$ for every f bounded by c and $1 \leq d \leq \min\{c', i\}$, where c and c' are as in Lemma 5. After computing $(\mathcal{A}_{\neg\varphi} \otimes K_1)_{f,1}$ for every f bounded by c, we check whether it includes, for some f, an accepting cycle reachable from a state in $Q_0 \times W_0^1$. By Lemma 5, it is sufficient to construct product automata only for functions bounded by c and depths at most c'.

We analyze complexity of this algorithm. For every $1 \leq i \leq n$, the algorithm constructs at most $(2c+1)^{|X|} \cdot c'$ copies of $\mathcal{A}_{\neg\varphi} \times K_i$. Every such construction is done in time $O(2^{|\varphi|} \cdot |K_i|)$. in addition, constructing $(\mathcal{A}_{\neg\varphi} \otimes K_i)_{f,d}$ is done in time linear in $|(\mathcal{A}_{\neg\varphi} \times K_i)_{f,d}|$. If φ is given in binary, this leads to a time complexity of $O(2^{|\varphi| \cdot |X|} \cdot |K|)$, as $c^{|X|} \cdot c' = O(2^{|\varphi| \cdot |X|})$. If φ is given in unary, the time complexity becomes $O(2^{|\varphi| + |X| \cdot \log(|\varphi|)} \cdot |K|)$, as $c^{|X|} \cdot c' = O(|\varphi|^{|X|}) = O(2^{|X| \cdot \log(|\varphi|)})$. Since $|X| \leq |\varphi|$, we are done. □

6 Discussion and Directions for Future Research

We introduced LTLA – a formalism for reasoning about systems with variables over $\mathbb{Z}$, and showed its applications in reasoning about systems over unbounded domains, especially over the domain of whole numbers, where arithmetic is supported, and in the setting of hierarchical systems. Our contribution enables the specification and verification of properties that refer to the internal structure of the hierarchical system and to its call-return behaviour. Our specification formalism does not require an a-priori knowledge of the maximal nesting depth and our model-checking algorithm avoids the exponential flattening of the system. Below we discuss some directions for future research.

LTLA satisfiability and synthesis for hierarchical systems The problem of ∃LTLA satisfiability for hierarchical systems is to decide whether there is a hierarchical system that satisfies a given ∃LTLA formula. The problem is more involved than general ∃LTLA satisfiability, as the value-components in the guessed computation should correspond to the nesting depth of states in a computation of a hierarchical system: it has to start with value 1 and to increase or decrease the value in at most 1 in every step. While we conjecture that the problem can be solved in PSPACE, essentially by combining the nonemptiness algorithm for NBWAs with a partition of $\mathbb{Z}$ to equivalence classes, we think that a more practical formulation of the satisfiability problem in the context of hierarchical systems should take into an account the goal of generating succinct hierarchical systems. Thus, it should be combined with some quality measures, as in [6], which direct the algorithm to search for solutions that use the hierarchy in a meaningful way. We find this, as well as a similar extension to the *synthesis* problem, to be of both theoretical and practical interests.

Branching time Similarly to the extension of LTL to LTLA, one could add arithmetics to branching time temporal logics [15]. In particular, we suggest to consider *CTL with arithmetics* (CTLA). Several variants of extensions of CTL with values over infinite domains have already been studied [13, 44]. An interesting problem is CTLA model checking for hierarchical systems. In [4], the authors describe a CTL model checking for hierarchical systems that avoids their flattening. The key idea is to bound the information that a compressed subsystem needs to maintain. It is shown in [4] that this information can be exponential in the number of exit states. As in the case of LTLA, a naive extension of this approach to CTLA requires taking many copies of each subsystem – one for each nesting depth. We believe that as in the case of LTLA, it is possible to avoid this blow up by keeping variables in the compressed subsystems.

Recursive systems A *recursive system* is a hierarchical system that allows unbounded nesting of components. In particular, there is no bound on the nesting depth of components. Verification of recursive systems is tightly related to reasoning about *pushdown systems* [1, 49]. Indeed, flattening of a recursive system results in a infinite-state system that behaves in a pushdown manner. Since LTLA handles values in $\mathbb{Z}$, it can naturally specify such systems. An interesting future work is to extend model-checking algorithms of recursive systems to specifications in LTLA.

Acknowledgmenets We thank Ofer Strichman for helpful discussions on the complexity of ILP.

References

[1] R. Alur, M. Benedikt, K. Etessami, P. Godefroid, T. W. Reps, and M. Yannakakis. Analysis of recursive state machines. *ACM Transactions on Programming Languagues and Systems*, 27(4):786–818, 2005.

[2] R. Alur, L. D'Antoni, J.V. Deshmukh, M. Raghothaman, and Y. Yuan. Regular functions and cost register automata. In *Proc. 28th Symp. on Logic in Computer Science*, pages 13–22, 2013.

[3] R. Alur, S. Kannan, and M. Yannakakis. Communicating hierarchical state machines. In *Proc. 26th Int. Colloq. on Automata, Languages, and Programming*, LNCS 1644, pages 169–178. Springer, 1999.

[4] R. Alur and M. Yannakakis. Model checking of hierarchical state machines. *ACM Transactions on Programming Languagues and Systems*, 23(3):273–303, 2001.

[5] B. Aminof, O. Kupferman, and A. Murano. Improved model checking of hierarchical systems. *Information and Computation*, 210:68–86, 2012.

[6] G. Avni and O. Kupferman. Synthesis from component libraries with costs. In *Proc. 25th Int. Conf. on Concurrency Theory*, LNCS 8704, pages 156–172. Springer, 2014.

[7] B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using qdds. In *Proc. 8th Int. Conf. on Computer Aided Verification*, LNCS 1102, pages 1–12. Springer, 1996.

[8] M. Bojańczyk, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data trees and xml reasoning. *Journal of the ACM*, 56(3):1–48, 2009.

[9] M. Bojanczyk, A. Muscholl, T. Schwentick, L. Segoufin, and C. David. Two-variable logic on words with data. In *Proc. 21st IEEE Symp. on Logic in Computer Science*, pages 7–16, 2006.

[10] A. Bouajjani, P. Habermehl, Y. Jurski, and M. Sighireanu. Rewriting systems with data. In *FCT*, pages 1–22, 2007.

[11] A. Bouajjani, P. Habermehl, and R R. Mayr. Automatic verification of recursive procedures with one integer parameter. *Theoretical Computer Science*, 295:85–106, 2003.

[12] A. Bouajjani, P. Habermehl, and T. Vojnar. Verification of parametric concurrent systems with prioritised FIFO resource management. *Formal Methods in System Design*, 32(2):129–172, 2008.

[13] C. Carapelle, A. Kartzow, and M. Lohrey. Satisfiability of ECTL* with constraints. *Journal of Computer and Systems Science*, 82(5):826–855, 2016.

[14] Y-F. Chen, O. Lengál, T. Tan, and Z. Wu. Register automata with linear arithmetic. In *Proc. 32nd IEEE Symp. on Logic in Computer Science*, pages 1–12, 2017.

[15] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logic of Programs*, LNCS 131, pages 52–71. Springer, 1981.

[16] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[17] H. Comon and V. Cortier. Flatness is not a weakness. In *Proc. 9th Annual Conf. of the European Association for Computer Science Logic*, LNCS 1862, pages 262–276. Springer, 2000.

[18] H. Comon and Y. Jurski. Multiple counters automata, safety analysis and presburger arithmetic. In *Proc. 10th Int. Conf. on Computer Aided Verification*, LNCS 1427, pages 268–279. Springer, 1998.

[19] W-P. de Roever, H. Langmaack, and A. Pnueli, editors. *Compositionality: The Significant Difference. Proceedings of Compositionality Workshop*, LNCS 1536, Springer, 1998.

[20] S. Demri. Linear-time temporal logics with presburger constraints: an overview. *Journal of Applied Non-Classical Logics*, 16(3-4):311–348, 2006.

[21] S. Demri. LTL over integer periodicity constraints. *Theoretical Computer Science*, 360(1-3):96–123, 2006.

[22] S. Demri and R. Gascon. Verification of qualitative Z constraints. *Theoretical Computer Science*, 409(1):24–40, 2008.

[23] S. Demri and R. Lazic. LTL with the freeze quantifier and register automata. In *Proc. 21st IEEE Symp. on Logic in Computer Science*, pages 17–26, 2006.

[24] S. Demri and R. Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3):16:1–16:30, 2009.

[25] M. Droste, W. Kuich, and H. Vogler (eds.). *Handbook of Weighted Automata*. Springer, 2009.

[26] D. Drusinsky and D. Harel. On the power of bounded concurrency I: Finite automata. *Journal of the ACM*, 41(3):517–539, 1994.

[27] R. Faran and O. Kupferman. A parametrized analysis of algorithms on hierarchical graphs. In *Descriptional Complexity of Formal Systems - 19th IFIP WG 1.02 International Conference*, LNCS 10316, pages 114–127. Springer, 2017.

[28] H. Frenkel, O. Grumberg, and S. Sheinvald. An automata-theoretic approach to modeling systems and specifications over infinite data. In *NASA Formal Methods - 9th International Symposium*, LNCS 10227, pages 1–18. Springer, 2017.

[29] O. Grumberg, O. Kupferman, and S. Sheinvald. Variable automata over infinite alphabets. In *Proc. 4th Int. Conf. on Language and Automata Theory and Applications*, LNCS 6031, pages 561–572. Springer, 2010.

[30] O. Grumberg, O. Kupferman, and S. Sheinvald. Model checking systems and specifications with parameterized atomic propositions. In *10th Int. Symp. on Automated Technology for Verification and Analysis*, pages 122–136, 2012.

[31] H.W. Lenstra Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983.

[32] M. Kaminski and N. Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.

[33] M. Kaminski and D. Zeitlin:. Extending finite-memory automata with non-deterministic reassignment. In *AFL*, pages 195–207, 2008.

[34] D. Kroening and O. Strichman. *Decision Procedures - An Algorithmic Point of View, Second Edition*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2016.

[35] O. Kupferman and T. Tamir. Properties and utilization of capacitated automata. In *Proc. 34th Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 29 of *LIPIcs*, pages 33–44. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2014.

[36] L. Libkin, W. Martens, and D. Vrgoc. Querying graphs with data. *Journal of the ACM*, 63(2):14:1–14:53, 2016.

[37] M.L. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall, 1 edition, 1967.

[38] F. Neven, T. Schwentick, and V. Vianu. Towards regular languages over infinite alphabets. In *26th Int. Symp. on Mathematical Foundations of Computer Science*, pages 560–572, 2001.

[39] A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.

[40] A. Pnueli. In transition from global to modular temporal reasoning about programs. In K. Apt, editor, *Logics and Models of Concurrent Systems*, volume F-13 of *NATO Advanced Science Institutes*, pages 123–144. Springer, 1985.

[41] A. Schrijver. *Theory of linear and integer programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999.

[42] Y. Shemesh and N.: Francez. Finite-state unification automata and relational languages. *Information and Computation*, 114:192–213, 1994.

[43] A.P. Sistla and E.M. Clarke. The complexity of propositional linear temporal logic. *Journal of the ACM*, 32:733–749, 1985.

[44] F. Song and Z. Wu. On temporal logics with data variable quantifications: Decidability and complexity. *Information and Computation*, 251:104–139, 2016.

[45] T. Tan. *Pebble Automata for Data Languages: Separation, Decidability, and Undecidability*. PhD thesis, Technion - Computer Science Department, 2009.

[46] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

[47] V. Vianu. Automatic verification of database-driven systems: a new frontier. In *ICDT '09*, pages 1–13, 2009.

[48] J. von zur Gathen and M. Sieveking. A bound on solutions of linear integer equalities and inequalities. *Proceedings of the American Mathematical Society*, 72(1):155–158, 1978.

[49] I. Walukiewicz. Pushdown processes: Games and model-checking. *Information and Computation*, 164(2):234–263, 2001.