Linear-Time Verification of Data-Aware Dynamic Systems with Arithmetic

Paolo Felli, Marco Montali, Sarah Winkler*

Free University of Bozen-Bolzano – Bolzano – Italy

Abstract

Combined modeling and verification of dynamic systems and the data they operate on has gained momentum in AI and in several application domains. We investigate the expressive yet concise framework of data-aware dynamic systems (DDS), extending it with linear arithmetic, and provide the following contributions. First, we introduce a new, semantic property of "finite summary", which guarantees the existence of a faithful finite-state abstraction. We rely on this to show that checking whether a witness exists for a linear-time, finite-trace property is decidable for DDSs with finite summary. Second, we demonstrate that several decidability conditions studied in formal methods and database theory can be seen as concrete, checkable instances of this property. This also gives rise to new decidability results. Third, we show how the abstract, uniform property of finite summary leads to modularity results: a system enjoys finite summary if it can be partitioned appropriately into smaller systems that possess the property. Our results allow us to analyze systems that were out of reach in earlier approaches. Finally, we demonstrate the feasibility of our approach in a prototype implementation.

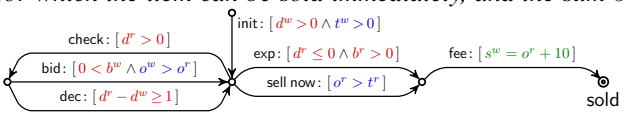
1 Introduction

The analysis of complex dynamic systems is a core research topic in AI. While process analysis has long focused on the control-flow perspective, in recent years a multi-perspective approach gained momentum, studying the interplay between control flow and data (Reichert 2012; Calvanese, de Giacomo, and Montali 2013; Calvanese et al. 2018; Deutsch et al. 2018). Verification in this setting is challenging, as it must deal with potentially infinitely many states.

This is aggravated in the presence of arithmetic, which is essential for practical applications (Deutsch et al. 2018): model checking of transition systems operating over simple, numeric data variables with arithmetic constraints is known to be undecidable, as it is easy to model a two-counter system. However, restrictions on the transition system have been shown to render certain verification tasks decidable. In particular, decidability has been obtained by confining the constraint language, as in the case of *monotonicity* constraints (Demri and D'Souza 2007) (e.g. $x \leq y$) and *gap-order* constraints (Mayr and Totzke 2016; Bozzelli and Pinchinat 2014) (e.g. $x - y \geq 2$), or by limiting the control flow, as in the case of *feedback freedom* (Damaggio, Deutsch, and Vianu 2012).

In this work, we focus on the framework of data-aware dynamic systems (DDSs) (de Leoni, Felli, and Montali 2020), an expressive yet concise model for process analysis, which we enrich with linear arithmetic. We call the resulting systems *DDSs with arithmetic (DDSAs)*, and study the verification problem for the linear-time, finite-trace temporal logic LTL_f (de Giacomo and Vardi 2013) extended with arithmetic constraints. The following is a motivating example.

Example 1.1. *Consider the process of an auction at an online market place. Its data variables are a timer d, the offer o by the last bidder, identified by b, a threshold price t for which the item can be sold immediately, and the sum s.*



The timer d is initialized to a number of days, and t is fixed (action init*). Then, while the timer did not expire (*check*), bids are taken (*bid*) or the timer may be decremented (*dec*). The auction ends if the timer expires and a bid was set (*exp*), or the offer exceeds t (*sell now*). Finally,* fee *sets s to the offer plus an auction fee. We will use our approach to verify that $\psi = \Box(\mathsf{sold} \wedge d > 0 \rightarrow o > t)$ holds, i.e., if the auction ends before the timer expires, the offer exceeds the threshold. The meaning of the colors will be clarified later.*

Our contribution is as follows. (1) First, we introduce the novel property of *finite summary*, and show that the above restrictions studied in the literature (i.e. (i) monotonicity constraints, (ii) gap-order constraints, (iii) feedback freedom) are instances of this property. We further generalize feedback freedom introducing a new, expressive property called (iv) *bounded lookback*. (2) Second, we prove that finite summary guarantees the existence of a *faithful, finite-state abstraction* for a DDSA by representing sets of states as logical constraints. This is used to show that checking existence of a witness for an LTL_f property is decidable. (3) Third, we illustrate a *modularity* result: if a DDSA $\mathcal{B}$ represents either the sequential, or parallel but variable-disjoint, execution of DDSAs with finite summary (possibly according to the different criteria (i)-(iv)), then also $\mathcal{B}$ enjoys this

*This work is partially supported by the UNIBZ projects DaCoMan, QUEST, SMART-APP, VERBA, and WineId.

property and is thus amenable to our verification technique.

To the best of our knowledge, LTL$_f$ model checking of such combinations of $(i) - (iv)$ is shown decidable for the first time (and the result is new for $(ii), (iv)$ individually).

To demonstrate feasibility, we implemented our approach in the tool ada, which tests for finite summary using $(i) - (iv)$, computes finite-state abstractions, and handles LTL$_f$ model checking using an SMT solver as backend.

Related Work. Verification of transition systems with arithmetic constraints has been studied in many areas including formal methods, database theory, and BPM. For monotonicity constraint (MC) systems, LTL model checking was proven decidable in (Demri and D'Souza 2007), even comparing variables multiple steps apart. An extended language is studied in (Demri 2006). DDSAs with MCs are also considered in (Felli, de Leoni, and Montali 2019) from the perspective of a finite-run semantics, giving an explicit procedure to compute finite, faithful abstractions. For gap-order constraint (GC) systems, reachability was shown decidable (Bozga, Gîrlea, and Iosif 2009). Also the existential fragment of CTL* with GCs is decidable, while the universal one is not (Bozzelli and Pinchinat 2014). A similar dichotomy was discovered for the EF and EG fragments of CTL (Mayr and Totzke 2016). We here consider LTL$_f$ model checking, a task suited to many applications (de Giacomo and Vardi 2013): For DDSAs with a finite summary, we prove decidability of our verification task, i.e., to check existence of a witness for an LTL$_f$ formula with constraints. Finite summary is based on the notion of *history constraints* from (Damaggio, Deutsch, and Vianu 2012), and we show that it generalizes their feedback freedom property, though their constraints may refer to a read-only database, a feature that we leave for future work. DDSAs generalize timed automata, and our abstraction shares with region graphs the representation of a "region" of reachable states by a formula (Alur and Dill 1994). The finite summary property does not cover timed automata with multiple clocks, while the one-clock-case is captured by MCs. Also in (Barrett, Demri, and Deters 2013) reachable states are abstracted by formulas, but our results are incomparable to both of these works. (Ghilardi et al. 2007) devise an abstract property that guarantees decidability of safety properties, if the given theory is compatible with a locally finite theory. For arithmetic this is not guaranteed, so that further work is required to compare their criterion with our work. Our method can be seen as a form of predicate abstraction, subject to a long line of research (e.g., (Clarke et al. 2004; Colón and Uribe 1998)); but in contrast to most works there, our abstraction is *strongly* preserving, i.e., our verification task is decidable.

Paper structure. In Sec. 2 we formalize DDSAs and our verification language and task. In Sec. 3 we develop the notion of finite summary and show how it yields finite state abstractions. Sec. 4 is devoted to our verification technique. In Sec. 5 we demonstrate four concrete classes implying finite summary, and in Sec. 6 we present modularity results. Sec. 7 describes our tool ada and concludes with directions for future work. All proofs and further examples can be found in an extended version (Felli, Montali, and Winkler 2021).

2 DDSs with Arithmetic

Here we enrich *data-aware dynamic systems (DDSs)* and the linear-time verification language from (de Leoni, Felli, and Montali 2020) with linear arithmetic constraints.

Model. We start by defining the set of arithmetic constraints over a domain D, which may be $\mathbb{Z}$, $\mathbb{Q}$, or $\mathbb{R}$:

Definition 2.1. A *constraint* c over a set V of variables is defined by the following grammar, where $k \in D$ and $v \in V$:

$e := v \mid k \mid e + e \mid e - e$

$c := e = e \mid e \neq e \mid e < e \mid e \leq e \mid c \wedge c$

The set of all constraints over domain D is denoted by $\mathcal{C}_D$. E.g., $x \neq 1$, $x < y - z$, and $x - y = 2$ are constraints over $\{x, y, z\}$ for domain $\mathbb{Z}$, $\mathbb{Q}$, or $\mathbb{R}$. From now on, V will be a fixed, finite set of variables. Two disjoint copies V^r and V^w of V, called the *read* and *write* variables, denote the variable values before and after a transition, respectively. We also write $\overline{V}$ for a vector that contains the variables V in an arbitrary but fixed order, and $\overline{V}^r$ and $\overline{V}^w$ for V^r and V^w ordered in the same way. Throughout this paper, by a *formula* φ we mean a boolean formula whose atoms are either propositional or constraints as in Def. 2.1. We are thus in the realm of SMT with linear arithmetic, which is decidable and admits *quantifier elimination*: if φ is a formula with free variables $X \cup \{y\}$, and atoms in $\mathcal{C}_D$ (cf. Def. 2.1), there is some φ' with free variables X that is equivalent to $\exists y.\varphi$, i.e., $\varphi' \equiv \exists y.\varphi$ (Presburger 1929). Here the relation $\equiv$ denotes logical equivalence. For a set C of constraints and a formula φ, we sometimes write $\varphi \wedge C$ for the formula $\varphi \wedge \bigwedge C$.

A *state variable assignment* α is a total function $\alpha \colon V \mapsto D$; we say that α *satisfies* a constraint c over V, written $\alpha \models c$, if the evaluation of c under α is true in D.

Definition 2.2. A *DDS with arithmetic (DDSA)* is a labelled transition system $\langle B, b_0, \mathcal{A}, T, F, V, \alpha_0, guard \rangle$, where:

- B is a finite set of *states*, with $b_0 \in B$ the initial one;
- $\mathcal{A}$ is a finite set of *actions*;
- $T \colon B \times \mathcal{A} \mapsto B$ is a *transition function*;
- $F \subseteq B$ is the set of *final states*;
- α_0 is the *initial state variable assignment*; and
- $guard \colon \mathcal{A} \mapsto \mathcal{C}_D$ specifies *executability constraints* on actions over variables $V^r \cup V^w$.

In Def. 2.2 we restrict to conjunctive guards: disjunction can be captured by multiple transitions between the same states. With this convention, the system in Ex. 1.1 can be transformed into an equivalent DDSA, and Fig. 1 shows further examples of DDSAs. Note that a guard simultaneously expresses a condition on the read variables, and an update on the written ones: for instance, $v^r < 7$ requires the current value of v to be less than 7, while $v^w - v^r \leq 7$ demands that the new value of v exceeds the current value by at most 7.

We denote a transition from state b to b' by executing an action $a \in \mathcal{A}$ as $b \xrightarrow{a} b'$. A *configuration* of $\mathcal{B}$ is a pair (b, α) where $b \in B$ and α is a state variable assignment.

A *guard assignment* β is a function $\beta \colon V^r \cup V^w \mapsto D$. As defined next, an action a transforms a configuration (b, α) into a new configuration (b', α') by changing state as defined by action a, and updating the state variable assignment in agreement with the action guard. In the new assignment α',

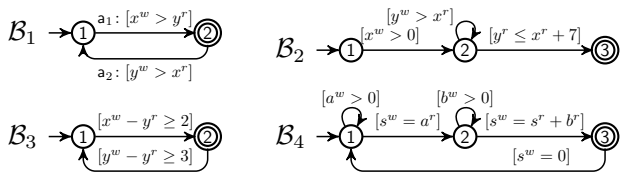


Figure 1: Simple DDSAs (with finite summary).

variables that are not written keep their previous value as per α, whereas written variables are updated according to the guard. Let $write(a) = \{x \mid x^w \in V^w$ occurs in $guard(a)\}$.

Definition 2.3. A DDSA $\mathcal{B} = \langle B, b_0, \mathcal{A}, T, F, V, \alpha_0, guard\rangle$ *admits a step* from configuration (b, α) to (b', α') via action a, denoted $(b, \alpha) \xrightarrow{a} (b', \alpha')$, if $b \xrightarrow{a} b'$ and the guard assignment β given by $\beta(v^r) = \alpha(v)$ and $\beta(v^w) = \alpha'(v)$ for all $v \in V$ satisfies the guard of a, i.e., $\beta \models guard(a)$ holds.

A *run* of length n is a sequence of steps $\rho\colon (b_0, \alpha_0) \xrightarrow{a_1} (b_1, \alpha_1) \xrightarrow{a_2} \cdots \xrightarrow{a_n} (b_n, \alpha_n)$, and ρ_i refers to (b_i, α_i). Note that a run always starts in the initial state (b_0, α_0).

Verification language. For a constraint set $\mathcal{C}$ over V and DDSA $\mathcal{B} = \langle B, b_0, \mathcal{A}, T, F, V, \alpha_0, guard\rangle$, let $\mathcal{L}_{\mathcal{BC}}$ be the language defined by the following grammar:

$$c \mid b \mid \psi \wedge \psi \mid \psi \vee \psi \mid \langle a\rangle\psi \mid \langle\cdot\rangle\psi \mid \Diamond\psi \mid \Box\psi \mid \psi \mathbin{\mathsf{U}} \psi$$

where $a \in \mathcal{A}$, $c \in \mathcal{C}$, and $b \in B$. Note that $\mathcal{L}_{\mathcal{BC}}$ does not support negation as we will also consider fragments where decidability is lost if constraints can be negated. However, if the set $\mathcal{C}$ is closed under negation,[1] $\mathcal{L}_{\mathcal{BC}}$ can express an arbitrary formula in negation normal form. We next lift the LTL$_f$ semantics (de Giacomo and Vardi 2013) to $\mathcal{L}_{\mathcal{BC}}$:

Definition 2.4. A run ρ of length n *satisfies* $\psi \in \mathcal{L}_{\mathcal{BC}}$, denoted $\rho \models \psi$, iff $\rho, 0 \models \psi$ holds, where for $0 \le i \le n$:

$\rho, i \models c$	iff $\rho_i = (b, \alpha)$ for some b and $\alpha \models c$;
$\rho, i \models b$	iff $\rho_i = (b, \alpha)$ for some α;
$\rho, i \models \psi_1 \wedge \psi_2$	iff $\rho, i \models \psi_1$ and $\rho, i \models \psi_2$;
$\rho, i \models \psi_1 \vee \psi_2$	iff $\rho, i \models \psi_1$ or $\rho, i \models \psi_2$;
$\rho, i \models \langle a\rangle\psi$	iff $i < n$, $\rho_i \xrightarrow{a} \rho_{i+1}$ and $\rho, i{+}1 \models \psi$;
$\rho, i \models \langle\cdot\rangle\psi$	iff $i < n$ and $\rho, i{+}1 \models \psi$;
$\rho, i \models \Diamond\psi$	iff $\rho, i \models \psi$ or ($i < n$ and $\rho, i{+}1 \models \Diamond\psi$);
$\rho, i \models \Box\psi$	iff $\rho, i \models \psi$ and ($i = n$ or $\rho, i{+}1 \models \Box\psi$);
$\rho, i \models \psi_1 \mathbin{\mathsf{U}} \psi_2$	iff $\rho, i \models \psi_2$ or ($i < n$ and both $\rho, i \models \psi_1$ and $\rho, i{+}1 \models \psi_1 \mathbin{\mathsf{U}} \psi_2$).

Verification problem. We use $\mathcal{L}_{\mathcal{BC}}$ to express properties over the finite traces of a DDSA $\mathcal{B}$. A run ρ is a *witness* for $\psi \in \mathcal{L}_{\mathcal{BC}}$ if (i) ρ ends in a final state of $\mathcal{B}$ and (ii) $\rho \models \psi$.

Definition 2.5 (Verification task)**.** Given a DDSA $\mathcal{B}$ and $\psi \in \mathcal{L}_{\mathcal{BC}}$, check whether there exists a witness ρ for ψ in $\mathcal{B}$.

If $\mathcal{C}$ is closed under negation, one can model check ψ by looking for a witness for $\neg\psi$, i.e., a counterexample.

Unsurprisingly, DDSAs can directly encode 2-counter Minsky machines, making the verification task undecidable.

Remark 2.6. *It is undecidable whether there exists a witness for a property $\Diamond b$ in a DDSA, for $b \in B$ a system state.*

[1] Here a constraint set $\mathcal{C}$ is *closed under negation* if for all $c \in \mathcal{C}$ there is some $c' \in \mathcal{C}$ such that $c' \equiv \neg c$.

3 DDSAa with Finite Summary

Instead of taming undecidability of verification by directly looking for decidable fragments, we introduce a *semantic property* called *finite summary*, and show that DDSAs with this property admit a faithful finite-state abstraction that preserves all properties expressible in our verification language. Throughout the section, we fix a DDSA $\mathcal{B} = \langle B, b_0, \mathcal{A}, T, F, V, \alpha_0, guard\rangle$ and a finite constraint set $\mathcal{C}$. We first consider paths in $\mathcal{B}$, called *symbolic runs*:

Definition 3.1. A *symbolic run* σ is a transition sequence $b_0 \xrightarrow{a_1} b_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} b_n$ where $b_i \in B$ and $a_i \in \mathcal{A}$; it *abstracts* any run of the form $(b_0, \alpha_0) \xrightarrow{a_1} (b_1, \alpha_1) \xrightarrow{a_2} \cdots \xrightarrow{a_n} (b_n, \alpha_n)$ i.e., a run with the same state and action sequence. The prefix of σ of i steps is denoted $\sigma|_i$.

For instance, for the DDSA $\mathcal{B}_1$ in Fig. 1 the sequence $1 \xrightarrow{\mathsf{a}_1} 2 \xrightarrow{\mathsf{a}_2} 1$ is a symbolic run. In this section, we aim to construct an abstract representation of the reachable configurations of $\mathcal{B}$, where we capture a set of configurations by a pair (b, φ) of a system state $b \in B$ and a formula φ with free variables V that describes the current state of the data. Our aim is to find a *finite* set of such pairs that covers all reachable configurations while being precise enough to decide our verification task. To that end, we next define the $update$ function as a uniform way to express how the current state, captured by a formula φ, changes by executing an action.

First, we define the *transition formula* Δ_a of action a as $\Delta_a(\overline{V}^r, \overline{V}^w) = guard(a) \wedge \bigwedge_{v \notin write(a)} v^w = v^r$. Intuitively, this formula states the conditions on variables *before and after* executing a: $guard(a)$ must be true and the values of all variables that are not written are propagated by inertia. Note that Δ_a has free variables $\overline{V}^r$ and $\overline{V}^w$; for variable vectors $\overline{X}$ and $\overline{Y}$ of the same length, let $\Delta_a(\overline{X}, \overline{Y})$ be the formula obtained from Δ_a by replacing $\overline{V}^r$ by $\overline{X}$ and $\overline{V}^w$ by $\overline{Y}$.

Definition 3.2. For a formula φ with free variables V and an action a, let $update(\varphi, a) = \exists \overline{U}.\varphi(\overline{U}) \wedge \Delta_a(\overline{U}, \overline{V})$, where $\overline{U}$ is a variable vector of the same length as $\overline{V}$ such that U is disjoint from V and variables in φ, to avoid variable capture.

For instance, for action a_1 in DDSA $\mathcal{B}_1$ of Fig. 1, $\Delta_{\mathsf{a}_1} = (x^w > y^r) \wedge (y^w = y^r)$; and for $\varphi = (x > 0) \wedge (y > x)$ we get $update(\varphi, \mathsf{a}_1) = \exists x'\, y'.(x' > 0) \wedge (y' > x') \wedge (x > y') \wedge (y = y')$. Using quantifier elimination, we get an equivalent, quantifier-free formula, for instance $(y > 0) \wedge (x > y)$.

A key notion for our approach are *history constraints*: formulas that sum up constraints collected along symbolic runs, possibly in combination with additional constraints that are needed for verification (i.e., constraints that occur in the property ψ to be checked). To express the latter, we consider *verification constraint sequences* $\mathbf{C}$ over constraint set $\mathcal{C}$, i.e., sequences $\mathbf{C} = \langle C_0, \ldots, C_n\rangle$ of sets $C_i \subseteq \mathcal{C}$. A prefix $\langle C_0, \ldots, C_m\rangle$ of $\mathbf{C}$ is denoted by $\mathbf{C}|_m$. Moreover, we denote by $C_{\alpha_0} = \{v = \alpha_0(v) \mid v \in V\}$ the set of *initial constraints*, to capture in a formula the initial assignment.

Definition 3.3. For a symbolic run $\sigma\colon b_0 \xrightarrow{a_1} b_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} b_n$, and verification constraint sequence $\mathbf{C} = \langle C_0, \ldots, C_n\rangle$, the *history constraint* $h(\sigma, \mathbf{C})$ is inductively defined by setting $h(\sigma, \mathbf{C}) = \bigwedge(C_{\alpha_0} \cup C_0)$ if $n = 0$, and $h(\sigma, \mathbf{C}) = update(h(\sigma|_{n-1}, \mathbf{C}|_{n-1}), a_n) \wedge C_n$ if $n > 0$.

Informally, the history constraint of a symbolic run is a formula that captures all variable constraints that must hold in the last state, i.e., it is a *summary* of the symbolic run, taking into account additional verification constraints $\mathbf{C}$ that will become relevant in Sec. 4. Note that symbolic runs may in fact feature a sequence of actions that is not executable due to guard conditions. In these cases history constraints are unsatisfiable. For simplicity, in what follows we do not rule out these explicitly (as it does not affect our results), though it is possible and in fact done in our implementation. We call $h(\sigma, \mathbf{C})$ a history constraint *of* $\mathcal{B}$ and $\mathcal{C}$ if σ is a symbolic run of $\mathcal{B}$ and $\mathbf{C}$ is a constraint sequence over $\mathcal{C}$. If no verification constraints are needed, we write $h(\sigma)$ for $h(\sigma, \langle\emptyset, \dots, \emptyset\rangle)$.

Example 3.4. *For $\mathcal{B}_1$ in Fig. 1 with domain $\mathbb{Q}$ and $\alpha_0(x) = \alpha_0(y) = 0$, let σ_k be the (unique) symbolic run of k steps, e.g. $\sigma_2\colon 1 \xrightarrow{\mathsf{a}_1} 2 \xrightarrow{\mathsf{a}_2} 1$. We get the history constraints*

$$\begin{aligned} h(\sigma_0) &= x=0 \wedge y=0 && (\varphi_0)\\ h(\sigma_1) &= \exists x_0 y_0.\ x_0=0 \wedge y_0=0 \wedge x>y_0 \wedge y=y_0 \\ &\equiv x>0 \wedge y=0 && (\varphi_1)\\ h(\sigma_2) &= \exists x_1 y_1 x_0 y_0.\ x_0=0 \wedge y_0=0 \wedge x_1>y_0 \wedge \\ &\quad\ y_1=y_0 \wedge y>x_1 \wedge x=x_1 \\ &\equiv x>0 \wedge y>x && (\varphi_2) \end{aligned}$$

where x_0, y_0, x_1, y_1 are fresh variables. The equivalence steps are obtained by simplification and quantifier elimination. In a similar way, we get for $\sigma_3\colon 1 \xrightarrow{\mathsf{a}_1} 2 \xrightarrow{\mathsf{a}_2} 1 \xrightarrow{\mathsf{a}_1} 2$ the constraint $h(\sigma_3) \equiv (y>0) \wedge (x>y)$, and for $\sigma_4\colon 1 \xrightarrow{\mathsf{a}_1} 2 \xrightarrow{\mathsf{a}_2} 1 \xrightarrow{\mathsf{a}_1} 2 \xrightarrow{\mathsf{a}_2} 1$ we get $h(\sigma_4) \equiv (x>0) \wedge (y>x)$. The fact that $h(\sigma_2)$ and $h(\sigma_4)$ are equivalent reflects that σ_2 and σ_4 are equivalent in our finite-state abstraction.

Next we relate history constraints and assignments in runs.

Lemma 3.5. For any symbolic run σ of length n and $\mathbf{C} = \langle C_0, \dots, C_n\rangle$, $h(\sigma, \mathbf{C})$ is satisfied by assignment α iff there is a run $(b_0, \alpha_0) \xrightarrow{a_1} \cdots \xrightarrow{a_n} (b_n, \alpha_n)$ that is abstracted by σ such that $\alpha = \alpha_n$ and $\alpha_i \models C_i$ for all i, $0 \le i \le n$.

This shows that history constraints faithfully summarize accumulated constraints in symbolic runs, and their satisfying assignments correspond to the results of actual runs. Both directions are proven by straightforward induction proofs. For instance, Lem. 3.5 states that since the assignment $\alpha(x) = 9$, $\alpha(y) = 7$ satisfies $h(\sigma_3)$ in Ex. 3.4, there is a run abstracted by σ_3 ending with this assignment. This is true, e.g., for $(1, \left[\begin{smallmatrix}x=0\\y=0\end{smallmatrix}\right]) \xrightarrow{\mathsf{a}_1} (2, \left[\begin{smallmatrix}x=1\\y=0\end{smallmatrix}\right]) \xrightarrow{\mathsf{a}_2} (1, \left[\begin{smallmatrix}x=1\\y=7\end{smallmatrix}\right]) \xrightarrow{\mathsf{a}_1} (2, \left[\begin{smallmatrix}x=9\\y=7\end{smallmatrix}\right])$.

Our finite summary property will express that all (infinitely many) symbolic runs can be faithfully described by a *finite* set of states (b, φ) of a system state $b \in B$ and a formula φ that summarizes accumulated constraints. To that end, we first define a *history set* as a set of such states that contains a representative for every history constraint:

Definition 3.6. A *history set* Φ for $\mathcal{B}, \mathcal{C}$ is a set of pairs (b, φ) of $b \in B$ and a formula φ such that for every history constraint $h(\sigma, \mathbf{C})$ of $\mathcal{B}, \mathcal{C}$ where σ has final state b, there is a $(b, \varphi) \in \Phi$ with $h(\sigma, \mathbf{C}) \equiv \varphi$ and Φ contains no other pairs.

The next result turns out to be convenient in the sequel to characterize history sets:

Lemma 3.7. Φ is a history set iff (1) for all $C \subseteq \mathcal{C}$, there is some $(b_0, \varphi_0) \in \Phi$ such that $\varphi_0 \equiv \bigwedge(C_{\alpha_0} \cup C)$, and (2) for all $(b, \varphi) \in \Phi$, $b \xrightarrow{a} b'$, and $C \subseteq \mathcal{C}$, there is some $(b', \varphi') \in \Phi$ such that $\varphi' \equiv \mathit{update}(\varphi, a) \wedge C$.

We will show that some of the DDSA classes that we consider in this paper admit a *finite* history set—systems with monotonicity constraints and bounded lookback—and this feature is sufficient to decide our verification problem. For other systems (e.g., gap-constraint systems) it is not possible to find finite history sets. However, we will prove that the verification problem is still decidable if the more liberal property of *finite summary* holds. Basically, this property expresses that there exists a suitable equivalence relation $\sim$ such that the *quotient* of a history set with respect $\sim$ is finite. Here, $\sim$ is considered *suitable* if it is preserved under steps of $\mathcal{B}$ and implies equisatisfiability; for practicality we also require decidability. These requirements are made formal in the following definition.

Definition 3.8. A *summary* for $(\mathcal{B}, \mathcal{C})$ is a pair $(\Phi, \sim)$ of a history set Φ for $\mathcal{B}, \mathcal{C}$ and equivalence relation $\sim$ on Φ s.t.
(1) $\sim$ contains $\equiv$ on Φ and is decidable,
(2) for all $(b, \varphi), (b, \psi) \in \Phi$ such that $\varphi \sim \psi$, (a) φ and ψ are equisatisfiable, and (b) for all transitions $b \xrightarrow{a} b'$ and $C \subseteq \mathcal{C}$, $[\mathit{update}(\varphi, a) \wedge C] \sim [\mathit{update}(\psi, a) \wedge C]$.

We say that $(\mathcal{B}, \mathcal{C})$ *has finite summary* if it admits a summary $(\Phi, \sim)$ where $\sim$ has finitely many equivalence classes.

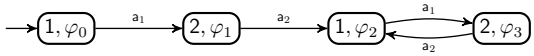
Here, $[\cdot]$ is a *representative* function for the given history set: if for a pair (b, ψ) there is some $(b, \varphi) \in \Phi$ with $\psi \equiv \varphi$, we can assume that $[\psi]$ is such a formula φ. A formula equivalent to $\mathit{update}(\varphi, a) \wedge C$ exists in Φ because of Lem. 3.7.

Intuitively, a DDSA has finite summary if it admits a finite-state abstraction that is expressive enough to account for all possible evolutions of $\mathcal{B}$ and properties in $\mathcal{C}$. We next show that $(\mathcal{B}, \mathcal{C})$ admits a finite summary if it has a finite history set, so one can pick $\equiv$ as equivalence relation.

Lemma 3.9. If $\mathcal{B}$ and $\mathcal{C}$ admit a finite history set Φ then $(\mathcal{B}, \mathcal{C})$ has finite summary $(\Phi, \equiv)$.

Proof. Def. 3.8 (1) follows from decidability of linear arithmetic and finiteness of Φ. For Def. 3.8 (2), we have that (a) $\varphi \equiv \psi$ implies equisatisfiability, and (b) we can write $\mathit{update}(\varphi, a) \wedge \bigwedge C = \exists \overline{U}.\varphi(\overline{U}) \wedge \chi$ and $\mathit{update}(\psi, a) \wedge \bigwedge C = \exists \overline{U}.\psi(\overline{U}) \wedge \chi$ for some χ, and these two formulas are clearly again equivalent as $\varphi \equiv \psi$. □

Example 3.10. *Continuing Ex. 3.4, it can be shown that $h(\sigma_{2i}) \equiv \varphi_2$ and $h(\sigma_{2i+1}) \equiv \varphi_3$ for all $i > 0$. Thus the set $\Phi = \{(1, \varphi_0), (2, \varphi_1), (1, \varphi_2), (2, \varphi_3)\}$ is a finite history set, and by Lem. 3.9 the tuple $(\Phi, \equiv)$ is a finite summary for $(\mathcal{B}_1, \emptyset)$. It can be visualized in a constraint graph, as done in (Felli, de Leoni, and Montali 2019):*



We conclude this section with another example where the history set is not finite but a finite summary can be found.

Example 3.11. *Consider the DDSA $\mathcal{B}_3$, and let σ_k be the symbolic run of k steps (there is only one). We have e.g. $h(\sigma_1) \equiv (x - y \ge 2) \wedge (y = 0)$, and $h(\sigma_2) \equiv (x \ge 2) \wedge (y \ge 3)$. In general, we obtain $h(\sigma_{2i}) \equiv (x \ge 3i - 1) \wedge$*

$(y \geq 3i)$ *and* $h(\sigma_{2i+1}) \equiv (x - y \geq 2) \wedge (y \geq 3i)$ *for all* $i \geq 1$. *Since* $h(\sigma_i) \not\equiv h(\sigma_j)$ *for* $i \neq j$, *the history set* $\Phi = \{h(\sigma_i) \mid i \geq 0\}$ *is not finite. However, in Sec. 5 (subsection on gap-order constraints) we will show that* $\mathcal{B}_3$ *admits a finite summary* $(\Phi, \sim_K)$, *where* $\sim_K$ *is the* cut-off equivalence relation *that considers formulas equivalent if they are syntactically equal after replacing all constants larger than some bound* K *by* K *itself.*

4 Checking the Existence of Witnesses

In order to express the requirements on a run of a DDSA $\mathcal{B}$ to satisfy an LTL_f formula ψ, we next define a nondeterministic automaton (NFA) $\mathcal{N}_\psi$. Then we combine $\mathcal{N}_\psi$ with $\mathcal{B}$ in a kind of product construction to check for the existence of witnesses for ψ.

To get the NFA, we perform a similar preprocessing step as in (de Leoni, Felli, and Montali 2020), and replace first all occurrences of subformulas $\langle a \rangle \psi'$ in ψ by $\langle \cdot \rangle (a \wedge \psi')$, adding a new proposition symbol for each action. For a run ρ of length n, we thus write $\rho, i \models a$ if $0 < i < n$ and $\rho, i-1 \models \langle a \rangle \top$. This modification allows us to consider fewer cases in the constructions and proofs below.

Technically, given $\psi \in \mathcal{L}_{\mathcal{BC}}$ we build the NFA $\mathcal{N}_\psi = (Q, \Sigma, \varrho, q_0, Q_F)$, where: *(i)* the set Q of states is a set of quoted formulas; *(ii)* $\Sigma = 2^S$ is the alphabet, where $S = B \cup \mathcal{A} \cup \mathcal{C}$; *(iii)* $\varrho \subseteq Q \times \Sigma \times Q$ is the transition relation; *(iv)* $q_0 \in Q$ is the initial state; *(v)* $Q_F \subseteq Q$ is the set of final states. Following (de Giacomo, de Masellis, and Montali 2014), we define ϱ using an auxiliary function δ and a new proposition λ that marks the last element of the trace. The input of δ is a (quoted) formula $\psi \in \mathcal{L}_{\mathcal{BC}} \cup \{\top, \bot\}$, and its output a set of tuples $(\text{“}\psi'\text{”}, \varsigma)$ where ψ' has the same type as ψ and $\varsigma \in 2^{S \cup \{\lambda, \neg\lambda\}}$. For two sets of such tuples R_1, R_2, and $\odot$ either $\wedge$ or $\vee$, let $R_1 \odot R_2 = \{(\text{“}\psi_1 \odot \psi_2\text{”}, \varsigma_1 \cup \varsigma_2) \mid (\text{“}\psi_1\text{”}, \varsigma_1) \in R_1, (\text{“}\psi_2\text{”}, \varsigma_2) \in R_2\}$, where we simplify $\psi_1 \odot \psi_2$ if possible. The function δ is as follows:

$\delta(\text{“}\top\text{”}) = \{(\text{“}\top\text{”}, \emptyset)\}$ and $\delta(\text{“}\bot\text{”}) = \{(\text{“}\bot\text{”}, \emptyset)\}$
$\delta(\text{“}p\text{”}) = \{(\text{“}\top\text{”}, \{p\}), (\text{“}\bot\text{”}, \emptyset)\}$ if $p \in \mathcal{C} \cup B \cup \mathcal{A}$
$\delta(\text{“}\psi_1 \vee \psi_2\text{”}) = \delta(\text{“}\psi_1\text{”}) \vee \delta(\text{“}\psi_2\text{”})$
$\delta(\text{“}\psi_1 \wedge \psi_2\text{”}) = \delta(\text{“}\psi_1\text{”}) \wedge \delta(\text{“}\psi_2\text{”})$
$\delta(\text{“}\langle \cdot \rangle \psi\text{”}) = \{(\text{“}\psi\text{”}, \{\neg\lambda\}), (\text{“}\bot\text{”}, \{\lambda\})\}$
$\delta(\text{“}\Diamond \psi\text{”}) = \delta(\text{“}\psi\text{”}) \vee \delta(\text{“}\langle \cdot \rangle \Diamond \psi\text{”})$
$\delta(\text{“}\Box \psi\text{”}) = \delta(\text{“}\psi\text{”}) \wedge (\delta(\text{“}\langle \cdot \rangle \Box \psi\text{”}) \vee \delta_\lambda)$
$\delta(\text{“}\psi_1 \mathbin{\mathsf{U}} \psi_2\text{”}) = \delta(\text{“}\psi_2\text{”}) \vee (\delta(\text{“}\psi_1\text{”}) \wedge \delta(\text{“}\langle \cdot \rangle (\psi_1 \mathbin{\mathsf{U}} \psi_2)\text{”}))$

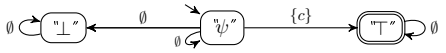
where δ_λ abbreviates $\{(\text{“}\top\text{”}, \{\lambda\}), (\text{“}\bot\text{”}, \{\neg\lambda\})\}$. While the symbol λ is needed for the construction, we can omit it from the NFA, and define $\mathcal{N}_\psi$ as follows:

Definition 4.1. Given a formula $\psi \in \mathcal{L}_{\mathcal{BC}}$, let the NFA $\mathcal{N}_\psi = (Q, \Sigma, \varrho, q_0, \{q_f, q_e\})$ be given by $q_0 = \text{“}\psi\text{”}$, $q_f = \text{“}\top\text{”}$ and q_e is an additional final state, and Q, ϱ are the smallest sets such that $q_0, q_f, q_e \in Q$ and whenever $q \in Q \setminus \{q_e\}$ and $(q', \varsigma) \in \delta(q)$ such that $\{\lambda, \neg\lambda\} \not\subseteq \varsigma$ then $q' \in Q$ and

(i) if $\lambda \notin \varsigma$ then $(q, \varsigma \setminus \{\lambda, \neg\lambda\}, q') \in \varrho$, and
(ii) if $\lambda \in \varsigma$ and $q' = \text{“}\top\text{”}$ then $(q, \varsigma \setminus \{\lambda, \neg\lambda\}, q_e) \in \varrho$.

This construction is similar to the one by (de Giacomo, de Masellis, and Montali 2014), but reflects that our verification language does not include negation. In fact it can be seen as a relaxation, in that $\delta(\text{“}p\text{”})$ contains $(\bot, \emptyset)$ rather than $(\bot, \{\neg p\})$, for any atom p. In this way, $\mathcal{N}_\psi$ cannot explicitly require atoms to be false; instead, the transition labels intuitively state *minimal requirements* for ψ to hold.

Example 4.2. *Let* $\psi = \Diamond c$ *for a constraint* $c = (y > 5)$. *By the definition of* δ, *we have* $\delta(\text{“}\Diamond c\text{”}) = \delta(\text{“}c\text{”}) \vee \delta(\text{“}\langle \cdot \rangle \Diamond c\text{”}) = \{(\text{“}\top\text{”}, \{c\}), (\text{“}\bot\text{”}, \emptyset)\} \vee \{(\text{“}\Diamond c\text{”}, \{\neg\lambda\}), (\text{“}\bot\text{”}, \{\lambda\})\} = \{(\text{“}\top\text{”}, \{c, \neg\lambda\}), (\text{“}\top\text{”}, \{c, \lambda\}), (\text{“}\Diamond c\text{”}, \{\neg\lambda\}), (\text{“}\bot\text{”}, \{\lambda\})\}$, *so the automaton* $\mathcal{N}_\psi$ *is as follows:*



Due to our relaxation, the self-loop on “ψ” *is labeled* $\emptyset$ *rather than* $\{\neg c\}$, *but nonetheless* $\mathcal{N}_\psi$ *works as expected: if* c *is true an accepting path exists, and if* c *is false no further possibilities arise.*

To express correctness of $\mathcal{N}_\psi$, we need some notions of consistency to express that a word w and a symbolic run are not contradictory with respect to actions and states. First, we call a symbol $\varsigma \in \Sigma$ *consistent* with transition $b \xrightarrow{a} b'$ if ς is disjoint from $\mathcal{A} \setminus \{a\}$ and $B \setminus \{b'\}$, namely if it contains no action symbol other than a nor state symbol other than b'. Let $constr(\varsigma) = \varsigma \cap \mathcal{C}$.

Definition 4.3. A word $w = \varsigma_0 \varsigma_1 \cdots \varsigma_n \in \Sigma^*$ is *consistent* with (a) a symbolic run $\sigma \colon b_0 \xrightarrow{a_1} b_2 \xrightarrow{a_1} \cdots \xrightarrow{a_n} b_n$ if ς_0 is disjoint from $B \setminus \{b_0\}$, and ς_i is consistent with $b_{i-1} \xrightarrow{a_i} b_i$ for $0 < i \leq n$. (b) a run ρ if it is consistent with the abstraction σ of ρ and α_i satisfies $constr(\varsigma_i)$, where $\rho_i = (b_i, \alpha_i)$.

These notions allow us to express correctness of $\mathcal{N}_\psi$:

Lemma 4.4. $\mathcal{N}_\psi$ accepts a word that is consistent with a run ρ iff $\rho \models \psi$.

Product construction. To check the existence of a witness for ψ in DDSA $\mathcal{B}$, we combine $\mathcal{N}_\psi$ with $\mathcal{B}$ to a cross-product automaton $\mathcal{N}_\mathcal{B}^\psi$, exploiting the notions from Sec. 3.

First, for technical reasons we add a dummy initial state b_0' to $\mathcal{B}$ and update its states to $B' = B \cup \{b_0'\}$ and its transitions to $T' = T \cup \{(b_0', a_0, b_0)\}$ for a fresh action a_0 with $guard(a_0) = \top$. We call the resulting DDSA $\mathcal{B}'$.

Definition 4.5. Let $\mathcal{B}' = \langle B', b_0', \mathcal{A}, T', F, V, \alpha_0, guard \rangle$ as above, $\mathcal{C}$ a constraint set, and $(\Phi, \sim)$ a summary for $(\mathcal{B}, \mathcal{C})$. For a formula $\psi \in \mathcal{L}_{\mathcal{BC}}$ and $\mathcal{N}_\psi$ as above, the *product automaton* $\mathcal{N}_\mathcal{B}^\psi = (P, \Sigma, R, p_0, P_F)$ is as follows:

- States in P are triples (b, q, φ) s.t. $b \in B'$, $q \in Q$, $\varphi \in \Phi$;
- The initial state is $p_0 = (b_0', q_0, \bigwedge C_{\alpha_0})$;
- There is a transition $(b, q, \varphi) \xrightarrow{a} (b', q', \varphi')$ in R iff $b \xrightarrow{a} b'$ in T', there is some $\varsigma \in \Sigma$ s.t. $q \xrightarrow{\varsigma} q'$ in $\mathcal{N}_\psi$, and
  - formula $\chi = update(\varphi, a) \wedge constr(\varsigma)$ is satisfiable, and $\varphi' \sim \chi$; in this way, χ captures all current constraints that are either inherited from $\mathcal{B}$ or stem from the transition of $\mathcal{N}_\psi$, given by $constr(\varsigma)$,
  - ς is consistent with $b \xrightarrow{a} b'$, and
  - $(b', q', \varphi') \in P_F$ iff $b' \in F$, $q' \in Q_F$.

Note that R is well-defined in the sense that for every such formula χ above, some φ' with $\varphi' \sim \chi$ and $(b', \varphi') \in \Phi$ exists, because Φ is a history set (cf. Lem. 3.7). Thus, if $(\Phi, \sim)$ is a *finite* summary, the construction in Def. 4.5 terminates. The next result states properties of the product construction, the induction proofs of both directions are straightforward.

Lemma 4.6. Let σ be a symbolic run of $\mathcal{B}$ and $w \in \Sigma^*$. There is a path π with $\sigma = \sigma(\pi)$ to a node (b, φ) in $\mathcal{N}_{\mathcal{B}}^{\psi}$ such that $\varphi \sim h(\sigma, w)$ iff w is accepted by $\mathcal{N}_{\psi}$, consistent with σ, and $h(\sigma, w)$ is satisfiable.

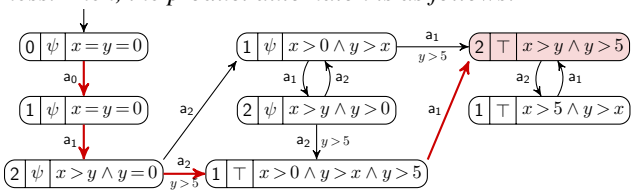
We next state our main result, where $h(\sigma, w)$ denotes $h(\sigma, \langle constr(\varsigma_0), \ldots, constr(\varsigma_n)\rangle)$ for word $w = \varsigma_0 \cdots \varsigma_n$.

Theorem 4.7. *Let $\psi \in \mathcal{L}_{\mathcal{BC}}$. The language of $\mathcal{N}_{\mathcal{B}}^{\psi}$ is non-empty iff there is a run of $\mathcal{B}$ that is a witness for ψ.*

Proof. ($\Longrightarrow$) Let π be a path to a final state p_f in $\mathcal{N}_{\mathcal{B}}^{\psi}$. By Lem. 4.6, there is an accepting transition sequence in $\mathcal{N}_{\psi}$ labeled $w = \varsigma_0 \ldots \varsigma_n$, and a symbolic run $\sigma(\pi)\colon b_0 \xrightarrow{a_1} b_1 \xrightarrow{*} b_n$ such that $h(\sigma(\pi), w)$ is satisfiable by some α, and ς_i is consistent with $b_{i-1} \xrightarrow{a_i} b_i$, for all i, so w is consistent with σ. By Lem. 3.5 (2), there is a run $\rho\colon (b_0, \alpha_0) \xrightarrow{a_1} \cdots \xrightarrow{a_n} (b_n, \alpha_n)$ abstracted by σ such that $\alpha = \alpha_n$ and $\alpha_i \models \bigwedge constr(\varsigma_i)$ for all i, $0 \leq i \leq n$. Thus w is consistent with ρ, and it follows from Lem. 4.4 that ρ is a witness. ($\Longleftarrow$) Let ρ be a witness for ψ, and σ its abstraction. By Lem. 4.4, $\mathcal{N}_{\psi}$ accepts a word w that is consistent with ρ. Consistency of w with ρ implies that w is also consistent with σ, and that $\alpha_i \models \bigwedge constr(\varsigma_i)$ for all i, $0 \leq i \leq n$. Thus α_n satisfies $h(\sigma, w)$ by Lem. 3.5 (1). By Lem. 4.6, the run of $\mathcal{N}_{\psi}$ labeled w and the symbolic run σ give rise to a path π in $\mathcal{N}_{\mathcal{B}}^{\psi}$ such that w is consistent with σ. As $\mathcal{N}_{\psi}$ accepts w, and the last state of σ is final, also π is accepting. □

We illustrate the product construction as well as the witness extraction on a simple example.

Example 4.8. *Consider the DDSA $\mathcal{B}_1$ from Fig. 1 and a formula $\psi = \Diamond(y > 5)$. We use the NFA $\mathcal{N}_{\psi}$ obtained in Ex. 4.2, removing the leftmost deadlock state for compactness. Then, the product automaton is as follows:*



Since $\mathcal{N}_{\mathcal{B}}^{\psi}$ has a final state (shown shaded), by Thm. 4.7 a witness for ψ exists. We follow the direction ($\Longrightarrow$) of the proof to construct a witness from the accepting path drawn in red: This path corresponds to the word $w = \langle \emptyset\ \emptyset\ \{y > 5\}\ \emptyset\rangle \in \Sigma^*$ accepted by $\mathcal{N}_{\psi}$, and the symbolic run $\sigma\colon 1 \xrightarrow{a_1} 2 \xrightarrow{a_2} 1 \xrightarrow{a_1} 2$ of $\mathcal{B}_1$. The formula φ in the final state is satisfiable and equivalent to $h(\sigma, w)$, and for any satisfying assignment of φ we obtain a witness run for ψ according to Lem. 3.5. For instance for $\alpha(x) = 9$, $\alpha(y) = 7$, one possible solution is $(1, \left[\begin{smallmatrix} x=0 \\ y=0 \end{smallmatrix}\right]) \xrightarrow{a_1} (2, \left[\begin{smallmatrix} x=1 \\ y=0 \end{smallmatrix}\right]) \xrightarrow{a_2} (1, \left[\begin{smallmatrix} x=1 \\ y=7 \end{smallmatrix}\right]) \xrightarrow{a_1} (2, \left[\begin{smallmatrix} x=9 \\ y=7 \end{smallmatrix}\right])$.

5 Conditions for Finite Summary

Thanks to Thm. 4.7 we can check for witnesses over DDSAs that admit a finite summary. Unfortunately, however:

Lemma 5.1. The finite summary property is undecidable.

Proof (sketch). Consider a DDSA $\mathcal{B}$ encoding a Minsky machine. $\mathcal{B}$ admits a finite summary iff the counter configurations are bounded, which is undecidable. □

In this section we identify relevant, sufficient conditions for finite summary. In short, these are the following (the equivalence relation used in Def. 3.8 is given in parentheses).

C1:	$\mathcal{B}$ is over *monotonicity constraints* ($\equiv$)	Thm. 5.2
C2:	$\mathcal{B}$ is over *gap-order constraints* ($\sim_K$)	Thm. 5.5
C3:	$\mathcal{B}$ is *feedback free* ($\equiv$)	Thm. 5.8
C4:	$\mathcal{B}$ has *bounded lookback* ($\equiv$)	Thm. 5.10

While **C1** and **C2** restrict the constraint language, **C3** and **C4** restrict the control flow (i.e., the shape of the DDSA). **C4** generalizes **C3** as well as the case where $\mathcal{B}$ is acyclic.

Before explaining these conditions, we point out that the DDSAs in Fig. 1 admit a finite summary. $\mathcal{B}_1$ can be seen as a monotonicity constraint (MC) system over $\mathbb{Q}$, or a gap-order constraint (GC) system over $\mathbb{Z}$, but **C3** and **C4** do not apply. $\mathcal{B}_2$ is feedback free and it can be shown to have also 2-bounded lookback. $\mathcal{B}_3$ is a GC system over $\mathbb{Z}$ but no other condition applies. $\mathcal{B}_4$ models a shopping process where two products with prices a and b are chosen by a customer, and the sum is computed in the variable s. $\mathcal{B}_4$ has 3-bounded lookback, but due to the self-reference of s in $s^w = s^r + b$, **C3** does not apply, and neither do **C1** or **C2**.

Monotonicity constraints (MCs) restrict Def. 2.1 as follows: MCs over variables V and domain D have the form $p \odot q$ where $p, q \in D \cup V$ and $\odot$ is one of $=, \neq, \leq, <, \geq$, or $>$. For MCs, we consider D to be $\mathbb{R}$ or $\mathbb{Q}$. An *MC-formula* is a boolean formula whose atoms are MCs. A DDSA is an MC-DDSA whose guards are conjunctions of MCs.

It is known that if φ is an MC-formula over constants $\mathcal{K}$ and variables $V \cup \{x\}$, then for a formula $\exists x.\, \varphi$, we can find a formula $\varphi' \equiv \exists x.\, \varphi$ such that φ' is an MC-formula over constants $\mathcal{K}$ and variables V, using a quantifier elimination procedure á la Fourier-Motzkin (Kroening and Strichman 2016, Sec. 5.4). In particular the set of constants $\mathcal{K}$ remains the same. This fact is crucial for the next result:

Theorem 5.2. *If $\mathcal{B}$ is an* MC*-DDSA and $\mathcal{C}$ a set of MCs then $(\mathcal{B}, \mathcal{C})$ admits a finite summary.*

Proof. Let $\mathcal{K}$ be the finite set of constants in $\mathcal{C}$, α_0, and guards of $\mathcal{B}$, and $\text{MC}_{\mathcal{K}}$ the set of quantifier-free formulas whose atoms are MCs over V, $\mathcal{K}$. In this way, $\text{MC}_{\mathcal{K}}$ is finite up to equivalence. We use Lem. 3.7 to show that $\Phi_{\text{MC}} := B \times \text{MC}_{\mathcal{K}}$ is a history set, which is also finite. First, for all $C \subseteq \mathcal{C}$, we have $\bigwedge(C_{\alpha_0} \cup C) \in \text{MC}_{\mathcal{K}}$. If $(b, \varphi) \in \Phi_{\text{MC}}$ and $b \xrightarrow{a} b'$ then $update(\varphi, a) \wedge C = \exists \overline{U}.\varphi(\overline{U}) \wedge \chi$ for some MC-formula χ over variables $U \cup V$ and constants $\mathcal{K}$. As remarked above, quantifier elimination can produce a formula φ' in $\text{MC}_{\mathcal{K}}$ such that $\varphi' \equiv \exists \overline{U}.\varphi(\overline{U}) \wedge \chi$. Thus $(\Phi_{\text{MC}}, \equiv)$ is a finite summary by Lem. 3.9. □

This result explains why the history set in Ex. 3.10 is finite. MCs over $\mathbb{Q}$ and $\mathbb{R}$ are closed under negation, so Thms. 4.7 and 5.2 imply decidability of LTL_f model checking. Note that the proof of Thm. 5.2 fails for domain $\mathbb{Z}$, as MCs over $\mathbb{Z}$ are *not* closed under quantifier elimination. Instead, they are covered by gap-order constraints, discussed next.

Gap-order constraints. Let X be a set of variables and $\mathcal{K} \subseteq \mathbb{Z}$ a finite set of constants such that $0 \in \mathcal{K}$. A *gap-order constraint* (GC) over X and $\mathcal{K}$ restricts Def. 2.1 to constraints of the form $x - y \geq k$ for $x, y \in X \cup \mathcal{K}$ and $k \in \mathbb{N}$. We call a *GC-formula* a quantifier-free formula whose atoms are GCs, and a GC-DDSA a DDSA where all guards are conjunctions of GCs. GC-DDSAs are known to generalize MC-DDSAs over $\mathbb{Z}$ (Bozzelli and Pinchinat 2014): for instance, $x = 3$ is expressible by $x - 3 \geq 0 \wedge 3 - x \geq 0$. However, it is known that relaxing the GC definition to allow also $x - y \leq k$ (or $k < 0$ in $x - y \geq k$) renders reachability in GC-DDSAs undecidable (Bozzelli and Pinchinat 2014).

In order to show that GC-DDSAs allow for a finite summary, we use the concept of a bounded approximation: Given the set of constants $\mathcal{K}$ and $K := max\{|c - c'| + 1 \mid c, c' \in \mathcal{K}\}$, the *$K$-bounded approximation* $\lfloor \varphi \rfloor_K$ of a GC-formula φ is obtained from φ by replacing all constraints $x - y \geq k$ where $k \geq K$ by $x - y \geq K$. The next lemma rephrases (Bozzelli and Pinchinat 2014, Props. 6 and 7):

Lemma 5.3. (1) A GC formula φ over variables X and constants $\mathcal{K}$ is satisfiable iff $\lfloor \varphi \rfloor_K$ is. (2) GC formulas φ over variables U and $\mathcal{K}$, and ψ over $U \cup V$ and $\mathcal{K}$ satisfy $\lfloor \exists \overline{U}.\varphi \wedge \psi \rfloor_K \equiv \lfloor \exists \overline{U}.\lfloor \varphi \rfloor_K \wedge \lfloor \psi \rfloor_K \rfloor_K$.

In the remainder of this section, let $\mathcal{B}$ be a GC-DDSA and $\mathcal{C}$ a constraint set, such that C_{α_0} and $\mathcal{C}$ consist of GCs over variables V and constants $\mathcal{K}$, and all guards of $\mathcal{B}$ are GCs over $V^r \cup V^w$ and $\mathcal{K}$, with the bound K as above. Below we use the fact that GC formulas are closed under quantifier elimination: if φ is a GC formula over variables $V \cup \{x\}$ and constants $\mathcal{K}$, we can find a GC formula φ' that is equivalent to $\exists x.\, \varphi$, quantifier-free, and over the variables V (though the constants in φ' need not be $\mathcal{K}$). For details, see (Revesz 1993), (Bozga, Gîrlea, and Iosif 2009, Thm. 2).

Let $\mathrm{GC}_{\mathcal{K}}$ be the set of quantifier-free formulas whose atoms are GCs over V and $\mathcal{K}$ and $\Phi_{\mathrm{GC}} = B \times \mathrm{GC}_{\mathcal{K}}$. As $\mathrm{GC}_{\mathcal{K}}$ may be infinite, we consider finite summary w.r.t. the equivalence relation $\sim_K$ defined as $\varphi \sim_K \psi$ iff $\lfloor \varphi \rfloor_K \equiv \lfloor \psi \rfloor_K$.

Example 5.4. *For $\mathcal{B}_3$ from Fig. 1 we have $\mathcal{K} = \{0, 2, 3\}$, so $K = 4$ (if $\mathcal{C} = \emptyset$, otherwise constraints in $\mathcal{C}$ need to be included). The history constraints $h(\sigma_4) \equiv (x \geq 5) \wedge (y \geq 6)$ and $h(\sigma_6) \equiv (x \geq 8) \wedge (y \geq 9)$ from Ex. 3.11 hence satisfy $h(\sigma_4) \sim_K h(\sigma_6)$ because their cutoff is equal, namely $\lfloor h(\sigma_4) \rfloor_K = \lfloor h(\sigma_6) \rfloor_K = (x \geq 4) \wedge (y \geq 4)$.*

Theorem 5.5. *$(\Phi_{\mathrm{GC}}, \sim_K)$ is a finite summary for $\mathcal{B}$ and $\mathcal{C}$.*

Proof (sketch). We use Lem. 3.7 to show that Φ_{GC} is a history set: first, for all $C \subseteq \mathcal{C}$, $\bigwedge(C_{\alpha_0} \cup C)$ is in $\mathrm{GC}_{\mathcal{K}}$. Next, for $(b, \varphi) \in \Phi_{\mathrm{GC}}$ and $b \xrightarrow{a} b'$, there is some GC-formula χ over $U \cup V$ and $\mathcal{K}$ such that $update(\varphi, a) \wedge C = \exists \overline{U}.\varphi(\overline{U}) \wedge \chi$. From quantifier elimination we get a GC-formula φ' over V and $\mathcal{K}$ with $\varphi' \equiv \exists \overline{U}.\varphi(\overline{U}) \wedge \chi$, so $\varphi' \in \mathrm{GC}_{\mathcal{K}}$. It remains to check Def. 3.8: Suppose $\varphi \sim_K \psi$, so $\lfloor \varphi \rfloor_K \equiv \lfloor \psi \rfloor_K$. Equisatisfiability of φ and ψ follows from Lem. 5.3 (1). We can write $update(\varphi, a) \wedge C = \exists \overline{U}.\varphi(\overline{U}) \wedge \chi$ and $update(\psi, a) \wedge C = \exists \overline{U}.\psi(\overline{U}) \wedge \chi$ for some GC-formula χ over $U \cup V$ and $\mathcal{K}$. Then $\lfloor \exists \overline{U}.\varphi(\overline{U}) \wedge \chi \rfloor_K \equiv \lfloor \exists \overline{U}.\psi(\overline{U}) \wedge \chi \rfloor_K$ follows using Lem. 5.3 (2). Finally, $(\Phi_{\mathrm{GC}}, \sim_K)$ has finitely many equivalence classes as the number of K-bounded GCs is finite. □

With Thm. 4.7 this shows that it is decidable whether a GC-DDSA admits a witness for an LTL_f formula, which is a novel result. It follows that model checking of a formula ψ is decidable if $\neg\psi$ is expressible in $\mathcal{L}_{\mathcal{BC}}$. However, the latter is not guaranteed for GC-DDSAs since GCs are not closed under negation. For instance, $\Box(x \geq y)$ can be checked since its negation is expressible as $\Diamond(y - x \geq 1)$; but $\Box(x - y \geq 2)$ cannot as its negation is not expressible in $\mathcal{L}_{\mathcal{BC}}$ with GCs.

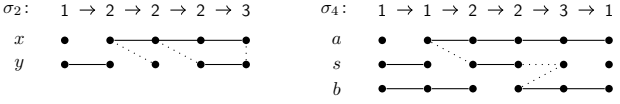
Feedback freedom (Damaggio, Deutsch, and Vianu 2012) achieves decidability by forbidding variable updates that depend on an unbounded history: it requires that for every dependency between two instances x_i, x_j of a variable x in a run, another "guard" variable y, keeps its value for the time span $[i, j]$ of the dependency. More precisely, let σ be a symbolic run of length n whose k-th action is a_k, and $\mathcal{C}$ a constraint set. The *computation graph* $G_{\sigma,\mathcal{C}}$ is the undirected graph with nodes $\mathcal{V} = \{v_i \mid v \in V \text{ and } 0 \leq i \leq n\}$ and an edge from x_i to y_j iff x_i and y_j occur in a common literal of $\Delta_{a_k}(\overline{V}_{k-1}, \overline{V}_k) \wedge C(\overline{V}_k)$, for some $C \subseteq \mathcal{C}$ and $i, j, k \leq n$. The subgraph of $G_{\sigma,\mathcal{C}}$ of all edges corresponding to equality literals $x_i = y_j$ for $x_i, y_j \in \mathcal{V}$ is denoted $E_{\sigma,\mathcal{C}}$.

Let $\equiv_E$ be the smallest equivalence relation on $\mathcal{V}$ containing $E_{\sigma,\psi}$, so that the equivalence classes of $\equiv_E$ are the connected components of $E_{\sigma,\psi}$. The equivalence class of $x_i \in \mathcal{V}$ is denoted $[\![x_i]\!]$, and the *span* of $[\![x_i]\!]$ is the set of affected instants, i.e., $span([\![x_i]\!]) = \{j \mid \exists v \in V \text{ with } v_j \in [\![x_i]\!]\}$.

Definition 5.6. For a DDSA $\mathcal{B}$ and constraint set $\mathcal{C}$, the pair $(\mathcal{B}, \mathcal{C})$ is *feedback-free* if for every symbolic run σ, every path in $G_{\sigma,\mathcal{C}}$ from x_i to x_j contains a node y such that $span([\![x_i]\!]) \cup span([\![x_j]\!]) \subseteq span([\![y]\!])$.

The next example illustrates this concept.

Example 5.7. *For runs σ_2 of $\mathcal{B}_2$ and σ_4 of $\mathcal{B}_4$ (cf. Fig. 1) and $\mathcal{C} = \{x > 5, s > 0\}$, we get the following graphs $G_{\sigma_i,\mathcal{C}}$:*



where edges in $E_{\sigma_i,\mathcal{C}}$ are drawn solid and others dotted. For σ_2, we have $span([\![y_2]\!]) \cup span([\![y_3]\!]) \subseteq span([\![x_1]\!])$. The graph is similar for other runs, so that $\mathcal{B}_2$ is feedback free; but $\mathcal{B}_4$ is not, as witnessed by the path from s_3 to s_4.

We postpone the proof of the next theorem, to show below that feedback freedom is a special case of *bounded lookback*.

Theorem 5.8. *Feedback freedom implies finite summary.*

Bounded lookback. We next show that a DDSA $\mathcal{B}$ has finite summary if, intuitively, at any point of a run of $\mathcal{B}$ the values of V depend on a bounded number of earlier steps. Throughout this section, we consider a DDSA $\mathcal{B}$ and constraint set $\mathcal{C}$. Moreover, we denote by $[\![G_{\sigma,\mathcal{C}}]\!]$ the graph obtained from $G_{\sigma,\mathcal{C}}$ by collapsing all edges in $E_{\sigma,\mathcal{C}}$.

Definition 5.9. The pair $(\mathcal{B}, \mathcal{C})$ has *bounded lookback* if there is some K such that for all symbolic runs σ of $\mathcal{B}$, all acyclic paths in $[\![G_{\sigma,\mathcal{C}}]\!]$ have length at most K.

For instance, after collapsing all solid (i.e., $E_{\sigma_4,\mathcal{C}}$) edges of $G_{\sigma_4,\mathcal{C}}$ in Ex. 5.7, the longest path has length 3. In fact, one can show that $(\mathcal{B}_4, \mathcal{C})$ has bounded lookback for $K = 3$.

Theorem 5.10. *Bounded lookback implies finite summary.*

Proof (sketch). Suppose $(\mathcal{B},\mathcal{C})$ has bounded lookback for some K. Let Ψ be the set of formulas with free variables V, quantifier depth at most $K \cdot |V|$, and vocabulary $\mathcal{C}, C_{\alpha_0}$, and guards of $\mathcal{B}$. As the quantifier depth is bounded and the set of atoms that can be taken from the vocabulary is finite, Ψ is finite up to equivalence. Induction on σ shows that $B \times \Psi$ is a history set (which implies the claim by Lem. 3.9): If σ is empty, $h(\sigma, \mathbf{C})$ is quantifier free and has all atoms in C_{α_0}, hence it is in Ψ. Otherwise, by induction hypothesis, $h(\sigma|_n, \mathbf{C}|_n)$ is equivalent to some $\varphi \in \Psi$, so $h(\sigma, \mathbf{C}) \equiv \exists \overline{U}.\varphi(\overline{U}) \wedge \chi =: \varphi'$ for some quantifier free χ. Let $[\![\varphi']\!]$ be the formula that is obtained from φ' by eliminating all equality literals $x = y$, and substituting all variables in an equivalence class by a representative. As $[\![\varphi']\!]$ encodes $[\![G_{\sigma,\mathcal{C}}]\!]$ and $(\mathcal{B},\mathcal{C})$ has K-bounded lookback, $[\![\varphi']\!]$ is equivalent to a formula that has quantifier depth at most $K \cdot |V|$. Hence, Ψ must contain a formula equivalent to $[\![\varphi']\!]$. □

Note that all acyclic DDSAs have bounded lookback, for K the number of states. For feedback-free systems, (Damaggio, Deutsch, and Vianu 2012, Lem. 5.4) shows that $[\![G_{\sigma,\mathcal{C}}]\!]$ is a tree of depth at most $|V|$, so that Thm. 5.8 follows from:

Lemma 5.11. If $(\mathcal{B},\mathcal{C})$ is feedback-free then it has $2|V|$-bounded lookback.

For a fixed K, bounded lookback is decidable in a similar way as feedback freedom (Damaggio, Deutsch, and Vianu 2012, Sec. 4.4), by enumerating all possible variable dependencies in symbolic runs of $\mathcal{B}$. While (Damaggio, Deutsch, and Vianu 2012) discovered that LTL model checking is decidable for feedback-free systems, the respective result—implied by Thms. 4.7 and 5.10—for the larger class of DDSAs with bounded lookback is new.

6 Modularity

In this section we show that a DDSA admits a finite summary if it is suitably decomposable into smaller systems that enjoy this property. As finite summary of the subsystems may be due to different criteria **C1**–**C4**, modularity results substantially extend applicability of our approach. As an arbitrary splitting of a DDSA $\mathcal{B}$ into subsystems with finite summary does not imply that $\mathcal{B}$ inherits the property, we consider two specific ways of decomposition for a DDSA $\mathcal{B} = \langle B, b_0, \mathcal{A}, T, F, V, \alpha_0, \mathit{guard}\rangle$:

Definition 6.1. Suppose $B = B_1 \cup B_2$, $B_1 \cap B_2 = \{b\}$, and T contains neither edges from B_2 to B_1, nor from $B_1 \setminus \{b\}$ to $B_2 \setminus \{b\}$; Let T_1 and T_2 be the projections of T to $B_1 \times \mathcal{A}$ and $B_2 \times \mathcal{A}$, respectively. Then $\mathcal{B}$ is *sequentially decomposable* into the DDSAs $\mathcal{B}_1 = \langle B_1, b_0, \mathcal{A}, T_1, \{b\}, V, \alpha_0, \mathit{guard}\rangle$ and $\mathcal{B}_2 = \langle B_2, b, \mathcal{A}, T_2, F, V \cup U, \alpha_U, \mathit{guard}\rangle$, where α_U is the assignment such that $\alpha_U(\overline{V}) = \overline{U}$, for some set of variables U such that $|U| = |V|$ and U is disjoint from V.

Definition 6.2. Let $V = V_1 \uplus V_2$ such that all constraints in $\{\mathit{guard}(a) \mid a \in \mathcal{A}\} \cup \mathcal{C}$ are over V_1 or V_2.
Then $(\mathcal{B},\mathcal{C})$ is *variable-decomposable* into $(\mathcal{B}_1, \mathcal{C}|_{V_1})$ and $(\mathcal{B}_2, \mathcal{C}|_{V_2})$ where $\mathcal{B}_i = \langle B, b_0, \mathcal{A}, T, F, V_i, \alpha_0|_{V_i}, \mathit{guard}_i\rangle$, and $\mathit{guard}_i(a)$ is $\mathit{guard}(a)$ if it is over V_i, and $\top$ otherwise.

Both ways of decomposition give rise to a modularity result:

Theorem 6.3. *Let $\mathcal{B}$ be a DDSA admitting a decomposition into $\mathcal{B}_1$ and $\mathcal{B}_2$ that is either (a) sequential and so that $(\mathcal{B}_i,\mathcal{C})$ has finite summary $(\Phi_i, \equiv)$, or (b) variable and so that $(\mathcal{B}_i, \mathcal{C}|_{V_i})$ has finite summary $(\Phi_i, \sim_i)$, for some $\mathcal{C}$ and both $i \in \{1,2\}$. Then $(\mathcal{B},\mathcal{C})$ admits a finite summary.*

Proof (sketch). (a) For $\Phi = \Phi_1 \cup \{\exists \overline{U}.\, \varphi_1(\overline{U}) \wedge \varphi_2 \mid \varphi_1 \in \Phi_1 \text{ and } \varphi_2 \in \Phi_2\}$, the pair $(\Phi, \equiv)$ is a finite summary. (b) We show that $\Phi = \{\varphi_1 \wedge \varphi_2 \mid \varphi_1 \in \Phi_1 \text{ and } \varphi_2 \in \Phi_2\}$ with $\sim_1$ and $\sim_2$ combined is a finite summary. □

We conclude this section by showing that Thm. 6.3 allows us to handle our motivating example Ex. 1.1. Note that decidability does not follow by any of the criteria **C1**–**C4** alone.

Example 6.4. *The system $\mathcal{B}$ of Ex. 1.1 is variable decomposable into a red GC-DDSA $\mathcal{B}_1$ over $\{b, d\}$, and a blue/green system $\mathcal{B}_2$ over $\{o, s, t\}$. $\mathcal{B}_2$ can in turn be sequentially split into a blue MC-DDSA $\mathcal{B}_{21}$, and the green single-step system $\mathcal{B}_{22}$ having 1-bounded lookback. By Thm. 6.3, $\mathcal{B}$ has finite summary because so do $\mathcal{B}_1$, $\mathcal{B}_{21}$, and $\mathcal{B}_{22}$. Then Thm. 4.7 applies to check that there is no witness for $\Diamond(\mathsf{sold} \wedge d > 0 \wedge o \leq t)$ (so property ψ in Ex. 1.1 holds). On the other hand, we can obtain a witness for $\Diamond(b = 1 \wedge o > t \wedge \Diamond(\mathsf{sold} \wedge b \neq 1))$, showing that a bid above the threshold t need not win.*

7 Conclusion

Implementation. We implemented our approach in the prototype `ada` (arithmetic DDS analyzer), available via a web interface,[2] where also source code and examples can be found. `ada` takes a DDSA $\mathcal{B}$ and an LTL_f formula ψ and checks whether $\mathcal{B}$ and the constraints $\mathcal{C}$ occurring in ψ admit a finite summary according to **C1**–**C4**, or if $\mathcal{B}$ is suitably decomposable (cf. Sec. 6). If finite summary is detected, `ada` visualizes the constraint graph, the NFA $\mathcal{N}_\psi$, and the automaton $\mathcal{N}_\mathcal{B}^\psi$, then extracts a witness for ψ if it exists (cf. Thm. 4.7). In the extended version (Felli, Montali, and Winkler 2021) we show results for relevant examples, including Ex. 1.1 and processes converted from Petri nets with data (Mannhardt et al. 2016). `ada` is written in Python and uses the Z3 SMT solver (de Moura and Bjørner 2008).

Future work. We see various possibilities for extensions: we expect that finite summary covers further known decidable cases, e.g. DDSAs with integer periodicity constraints, which are closed under quantifier elimination (Demri 2006); and flat systems with Presburger-definable loop effects (Barrett, Demri, and Deters 2013). For these as well as the criteria **C1**–**C4**, it would be also of great interest to investigate the complexity bounds implied by our method. Further decomposition results would be useful, too, e.g. limited forms of parallel execution. Next, we want to study whether our techniques apply to branching-time properties, as well as transition systems with full-fledged relational databases in the vein of (Deutsch et al. 2018; Calvanese et al. 2020). The area of BPM provides a rich source of practical applications: processes are often specified as Petri nets with data, which can be expressed as DDSAs (Mannhardt et al. 2016).

[2] http://tiny.one/adatool

References

Alur, R.; and Dill, D. 1994. A theory of timed automata. *Theor. Comput. Sci.*, 126(2): 183–235.

Barrett, C.; Demri, S.; and Deters, M. 2013. Witness Runs for Counter Machines. In *Proc. 10th FroCoS*, volume 8152 of *LNCS*, 120–150.

Bozga, M.; Gîrlea, C.; and Iosif, R. 2009. Iterating Octagons. In *Proc. 15th TACAS*, volume 5505 of *LNCS*, 337–351.

Bozzelli, L.; and Pinchinat, S. 2014. Verification of gap-order constraint abstractions of counter systems. *Theor. Comput. Sci.*, 523: 1–36.

Calvanese, D.; de Giacomo, G.; and Montali, M. 2013. Foundations of data-aware process analysis: a database theory perspective. In *Proc. 32nd PODS*, 1–12.

Calvanese, D.; de Giacomo, G.; Montali, M.; and Patrizi, F. 2018. First-order μ-calculus over generic transition systems and applications to the situation calculus. *Inf. Comput.*, 259(3): 328–347.

Calvanese, D.; Ghilardi, S.; Gianola, A.; Montali, M.; and Rivkin, A. 2020. SMT-based verification of data-aware processes: a model-theoretic approach. *Math. Struct. Comput. Sci.*, 30(3): 271–313.

Clarke, E. M.; Kroening, D.; Sharygina, N.; and Yorav, K. 2004. Predicate Abstraction of ANSI-C Programs Using SAT. *Formal Methods Syst. Des.*, 25(2-3): 105–127.

Colón, M.; and Uribe, T. E. 1998. Generating Finite-State Abstractions of Reactive Systems Using Decision Procedures. In *Proc. 10th CAV*, volume 1427 of *LNCS*, 293–304.

Damaggio, E.; Deutsch, A.; and Vianu, V. 2012. Artifact systems with data dependencies and arithmetic. *ACM Trans. Database Syst.*, 37(3): 22:1–22:36.

de Giacomo, G.; de Masellis, R.; and Montali, M. 2014. Reasoning on LTL on Finite Traces: Insensitivity to Infiniteness. In *Proc. 28th AAAI*, 1027–1033.

de Giacomo, G.; and Vardi, M. Y. 2013. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *Proc. 23rd IJCAI*, 854–860.

de Leoni, M.; Felli, P.; and Montali, M. 2020. Strategy Synthesis for Data-Aware Dynamic Systems with Multiple Actors. In *Proc. 17th KR*, 315–325.

de Moura, L.; and Bjørner, N. 2008. Z3: An Efficient SMT Solver. In *Proc. 14th TACAS*, volume 4963 of *LNCS*, 337–340.

Demri, S. 2006. LTL over integer periodicity constraints. *Theor. Comput. Sci.*, 360(1-3): 96–123.

Demri, S.; and D'Souza, D. 2007. An automata-theoretic approach to constraint LTL. *Inform. Comput.*, 205(3): 380–415.

Deutsch, A.; Hull, R.; Li, Y.; and Vianu, V. 2018. Automatic verification of database-centric systems. *ACM SIGLOG News*, 5(2): 37–56.

Felli, P.; de Leoni, M.; and Montali, M. 2019. Soundness Verification of Decision-Aware Process Models with Variable-to-Variable Conditions. In *Proc. 19th ACSD*, 82–91. IEEE.

Felli, P.; Montali, M.; and Winkler, S. 2021. Linear-Time Verification of Data-Aware Dynamic Systems with Arithmetic (extended version). Available from https://tinyurl.com/adasoundness/downloads/arithdpnLTLf.pdf.

Ghilardi, S.; Nicolini, E.; Ranise, S.; and Zucchelli, D. 2007. Combination Methods for Satisfiability and Model-Checking of Infinite-State Systems. In *Proc. 21st CADE*, volume 4603 of *LNCS*, 362–378. Springer.

Kroening, D.; and Strichman, O. 2016. *Decision Procedures – An Algorithmic Point of View, Second Edition*. Springer.

Mannhardt, F.; de Leoni, M.; Reijers, H.; and van der Aalst, W. 2016. Balanced multi-perspective checking of process conformance. *Computing*, 98(4): 407–437.

Mayr, R.; and Totzke, P. 2016. Branching-Time Model Checking Gap-Order Constraint Systems. *Fundam. Informaticae*, 143(3-4): 339–353.

Presburger, M. 1929. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du I congres de Mathematiciens des Pays Slaves*, 92–101.

Reichert, M. 2012. Process and Data: Two Sides of the Same Coin? In *OTM 2012*, volume 7565 of *LNCS*, 2–19.

Revesz, P. Z. 1993. A Closed-Form Evaluation for Datalog Queries with Integer (Gap)-Order Constraints. *Theor. Comput. Sci.*, 116(1): 117–149.