



PDF Download
3776661.pdf
10 March 2026
Total Citations: 0
Total Downloads: 58

Latest updates: <https://dl.acm.org/doi/10.1145/3776661>

RESEARCH-ARTICLE

General Decidability Results for Systems with Continuous Counters

A. R. BALASUBRAMANIAN, Max Planck Institute for Software Systems, Saarbrücken, Saarland, Germany

MATTHEW HAGUE, Royal Holloway, University of London, Egham, Surrey, U.K.

RUPAK MAJUMDAR, Max Planck Institute for Software Systems, Saarbrücken, Saarland, Germany

RAMANATHAN S. THINNIYAM, Uppsala University, Uppsala, Uppsala, Sweden

GEORG ZETZSCHE, Max Planck Institute for Software Systems, Saarbrücken, Saarland, Germany

Open Access Support provided by:

Max Planck Institute for Software Systems

Uppsala University

Royal Holloway, University of London

Published: 08 January 2026
Accepted: 06 November 2025
Received: 10 July 2025

[Citation in BibTeX format](#)

General Decidability Results for Systems with Continuous Counters

A. R. BALASUBRAMANIAN, Max Planck Institute for Software Systems (MPI-SWS), Germany

MATTHEW HAGUE, Royal Holloway, University of London, United Kingdom

RUPAK MAJUMDAR, Max Planck Institute for Software Systems (MPI-SWS), Germany

RAMANATHAN S. THINNIYAM, Uppsala University, Sweden

GEORG ZETZSCHE, Max Planck Institute for Software Systems (MPI-SWS), Germany

Counters that hold natural numbers are ubiquitous in modeling and verifying software systems; for example, they model dynamic creation and use of resources in concurrent programs. Unfortunately, such discrete counters often lead to extremely high complexity. Continuous counters are an efficient over-approximation of discrete counters. They are obtained by relaxing the original counters to hold values over the non-negative *rational* numbers.

This work shows that continuous counters are extraordinarily well-behaved in terms of decidability. Our main result is that, despite continuous counters being infinite-state, the language of sequences of counter instructions that can arrive in a given target configuration, is regular. Moreover, a finite automaton for this language can be computed effectively. This implies that a wide variety of transition systems can be equipped with continuous counters, while maintaining decidability of reachability properties. Examples include higher-order recursion schemes, well-structured transition systems, and decidable extensions of discrete counter systems.

We also prove a non-elementary lower bound for the size of the resulting finite automaton.

CCS Concepts: • **Theory of computation** → **Models of computation**.

Additional Key Words and Phrases: Continuous Counters, Vector Addition System, Regular Language, Decidability

ACM Reference Format:

A. R. Balasubramanian, Matthew Hague, Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetsche. 2026. General Decidability Results for Systems with Continuous Counters. *Proc. ACM Program. Lang.* 10, POPL, Article 19 (January 2026), 28 pages. <https://doi.org/10.1145/3776661>

1 Introduction

Counters are ubiquitous in modeling and verifying software systems. They model dynamic creation and use of resources in concurrent systems. For example, a fundamental type of infinite-state system with counters is a *vector addition system* (VAS). These consist of finitely many counters that can be incremented and decremented in a coordinated way. VAS, and related models such as Petri nets, have a long record of modeling resources in concurrent systems, including manufacturing systems and discrete control [37], business workflows [57], hardware design [58], and biological

Authors' Contact Information: A. R. Balasubramanian, Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany, bayikudi@mpi-sws.org; Matthew Hague, Royal Holloway, University of London, Egham, United Kingdom, Matthew.Hague@rhul.ac.uk; Rupak Majumdar, Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany, rupak@mpi-sws.org; Ramanathan S. Thinniyam, Uppsala University, Uppsala, Sweden, ramanathan.s.thinniyam@it.uu.se; Georg Zetsche, Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany, georg@mpi-sws.org.



This work is licensed under a Creative Commons Attribution 4.0 International License.

© 2026 Copyright held by the owner/author(s).

ACM 2475-1421/2026/1-ART19

<https://doi.org/10.1145/3776661>

pathways [42]. In concurrent programming, they can model and analyze dynamic thread creation [6, 11, 26, 28], numerical data types [33], network broadcasts [21], and population protocols [22].

However, analyzing systems with discrete counters encounters two crucial challenges. First, when the size of the counters is very large, e.g., modeling packets in a network or individuals in a large population, individual discrete interactions are too fine-grained. In such situations, one would like a “fluid limit” that captures the limit behaviors as counters go to infinity. Second, the complexity of reachability for the simplest model, vector addition systems (VAS), which have natural-number counters, is already Ackermann-complete [16, 46, 48]. As soon as we combine such counters with more expressive infinite-state data structures, algorithmic tools become extremely rare. Such combinations are natural in concurrency theory: for example, combining pushdown systems with counters (pushdown VAS) lets us model systems with both recursion and concurrency, but decidability of reachability remained a longstanding open problem until a very recent breakthrough [30]. Reachability becomes undecidable very quickly when we extend systems (that themselves have decidable reachability) even with very weak counters. For example, lossy channel systems (LCS for short; which model unreliable communication channels) and higher-order pushdown automata (HOPA; a standard model for programs with higher-order recursion) are well-known to have decidable reachability, but equipping them even with very weak kinds of counters makes reachability undecidable ([59, Prop. 7] and [41, Thm. 4] for HOPA, and [2, p. 7] for LCS).

Continuous counters [19] provide a fluid relaxation of counter machines. In a continuous counter machine, any transition that adds a vector $\mathbf{x} \in \mathbb{Z}^d$ to the counters, can be executed with any *non-zero fraction*: we are allowed to pick a rational $0 < \alpha \leq 1$ and add $\alpha \cdot \mathbf{x}$ to the counters. As a result, the counters may thus assume values in \mathbb{Q}_+ , the non-negative rationals. This means that the possible behaviors of the relaxation are an overapproximation of the original machine. This idea of relaxing discrete counters to continuous ones goes back to David and Alla, who introduced continuous semantics in the context of studying fluid limits of Petri nets already in the 1980s [18].

Just as linear programming is a relaxation of integer linear programming with better algorithmic complexity, continuous relaxations of counter machines have much better complexity in many cases. For example, reachability in Petri nets with continuous semantics can be decided in polynomial time [25]; reachability in finite systems with continuous counters is NP-complete [13]; reachability in pushdown continuous counter systems is NEXP-complete [10], and there is an almost complete description of the complexity of systems with affine continuous counters [7].

The continuous relaxation has found applications in many fields, such as modeling client-server systems [49], chemical reaction networks [39] and biological networks [31]. Also, continuous Petri nets are well-studied in biochemical settings, with a variety of semantics and interpretations [35, 36].

In certain cases, continuous counters not only overapproximate the original system, but even allow an *exact representation of system behavior*. For example, we consider leaderless rendez-vous protocols, which are a model of distributed systems composed of arbitrarily many identical finite-state agents interacting in pairs. These interactions can be modeled using a (discrete) counter system where each of the counters is used to keep track of the number of agents in each state. It turns out that important questions such as reachability and coverability for such protocols can be solved *exactly* by using the continuous relaxation [8]. Furthermore, even more involved analysis problems such as the cut-off problem can be solved by combining the continuous relaxation along with the so-called integer relaxation [8]. All of these algorithms run in polynomial time, which highlights the importance of the continuous relaxation as an important tool in algorithmic verification.

Finally, continuous relaxations have proven to be useful for the problem of coverability in Petri nets. In [23], continuous relaxations were used in an SMT approach to solve (i.e. prove safety of) 94 of 115 instances of Petri net benchmarks. Maintaining integer counters only helped prove 2

additional instances (with higher runtimes). In contrast, coverability tools only scaled to about 60 instances. The continuous relaxation has also been used as an optimization in coverability algorithms for VAS [12]. Thus, better analysis tools for continuous counters will yield better algorithms for discrete counter systems in practice.

Given these positive results, it is natural to ask how far one can push the decidability frontier for continuous counters: Is reachability decidable if we extend well-known infinite-state models with continuous counters? This is an important concern in modeling and analysis, because we often take fluid limits for certain components while maintaining the discrete structure for other components.

Our main results show that indeed, continuous counters have extraordinarily good decidability properties. In a nutshell, decidable models remain decidable when continuous counters are added, as long as the models are closed under taking products with finite-state systems.

1.1 Outline of Main Results

To make our results precise, we need some terminology. In a continuous counter machine, a transition that adds a vector $\mathbf{x} \in \mathbb{Z}^d$ can be scaled by any rational $0 < \alpha \leq 1$, that is $\alpha \cdot \mathbf{x}$ is added. We refer to this as *fractional firing*. A system that has access to d continuous counters has transitions which are labeled with vectors from a finite set $\Sigma \subset \mathbb{Z}^d$, which represent the set of counter instructions. If a sequence $w \in \Sigma^*$ can lead from counter configuration $\mathbf{x} \in \mathbb{Q}_+^d$ to $\mathbf{y} \in \mathbb{Q}_+^d$ via fractional firing, then we write $\mathbf{x} \xrightarrow{w} \mathbf{y}$.

Our main result is that for any dimension d , any given finite $\Sigma \subset \mathbb{Z}^d$ and any $\mathbf{x}, \mathbf{y} \in \mathbb{Q}_+^d$, the set of all words $w \in \Sigma^*$ with $\mathbf{x} \xrightarrow{w} \mathbf{y}$ is an *effectively regular language*.

This is very surprising: The set of configurations attainable between \mathbf{x} and \mathbf{y} is infinite, and involves arbitrarily large numbers. Thus, regularity strongly contrasts with almost all kinds of infinite-state systems studied so far: In almost all known types of infinite-state systems, the set of transition sequences between two configurations is in general non-regular. This holds for any type of discrete counters that can be incremented and decremented (such as classical VAS, but also the integer, bidirected, and reversal-bounded variants of VAS, etc.)¹, but also for pushdown automata² (higher-order or not), lossy-channel systems. To our knowledge, the only exception to this is timed automata [4] (see, e.g. [5] for the finite-word setting)³.

Our construction yields an *Ackermannian upper bound* on the size of an NFA recognising the language. We also prove a *non-elementary lower bound* on the size of these NFAs. (This already indicates that our regularity proof requires substantially different insights than the exponential construction for timed automata [4, 5]; and in fact, the proof uses an entirely different approach.)

1.2 Implications of the Main Result

Our main result implies a very general decidability result for infinite-state systems extended with continuous counters: For any class of infinite-state systems that is closed under taking products with finite-state systems, reachability is decidable if and only if it is decidable when equipped with additional continuous counters. Thus, it follows that, e.g., lossy channel systems and higher-order

¹This is because applying transitions in discrete counter systems is forward- and backward-deterministic: In a forward- and backward-deterministic Σ -labeled transition system where between two configurations c_1 and c_2 , infinitely many configurations can be visited, the set of $w \in \Sigma^*$ leading from c_1 to c_2 must be non-regular (a simple consequence of the Myhill-Nerode theorem). Similarly arguments apply to almost any type of infinite-state system.

²This is not to be confused with the fact that in a pushdown automaton, the set of reachable *stack contents* is regular [14]: The set of transition sequences, e.g. from empty stack to empty stack can be the Dyck language, which is not regular.

³However, note that a substantial difference between timed automata and CVAS is that once a clock in a timed automaton surpasses all constants occurring in guards, its value becomes irrelevant. In a CVAS, when we reach any value x we need at least x decrement steps to come back to zero. This makes regularity in CVAS rather unexpected.

pushdown automata (even with the collapse operation [34]), when extended with continuous counters, still have decidable reachability.

Our result also applies to the very general class of well-structured transition systems (WSTS) [1, 24]. Here, the state space is ordered by a well-quasi ordering (WQO) \leq that enjoys some compatibility with the transition relation. Under mild assumptions, many questions are decidable for WSTS [1, 24]. An example is the *coverability problem*: Given a configuration \mathbf{c} of a WSTS, can we reach a configuration \mathbf{c}' with $\mathbf{c} \leq \mathbf{c}'$? In this case, \mathbf{c} is said to be *coverable*. Our result implies that given such a WSTS whose transitions carry labels from some finite $\Sigma \subset \mathbb{Z}^d$, one can even decide whether \mathbf{c} is coverable by a run that, working on d continuous counters, reaches a given $\mathbf{y} \in \mathbb{Q}_+^d$. This is because, by our result, the d continuous counters can be simulated by finitely many states, and WSTS are well-known to be closed under taking products with finite-state systems [24].

Our general decidability result starkly contrasts with discrete counters. There, many infinite-state models (e.g. LCS or HOPA) have undecidable reachability when extended with weak kinds of counters (see references above). Until very recently, the only known situation where combining (discrete) counters with another type of data structure retains decidability is pushdown systems with reversal-bounded counters [33]. The recent breakthrough on PVAAS reachability generalizes the result to all counters [30].

Our result also makes the vast collection of algorithmic tools for regular languages (see, e.g. [3, 20, 29, 54, 56]) available to the languages of continuous counters. This includes not only combinatorial methods working with finite automata [20] and regular expressions [29, 54], but also algebraic tools working with finite semigroups [56], and topological approaches using topological semigroups [3].

For space reasons, detailed proofs can be found in the full version of this paper [9].

2 Preliminaries and Main Results

In this section, we recall the basic definitions of our central object of study, namely, continuous vector addition system (CVAS for short). Then we formally state our main results.

2.1 Preliminaries

Throughout the paper, we use \mathbb{N} , \mathbb{Z} , \mathbb{Q} , and \mathbb{Q}_+ to denote the natural numbers, integers, rational numbers, and non-negative rational numbers, respectively.

A d -dimensional *Continuous Vector Addition System* (d -CVAS or simply CVAS) is a finite set $\Sigma \subset \mathbb{Z}^d$ of vectors called *transitions*. The operational semantics of a CVAS is given by means of its *configurations*, which we define below.

A *configuration* of Σ is a tuple $\mathbf{x} \in \mathbb{Q}_+^d$, which intuitively denotes the current value of each of the d continuous counters. Let $t \in \Sigma$ be a transition and let $\alpha \in (0, 1]$ be a rational number. A *step* from a configuration \mathbf{x} to \mathbf{y} by means of the pair (α, t) (denoted $\mathbf{x} \xrightarrow{\alpha t} \mathbf{y}$) is possible iff $\mathbf{y} = \mathbf{x} + \alpha t$. In this case, α is called the *firing fraction* of this step. Since each configuration must be in \mathbb{Q}_+^d , a step is only possible if no component of the configuration becomes negative. If $\mathbf{x} \xrightarrow{\alpha t} \mathbf{y}$ for some $\alpha \in (0, 1]$, then we also write $\mathbf{x} \xrightarrow{t} \mathbf{y}$.

A *run* of Σ is a finite sequence of steps $\mathbf{x}_0 \xrightarrow{\alpha_1 t_1} \mathbf{x}_1 \xrightarrow{\alpha_2 t_2} \dots \xrightarrow{\alpha_n t_n} \mathbf{x}_n$. Here, the sequence $\alpha_1 t_1, \dots, \alpha_n t_n$ is called a *firing sequence*. If such a firing sequence exists between \mathbf{x}_0 and \mathbf{x}_n , we say that \mathbf{x}_n is *reachable* from \mathbf{x}_0 (written $\mathbf{x}_0 \xrightarrow{\alpha_1 t_1, \alpha_2 t_2, \dots, \alpha_n t_n} \mathbf{x}_n$ or $\mathbf{x}_0 \xrightarrow{*} \mathbf{x}_n$).

For a word $w \in \Sigma^*$ and configurations $\mathbf{x}, \mathbf{y} \in \mathbb{Q}_+^d$, we write $\mathbf{x} \xrightarrow{w} \mathbf{y}$ if $w = t_1 \dots t_n$ and there exist rational fractions $\alpha_1, \dots, \alpha_n \in (0, 1]$ such that $\mathbf{x} \xrightarrow{\alpha_1 t_1, \dots, \alpha_n t_n} \mathbf{y}$. We also write $\mathbf{x} \xrightarrow{\alpha w} \mathbf{y}$ where $\alpha = \alpha_1, \alpha_2, \dots, \alpha_n$ is the sequence of firing fractions. It will always be clear from the context whether

α is a single firing fraction or a sequence of firing fractions. In the sequel, we often view Σ as an alphabet and write *letters* to mean transitions.

Finally, for a CVAS $\Sigma \subset \mathbb{Z}^d$ and configurations $\mathbf{x}, \mathbf{y} \in \mathbb{Q}_+^d$, we define its transition language as

$$L_\Sigma^{\mathbf{x}, \mathbf{y}} := \{w \in \Sigma^* \mid \mathbf{x} \xrightarrow{w} \mathbf{y}\}$$

Example 2.1. Let $\Sigma = \{a, b, c\}$ with $a = (1, 0, 0)$, $b = (-1, 1, 0)$ and $c = (0, -1, 1)$. Let $\mathbf{x} = \langle 0, 0, 0 \rangle$ and $\mathbf{y} = \langle 0, 1/4, 1/4 \rangle$. It is easy to see that $abbc \in L_\Sigma^{\mathbf{x}, \mathbf{y}}$ because of the following run:

$$\langle 0, 0, 0 \rangle \xrightarrow{\frac{1}{2}a} \left\langle \frac{1}{2}, 0, 0 \right\rangle \xrightarrow{\frac{1}{4}b} \left\langle \frac{1}{4}, \frac{1}{4}, 0 \right\rangle \xrightarrow{\frac{1}{4}b} \left\langle 0, \frac{1}{2}, 0 \right\rangle \xrightarrow{\frac{1}{4}c} \left\langle 0, \frac{1}{4}, \frac{1}{4} \right\rangle$$

On the other hand, note that $bbc \notin L_\Sigma^{\mathbf{x}, \mathbf{y}}$. Indeed, suppose there exist fractions $\alpha_1 > 0, \alpha_2 > 0, \alpha_3 > 0$ such that $\mathbf{x} \xrightarrow{\alpha_1 b, \alpha_2 b, \alpha_3 c} \mathbf{y}$. Note that the first counter of \mathbf{x} is zero and also that $\alpha_1 b$ decrements the first counter by α_1 . Hence, the first counter becomes negative after firing $\alpha_1 b$ from \mathbf{x} , which is a contradiction to the requirement that configurations contain only non-negative values.

Now that we have all the necessary definitions, we now move on to stating our main result.

2.2 Main Result - Effective Regularity

The main result of this paper is the following

THEOREM 2.2. *For every CVAS $\Sigma \subset \mathbb{Z}^d$ and configurations $\mathbf{x}, \mathbf{y} \in \mathbb{Q}_+^d$, the language $L_\Sigma^{\mathbf{x}, \mathbf{y}}$ is regular. Moreover, given Σ, \mathbf{x} and \mathbf{y} , one can effectively compute an NFA for it.*

We now discuss some implications of this result. Theorem 2.2 implies that for any class of labeled transition systems (LTS) that (i) has decidable reachability and (ii) is closed under taking products with finite-state systems, the following problem is decidable: Given an LTS S over an alphabet $\Sigma \subset \mathbb{Z}^d$, CVAS configurations $\mathbf{x}, \mathbf{y} \in \mathbb{Q}_+^d$, and states s, t of S , is there a run of S from s to t , such that it can take the CVAS configuration \mathbf{x} into \mathbf{y} ? Indeed, a decision procedure for this problem is as follows: First, construct the NFA for $L_\Sigma^{\mathbf{x}, \mathbf{y}}$ using Theorem 2.2, then take the product of this NFA with S using (ii), and then finally check reachability of this product system using (i).

Let us formalize this result in language-theoretic terms. We say that a class C of languages has *decidable regular intersection* if given $L \subseteq \Sigma^*$ from C and a regular language $R \subseteq \Sigma^*$, it is decidable whether $L \cap R = \emptyset$. This is the case, e.g., if C is the class of languages of a class of LTS with the above properties (i) and (ii).

COROLLARY 2.3. *Suppose C is a class of languages with decidable regular intersection. Then, given a language $L \subseteq \Sigma^*$ in C over some $\Sigma \subset \mathbb{Z}^d$, and configurations $\mathbf{x}, \mathbf{y} \in \mathbb{Q}_+^d$, we can decide whether L intersects $L_\Sigma^{\mathbf{x}, \mathbf{y}}$.*

Corollary 2.3 applies to a wide range of (languages of) infinite-state systems. For example, it applies to the large class of well-structured transition systems (WSTS) [1, 24]. They can be viewed as language-accepting devices [27] (by considering all labels of runs that arrive in some upward closed set of configurations), and since they are closed under taking products with finite-state systems (which are well-structured), their corresponding language class is closed under intersection with regular languages. In particular, this means *reachability is decidable in lossy channel-systems equipped with continuous counters*.

Another application concerns higher-order pushdown automata [17, 50, 51] (even with the collapse operation [34]). Their language class is closed under regular intersection, since their finitely many control states admit a product construction with a finite automaton. Moreover, their emptiness problem is decidable: With collapse, they are equivalent to higher-order recursion

schemes (HORS) [34, Thm. 4.1], and for HORS, even monadic second-order logic is decidable [53, Thm. 16]. Thus, Corollary 2.3 implies that *in higher-order pushdown automata (even with collapse) equipped with continuous counters, reachability is decidable*. This is in contrast to discrete counters: It is known that already for second-order pushdown automata (a special case of higher-order pushdown automata), even adding very simple kinds of discrete counters leads to an undecidable reachability problem (see [59, Prop. 7] and [41, Thm. 4]).

Continuous Petri nets and CVASS. Closely related to CVAS, there are two other formalisms that overapproximate discrete counter systems with continuous semantics: *continuous Petri nets* (CPN) and *continuous vector addition systems with states* (CVASS). The latter are almost the same as CVAS, except that the configurations also feature a control-state, and each transition can only be taken in a particular control-state (and updates the control-state). From Theorem 2.2, it follows immediately that *languages of CVASS are regular as well*, since the language of a CVASS is just the language of a CVAS, intersected with a regular language.

CPN are slightly different: Each transition is not just a vector in \mathbb{Z}^d , but *two* vectors $\text{pre}, \text{post} \in \mathbb{N}^d$. It is applied by picking a fraction $\alpha \in (0, 1]$, then first subtracting $\alpha \cdot \text{pre}$ and then adding $\alpha \cdot \text{post}$. Crucially, after subtracting $\alpha \cdot \text{pre}$, the configuration vector must be non-negative. While in the discrete setting, Petri nets are easily translated into VASS, it is not obvious that CPN can be translated into CVASS. However, Blondin and Haase [13, proof of Prop. 4.1] provide a language-preserving translation from CPN to CVASS. Thus, *languages of continuous Petri nets are regular as well*.

2.3 A Non-Elementary Lower Bound

We now make a remark on the complexity of computing an NFA for $L_\Sigma^{x,y}$. As we shall show in Section 4, the construction from Theorem 2.2 only yields an Ackermannian upper bound for the NFA for $L_\Sigma^{x,y}$. Usually, one would expect that if all languages of a class of systems is regular, then there should be an elementary upper bound (just as, e.g. in the region construction for timed automata [4]). However, as our second result, we show that this is not the case: There is a non-elementary lower bound for the size of NFAs for $L_\Sigma^{x,y}$. More precisely, let $\text{exp}_h: \mathbb{N} \rightarrow \mathbb{N}$ be the h -fold exponentiation function, i.e. $\text{exp}_0(n) = n$ and $\text{exp}_{h+1}(n) = 2^{\text{exp}_h(n)}$. We define the *size* of a configuration $x \in \mathbb{Q}_+^d$ to be the largest absolute value of a numerator or denominator occurring in some component in x . Hence, a configuration $x \in \mathbb{Q}_+^d$ of size s can be represented using at most $O(\log(s) \cdot d)$ bits. We then show that

THEOREM 2.4. *For each $h \geq 2$, there is a $5h$ -dimensional CVAS Σ_h of size $O(h)$ with the following property: For each n , there are configurations x_n, y_n of size at most n , such that any NFA for $L_{\Sigma_h}^{x_n, y_n}$ requires at least $\text{exp}_h(n)$ states.*

This theorem shows that CVAS can be extremely succinct in representing regular languages. This is perhaps surprising, given that checking whether a language of a CVAS is non-empty can be done in polynomial-time [25]. Note that with these results, we leave a complexity gap between non-elementary and Ackermannian. In particular, we leave open whether there is a primitive-recursive construction of an NFA for $L_\Sigma^{x,y}$.

3 Examples and Behaviours of CVAS

Before presenting our proofs, we give examples to show the behaviour of CVAS and motivate our definitions. In this section, let us consider the same CVAS that we had discussed in Example 2.1. That is, we consider the following 3-dimensional CVAS with transitions $\Sigma = \{a, b, c\}$ given by

$$a = (1, 0, 0), \quad b = (-1, 1, 0), \quad c = (0, -1, 1)$$

Let us also set $\mathbf{x} = \langle 0, 0, 0 \rangle$ and $\mathbf{y} = \langle 0, 1/4, 1/4 \rangle$. We will now observe some behaviours of this CVAS for \mathbf{x} and \mathbf{y} . These observations will lead to the concepts of path-schemes, bubbles, stars, and gatherings that are used in our proofs and that we shall introduce at the end of this section.

3.1 Duplication

As a first step, we notice that we have the following run between \mathbf{x} and \mathbf{y} , which proves that $abcabc \in L_{\Sigma}^{\mathbf{x}, \mathbf{y}}$:

$$\begin{array}{ccccccc} \langle 0, 0, 0 \rangle & \xrightarrow{\frac{1}{4}a} & \langle \frac{1}{4}, 0, 0 \rangle & \xrightarrow{\frac{1}{8}b} & \langle \frac{1}{8}, \frac{1}{8}, 0 \rangle & \xrightarrow{\frac{1}{16}c} & \langle \frac{1}{8}, \frac{1}{16}, \frac{1}{16} \rangle \\ & \xrightarrow{\frac{1}{4}a} & \langle \frac{3}{8}, \frac{1}{16}, \frac{1}{16} \rangle & \xrightarrow{\frac{3}{8}b} & \langle 0, \frac{7}{16}, \frac{1}{16} \rangle & \xrightarrow{\frac{3}{16}c} & \langle 0, \frac{1}{4}, \frac{1}{4} \rangle \end{array} \quad (1)$$

One of the important properties of CVAS runs is that it allows immediate duplication of transitions. For example, by halving the fraction of each transition in run 1 and then inserting a copy of each transition next to its original, we get the following run over $aabbccaabbcc$:

$$\langle 0, 0, 0 \rangle \xrightarrow{\frac{1}{8}a \frac{1}{8}a \frac{1}{16}b \frac{1}{16}b \frac{1}{32}c \frac{1}{32}c \frac{1}{8}a \frac{3}{16}b \frac{3}{16}b \frac{3}{32}c \frac{3}{32}c} \left\langle 0, \frac{1}{4}, \frac{1}{4} \right\rangle$$

This shows that $aabbccaabbcc \in L_{\Sigma}^{\mathbf{x}, \mathbf{y}}$. More importantly, this principle can now be applied to $aabbccaabbcc$ itself (duplicating the second a in aa , the second b in bb etc.). Using \hat{a} , \hat{b} , \hat{c} to mark the new copies of a , b , and c , we can prove that $aa\hat{a}bb\hat{b}cc\hat{c}aa\hat{a}bb\hat{b}cc\hat{c} \in L_{\Sigma}^{\mathbf{x}, \mathbf{y}}$. Generalizing this argument allows us to prove that $aa^*bb^*cc^*aa^*bb^*cc^* \in L_{\Sigma}^{\mathbf{x}, \mathbf{y}}$. In fact, whenever $w_1w_2 \dots w_k \in L_{\Sigma}^{\mathbf{x}, \mathbf{y}}$ for any k words w_1, w_2, \dots, w_k , then $w_1w_1^*w_2w_2^* \dots w_kw_k^* \in L_{\Sigma}^{\mathbf{x}, \mathbf{y}}$ as well.⁴

3.2 Padding Runs

Remarkably, we have more freedom to manipulate runs. To illustrate this, we take the run from 1 over the word $abcabc$. We can pad this word by inserting the transitions b, a, c in the middle (again marked by the hat symbol to help the reader) to get the word $abc\hat{b}\hat{a}\hat{c}abc$. We shall now modify the run from 1 to get a run over $abc\hat{b}\hat{a}\hat{c}abc$, thereby showing that this new word is also in $L_{\Sigma}^{\mathbf{x}, \mathbf{y}}$.

To get this new run, we need to redistribute the firing fractions of the transitions in run 1 to accommodate for their additional instances. We do this by reducing the firing fractions of the final instances of a, b and c and moving the slack to the inserted bac . In general, the goal is to adjust the fractions to maintain the same overall effect of each transition, but without any component going below 0 due to the adjustments.

$$\begin{array}{ccccccc} \langle 0, 0, 0 \rangle & \xrightarrow{\frac{1}{4}a} & \langle \frac{1}{4}, 0, 0 \rangle & \xrightarrow{\frac{1}{8}b} & \langle \frac{1}{8}, \frac{1}{8}, 0 \rangle & \xrightarrow{\frac{1}{16}c} & \langle \frac{1}{8}, \frac{1}{16}, \frac{1}{16} \rangle \\ & \xrightarrow{\frac{1}{16}b} & \langle \frac{1}{16}, \frac{1}{8}, \frac{1}{16} \rangle & \xrightarrow{\frac{1}{8}a} & \langle \frac{3}{16}, \frac{1}{8}, \frac{1}{16} \rangle & \xrightarrow{\frac{1}{16}c} & \langle \frac{3}{16}, \frac{1}{16}, \frac{1}{8} \rangle \\ & \xrightarrow{\frac{1}{8}a} & \langle \frac{5}{16}, \frac{1}{16}, \frac{1}{8} \rangle & \xrightarrow{\frac{5}{16}b} & \langle 0, \frac{3}{8}, \frac{1}{8} \rangle & \xrightarrow{\frac{1}{8}c} & \langle 0, \frac{1}{4}, \frac{1}{4} \rangle \end{array} \quad (2)$$

Similarly, we can prove that $abc\hat{b}\hat{a}\hat{c}abc$ also belongs to $L_{\Sigma}^{\mathbf{x}, \mathbf{y}}$. This principle of redistributing the firing fractions is quite general and will allow us to conclude a broad result in Section 6 (namely Theorem 6.1, the Lifting Runs Theorem) that allows us to show effective regularity.

⁴This principle shows that the language of a CVAS is either empty or infinite. As a consequence, there are regular languages that are not CVAS languages.

3.3 Limitations on Run Padding

In the examples above, we could add new transitions, but we only added them after the first instance of each transition in the original run. More precisely, in Section 3.1, we went (using hat to mark additions) from $abcabc$ to $a\hat{a}b\hat{b}c\hat{c}a\hat{a}b\hat{b}c\hat{c}$. Similarly, in Section 3.2 we went from $abcabc$ to $abc\hat{b}\hat{a}\hat{c}abc$. In both cases, we added transitions only after they had already appeared. This is not by accident, as the following argument shows, which limits the kinds of padding that we can perform on runs.

Suppose we consider the word $\hat{b}abcabc$. This adds a b before its first appearance in $abcabc$. It is easy to see that there *cannot* be any firing sequence that would enable this new word to start from $\langle 0, 0, 0 \rangle$ and reach $\langle 0, 1/4, 1/4 \rangle$. This is because b decrements the first counter and so it would be impossible to choose a positive non-zero fraction to fire b from $\langle 0, 0, 0 \rangle$ without making the first counter negative. By the same argument, we can establish similar claims for $\hat{c}abcabc$ and $\hat{a}bcabc$.

We can now see that a similar limitation also applies to the last appearance of transitions, i.e., we cannot always pad transitions after their last appearance in the original word. For instance, consider the word $abcabc\hat{a}$ obtained from $abcabc$ by padding an a at the end. Once again, we can see that there *cannot* be any firing sequence that would enable this new word to start from $\langle 0, 0, 0 \rangle$ reach $\langle 0, 1/4, 1/4 \rangle$. This is because a increments the first counter and so it would be impossible to choose a positive non-zero fraction to fire a without making the first counter strictly bigger than zero. By the same argument, we can also establish a similar claim for $abcab\hat{a}c$.

To sum up, while Section 3.1 and Section 3.2 gave us quite some freedom to pad words and insert transitions, here we have seen some limits on padding, by looking at the first and last appearance of transitions. This naturally leads us to the notion of *gatherings* which we discuss next.

3.4 Gatherings and the Lifting Runs Theorem

So far we have seen that it might be possible to adjust a run by adding additional transitions, as long as we add any new instance of a transition after its first appearance and before its last appearance in the run. This tells us that tracking the first and last appearances of a transition is important. Let us formalize this by means of *first-* and *last-appearance records*.

Let w be a word. For each transition a that appears in w , let $f_a \in [1, |w|]$ (resp. $\ell_a \in [1, |w|]$) be the index of the first (resp. last) appearance of a in w . The *first-appearance* (resp. *last-appearance*) *record* of w is the unique sequence of transitions a_1, a_2, \dots, a_n (resp. b_1, b_2, \dots, b_n) such that $f_{a_1} < f_{a_2} < \dots < f_{a_n}$ (resp. $\ell_{b_1} < \ell_{b_2} < \dots < \ell_{b_n}$).

First- and last-appearance will be used in Section 6 to generalize the reasoning in Section 3.2 for showing that, under some broad conditions, if we have a run over a word w between x and y , then we also have a run between x and y over *any word* w' obtained by inserting transitions into w whilst preserving its first- and last-appearance records (Theorem 6.4).

The first- and last-appearance records of a word are part of the crucial notion of *gatherings*: Informally, a gathering is a set of words that have a particular first- and last-appearance record and where the first appearances occur to the left of all last appearances. Formally, a *gathering* is an expression of the form $X_{a_1 \dots a_n}^{b_1 \dots b_n}$ where a_1, \dots, a_n and b_1, \dots, b_n are letters such that $\{a_1, \dots, a_n\} = \{b_1, \dots, b_n\}$. Intuitively, a_1, \dots, a_n is the order in which each transition should appear first, and b_1, \dots, b_n is the order in which each transition should appear last. Additionally, the last instance of b_1 appears after the first appearance of a_n in the gathering. In full, a word w over the alphabet $\{a_1, \dots, a_n\} = \{b_1, \dots, b_n\}$ is said to *match* this gathering if (i) all first appearances of letters are to the left of all last appearances of letters in w and (ii) it has the same first- and last-appearance records as the one specified by the gathering. Note that each of a_1, \dots, a_n must appear in the word, and these characters may not necessarily comprise the whole of Σ . The term gathering comes from the fact that in such a word, “everyone comes before everyone leaves”.

Example 3.1. Suppose we have the gathering X_{abc}^{abc} . Then the words $abcabc$, $abcbcaabc$ match X_{abc}^{abc} , but the words $abcacb$, $abcabcacb$ do not: since the way the letters appear last in $abcacb$ and $abcabcacb$ is a, c, b . Moreover, due to condition (i), $ababc$ does not match because the last appearance of a is before the first appearance of c . If we have the gathering X_{ab}^{ba} , then the word $abba$ matches X_{ab}^{ba} , but aba and $abab$ do not match X_{ab}^{ba} .

In Theorem 6.4, the Lifting Runs Theorem for Gatherings, we see that, by using these ideas, from a gathering that intersects $L_{\Sigma}^{x,y}$ we can find an infinite regular language that is contained in $L_{\Sigma}^{x,y}$. This allows us to make strides towards a regular representation of $L_{\Sigma}^{x,y}$.

However, to construct a regular representation of $L_{\Sigma}^{x,y}$, gatherings alone are too simple. As a next step, we synthesize the ideas presented so far to introduce *path-schemes*. The Lifting Runs Theorem for Gatherings is generalised to path-schemes in Theorem 6.1 (The Lifting Runs Theorem).

3.5 Path-Schemes

In Section 3.4, we saw that it is important to track *gatherings* in order to characterize the language of a CVAS. In Section 3.1 we also saw that it is sometimes necessary to track languages of the form $w_0\Sigma_1^*w_1\Sigma_2^*\dots\Sigma_n^*w_n$ for words w_0, w_1, \dots, w_n and subsets $\Sigma_1, \Sigma_2, \dots, \Sigma_n \subseteq \Sigma$. We refer to the Σ_i^* components as *stars*. Let us now combine both of these ideas into the notion of a *path-scheme*.

Let us fix a set of transitions Σ and two configurations \mathbf{x} and \mathbf{y} . A *path-scheme* over Σ defines a regular language of words and is given by an expression $\rho = u_0X_1u_1\dots X_nu_n$ where each $u_i \in \Sigma^*$ and each X_i is a *bubble*. A bubble is either,

- a *star* X_A for a set of characters $A \subseteq \Sigma$, or
- a *gathering* $X_{a_1\dots a_n}^{b_1\dots b_n}$ where $\{a_1, \dots, a_n\} = \{b_1, \dots, b_n\} \subseteq \Sigma$ and a_1, \dots, a_n are pairwise distinct (and thus also b_1, \dots, b_n).

The language of a star $L(X_A)$ is the set of words A^* . The language of a gathering $L(X_{a_1\dots a_n}^{b_1\dots b_n})$ is the set of all words that match the gathering, i.e., all w such that $w = a_1w_1\dots a_nw_nb_1v_1\dots b_nv_n$ where for all i we have $w_i \in \{a_1, \dots, a_i\}^*$ and $v_i \in (\{a_1, \dots, a_n\} \setminus \{b_1, \dots, b_i\})^*$. For words $w \in L(X_{a_1\dots a_n}^{b_1\dots b_n})$, we define its *center* as $\text{center}(w) = w_n$. That is, $\text{center}(w)$ is the part of w between the first occurrence of a_n and the last occurrence of b_1 where all characters in the gathering can occur freely.

The language $L(\rho)$ of a path-scheme $\rho = u_0X_1u_1\dots X_nu_n$ is the set of words of the form $u_0w_1\dots w_nu_n$ where $w_i \in L(X_i)$ for all i . In such a case, we say that (w_1, \dots, w_n) is a ρ -*factorization* or short ρ -*factor* of w . A path-scheme ρ is said to be *pre-perfect* if all of its bubbles are gatherings. ρ is said to be *perfect* if it is pre-perfect and in addition $L(\rho) \cap L_{\Sigma}^{x,y} \neq \emptyset$.

We will prove that the language of a CVAS can be (effectively) represented as a finite union of perfect path-schemes. This will immediately imply that the language of a CVAS is (effectively) regular. In the next section, we introduce the main ideas and give an overview of this proof.

4 Key Concepts and Proof Overview

4.1 Proof Overview

Our NFA construction for Theorem 2.2 proceeds as follows. It maintains a list of path-schemes, which initially just consists of a single path-scheme X_{Σ} , representing Σ^* . As we saw in Section 3.5, a path-scheme ρ gives rise to a set $L(\rho) \subseteq \Sigma^*$ of transition sequences. Here, it may still be the case that a path-scheme contains no word from $L_{\Sigma}^{x,y}$.

One step in our procedure is to decompose each path-scheme into finitely many perfect path-schemes. By definition, a perfect path-scheme intersects $L_{\Sigma}^{x,y}$. This is the *first decomposition* procedure. However, just finding perfect path-schemes using the first decomposition does not yield

effective regularity: Instead, we show in addition that each perfect path-scheme ρ not only contains a *single run*, but in fact contains an infinite regular subset $R_\rho \subseteq L(\rho)$ with $R_\rho \subseteq L_\Sigma^{\mathbf{x},\mathbf{y}}$.

However, the set R_ρ may not capture all of $L(\rho)$. Therefore, our proof uses a *second decomposition* to turn a perfect ρ into finitely many (not necessarily perfect) path-schemes $\sigma, \rho_1, \dots, \rho_m$ where σ captures R_ρ and ρ_1, \dots, ρ_m capture the remaining transition sequences of ρ . This step employs the Lifting Runs Theorem (Theorem 6.1) to identify R_ρ , from which σ and ρ_1, \dots, ρ_m are computed.

Our construction then alternates between these two decomposition steps. Termination is guaranteed by the fact that each decomposition step yields path-schemes that are smaller—in some appropriate lexicographical ordering—than the path-scheme that is decomposed.

Having described our high-level strategy, we now state the precise theorems corresponding to the first and the second decomposition mentioned above.

4.2 The Two Decomposition Steps

We will describe here what our two decomposition steps achieve. The resulting theorems — Theorems 4.2 and 4.3—will be stated here and proven in Section 5 and Section 6, respectively.

For the rest of this section, let us fix a CVAS with set of transitions Σ and two configurations \mathbf{x} and \mathbf{y} . We now state the two main theorems that we need to prove our main result. In order to state these two theorems, we first need to set up a key definition.

As mentioned before, to prove termination of our construction, we need to impose a lexicographic ordering on path-schemes. To this end, we define the *weight* of a path-scheme $W(\rho)$ as a vector in $\mathbb{N}^{|\Sigma|}$ such that the i^{th} component of $W(\rho)$ denotes the number of bubbles in ρ with exactly i distinct elements in it. More formally, for each $1 \leq i \leq |\Sigma|$, we first let \bar{i} be the vector that is 1 in the i^{th} component and 0 elsewhere. Then, we inductively define weight as

- $W(X_A) = \overline{|A|}$.
- $W(X_{a_1 \dots a_n}^{b_1 \dots b_n}) = \bar{n}$.
- $W(u_0 X_1 \dots X_n u_n) = \sum_i W(X_i)$.

Example 4.1. Let $\Sigma = \{a, b, c\}$. The star $X_{a,b}$ has weight $(0, 1, 0)$ as it contains a single bubble with 2 elements in its support. Similarly, the gathering $X_{a,b,c}^{b,a,c}$ has weight $(0, 0, 1)$. For the path-scheme $aX_{a,b}cX_{a,b,c}^{b,a,c}$, its weight is the sum of the weights of its bubbles, which is $(0, 1, 1)$.

Now, given two path-schemes σ and ρ , we say that $W(\sigma) \leq_{\text{lex}} W(\rho)$ if $W(\sigma)$ is *lexicographically smaller* than $W(\rho)$, i.e., either the $|\Sigma|^{\text{th}}$ component of $W(\sigma)$ is strictly smaller than the $|\Sigma|^{\text{th}}$ component of $W(\rho)$ or they are equal and the $(|\Sigma| - 1)^{\text{th}}$ component of $W(\sigma)$ is strictly smaller than the $(|\Sigma| - 1)^{\text{th}}$ component of $W(\rho)$ and so on. We say that $W(\sigma) <_{\text{lex}} W(\rho)$ if $W(\sigma)$ is *strictly lexicographically smaller* than $W(\rho)$, i.e., if $W(\sigma) \leq_{\text{lex}} W(\rho)$ and $W(\sigma) \neq W(\rho)$.

We now introduce the two main theorems needed to prove our effective regularity result. Both of these results decompose path-schemes into “simpler” path-schemes whilst still preserving certain properties regarding $L_\Sigma^{\mathbf{x},\mathbf{y}}$. The first one states that any path-scheme ρ can be decomposed into a finite number of simpler *perfect* path-schemes that together preserves the intersection $L(\rho) \cap L_\Sigma^{\mathbf{x},\mathbf{y}}$.

THEOREM 4.2 (PATH-SCHEME DECOMPOSITION). *Given a path-scheme ρ , we can compute a finite set of perfect path-schemes ρ_1, \dots, ρ_m such that $L(\rho) \cap L_\Sigma^{\mathbf{x},\mathbf{y}} = (\bigcup_i L(\rho_i)) \cap L_\Sigma^{\mathbf{x},\mathbf{y}}$ and $W(\rho_i) \leq_{\text{lex}} W(\rho)$ for each i .*

Note that if $L(\rho) \cap L_\Sigma^{\mathbf{x},\mathbf{y}}$ is empty, then the above theorem guarantees that so is $(\bigcup_i L(\rho_i)) \cap L_\Sigma^{\mathbf{x},\mathbf{y}}$. However, since ρ_1, \dots, ρ_m are required to be perfect path-schemes, in this case we will have $m = 0$.

The second theorem states that any perfect path-scheme ρ can be decomposed into a finite number of path-schemes such that one of them completely sits inside $L_\Sigma^{\mathbf{x},\mathbf{y}}$ and all the others have

strictly smaller weight than ρ . This theorem relies on the Lifting Runs Theorem (Theorem 6.1) to identify the required path-schemes.

THEOREM 4.3 (PERFECT PATH-SCHEME DECOMPOSITION). *Given a perfect path-scheme ρ , we can compute a path-scheme σ and a finite set of path-schemes ρ_1, \dots, ρ_m such that $L(\rho) = L(\sigma) \cup \bigcup_i L(\rho_i)$ and $L(\sigma) \subseteq L_\Sigma^{\mathbf{x}, \mathbf{y}}$ and $W(\rho_i) <_{\text{lex}} W(\rho)$ for all i .*

We shall now see how these two theorems can be used to prove our effective regularity result.

4.3 Proof of Effective Regularity

Assuming the above two theorems, we show that $L_\Sigma^{\mathbf{x}, \mathbf{y}}$ is effectively regular. Here (and later in the proof), we employ a result by Blondin and Haase [13, Thm 4.9], which allows us to decide, for a given CVAS Σ and a regular language $R \subseteq \Sigma^*$, whether the language $L_\Sigma^{\mathbf{x}, \mathbf{y}}$ intersects R .

PROPOSITION 4.4 (BLONDIN & HAASE). *Given a CVAS $\Sigma \subset \mathbb{Z}^d$, configurations $\mathbf{x}, \mathbf{y} \in \mathbb{Q}_+^d$ and a regular language R , the problem of deciding whether $R \cap L_\Sigma^{\mathbf{x}, \mathbf{y}} \neq \emptyset$ is decidable (and NP-complete).*

This is because in [13, Thm 4.9], Blondin and Haase show that reachability in ‘‘CVAS with states’’, which are CVAS with finitely many control states, is NP-complete. Proposition 4.4 implies that given a path-scheme ρ , it is decidable whether $L(\rho) \cap L_\Sigma^{\mathbf{x}, \mathbf{y}} \neq \emptyset$ (since $L(\rho)$ is a regular language).

To show effective regularity, using Proposition 4.4, we first check if $\Sigma^* \cap L_\Sigma^{\mathbf{x}, \mathbf{y}} \neq \emptyset$, i.e., we check if $L_\Sigma^{\mathbf{x}, \mathbf{y}}$ is non-empty. If it is empty, then we are already done. Hence, for the rest of this proof, we assume that $L_\Sigma^{\mathbf{x}, \mathbf{y}} \neq \emptyset$.

We start with a tree T_0 and then progressively add leaves to construct trees T_1, T_2, \dots with each tree T_i satisfying the following invariants:

- C0 Each node of the tree will be labelled by one path-scheme ρ such that $L(\rho) \cap L_\Sigma^{\mathbf{x}, \mathbf{y}} \neq \emptyset$. This path-scheme will either be marked or unmarked. If it is at an odd level, then it will be a perfect path-scheme.
- C1 For any node ρ at an odd level, if its parent is ρ' , then $W(\rho) \leq_{\text{lex}} W(\rho')$.
- C2 For any unmarked node ρ at an even level, if its parent is ρ' , then $W(\rho) <_{\text{lex}} W(\rho')$.
- C3 If a node ρ is marked, then $L(\rho) \subseteq L_\Sigma^{\mathbf{x}, \mathbf{y}}$ and it will not have any children.
- C4 The union of the languages at the leaf nodes will be an overapproximation of $L_\Sigma^{\mathbf{x}, \mathbf{y}}$, i.e., a super-set of $L_\Sigma^{\mathbf{x}, \mathbf{y}}$.

We begin by setting T_0 to be the tree consisting of just the unmarked root node labelled with the path-scheme X_Σ . Clearly, the five conditions are satisfied by T_0 (where we take the root node to be a node at level 0). Suppose we have already constructed T_k . If all the leaves of T_k are marked, then we stop the construction. Otherwise, let ρ be an unmarked leaf of T_k . There are two possibilities:

- ρ is a leaf at an even level: In this case, we apply the Path-Scheme Decomposition theorem to ρ to get a finite set of perfect path-schemes ρ_1, \dots, ρ_m such that $L(\rho) \cap L_\Sigma^{\mathbf{x}, \mathbf{y}} = (\bigcup_i L(\rho_i)) \cap L_\Sigma^{\mathbf{x}, \mathbf{y}}$ and $W(\rho_i) \leq_{\text{lex}} W(\rho)$ for each i . We then attach each ρ_i as an unmarked leaf to ρ . Let this new tree be T_{k+1} .
By construction, the invariants C0, C1, C2 and C3 are immediately satisfied. For invariant C4, notice that $L(\rho) \cap L_\Sigma^{\mathbf{x}, \mathbf{y}} = (\bigcup_i L(\rho_i)) \cap L_\Sigma^{\mathbf{x}, \mathbf{y}}$. Since C4 was satisfied by the tree T_k and since the leaf ρ in T_k is replaced by the leaves ρ_1, \dots, ρ_m in T_{k+1} , it follows that C4 is also satisfied by the tree T_{k+1} .
- ρ is a leaf at an odd level: By invariant C0, ρ must be a perfect path-scheme. We apply the Perfect Path-Scheme Decomposition theorem to ρ to get a path-scheme σ and a finite set of path-schemes ρ_1, \dots, ρ_m such that $L(\rho) = L(\sigma) \cup \bigcup_i L(\rho_i)$ and $L(\sigma) \subseteq L_\Sigma^{\mathbf{x}, \mathbf{y}}$ and $W(\rho_i) <_{\text{lex}} W(\rho)$ for all i . We then attach σ as a marked leaf. Further, we add each ρ_i that

satisfies $L(\rho_i) \cap L_\Sigma^{\mathbf{x}, \mathbf{y}} \neq \emptyset$ (we check this using Proposition 4.4) as an unmarked leaf to ρ . Let this new tree be T_{k+1} .

By construction, the invariants C0, C1, C2 and C3 are once again immediately satisfied. For invariant C4, note that the only reason we do not add some ρ_i as a leaf to ρ is if $L(\rho_i) \cap L_\Sigma^{\mathbf{x}, \mathbf{y}} = \emptyset$. Hence, if $\sigma, \rho_{i_1}, \rho_{i_2}, \dots, \rho_{i_q}$ are the path-schemes which we added as a leaf to ρ , then $L(\rho) \cap L_\Sigma^{\mathbf{x}, \mathbf{y}} = (L(\sigma) \cup \bigcup_{i_j} L(\rho_{i_j})) \cap L_\Sigma^{\mathbf{x}, \mathbf{y}}$. Since C4 was satisfied by the tree T_k and since the leaf ρ in T_k is replaced by the leaves $\sigma, \rho_{i_1}, \dots, \rho_{i_q}$ in T_{k+1} , it follows that C4 is satisfied by T_{k+1} as well.

We claim that the construction of T_0, T_1, \dots halts after a finite number of steps. Towards a contradiction, suppose T_0, T_1, \dots does not terminate. Note that each T_i satisfies all the five invariants, is finitely-branching and is obtained by adding a finite number of leaves to T_{i-1} . Hence, from T_0, T_1, \dots we get a finitely-branching infinite tree T that also satisfies all of the five invariants. By König's lemma, T must have an infinite path. By invariant C3, this path has no marked nodes. By C1 and C2, the weight of an unmarked node at an even level along this path strictly decreases (with respect to the $<_{\text{lex}}$ ordering) when compared with the weight of its grandparent. By the well-ordering of the lexicographic ordering, this path cannot be infinite, leading to a contradiction.

Hence, the construction halts at some point (say T_m). By definition of the construction, this can happen iff there are no unmarked leaves in T_m . Let ρ_1, \dots, ρ_ℓ be the set of all leaves in T_m . By invariant C4, we have $L_\Sigma^{\mathbf{x}, \mathbf{y}} \subseteq \bigcup_{1 \leq i \leq \ell} L(\rho_i)$ and by invariant C3, we have $\bigcup_{1 \leq i \leq \ell} L(\rho_i) \subseteq L_\Sigma^{\mathbf{x}, \mathbf{y}}$. Hence, $L_\Sigma^{\mathbf{x}, \mathbf{y}} = \bigcup_{1 \leq i \leq \ell} L(\rho_i)$, and thus $L_\Sigma^{\mathbf{x}, \mathbf{y}}$ is effectively regular.

We now conclude with a brief discussion on the complexity of our construction, i.e., the size of the final set of path-schemes that we obtain.

Complexity. By a straightforward application of length function theorems for ordinals [55, Theorem 3.5], we can show that the NFA for $L_\Sigma^{\mathbf{x}, \mathbf{y}}$ can be computed in Ackermannian time. Moreover, if we fix the number $h = |\Sigma|$ of transitions in our CVA, then we obtain a primitive recursive construction. This is because termination in our construction is guaranteed by descent w.r.t. the lexicographical ordering $(\mathbb{N}^{|\Sigma|}, \leq_{\text{lex}})$, which is the ordinal $\omega^{|\Sigma|}$, and each construction of path-schemes runs in elementary time. This elementary time bound in particular yields an elementary control function for the application of the length function theorem for ordinals. A detailed argument can be found in the full version of this paper.

Division of Proof into Sections. Hence, in order to complete the proof of effective regularity (Theorem 2.2), it suffices to prove Theorems 4.2 and 4.3, which we do in Sections 5 and 6 respectively.

5 Path-Scheme Decomposition

In this section, we will prove Theorem 4.2. We begin by proving a slightly weaker claim for the special case of *stars*.

LEMMA 5.1 (STAR DECOMPOSITION). *Given a star X_A , we can compute a finite set of pre-perfect path-schemes ρ_1, \dots, ρ_m such that $L(X_A) = L(\rho_1) \cup \dots \cup L(\rho_m)$ and $W(\rho_i) \leq_{\text{lex}} W(X_A)$.*

PROOF SKETCH. We proceed inductively w.r.t. the size of A . The base case is when $|A| = 0$. Hence $A = \emptyset$, and so we can pick the pre-perfect path-scheme $\rho = \epsilon$ consisting of just the empty word. It is then easy to see that $L(X_A) = A^* = \{\epsilon\}$ and also that $W(\rho) = W(X_A)$.

For the induction step, suppose $|A| > 0$. Then we first claim that we can decompose A^* as

$$A^* = \bigcup_G L(G) \cup \bigcup_{\substack{B \subseteq A \\ C \subseteq A}} B^* C^* \cup \bigcup_{\substack{a \in A \\ B \subseteq A \setminus \{a\} \\ C \subseteq A \setminus \{a\}}} B^* a C^*, \quad (3)$$

where G ranges over all gatherings $X_{a_1, \dots, a_n}^{b_1, \dots, b_n}$ with $A = \{a_1, \dots, a_n\} = \{b_1, \dots, b_n\}$. The inclusion “ \supseteq ” is trivial, so let us prove the other direction “ \subseteq ”, i.e., we prove that every word in A^* belongs to one of the terms on the right-hand side of the above equation.

Consider a word $w \in A^*$. If some letter $a \in A$ occurs less than twice in w , then w belongs either (i) to B^*aC^* for some $B \subseteq A \setminus \{a\}$, $C \subseteq A \setminus \{a\}$ or (ii) to B^* for some $B \subseteq A$. Thus, let us assume that every letter occurs at least twice in w . Let $A = \{a_1, \dots, a_n\}$ and for each $1 \leq i \leq n$, let $f_i \in [1, |w|]$ be the position of the first occurrence of a_i in w , and $\ell_i \in [1, |w|]$ be the last occurrence of a_i in w . We now consider two cases:

- (i) Some first occurrence is to the right of some last occurrence, i.e. there are i, j such that $\ell_j < f_i$. In this case it is easy to see that w belongs to B^*C^* , where $B = A \setminus \{a_i\}$ and $C = A \setminus \{a_j\}$ and so we are done.
- (ii) Every first occurrence is to the left of every last occurrence, i.e. $f_i < \ell_j$ for all i, j . In this case we claim that there is a gathering G such that $w \in L(G)$. Indeed, let i_1, \dots, i_n and j_1, \dots, j_n be permutations of $\{1, \dots, n\}$ such that $f_{i_1} < f_{i_2} < \dots < f_{i_n} < \ell_{j_1} < \ell_{j_2} < \dots < \ell_{j_n}$. Then, it is easily seen that w belongs to the gathering X whose first-appearance record is a_{i_1}, \dots, a_{i_n} and whose last-appearance record is a_{j_1}, \dots, a_{j_n} . Hence, in this case as well, we are done.

With (3) at our disposal, in order to prove the induction step for X_A , it suffices to prove the following: For each term S that appears in the right-hand side of (3), we can decompose S as a finite set of pre-perfect path-schemes $\sigma_1, \dots, \sigma_k$ such that each $W(\sigma_i) \leq_{\text{lex}} W(X_A)$. This follows by a simple case analysis on S . The intuition is that S is either the language of a gathering G , or a term of the form B^*C^* or B^*aC^* for some $B, C \subseteq A$. In the first case, gatherings are pre-perfect path-schemes themselves. In the latter two cases, we use the induction hypothesis to get pre-perfect path-schemes for B^* and C^* . From these, we can construct a set of pre-perfect path-schemes for each language of the form B^*C^* and B^*aC^* . Note that inductively, the pre-perfect path-schemes for B^* , C^* have weight $\leq_{\text{lex}} \overline{|A|} - 1$. Thus, a concatenation of such schemes will have a weight vector that is strictly less than $\overline{|A|}$. Moreover, each gathering G has weight $\overline{|A|}$. Thus, every pre-perfect path-scheme we construct has a weight vector that is at most $\overline{|A|} = W(X_A)$. A detailed proof of this case analysis on S is given in the full version of this paper.

This completes the induction step and therefore concludes the proof of the lemma. \square

Using this lemma, we now prove Theorem 4.2.

PROOF OF THEOREM 4.2. First, notice that it suffices to only compute pre-perfect path-schemes ρ_1, \dots, ρ_m with $L(\rho) = \bigcup_{i=1}^m L(\rho_i)$ and $W(\rho_i) \leq_{\text{lex}} W(\rho)$ for each i . Indeed, if we can compute such pre-perfect path-schemes, then we can check whether each ρ_i is perfect. This amounts to checking whether $L(\rho_i) \cap L_{\Sigma}^{x,y} \neq \emptyset$, which is decidable (even in NP) by Proposition 4.4. Let $\rho_{i_1}, \dots, \rho_{i_k}$ be the perfect path-schemes from this list. It is then easy to see that $L(\rho) \cap L_{\Sigma}^{x,y} = (\cup_{i_j} L(\rho_{i_j})) \cap L_{\Sigma}^{x,y}$ and $W(\rho_{i_j}) \leq_{\text{lex}} W(\rho)$ for each i_j , thereby yielding the desired list of perfect path-schemes for ρ .

Now let ρ be a path-scheme. Let X_{A_1}, \dots, X_{A_s} be the occurrences of stars in ρ . For each star X_{A_i} in ρ , we compute a list $\sigma_{i,1}, \dots, \sigma_{i,t}$ of perfect path-schemes using Lemma 5.1 (by duplicating the path-schemes if necessary, we may assume that we get the same number t of schemes for each A_i).

For each function $f: [1, s] \rightarrow [1, t]$, consider the perfect path-scheme ρ_f obtained from ρ by replacing each X_{A_i} with $\sigma_{i,f(i)}$. It is then easy to see that $L(\rho) = \bigcup_f \rho_f$, where f ranges over all functions $[1, s] \rightarrow [1, t]$. It remains to argue that $W(\rho_f) \leq_{\text{lex}} W(\rho)$. For this, we observe that since $W(\sigma_{i,f(i)}) \leq_{\text{lex}} W(X_{A_i})$, it follows that replacing X_{A_i} with $\sigma_{i,f(i)}$ in any path-scheme will not increase the weight: This is because if $\mathbf{u} \leq_{\text{lex}} \mathbf{v}$ for some vectors $\mathbf{u}, \mathbf{v} \in \mathbb{N}^k$, then $\mathbf{w} + \mathbf{u} \leq_{\text{lex}} \mathbf{w} + \mathbf{v}$ for every $\mathbf{w} \in \mathbb{N}^k$. Applying this observation s times yields that $W(\rho_f) \leq_{\text{lex}} W(\rho)$. \square

6 Perfect Path-Scheme Decomposition

In this section, we will prove Theorem 4.3. In order to prove this theorem, we first set up some notation and introduce some key definitions.

Recall that a word w is said to be less than or equal to another word w' over the subword ordering (denoted by $w \leq w'$) if w can be obtained from w' by deleting some letters in it. Now, let X be a gathering and let $w \in L(X)$. We define $(w)_X^\uparrow$ to be the set of all words $w' \in L(X)$ such that $\text{center}(w) \leq \text{center}(w')$ with respect to the subword ordering \leq . Hence, $(w)_X^\uparrow$ is the set of all words in $L(X)$ whose center component is larger than or equal to the center component of w .

We extend the above notion to pre-perfect path-schemes. Let $\rho = u_0 X_1 \dots X_n u_n$ be a pre-perfect path-scheme and let $\vec{w} = (w_1, \dots, w_n)$ be a ρ -factor. We let $(\vec{w})_\rho^\uparrow$ be the set of words given by $\{u_0 w'_1 u_1 w'_2 \dots w'_n u_n : \forall 1 \leq i \leq n, w'_i \in (w_i)_{X_i}^\uparrow\}$. Intuitively, $(\vec{w})_\rho^\uparrow$ is the set of all words obtained by replacing each w_i with some $w'_i \in (w_i)_{X_i}^\uparrow$. We call $(\vec{w})_\rho^\uparrow$ the ρ -upward closure of \vec{w} .

With these notations set up, we are now ready to state the three main ingredients which will together imply Theorem 4.3. To state these ingredients, we first set up some context. Recall that to prove Theorem 4.3, we need to split a perfect path-scheme ρ into path-schemes $\sigma, \rho_1, \dots, \rho_m$ such that $L(\sigma)$ is contained entirely in $L_\Sigma^{x,y}$ and the weight of each ρ_i is strictly less than the weight of ρ . The three ingredients will help us find the path-schemes $\sigma, \rho_1, \dots, \rho_m$. The first two are about finding σ and the last one is about finding ρ_1, \dots, ρ_m .

First Ingredient: Lifting Runs Theorem. The Lifting Runs Theorem states that from any perfect path-scheme ρ , we can extract a word w and a ρ -factor \vec{w} such that the entire set $(\vec{w})_\rho^\uparrow$ is contained in $L_\Sigma^{x,y}$. Intuitively this result allows us to *lift* the ρ -factor \vec{w} to its ρ -upward closure $(\vec{w})_\rho^\uparrow$.

THEOREM 6.1 (LIFTING RUNS THEOREM). *Suppose ρ is a perfect path-scheme. Then, we can compute a word $w \in L(\rho) \cap L_\Sigma^{x,y}$ and a corresponding ρ -factor \vec{w} such that $(\vec{w})_\rho^\uparrow \subseteq L(\rho) \cap L_\Sigma^{x,y}$.*

Second Ingredient: Path-Scheme for ρ -Upward Closures. Given \vec{w} , obtained from the Lifting Runs Theorem, we can partition $L(\rho)$ into two: $(\vec{w})_\rho^\uparrow$ and $L(\rho) \setminus (\vec{w})_\rho^\uparrow$. Our second result proves that it is always possible to construct a path-scheme σ that captures exactly the set $(\vec{w})_\rho^\uparrow$.

THEOREM 6.2 (PATH-SCHEME THEOREM FOR ρ -UPWARD CLOSURES). *Given a perfect path-scheme ρ and a ρ -factor \vec{w} , we can compute a path-scheme σ such that $L(\sigma) = (\vec{w})_\rho^\uparrow$.*

Third Ingredient: Path-Schemes for Complement of ρ -Upward Closures. We characterize the complement of this ρ -upward closure, i.e., $L(\rho) \setminus (\vec{w})_\rho^\uparrow$, and get the desired ρ_1, \dots, ρ_m for proving Theorem 4.3. In particular, we can compute a finite set of path-schemes ρ_1, \dots, ρ_m such that their union *over-approximates* $L(\rho) \setminus (\vec{w})_\rho^\uparrow$ and the weight of each ρ_i is strictly less than the weight of ρ .

THEOREM 6.3 (PATH-SCHEMES THEOREM FOR COMPLEMENTS). *Given a perfect path-scheme ρ and a ρ -factor \vec{w} , we can compute a finite set of path-schemes ρ_1, \dots, ρ_m such that $L(\rho) \setminus (\vec{w})_\rho^\uparrow \subseteq \bigcup_i L(\rho_i) \subseteq L(\rho)$ and $W(\rho_i) <_{\text{lex}} W(\rho)$ for all i .*

In the remainder of this section, we will prove our three ingredients. First, we first show how they together imply Theorem 4.3. (A pictorial representation of this implication can be found in Fig. 1).

PROOF OF THEOREM 4.3. Let ρ be a perfect path-scheme. By the Lifting Runs Theorem, we can compute a word $w \in L(\rho)$ and a ρ -factor \vec{w} such that $(\vec{w})_\rho^\uparrow \subseteq L(\rho) \cap L_\Sigma^{x,y}$. Then, by the Path-Scheme Theorem for ρ -Upward Closures, we can compute a path-scheme σ such that $L(\sigma) = (\vec{w})_\rho^\uparrow$. Finally,

by the Path-Schemes Theorem for Complements, we can compute path-schemes ρ_1, \dots, ρ_m such that $L(\rho) \setminus (\vec{w})_\rho^\uparrow \subseteq \bigcup_i L(\rho_i) \subseteq L(\rho)$ and $W(\rho_i) <_{\text{lex}} W(\rho)$ for all i . By construction, we have that $L(\rho) = L(\sigma) \cup \bigcup_i L(\rho_i)$, $L(\sigma) \subseteq L_\Sigma^{\mathbf{x}, \mathbf{y}}$ and $W(\rho_i) <_{\text{lex}} W(\rho)$ for all i , proving Theorem 4.3. \square

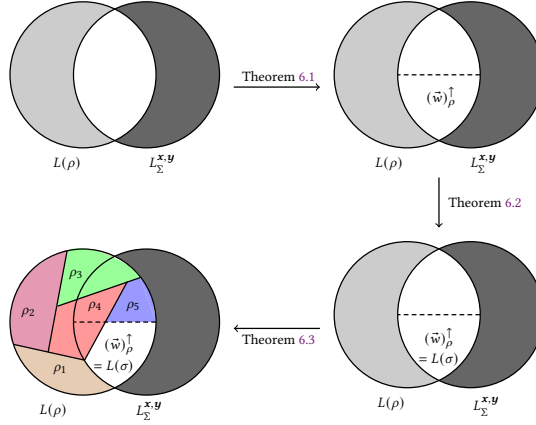


Fig. 1. Venn-diagram illustration of the proof of Theorem 4.3. Theorem 6.1 allows us to find a ρ -upward closed set $(\vec{w})_\rho^\uparrow$ that belongs to both the circles, i.e., to both $L(\rho)$ and $L_\Sigma^{\mathbf{x}, \mathbf{y}}$. Theorem 6.2 finds a path-scheme σ that equals this set. Along with this σ , Theorem 6.3 then allows us to split the left circle into multiple fragments (here ρ_1, \dots, ρ_5) such that each ρ_i has weight strictly smaller than ρ . Notice that the fragments need not be disjoint with σ ; for example, ρ_4 intersects with σ .

6.1 Proof of the First Ingredient: Lifting Runs Theorem

In this subsection, we will prove our first ingredient, namely the Lifting Runs Theorem (Theorem 6.1). First we prove the following special case of this theorem for *gatherings*.

THEOREM 6.4 (LIFTING RUNS THEOREM FOR GATHERINGS). *Suppose $X = X_{b_1, \dots, b_n}^{a_1, \dots, a_n}$ is a gathering such that $L(X) \cap L_\Sigma^{\mathbf{x}, \mathbf{y}} \neq \emptyset$. Then, we can compute a word $w \in L(X) \cap L_\Sigma^{\mathbf{x}, \mathbf{y}}$ such that $(w)_X^\uparrow \subseteq L(X) \cap L_\Sigma^{\mathbf{x}, \mathbf{y}}$.*

PROOF SKETCH. To prove this theorem, we begin with a fact from the proof of [13, Proposition 4.5]. It states that if $L(X) \cap L_\Sigma^{\mathbf{x}, \mathbf{y}} \neq \emptyset$ then we can compute a word $w \in L(X) \cap L_\Sigma^{\mathbf{x}, \mathbf{y}}$ such that $w = u \cdot \text{center}(w) \cdot v$ where $u = a_1 a_2 \dots a_n$, $v = b_1 b_2 \dots b_n$, the set of letters that appear in $\text{center}(w)$ is exactly $\{a_1, \dots, a_n\}$ and the following property is satisfied: There exist two vectors \mathbf{x}' , \mathbf{y}' and three sequences of firing fractions α, β, γ such that $\mathbf{x} \xrightarrow{\alpha u} \mathbf{x}' \xrightarrow{\beta \text{center}(w)} \mathbf{y}' \xrightarrow{\gamma v} \mathbf{y}$ and

- If a counter is non-zero at any point in the run $r_1 = \mathbf{x} \xrightarrow{\alpha u} \mathbf{x}'$ then it stays non-zero from that point onwards in r_1 .
- If a counter is zero at any point in the run $r_3 = \mathbf{y}' \xrightarrow{\gamma v} \mathbf{y}$ then it stays zero from that point onwards in r_3 .
- If a counter was non-zero at any point in either r_1 or r_3 , then it stays non-zero along the run $r_2 = \mathbf{x}' \xrightarrow{\beta \text{center}(w)} \mathbf{y}'$.⁵

⁵Technically, this fact was proven for cyclic reachability in CVAS with finitely many control states, i.e., reachability in CVAS with states with the same starting and the final state. However, the same fact applies in our setting as well, because any gathering X over a CVAS Σ can be converted into a CVAS with states where the starting and the final state are the same.

With these properties at our disposal, we can prove that not only is $w \in L(X) \cap L_\Sigma^{\mathbf{x}, \mathbf{y}}$ but that we have the stronger property $(w)_X^\uparrow \subseteq L(X) \cap L_\Sigma^{\mathbf{x}, \mathbf{y}}$. To this end suppose $w' \in (w)_X^\uparrow$. By definition, $w' \in L(X)$ and so it suffices to prove that $w' \in L_\Sigma^{\mathbf{x}, \mathbf{y}}$. Since X is a gathering and since $w' \in (w)_X^\uparrow$, we can split w' as $w' = u' \cdot \text{center}(w') \cdot v'$ such that $\text{center}(w) \leq \text{center}(w')$. Furthermore, since $u = a_1 a_2 \dots a_n$ and $v = b_1 b_2 \dots b_n$, by definition of a gathering, we also have that $u \leq u'$ and $v \leq v'$.

We can now argue that there exist sequences of firing fractions α', β' and γ' such that $\mathbf{x} \xrightarrow{\alpha' u'} \mathbf{x}' \xrightarrow{\beta' \text{center}(w')} \mathbf{y}' \xrightarrow{\gamma' v'} \mathbf{y}$. Intuitively, we do this as follows: For each transition t , we show that we can take away a small part of its firing fraction from the sequences $\alpha u, \beta \text{center}(w)$ and γv and redistribute this small part amongst its additional occurrences present in $u', \text{center}(w')$ and v' respectively. When this redistribution is done naively and arbitrarily, it might happen that some counter value becomes negative at some point. To prevent this, we use the special properties of w to show that there is a way of redistribution which forces the counters to stay non-negative at each step. The formal details of this redistribution can be found in the full version of this paper. \square

Example 6.5. We give here an example that illustrates the main techniques behind the redistribution of firing fractions in the proof of Theorem 6.4. Let us consider the example from Section 3 where $\Sigma = \{a, b, c\}$ with $a = (1, 0, 0)$, $b = (-1, 1, 0)$ and $c = (0, -1, 1)$. Let X be the gathering $X_{a,b,c}^{a,b,c}$ and let $\mathbf{x} = \langle 0, 0, 0 \rangle$, $\mathbf{y} = \langle 0, \frac{1}{4}, \frac{1}{4} \rangle$. Let $w = abc bac abc \in L(X)$. Notice that w can be split as $abc \underline{bac} abc$ where the first part is its first-appearance record, the middle part is its center and the last part is its last-appearance record. Corresponding to this split, we also saw the following three runs in Eq. (2)

$$\langle 0, 0, 0 \rangle \xrightarrow{\frac{1}{4}a} \left\langle \frac{1}{4}, 0, 0 \right\rangle \xrightarrow{\frac{1}{8}b} \left\langle \frac{1}{8}, \frac{1}{8}, 0 \right\rangle \xrightarrow{\frac{1}{16}c} \left\langle \frac{1}{8}, \frac{1}{16}, \frac{1}{16} \right\rangle \quad (4)$$

$$\left\langle \frac{1}{8}, \frac{1}{16}, \frac{1}{16} \right\rangle \xrightarrow{\frac{1}{16}b} \left\langle \frac{1}{16}, \frac{1}{8}, \frac{1}{16} \right\rangle \xrightarrow{\frac{1}{8}a} \left\langle \frac{3}{16}, \frac{1}{8}, \frac{1}{16} \right\rangle \xrightarrow{\frac{1}{16}c} \left\langle \frac{3}{16}, \frac{1}{16}, \frac{1}{8} \right\rangle \quad (5)$$

$$\left\langle \frac{3}{16}, \frac{1}{16}, \frac{1}{8} \right\rangle \xrightarrow{\frac{1}{8}a} \left\langle \frac{5}{16}, \frac{1}{16}, \frac{1}{8} \right\rangle \xrightarrow{\frac{5}{16}b} \left\langle 0, \frac{3}{8}, \frac{1}{8} \right\rangle \xrightarrow{\frac{1}{8}c} \left\langle 0, \frac{1}{4}, \frac{1}{4} \right\rangle \quad (6)$$

Notice that if we set $\mathbf{x}' = \langle \frac{1}{8}, \frac{1}{16}, \frac{1}{16} \rangle$ and $\mathbf{y}' = \langle \frac{3}{16}, \frac{1}{16}, \frac{1}{8} \rangle$, then this word w and these three runs together satisfy the properties mentioned in the proof of Theorem 6.4. Hence, according to this theorem, any word in $(w)_X^\uparrow$ must also be fireable from \mathbf{x} to reach \mathbf{y} .

Let $w' = \underline{abc} \underline{bbac} \underline{acbc} \in (w)_X^\uparrow$. To show that Theorem 6.4 applies to w' , we must now construct firing fractions α', β' and γ' such that

$$\langle 0, 0, 0 \rangle \xrightarrow{\alpha' abc} \left\langle \frac{1}{8}, \frac{1}{16}, \frac{1}{16} \right\rangle \xrightarrow{\beta' bbac} \left\langle \frac{3}{16}, \frac{1}{16}, \frac{1}{8} \right\rangle \xrightarrow{\gamma' acbc} \left\langle 0, \frac{1}{4}, \frac{1}{4} \right\rangle$$

In the proof of Theorem 6.4, we get these three runs for w' by modifying the corresponding runs for w . We first see how to construct α' by modifying the run in Eq. (4). First, we inspect Eq. (4) and pick a small enough $\epsilon > 0$ so that we can fire the modified run $(\frac{1}{4} - \epsilon)a, \frac{1}{8}b, \frac{1}{16}c$ from $\langle 0, 0, 0 \rangle$ without any counter values becoming negative. It is always possible to find such an ϵ because this run satisfies the property that once a counter becomes non-zero, it stays non-zero throughout. (In this case, it suffices to take $\epsilon = \frac{1}{16}$.) Then, we plug back the remaining ϵa portion into this modified run to get the same effect as the original run. Concretely speaking we get the following run.

$$\langle 0, 0, 0 \rangle \xrightarrow{\frac{3}{16}a} \left\langle \frac{3}{16}, 0, 0 \right\rangle \xrightarrow{\frac{1}{8}b} \left\langle \frac{1}{16}, \frac{1}{8}, 0 \right\rangle \xrightarrow{\frac{1}{16}a} \left\langle \frac{1}{8}, \frac{1}{8}, 0 \right\rangle \xrightarrow{\frac{1}{16}c} \left\langle \frac{1}{8}, \frac{1}{16}, \frac{1}{16} \right\rangle$$

Doing exactly the same procedure, we can modify Eq. (5) to get the following run.

$$\left\langle \frac{1}{8}, \frac{1}{16}, \frac{1}{16} \right\rangle \xrightarrow{\frac{1}{32}b} \left\langle \frac{3}{32}, \frac{3}{32}, \frac{1}{16} \right\rangle \xrightarrow{\frac{1}{32}b} \left\langle \frac{1}{16}, \frac{1}{8}, \frac{1}{16} \right\rangle \xrightarrow{\frac{1}{8}a} \left\langle \frac{3}{16}, \frac{1}{8}, \frac{1}{16} \right\rangle \xrightarrow{\frac{1}{16}c} \left\langle \frac{3}{16}, \frac{1}{16}, \frac{1}{8} \right\rangle$$

Finally, we do the same procedure, but in reverse for the last run. More precisely, we pick $\epsilon > 0$ so that it is possible to fire the modified run $\frac{1}{8}a, \frac{5}{16}b, (\frac{1}{8} - \epsilon)c$ to reach $\langle 0, \frac{1}{4}, \frac{1}{4} \rangle$ without any counter values becoming negative. It is always possible to take such an $\epsilon > 0$ because this run satisfies the property that once a counter becomes zero, it stays zero throughout. (In this case, we see that it suffices to take $\epsilon = \frac{1}{32}$). We plug the remaining ϵc portion into this modified run to get the following

$$\left\langle \frac{3}{16}, \frac{1}{16}, \frac{1}{8} \right\rangle \xrightarrow{\frac{1}{8}a} \left\langle \frac{5}{16}, \frac{1}{16}, \frac{1}{8} \right\rangle \xrightarrow{\frac{1}{32}c} \left\langle \frac{5}{16}, \frac{1}{32}, \frac{5}{32} \right\rangle \xrightarrow{\frac{5}{16}b} \left\langle 0, \frac{11}{32}, \frac{5}{32} \right\rangle \xrightarrow{\frac{3}{32}c} \left\langle 0, \frac{1}{4}, \frac{1}{4} \right\rangle$$

Altogether, this shows that we have the required three runs for w' and proves that $w' \in L_{\Sigma}^{\mathbf{x}, \mathbf{y}}$.

We stress that this procedure of redistributing runs is possible only because of the special properties of the runs. For instance, suppose instead of Eq. (4), we had the following run

$$\langle 0, 0, 0 \rangle \xrightarrow{\frac{1}{4}a} \left\langle \frac{1}{4}, 0, 0 \right\rangle \xrightarrow{\frac{1}{4}b} \left\langle 0, \frac{1}{4}, 0 \right\rangle \xrightarrow{\frac{1}{8}c} \left\langle 0, \frac{1}{8}, \frac{1}{8} \right\rangle$$

Then, we cannot redistribute the fraction $1/4$ for a without either making the first counter go negative or without also redistributing the fraction $1/4$ for b . Hence it would not be possible to modify this run to also get a run for abc between $\langle 0, 0, 0 \rangle$ and $\langle 0, \frac{1}{8}, \frac{1}{8} \rangle$ by the above procedure.

We can now prove the Lifting Runs theorem for any perfect path-scheme.

PROOF OF THEOREM 6.1. Let ρ be any perfect path-scheme of the form $\rho = u_0 X_1 \dots X_n u_n$. Since it is perfect, there is a word $w \in L(\rho) \cap L_{\Sigma}^{\mathbf{x}, \mathbf{y}}$. Let $\vec{w} = (w_1, \dots, w_n)$ be a ρ -factor of w . Hence, for each $0 \leq i \leq n$, there exists $\mathbf{x}_i, \mathbf{y}_i$ with $\mathbf{x}_0 = \mathbf{x}$ and $\mathbf{y}_n = \mathbf{y}$ such that

$$u_0 \in L_{\Sigma}^{\mathbf{x}_0, \mathbf{y}_0}, w_1 \in L_{\Sigma}^{\mathbf{y}_0, \mathbf{x}_1}, u_1 \in L_{\Sigma}^{\mathbf{x}_1, \mathbf{y}_1}, \dots, w_n \in L_{\Sigma}^{\mathbf{y}_{n-1}, \mathbf{x}_n}, u_n \in L_{\Sigma}^{\mathbf{x}_n, \mathbf{y}_n} \quad (7)$$

Applying Theorem 6.4 to X_1, \dots, X_n , we can compute $w'_1 \in L(X_1), \dots, w'_n \in L(X_n)$, such that

$$(w'_1)_{X_1}^{\uparrow} \subseteq L(X_1) \cap L_{\Sigma}^{\mathbf{y}_0, \mathbf{x}_1}, (w'_2)_{X_2}^{\uparrow} \subseteq L(X_2) \cap L_{\Sigma}^{\mathbf{y}_1, \mathbf{x}_2}, \dots, (w'_n)_{X_n}^{\uparrow} \subseteq L(X_n) \cap L_{\Sigma}^{\mathbf{y}_{n-1}, \mathbf{x}_n} \quad (8)$$

Let $w' = u_0 w'_1 u_1 w'_2 \dots w'_n u_n$ with the ρ -factor $\vec{w}' = (w'_1, w'_2, \dots, w'_n)$. By definition of $(\vec{w}')_{\rho}^{\uparrow}$ and Eq. (7) and Eq. (8) it follows that $(\vec{w}')_{\rho}^{\uparrow} \subseteq L(\rho) \cap L_{\Sigma}^{\mathbf{x}, \mathbf{y}}$.

The above procedure gives a way to compute w' when we know $\mathbf{y}_0, \mathbf{x}_1, \mathbf{y}_1, \dots, \mathbf{y}_{n-1}, \mathbf{x}_n$. It remains to compute $\mathbf{y}_0, \mathbf{x}_1, \mathbf{y}_1, \dots, \mathbf{y}_{n-1}, \mathbf{x}_n$. We can use an enumerate-and-check strategy: We enumerate $(2n)$ -tuples of rational vectors and then check for each $0 \leq i \leq n$, whether $u_i \in L_{\Sigma}^{\mathbf{x}_i, \mathbf{y}_i}$ and $L(X_i) \cap L_{\Sigma}^{\mathbf{y}_{i-1}, \mathbf{x}_i} \neq \emptyset$. The former asks if a given word is in the intersection of a regular language and a CVAS language. The latter asks if the intersection of a regular language and a CVAS language is non-empty. Since both these problems are decidable (even in NP) by Proposition 4.4, it follows that we can decide whether the current tuple of vectors satisfies the required conditions. Since ρ is a perfect path-scheme, we are guaranteed that one of our guesses will succeed and so our algorithm will terminate at some point and compute the required sequence of vectors. This proves Theorem 6.1. \square

6.2 Proof of the Second Ingredient: Path-Scheme Theorem for ρ -Upward Closures

In this subsection, we will prove our second ingredient, namely the Path-Scheme Theorem for ρ -Upward Closures (Theorem 6.2). To prove this theorem, we first need some definitions, which

we state here. Let $X = X_{a_1 \dots a_n}^{b_1 \dots b_n}$ be a gathering and let $A = \{a_1, \dots, a_n\}$. We let $\text{flat}(X)$ be the path-scheme denoted by

$$a_1 X_{\{a_1\}} a_2 X_{\{a_1, a_2\}} \dots a_n X_A b_1 X_{A \setminus \{b_1\}} b_2 X_{A \setminus \{b_1, b_2\}} \dots X_{A \setminus \{b_1, \dots, b_{n-1}\}} b_n$$

Note that $\text{flat}(X)$ is a path-scheme that, by definition, accepts the same set of words as X , i.e., $L(\text{flat}(X)) = L(X)$. Furthermore, if $w \in L(\text{flat}(X))$ and $\vec{w} = (w_1, \dots, w_{2n-1})$ is a $\text{flat}(X)$ -factor of w , it is easily seen that the part w_n in \vec{w} corresponds to the bubble X_A and is exactly equal to the center of w . For this reason, we will call the bubble X_A as the central bubble of $\text{flat}(X)$.

Finally, in our proof of this theorem, we will construct new path-schemes by substituting bubbles with path-schemes. To this end, we define a notion of substitution for path-schemes as follows: Suppose $\rho = u_0 X_1 \dots X_n u_n$ and $\rho' = v_0 Y_1 \dots Y_m v_m$ are two path-schemes. We define $\rho[\rho'/j]$ to be the path-scheme obtained by replacing the j^{th} bubble in ρ (i.e., X_j) with the path-scheme ρ' . I.e.,

$$\rho[\rho'/j] = u_0 X_1 \dots X_{j-1} u_{j-1} v_0 Y_1 \dots Y_m v_m u_j X_{j+1} \dots X_n u_n.$$

Our strategy to now prove Theorem 6.2 is the same as the one we used to prove Theorem 6.1: First, prove the theorem for gatherings. Then use this to prove the theorem for the general case.

THEOREM 6.6 (PATH-SCHEME THEOREM FOR ρ -UPWARD CLOSURES OF GATHERINGS). *Given a gathering $X = X_{a_1 \dots a_n}^{b_1 \dots b_n}$ and a word $w \in L(X)$, we can compute a path-scheme ρ such that $L(\sigma) = (w)_X^\uparrow$.*

PROOF. By the definition of $\text{flat}(X)$ and ξ , it follows that σ is precisely the path-scheme given by

$$a_1 X_{\{a_1\}} a_2 X_{\{a_1, a_2\}} \dots a_n X_{A c_1} X_{A c_2} \dots X_{A c_m} X_A b_1 X_{A \setminus \{b_1\}} b_2 X_{A \setminus \{b_1, b_2\}} \dots X_{A \setminus \{b_1, \dots, b_{n-1}\}} b_n$$

It follows that any word w' belongs to $L(\sigma)$ iff w' can be written as

$$w' = a_1 w'_1 a_2 w'_2 \dots a_n w'' b_1 w'_n \dots w'_{2n-2} b_n$$

where

- For each $i \in [1, n-1]$, $w'_i \in L(X_{\{a_1, \dots, a_i\}}) = \{a_1, \dots, a_i\}^*$,
- $w'' \in L(X_{A c_1} X_{A c_2} \dots X_{A c_m} X_A) = L(\xi) \subseteq L(X_A)$,
- And for each $i \in [n, 2n-2]$, $w'_i \in L(X_{A \setminus \{b_1, \dots, b_{i-n+1}\}}) = (A \setminus \{b_1, \dots, b_{i-n+1}\})^*$.

Hence, $w' \in L(\sigma)$ iff $w' \in L(X)$ and $\text{center}(w') \in L(\xi)$. Since a word is in $L(\xi)$ iff it is larger than or equal to $\text{center}(w)$ with respect to the subword ordering, it follows that $w' \in L(\sigma)$ iff $w' \in L(X)$ and $\text{center}(w) \leq \text{center}(w')$. The latter is the definition of $(w)_X^\uparrow$ and so the proof is complete. \square

We can now prove the Path-Scheme Theorem for ρ -Upward Closures.

PROOF OF THEOREM 6.2. Let $\rho = u_0 X_1 u_1 \dots X_n u_n$ be a perfect path-scheme and take some ρ -factor $\vec{w} = (w_1, w_2, \dots, w_n)$. By Theorem 6.6, for each X_i and w_i , we can compute a path-scheme σ_i such that $L(\sigma_i) = (w_i)_{X_i}^\uparrow$. By definition of $(\vec{w})_\rho^\uparrow$ it then follows that $(\vec{w})_\rho^\uparrow = L(\sigma)$ where σ is given by $u_0 \sigma_1 u_1 \sigma_2 \dots \sigma_n u_n$. This completes the proof of Theorem 6.2. \square

6.3 Proof of the Third Ingredient: Path-Scheme Theorem for Complements

In this subsection, we prove our third ingredient, the Path-Scheme Theorem for Complements (Theorem 6.3). The strategy is similar to that of Theorem 6.2: First, prove the theorem for the special case of gatherings and then prove the general case. We begin with the case of gatherings.

THEOREM 6.7 (PATH-SCHEME THEOREM FOR COMPLEMENT OVER GATHERINGS). *Given a gathering $X = X_{a_1 \dots a_n}^{b_1 \dots b_n}$ and a word $w \in L(X)$, we can compute a finite set of path-schemes ρ_1, \dots, ρ_m such that $L(X) \setminus (w)_X^\uparrow = \bigcup_i L(\rho_i)$ and $W(\rho_i) <_{\text{lex}} W(X)$ for all i .*

PROOF SKETCH. Let $\text{center}(w) = c_1 \dots c_m$ and $A = \{a_1, \dots, a_n\}$. For any $a \in A$, let $\bar{a} = A \setminus \{a\}$. Note that a word is not larger than or equal to $\text{center}(w)$ (with respect to the subword ordering) iff the largest prefix of $\text{center}(w)$ that is a subword is strictly smaller than $\text{center}(w)$.

Hence, for every $i \in [1, m]$, we let σ_i be the path-scheme given by $X_{\bar{c}_1}c_1X_{\bar{c}_2}c_2 \dots X_{\bar{c}_i}$. By the above, $\bigcup_{i \in [1, m]} L(\sigma_i)$ is exactly the set of all words that are not larger than or equal to $\text{center}(w)$.

Now the set of words in $L(X) \setminus (w)_X^\uparrow$ are precisely those words in $L(X)$ whose center component belongs to $\bigcup_{i \in [1, m]} L(\sigma_i)$. Recall that $\text{flat}(X)$ is a path-scheme such that $L(\text{flat}(X)) = L(X)$. Furthermore, the central bubble of $\text{flat}(X)$ captures precisely the center of all words in $L(X)$. Hence, if we replace the central bubble of $\text{flat}(X)$ with $\sigma_1, \dots, \sigma_m$ to get path-schemes ρ_1, \dots, ρ_m it will follow that $L(X) \setminus (w)_X^\uparrow = \bigcup_{i \in [1, m]} L(\rho_i)$. Since the central bubble of $\text{flat}(X)$ is its n^{th} component, it then suffices to set each ρ_i to be $\rho_i = \text{flat}(X)[\sigma_i/n]$. The formal proof behind the correctness of the equality $L(X) \setminus (w)_X^\uparrow = \bigcup_{i \in [1, m]} L(\rho_i)$ can be found in the full version of this paper.

For any i , notice that any bubble of ρ_i is either a bubble of $\text{flat}(X)$ that is not the central bubble or a bubble of σ_i . By construction, in either case, it follows that, each bubble of ρ_i is actually a star of the form X_S with $|S| < |A|$. It follows then that the weight of each bubble of ρ_i has only 0 in the j^{th} component for any $j \geq |A|$. Hence, the overall weight of ρ_i also has only 0 in the j^{th} component for any $j \geq |A|$. This implies that the weight of each ρ_i is lexicographically strictly smaller than the weight of X (which has a 1 in the $|A|^{\text{th}}$ component). This then completes the proof. \square

We can now prove the Path-Schemes Theorem for Complements.

PROOF OF THEOREM 6.3. Let $\rho = u_0X_1u_1 \dots X_nu_n$ be a perfect path-scheme and let \vec{w} be a ρ -factor given by $\vec{w} = (w_1, \dots, w_n)$. By Theorem 6.7, for each X_i and w_i , we can compute a finite number of path-schemes $\sigma_i^1, \dots, \sigma_i^{m_i}$ such that $\bigcup_{j \in [1, m_i]} L(\sigma_i^j) = L(X_i) \setminus (w_i)_{X_i}^\uparrow$ and $W(\sigma_i^j) <_{\text{lex}} W(X_i)$. Now, for each $i \in [1, n]$ and each $j \in [1, m_i]$, we can create a path-scheme $\rho_i^j = \rho[\sigma_i^j/i]$.

First, we claim that if $w' \in L(\rho) \setminus (\vec{w})_\rho^\uparrow$, then $w' \in L(\rho_i^j)$ for some i, j . Indeed, if $w' \in L(\rho) \setminus (\vec{w})_\rho^\uparrow$ then for any ρ -factor $\vec{w}' = (w'_1, \dots, w'_n)$ there must be some i such that $\text{center}(w_i) \not\leq \text{center}(w'_i)$. Hence, $w'_i \in L(X_i) \setminus (w_i)_{X_i}^\uparrow$ and so $w'_i \in L(\sigma_i^j)$ for some j . It then follows that $w' \in L(\rho_i^j)$. Hence, we can conclude that $L(\rho) \setminus (\vec{w})_\rho^\uparrow \subseteq \bigcup_{i,j} L(\rho_i^j)$.

Second, since each ρ_i^j was obtained from ρ by substituting X_i with σ_i^j and since $L(\sigma_i^j) \subseteq L(X_i)$, it follows that $L(\rho_i^j) \subseteq L(\rho)$. Hence we can conclude that $\bigcup_{i,j} L(\rho_i^j) \subseteq L(\rho)$.

Finally for each i, j we have that $W(\rho_i^j) = W(\rho) - W(X_i) + W(\sigma_i^j)$. Since $W(\sigma_i^j) <_{\text{lex}} W(X_i)$, it follows that $W(\rho_i^j) <_{\text{lex}} W(\rho)$. This completes the proof of Theorem 6.3. \square

7 Comparison with the KLM Decomposition

In the previous three sections, we have presented our effective regularity proof for CVAS languages. Our construction bears a striking similarity with the KLM-decomposition⁶ from the theory of (discrete) vector addition systems [43, 44, 52]. Since we find it surprising that a KLM-like decomposition (i) can be used for a regularity proof and (ii) applies to the continuous setting, we use this section to point out similarities and differences between our construction and the KLM-decomposition. *Readers who only want to understand the new proofs may safely skip this section.*

Vector Addition Systems. Intuitively, a vector addition system is the same as a CVAS, except that the firing fraction at each step must be exactly equal to 1. Formally, a *vector addition system* (VAS) is

⁶The authors of this work discovered the similarity only after obtaining the results of this paper. However, some structural aspects and some terms in the construction (such as “perfect” for path-schemes) have been chosen after this discovery to match the KLM-decomposition.

syntactically the same as a CVAS, and is given by a finite set of transitions $\Sigma \subset \mathbb{Z}^d$. A configuration of a VAS is a vector in \mathbb{N}^d . A step from a configuration \mathbf{x} to \mathbf{y} by means of a transition t is possible if and only if $\mathbf{y} = \mathbf{x} + t$. Since each configuration must be an element in \mathbb{N}^d , a step is only possible if no component of the configuration becomes negative. Similar to CVAS, we can define runs and reachability in a VAS. The *reachability problem for VAS* is to decide, given a set of transitions Σ and two configurations \mathbf{x} and \mathbf{y} , whether \mathbf{x} can reach \mathbf{y} .

A landmark result in the theory of VAS is that reachability is decidable and in fact, Ackermann-complete [16, 46, 48]. The first known algorithm for reachability in VAS is the so-called *KLM-decomposition* (named after three key contributors Kosaraju [43], Lambert [44], and Mayr [52]).

Why is the Similarity Surprising? At first glance, it might seem natural that ideas for VAS also apply to CVAS. However, compared to other VAS overapproximations like \mathbb{Z} -VASS (where the counters can become negative), it is not obvious that a CVAS can be translated into a VAS with the same language. For example, consider the following decision problem: Given a VAS, configurations $\mathbf{x}, \mathbf{y} \in \mathbb{N}^d$ and a number ℓ (represented in binary), decide whether there is a run of length ℓ from \mathbf{x} to \mathbf{y} . In the VAS setting, this problem is PSPACE-complete (since all intermediate counter values are at most exponential). However, for CVAS, this problem is NEXP-complete [10, Theorems 1.1 and 6.1]. Hence, unless PSPACE = NEXP, one cannot translate a CVAS into a language-equivalent VAS in polynomial time. Of course, Theorem 2.2 implies that CVAS languages are also VAS languages (since the former are regular), but we are not aware of any proof of this fact besides Theorem 2.2.

An Overview of KLM. Roughly speaking, the KLM-decomposition works as follows (we adopt the terminology of Leroux and Schmitz [47]). It maintains a list of structures called *marked witness graph sequences* (MWGS). Each MWGS describes a set of *pre-runs*, which intuitively are sequences of transitions that are not necessarily valid VAS runs. In the beginning, the list consists of a single MWGS—essentially the VAS itself.

A crucial notion for MWGSes is that of a “perfect MWGS”. A key insight in the KLM-decomposition is that a perfect MWGS has a valid run among its pre-runs. Moreover, whether an MWGS is perfect can be decided using simpler algorithmic tools: (i) a coverability check, and (ii) feasibility of a system of integer linear inequalities.

The KLM algorithm first checks perfectness of each of the current MWGS. If one of them is perfect, it concludes the existence of a valid run. On the other hand, for every MWGS ξ that is *not* perfect, the algorithm can decompose ξ into finitely many MWGS that are smaller w.r.t. some lexicographical ordering. This implies that after finitely many steps, all remaining MWGS (if any) must be perfect. Hence, if at the very end, some perfect MWGS remains, we can conclude the existence of a run; otherwise, we can conclude that no run exists.

Similarities. Let us first highlight similarities between our construction and the KLM-decomposition. Our construction follows a similar overall strategy: While KLM decomposes MWGS, our algorithm decomposes path-schemes. Moreover perfectness in MWGS has a correspondence in perfectness of path-schemes, since the perfectness condition in Section 3.5 consists of a (i) “coverability part” and (ii) a “linear inequalities” part. Indeed, recall that a path-scheme ρ is perfect if it is pre-perfect and $L(\rho) \cap L_{\Sigma}^{\mathbf{x}, \mathbf{y}} \neq \emptyset$. First, pre-perfectness is closely related to coverability (of the zero vector) in CVAS along a run over a given word $w \in \Sigma^*$. This is because whether some word $w \in \Sigma^*$ can cover the zero vector (starting from a given initial vector) depends only on the order of first appearances of letters in w [10, Lemma 3.10]. Similarly, last appearances determine “backwards coverability”. Furthermore, the proof of Proposition 4.4 (specifically, the NP upper bound in [13, Thm 4.9]) is based on expressing $L(\rho) \cap L_{\Sigma}^{\mathbf{x}, \mathbf{y}} \neq \emptyset$ as solvability of systems of linear inequalities.

Another similarity is that KLM-based proofs have a step that constructs a run (or a subset of runs) inside a perfect MWGS. Examples include Lambert’s Iteration Lemma [44, Lemma 4.1] or Leroux and Schmitz’s discovery that the pre-runs of perfect MWGS actually form an *adherent ideal* of the set of all pre-runs below valid runs [47, Lemma VII.2, Theorem VII.5]. In our proof, the construction of a regular subset R_ρ of $L_\Sigma^{x,y}$ in Theorem 6.4, which is based on [13, Proposition 4.5], is similar (in spirit and in technique) to these steps in KLM-based proofs.

Differences. Let us now point out differences between our proof and existing variants of the KLM-decomposition.

First, our decomposition does not stop when all path-schemes are perfect. Instead, it is possible that the regular sets R_ρ do not cover the entire set $L_\Sigma^{x,y}$: Classical KLM-decompositions only construct over- and/or underapproximations of the set of runs⁷, whereas we need to find an exact representation. This necessitates the *second decomposition step* (Theorem 4.3): In addition to constructing regular sets $R_\rho \subseteq L_\Sigma^{x,y}$ of perfect path-schemes, we then need to decompose the perfect path-schemes further into path-schemes that cover those runs that have not been caught by the already-built sets R_ρ .

Second, the steps to achieve perfectness of path-schemes (Theorem 4.2) are very different: In KLM, one decomposes by bounding counters or by bounding the number of transition occurrences (which is anyway not possible in the continuous semantics). Instead, our construction relies on a word-combinatorial observation: Lemma 5.1.

Third, in order for both decomposition steps to work together (and ensure termination), our proof requires the novel notion of gatherings, which is built so that it (i) yields subsets of the CVAS language (Theorem 6.4) and (ii) can be achieved by decomposing arbitrary path-schemes (Theorem 4.2). Note that gatherings are different from just the set of words with a particular first- and last appearance record (as in [10, Lemma 3.10]): In a gathering, it is important that all first appearances come before all last appearances.

8 Lower Bound

In this section, we prove Theorem 2.4. That is, we construct a CVAS $\Sigma_h \subset \mathbb{Z}^{5h}$ of size $O(h)$ and configurations $\mathbf{x}_n, \mathbf{y}_n \in \mathbb{Q}_+^{5h}$ of size $\leq n$ such that any NFA for $L_{\Sigma_h}^{\mathbf{x}_n, \mathbf{y}_n}$ has at least $\exp_h(n)$ states.

Let us first give an overview of the constructed CVAS. We will describe Σ_h implicitly by constructing transition sequences $w_1, \dots, w_h, r_1, \dots, r_h$, and u over the transition set Σ_h . Thus, Σ_h will just consist of those transitions occurring in w_1, \dots, w_h (which have 6 transitions each), r_1, \dots, r_h (which have one transition each), and u (which has $2h$ transitions). We will be interested in transition sequences of the form that are “short” in the sense of Definition 8.1.

$$w_1^{\ell_1} r_1 w_2^{\ell_2} r_2 \cdots w_h^{\ell_h} r_h u, \quad (9)$$

Definition 8.1 (Short run). A run as in (9) is called *short* if $\ell_1 \leq n$ and $\ell_{i+1} \leq \ell_i \cdot 2^{\ell_i}$ for $i \in [1, h-1]$.

In fact, our construction will ensure that there is only one short run from \mathbf{x}_n to \mathbf{y}_n of the form in (9) and that this run will “max out” the bounds above.

Definition 8.2 (Maxed out run). A short run as in (9) is called *maxed out* if $\ell_1 = n$ and $\ell_{i+1} = \ell_i \cdot 2^{\ell_i}$ for each $i \in [1, h-1]$.

Once we define all details of $\Sigma_h, \mathbf{x}_n, \mathbf{y}_n$, we will show:

⁷Examples of overapproximations are semilinear overapproximations (which are implicit in the various KLM-based algorithms for reachability, including the new algorithm for PVASS by Guttenberg, Keskin, and Meyer [30]), regular overapproximations [15, 32], or integer VASS overapproximations [40]. Underapproximations are given, e.g. by the Iteration Lemma [44, Lemma 4.1] and related facts, see above.

LEMMA 8.3. *The maxed out run leads from \mathbf{x}_n to \mathbf{y}_n . Moreover, the only short run of the form in (9) from \mathbf{x}_n to \mathbf{y}_n is the maxed out run.*

In fact, Lemma 8.3 suffices to obtain the stated lower bound:

LEMMA 8.4. *Every NFA for $L_{\Sigma_h}^{\mathbf{x}_n, \mathbf{y}_n}$ has at least $\exp_h(n)$ states.*

PROOF. Consider an NFA for $L_{\Sigma_h}^{\mathbf{x}_n, \mathbf{y}_n}$ with s states. Since the maxed out run leads \mathbf{x}_n to \mathbf{y}_n , the NFA accepts $w_1^{\ell_1} r_1 \cdots w_h^{\ell_h} r_h u$ with $\ell_1 = n$ and $\ell_{i+1} = \ell_i \cdot 2^{\ell_i}$ for $i \in [1, h-1]$. If $s < \ell_h$, then in this run, we can remove some infix w_h^r with $r > 0$. This shorter word implies the existence of a short run of the form in (9) that is not maxed out, contradicting Lemma 8.3. Hence, we have $s \geq \ell_h \geq \exp_h(n)$. \square

Counters and Configurations. Let us now describe the $5h$ counters and the initial and final configurations \mathbf{x}_n and \mathbf{y}_n . Our CVAS will have counters

$$\underbrace{x_1, \dots, x_h, y_1, \dots, y_h}_{\text{high-precision counters}}, \underbrace{\bar{x}_1, \dots, \bar{x}_h, \bar{y}_1, \dots, \bar{y}_h}_{\text{complement counters}}, \underbrace{\text{step}_1, \dots, \text{step}_h}_{\text{step counters}}. \quad (10)$$

Here, $\bar{x}_1, \dots, \bar{x}_h, \bar{y}_1, \dots, \bar{y}_h$ will act as ‘‘complement counters’’ to $x_1, \dots, x_h, y_1, \dots, y_h$ (they carry complementary values). Furthermore, the $x_i, \bar{x}_i, y_i, \bar{y}_i$ for $i = 1, \dots, h$ are ‘‘high precision counters’’, since they will carry numbers of non-elementary precision. Finally, the ‘‘step counters’’ $\text{step}_1, \dots, \text{step}_h$ will reflect the length of the run, from which we impose lower bounds on the run length.

The initial and final configurations, \mathbf{x}_n and \mathbf{y}_n , are defined as follows. Let \mathbf{x}_n be the configuration where $x_1, \dots, x_h, y_1, \dots, y_h$ all carry $\frac{1}{n}$, and all other counters carry 0. Furthermore, let \mathbf{y}_n be the configuration where each step_i carries 4 and all other counters carry 0.

Shape of Maxed Out Runs. Initially, in \mathbf{x}_n , the counters $x_1, \dots, x_h, y_1, \dots, y_h$ will all be $\frac{1}{n}$. In the maxed out run, the $w_1^* r_1$ portion will (overall) not change x_1, y_1 , but it will turn $x_2, \dots, x_h, y_2, \dots, y_h$ into $\frac{1}{n2^n}$, with $\ell_1 = n$ iterations of w_1 . Similarly, the $w_2^* r_2$ portion of this run will not change x_1, x_2, y_1, y_2 , but it will change $\frac{1}{n2^n}$ into $\frac{1}{n2^n \cdot 2^{n \cdot 2^n}}$ in the counters $x_3, \dots, x_h, y_3, \dots, y_h$. This will be done in $w_2^{\ell_2} r_2$ with $\ell_2 = n \cdot 2^n$. Then, this pattern continues for $i = 3, \dots, h$.

More generally, we will show that from a configuration where $x_i, \dots, x_h, y_i, \dots, y_h$ all have value $\frac{1}{k}$, there is a unique run $w_i^{\ell} r_i$ with $\ell \leq k$ that arrives in a configuration where step_i is 4. More precisely, an (i, k) -short run is one with

- (1) a transition sequence $w_i^{\ell} r_i$ with $\ell \leq k$
- (2) it starts in a configuration where $x_i, \dots, x_h, y_i, \dots, y_h$ all carry $\frac{1}{k}$, and $\bar{x}_i, \dots, \bar{x}_h, \bar{y}_i, \dots, \bar{y}_h$ all carry 0, and $\text{step}_i, \dots, \text{step}_h$ all carry 0,
- (3) it ends in a configuration where step_i is 4 and \bar{x}_i, \bar{y}_i carry 0.

We say that an (i, k) -short run is (i, k) -maxed out if

- (1) $\ell = k$
- (2) at the end of the run, the counters $x_i, \dots, x_h, y_i, \dots, y_h$ carry $\frac{1}{k2^k}$, the counters $\bar{x}_i, \dots, \bar{x}_h, \bar{y}_i, \dots, \bar{y}_h$, and $\text{step}_{i+1}, \dots, \text{step}_h$ all carry 0.

Once we define all the transitions, we will prove:

LEMMA 8.5. *For every $i \in [1, h]$ and $k \geq 0$, there is an (i, k) -maxed out run. Moreover, every (i, k) -short run is (i, k) -maxed out.*

Transition Notation. To make the effects of the transitions easily visible, we introduce some notation. Recall that our CVAS has $5h$ counters (see (10)). We use the names of a counter to indicate the effect of incrementing this counter. For example, x_i is the vector in \mathbb{Z}^{5h} that has 1 in the coordinate for counter x_i . This also means, e.g., that $x_i + \text{step}_i$ would be a transition that adds 1 to the counters x_i and step_i , but leaves all others unchanged. For example, the transition sequence u is

$$u = f_1 g_1 \cdots f_h g_h, \quad f_j = -x_j, \quad g_j = -y_j,$$

for every $j = 1, \dots, h$. Thus, if the counters $x_1, \dots, x_h, y_1, \dots, y_h$ carry values ≤ 1 , the sequence u allows us to reset them to zero.

The sequences $w_1, r_1, \dots, w_h, r_h$ will satisfy Property 8.6 below. We next show how Lemma 8.5 implies Lemma 8.3. Then, we set up $w_1, \dots, w_h, r_1, \dots, r_h$ to satisfy Lemma 8.5 and Property 8.6.

PROPERTY 8.6. *The sequences w_i and r_i have no effect on $x_j, \bar{x}_j, y_j, \bar{y}_j, \text{step}_j$ for $j < i$.*

PROOF OF LEMMA 8.3. First, by initially firing a $(1, n)$ -maxed out run, then a $(2, n2^n)$ -maxed out run, etc., and then resetting all x_j, y_j (for $j = 1, \dots, h$) using u , we obtain a maxed out run from \mathbf{x}_n to \mathbf{y}_n . Note that here, u is able to reset all x_j, y_j since their values are ≤ 1 .

For the second statement, take a short run along the transition sequence $w_1^{\ell_1} r_1 \cdots w_h^{\ell_h} r_h u$. Then the configuration reached after the prefix $w_1^{\ell_1} r_1$ must agree with \mathbf{y}_n in the counters $\text{step}_1, \bar{x}_1, \bar{y}_1$, because the latter are not touched during $w_2^{\ell_2} r_2 \cdots w_h^{\ell_h} r_h u$. Hence, the run along $w_1^{\ell_1} r_1$ is a short $(1, n)$ -run. By Lemma 8.5, the run along $w_1^{\ell_1} r_1$ is a $(1, n)$ -maxed out run. Thus, after $w_1^{\ell_1} r_1$, our run reaches a configuration where $x_1, \dots, x_h, y_1, \dots, y_h$ carry $\frac{1}{n2^n}$, the counters $\bar{x}_1, \dots, \bar{x}_h, \bar{y}_1, \dots, \bar{y}_h$ carry 0, and $\ell_1 = n$. By repeating this argument, we get that $w_i^{\ell_i} r_i$ is a (i, M_i) -short run, and hence a (i, M_i) -maxed out run, for $M_1 = n$ and $M_{i+1} = M_i 2^{M_i}$ for $i \in [1, h-1]$. In particular, we obtain $\ell_i = M_i$ for all i . Overall, this implies that our run along $w_1^{\ell_1} r_1 \cdots w_h^{\ell_h} r_h u$ is a maxed out run. \square

The Three Gadgets. When describing the transition sequences w_1, \dots, w_h and r_1, \dots, r_h , some more notation will be useful: Since \bar{x}_i is meant as a complement counter to x_i , most transitions will have opposite effect on these two counters (and likewise with \bar{y}_i and y_i). Therefore, we define $\hat{x}_i := x_i - \bar{x}_i$ and $\hat{y}_i := y_i - \bar{y}_i$. Thus, a transition with effect \hat{x}_i adds 1 to x_i and subtracts 1 from \bar{x}_i .

Let us now describe the words $w_1, \dots, w_h, r_1, \dots, r_h$, together with the (i, k) -maxed out runs. We will show afterwards that the described (i, k) -maxed out run is indeed the only (i, k) -short run. First, we have $w_i = t_{i,1} \cdots t_{i,6}$, where the first three transitions are:

$$t_{i,1} = -2\hat{x}_i - \hat{y}_i - 2 \sum_{j=i+1}^h (\hat{x}_j + \hat{y}_j), \quad t_{i,2} = \hat{x}_i + \text{step}_i, \quad t_{i,3} = -\hat{x}_i + \text{step}_i.$$

Thus, $t_{i,1}$ subtracts some value $\alpha \in (0, 1]$ from y_i , and subtracts 2α from x_i and all counters $x_{i+1}, y_{i+1}, \dots, x_h, y_h$. In our (i, k) -maxed out run, this is applied with $\alpha = \frac{1}{2k}$. This depletes the counters x_i and $x_{i+1}, y_{i+1}, \dots, x_h, y_h$, but sets y_i to $\frac{1}{2k}$. Because of opposite effects on the complement counters, this sets \bar{x}_i and $\bar{x}_{i+1}, \bar{y}_{i+1}, \dots, \bar{x}_h, \bar{y}_h$ to $\frac{1}{k}$. Moreover, \bar{y}_i becomes $\frac{1}{2k}$.

In $t_{i,2} t_{i,3}$, we move mass back and forth between x_i and \bar{x}_i , while adding the mass (twice) to step_i . In our (i, k) -maxed out run, $t_{i,2} t_{i,3}$ moves all the mass $(\frac{1}{k})$ from \bar{x}_i to x_i (in $t_{i,2}$), and then back (in $t_{i,3}$); and we add $\frac{2}{k}$ to step_i . Thus, compared to the configuration after $t_{i,1}$, we only added $\frac{2}{k}$ to step_i .

The second part of w_i , namely $t_{i,4} t_{i,5} t_{i,6}$, works as follows:

$$t_{i,4} = \sum_{j=i}^h \hat{x}_j + \hat{y}_j \quad t_{i,5} = -\hat{y}_i + \text{step}_i \quad t_{i,6} = \hat{y}_i + \text{step}_i$$

In $t_{i,4}$, we fill the counters $x_i, y_i, \dots, x_h, y_h$ back up, but only as far as the capacity of y_i (as guaranteed using the complement counter \bar{y}_i) allows: In our (i, k) -maxed out run, before $t_{i,4}$, we have y_i at $\frac{1}{2k}$, and the counters x_i and $x_{i+1}, y_{i+1}, \dots, x_h, y_h$ at 0. Thus, with $t_{i,4}$, we can increase all counters $x_i, y_i, \dots, x_h, y_h$ by $\frac{1}{2k}$. This means, we get $\frac{1}{2k}$ in x_i and in $x_{i+1}, y_{i+1}, \dots, x_h, y_h$. And we get $\frac{1}{k}$ in y_i .

Along $t_{i,5}t_{i,6}$, we then have a “back and forth” again, but now between y_i and \bar{y}_i . In our (i, k) -maxed out run, we move the entire $\frac{1}{k}$ back and forth. We thus have the same configuration as after $t_{i,4}$, but step $_i$ now carries $\frac{2}{k} + \frac{2}{k} = \frac{4}{k}$.

We have now described our (i, k) -maxed out run along the first copy of w_i . Note that overall, it had the same effect on x_i and on $x_{i+1}, y_{i+1}, \dots, x_h, y_h$: It replaced the value $\frac{1}{k}$ with the value $\frac{1}{2k}$. Moreover, the run did not change y_i . And it added $\frac{4}{k}$ to step $_i$. To describe the later iterations of w_i , we use a simpler notation for configurations: The abovementioned configuration is written as

$$\begin{array}{c} \text{content of } x_i \text{ and of } x_{i+1}, y_{i+1}, \dots, x_h, y_h \quad \text{content of } y_i \\ \downarrow \qquad \qquad \qquad \downarrow \\ \left\langle \frac{1}{2k}; \frac{1}{k}; \frac{4}{k} \right\rangle \\ \uparrow \\ \text{content of step}_i \end{array} \quad (11)$$

(Here, we use semicolons instead of commas to distinguish this representation from a three-dimensional configuration.) In the j -th iterations of w_i for $j \geq 1$, our (i, k) -maxed out run will perform the following transformation:

$$\left\langle \frac{1}{2^j k}; \frac{1}{k}; \frac{4j}{k} \right\rangle \xrightarrow{t_{i,1}} \left\langle 0; \frac{1}{k} - \frac{1}{2^{j+1}k}; \frac{4j}{k} \right\rangle \xrightarrow{t_{i,2}t_{i,3}} \left\langle 0; \frac{1}{k} - \frac{1}{2^{j+1}k}; \frac{4j+2}{k} \right\rangle \xrightarrow{t_{i,4}t_{i,5}t_{i,6}} \left\langle \frac{1}{2^{j+1}k}; \frac{1}{k}; \frac{4(j+1)}{k} \right\rangle \quad (12)$$

and overall we went from $\langle \frac{1}{2^j k}; \frac{1}{k}; \frac{4j}{k} \rangle$ to $\langle \frac{1}{2^{j+1}k}; \frac{1}{k}; \frac{4(j+1)}{k} \rangle$. Thus, after w_i^k , our (i, k) -maxed out run arrives in $\langle \frac{1}{2^k k}; \frac{1}{k}; 4 \rangle$. After executing w_i^k , our (i, k) -maxed out run has the reset transition r_i :

$$r_i = -\bar{x}_i - \sum_{j=i+1}^h (\bar{x}_j + \bar{y}_j).$$

This allows us to reset all the complement counters to 0, ready for the next stage of w_{i+1} . This completes our description of $w_1, r_1, \dots, w_h, r_h$ and of our (i, k) -maxed out run. Moreover, Property 8.6 is clearly satisfied. For Lemma 8.5, it remains to show that this run is the only (i, k) -short run.

PROOF OF LEMMA 8.5. Consider an (i, k) -short run along $w_i^\ell r_i$. The transitions in w_i keep constant both (i) the sum of x_i and \bar{x}_i and (ii) the sum of y_i and \bar{y}_i . These two sums remain $\frac{1}{k}$ throughout the execution of w_i^ℓ . This means, the transitions $t_{i,2}, t_{i,3}, t_{i,5}$, and $t_{i,6}$ can use a firing fraction of at most $\frac{1}{k}$. We call these the “step $_i$ -transitions”, because they modify step $_i$. Hence, if we denote by s the sum of all firing fractions of step $_i$ -transitions, then $s \leq \ell \cdot 4 \cdot \frac{1}{k}$.

Moreover, s is the final value in step $_i$, which is 4. Hence $4 = s \leq \ell \cdot 4 \cdot \frac{1}{k}$. Since $\ell \leq k$, this is only possible if $\ell = k$ and all fractions of step $_i$ -transitions are $\frac{1}{k}$. But, to fire $t_{i,2}$ with fraction $\frac{1}{k}$, the $t_{i,1}$ directly before must leave x_i empty. Likewise, to fire $t_{i,5}$ with fraction $\frac{1}{k}$, the $t_{i,4}$ directly before must leave y_i at full capacity, i.e. $\frac{1}{k}$. In short, each $t_{i,1}$ must empty x_i and each $t_{i,4}$ must fill y_i to $\frac{1}{k}$.

By induction on j , this implies that the j -th iteration of w_i must have exactly the effect described in (12). That is, after $\ell = k$ iterations, we reach $\langle \frac{1}{2^k k}; \frac{1}{k}; 4 \rangle$. This implies that the counters \bar{x}_i and $\bar{x}_{i+1}, \bar{y}_{i+1}, \dots, \bar{x}_h, \bar{y}_h$ all carry $\frac{1}{k} - \frac{1}{2^k k}$. After $w_i^\ell = w_i^k$, the transition r_i must be fired with some fraction

$\alpha \in (0, 1]$. After this, the counter \bar{x}_i will carry $\frac{1}{k} - \frac{1}{2^k k} - \alpha$. Since \bar{x}_i is 0 in the final configuration of our (i, k) -short run, we must have $\alpha = \frac{1}{k} - \frac{1}{2^k k}$. In particular, all \bar{x}_i and $\bar{x}_{i+1}, \bar{y}_{i+1}, \dots, \bar{x}_h, \bar{y}_h$ are reset to 0 by r_i . This means, our (i, k) -short run was in fact an (i, k) -maxed out run. \square

REMARK 8.7. *The requirement of being short (by our definition of “short”) is crucial to force the non-elementary length of the run $w_1^{\ell_1} r_1 \cdots w_h^{\ell_h} r_h u$. Indeed, the proof of [13, Thm. 4.9] implies that in every CVAS with states, if there is a run between two configurations, then there is one of at most exponential length. In particular, there must be such a run from x_n to y_n of the form (9). Indeed, one can choose $\ell_i = 2^i n$ for each $i = 1, \dots, h$ (and since $2n > n$, this is not short by our definition). A detailed argument can be found in the full version of this paper.*

9 Conclusion

We have shown that the language of every CVAS is effectively regular (Theorem 2.2). This clarifies the decidability landscape of reachability in systems with continuous counters (Corollary 2.3). While the decidability aspects are now understood, we view these results as the starting point for investigating the complexity of reachability in a wide range of systems with continuous counters.

Complexity. There are many complexity aspects that remain unclear. For example, what is the exact complexity of computing NFAs for $L_\Sigma^{x,y}$? We have a non-elementary lower bound, and our construction yields an Ackermannian upper bound. It would be interesting to investigate whether there is a primitive-recursive construction.

We already know situations where computing an NFA for $L_\Sigma^{x,y}$ (which has non-elementary size) does not yield optimal complexity for reachability in systems with continuous counters. For example, in finite-state systems equipped with continuous counters, reachability is NP-complete [13, Theorem 4.14], and for pushdown systems with continuous counters, reachability is NEXP-complete [10, Theorem 1.1]. This suggests that for order- k pushdown systems with continuous counters, reachability should be k -NEXP-complete, but our results only yield an Ackermannian upper bound.

Demystification. It would also be interesting to provide a simple, abstract description of how the path-scheme decomposition in our construction relates to the CVAS. For the KLM decomposition, such a description was obtained by Leroux and Schmitz [47]. Their description—the so-called “demystification”—casts the decomposition as a computation of ideals. However, this was 30 years after the KLM decomposition was first described. Hence, this might require deep conceptual insights.


For example, we can ask if there is a counterpart to the run embedding of Jančar [38] and Leroux [45] that would make our construction an ideal decomposition. Also, since our regular language is a finite union of sets that look similar to upward closures: Is there a well-quasi ordering such that $L_\Sigma^{x,y}$ is upward closed? A candidate for these two questions might be the DFA-defined well-quasi orderings from [60, p. 3], where the DFA is induced by a suitable pre-perfect path-scheme.

On the one hand, such a demystification would conceptually clarify our proof. On the other hand, it might also deepen our understanding of VAS: Because of the alternation of decomposition steps, our construction yields a precise description of firing sequences, whereas the KLM decomposition for VAS only provides an overapproximation (i.e. an ideal decomposition over the set of pre-runs). Perhaps the two approaches can be merged into a more general construction for VAS that lets us understand the set of runs of a VAS as a regular set, once runs are equipped with appropriate structure — just as context-free languages are yields of regular sets of trees.

Acknowledgments

We thank the anonymous reviewers for their suggested improvements. A part of the work was done when the first author was at TUM, Germany.

This research was sponsored in part by the Deutsche Forschungsgemeinschaft project [389792660 TRR 248-CPEC](#).

 Funded by the European Union (ERC, FINABIS, [101077902](#)). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

References

- [1] Parosh Aziz Abdulla, Kárlis Čerāns, Bengt Jonsson, and Yih-Kuen Tsay. 2000. Algorithmic Analysis of Programs with Well Quasi-ordered Domains. *Inf. Comput.* 160, 1-2 (2000), 109–127. [doi:10.1006/inco.1999.2843](#)
- [2] C. Aiswarya. 2020. On Network Topologies and the Decidability of Reachability Problem. In *Networked Systems - 8th International Conference, NETYS 2020, Marrakech, Morocco, June 3-5, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12129)*, Chryssis Georgiou and Rupak Majumdar (Eds.). Springer, 3–10. [doi:10.1007/978-3-030-67087-0_1](#)
- [3] Jorge Almeida and Alfredo Costa. 2021. Profinite topologies. In *Handbook of Automata Theory*, Jean Éric Pin (Ed.). European Mathematical Society, Berlin, Germany, Chapter 17. [doi:10.4171/AUTOMATA-1/17](#)
- [4] Rajeev Alur and David L Dill. 1994. A theory of timed automata. *Theoretical computer science* 126, 2 (1994), 183–235. [doi:10.1016/0304-3975\(94\)90010-8](#)
- [5] Rajeev Alur and Parthasarathy Madhusudan. 2004. Decision problems for timed automata: A survey. In *Formal Methods for the Design of Real-Time Systems*. Springer, 1–24. [doi:10.1007/978-3-540-30080-9_1](#)
- [6] Mohamed Faouzi Atig, Ahmed Bouajjani, and Shaz Qadeer. 2009. Context-Bounded Analysis for Concurrent Programs with Dynamic Creation of Threads. In *Tools and Algorithms for the Construction and Analysis of Systems, 15th International Conference, TACAS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5505)*, Stefan Kowalewski and Anna Philippou (Eds.). Springer, 107–123. [doi:10.1007/978-3-642-00768-2_11](#)
- [7] A. R. Balasubramanian. 2024. Decidability and Complexity of Decision Problems for Affine Continuous VASS. In *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2024, Tallinn, Estonia, July 8-11, 2024*, Pawel Sobocinski, Ugo Dal Lago, and Javier Esparza (Eds.). ACM, 7:1–7:13. [doi:10.1145/3661814.3662124](#)
- [8] A. R. Balasubramanian, Javier Esparza, and Mikhail A. Raskin. 2021. Finding Cut-Offs in Leaderless Rendez-Vous Protocols is Easy. In *Foundations of Software Science and Computation Structures - 24th International Conference, FOSSACS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 12650)*, Stefan Kiefer and Christine Tasson (Eds.). Springer, 42–61. [doi:10.1007/978-3-030-71995-1_3](#)
- [9] A. R. Balasubramanian, Matthew Hague, Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetsche. 2025. General Decidability Results for Systems with Continuous Counters. arXiv:2511.21559 [cs.FL] <https://arxiv.org/abs/2511.21559>
- [10] A. R. Balasubramanian, Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetsche. 2024. Reachability in Continuous Pushdown VASS. *Proc. ACM Program. Lang.* 8, POPL (2024), 90–114. [doi:10.1145/3633279](#)
- [11] Pascal Baumann, Rupak Majumdar, Ramanathan S. Thinniyam, and Georg Zetsche. 2020. The Complexity of Bounded Context Switching with Dynamic Thread Creation. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference) (LIPIcs, Vol. 168)*, Artur Czumaj, Anuj Dawar, and Emanuela Merelli (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 111:1–111:16. [doi:10.4230/LIPIcs.ICALP.2020.111](#)
- [12] Michael Blondin, Alain Finkel, Christoph Haase, and Serge Haddad. 2016. Approaching the Coverability Problem Continuously. In *Tools and Algorithms for the Construction and Analysis of Systems (Lecture Notes in Computer Science)*, Marsha Chechik and Jean-François Raskin (Eds.). Springer, Berlin, Heidelberg, 480–496. [doi:10.1007/978-3-662-49674-9_28](#)
- [13] Michael Blondin and Christoph Haase. 2017. Logics for Continuous Reachability in Petri Nets and Vector Addition Systems with States. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, Reykjavik, Iceland, 1–12. [doi:10.1109/LICS.2017.8005068](#)
- [14] Ahmed Bouajjani, Javier Esparza, and Oded Maler. 1997. Reachability Analysis of Pushdown Automata: Application to Model-Checking. In *CONCUR '97: Concurrency Theory, 8th International Conference, Warsaw, Poland, July 1-4, 1997, Proceedings (Lecture Notes in Computer Science, Vol. 1243)*, Antoni W. Mazurkiewicz and Józef Winkowski (Eds.). Springer, 135–150. [doi:10.1007/3-540-63141-0_10](#)
- [15] Wojciech Czerwinski, Piotr Hofman, and Georg Zetsche. 2018. Unboundedness Problems for Languages of Vector Addition Systems. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic (LIPIcs, Vol. 107)*, Ioannis Chatzigiannakis, Christos Kaklamani, Dániel Marx, and Donald

- Sannella (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 119:1–119:15. doi:10.4230/LIPICS.ICALP.2018.119
- [16] Wojciech Czerwiński and Łukasz Orlikowski. 2021. Reachability in Vector Addition Systems is Ackermann-complete. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*. IEEE, 1229–1240. doi:10.1109/FOCS52979.2021.00120
- [17] Werner Damm and Andreas Goerdt. 1986. An automata-theoretical characterization of the OI-hierarchy. *Information and control* 71, 1-2 (1986), 1–32. doi:10.1016/S0019-9958(86)80016-X
- [18] René David and Hassane Alla. 1987. Continuous Petri nets. In *Proc. 8th European Workshop on Appli. & Theory of Petri nets, 1987*.
- [19] René David and Hassane Alla. 2010. *Discrete, Continuous, and Hybrid Petri Nets*. Springer. doi:10.1007/978-3-642-10669-9
- [20] Jean Éric Pin. 2021. Finite Automata. In *Handbook of Automata Theory*, Jean Éric Pin (Ed.). European Mathematical Society, Berlin, Germany, Chapter 1. doi:10.4171/AUTOMATA-1/1
- [21] Javier Esparza, Alain Finkel, and Richard Mayr. 1999. On the Verification of Broadcast Protocols. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*. IEEE Computer Society, 352–359. doi:10.1109/LICS.1999.782630
- [22] Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. 2015. Verification of Population Protocols. In *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1.4, 2015 (LIPIcs, Vol. 42)*, Luca Aceto and David de Frutos-Escrig (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 470–482. doi:10.4230/LIPICS.CONCUR.2015.470
- [23] Javier Esparza, Ruslán Ledesma-Garza, Rupak Majumdar, Philipp J. Meyer, and Filip Niksic. 2014. An SMT-Based Approach to Coverability Analysis. In *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014, Proceedings (Lecture Notes in Computer Science, Vol. 8559)*, Armin Biere and Roderick Bloem (Eds.). Springer, 603–619. doi:10.1007/978-3-319-08867-9_40
- [24] Alain Finkel and Philippe Schnoebelen. 2001. Well-structured transition systems everywhere! *Theoretical Computer Science* 256, 1-2 (2001), 63–92. doi:10.1016/S0304-3975(00)00102-X
- [25] Estibaliz Fraca and Serge Haddad. 2015. Complexity Analysis of Continuous Petri Nets. *Fundam. Informaticae* 137, 1 (2015), 1–28. doi:10.3233/FI-2015-1168
- [26] Pierre Ganty and Rupak Majumdar. 2012. Algorithmic verification of asynchronous programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 34, 1 (2012), 1–48. doi:10.1145/2160910.2160915
- [27] Gilles Geeraerts, Jean-François Raskin, and Laurent Van Begin. 2007. Well-structured languages. *Acta Informatica* 44, 3-4 (2007), 249–288. doi:10.1007/S00236-007-0050-3
- [28] Steven M. German and A. Prasad Sistla. 1992. Reasoning about systems with many processes. *Journal of the ACM (JACM)* 39, 3 (1992), 675–735. doi:10.1145/146637.146681
- [29] Hermann Gruber, Jonathan Lee, and Jeffrey Shallit. 2021. Enumerating regular expressions and their languages. In *Handbook of Automata Theory*, Jean Éric Pin (Ed.). European Mathematical Society, Berlin, Germany, Chapter 13. doi:10.4171/AUTOMATA-1/13
- [30] Roland Guttenberg, Eren Keskin, and Roland Meyer. 2025. PVASS reachability is decidable. arXiv:2504.05015 [cs.LO] <https://arxiv.org/abs/2504.05015>
- [31] Stefan Haar and Juri Kolčák. 2026. Continuous Petri Nets Faithfully Fluidify Most Permissive Boolean Networks. In *Computational Methods in Systems Biology*, François Fages and Sabine Pères (Eds.). Springer Nature Switzerland, Cham, 89–105. doi:10.1007/978-3-032-01436-8_6
- [32] Peter Habermehl, Roland Meyer, and Harro Wimmel. 2010. The Downward-Closure of Petri Net Languages. In *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 6199)*, Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis (Eds.). Springer, 466–477. doi:10.1007/978-3-642-14162-1_39
- [33] Matthew Hague and Anthony Widjaja Lin. 2011. Model Checking Recursive Programs with Numeric Data Types. In *Computer Aided Verification*, Ganesh Gopalakrishnan and Shaz Qadeer (Eds.). Vol. 6806. Springer Berlin Heidelberg, Berlin, Heidelberg, 743–759. doi:10.1007/978-3-642-22110-1_60
- [34] Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. 2017. Collapsible Pushdown Automata and Recursion Schemes. *ACM Trans. Comput. Log.* 18, 3 (2017), 25:1–25:42. doi:10.1145/3091122
- [35] Monika Heiner, David Gilbert, and Robin Donaldson. 2008. Petri Nets for Systems and Synthetic Biology. In *Formal Methods for Computational Systems Biology*, Marco Bernardo, Pierpaolo Degano, and Gianluigi Zavattaro (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 215–264. doi:10.1007/978-3-540-68894-5_7
- [36] Mostafa Herajy, Monika Heiner, Anna Gambin, and Monika Heiner. 2018. Adaptive and Bio-semantics of Continuous Petri Nets: Choosing the Appropriate Interpretation. *Fundam. Inf.* 160, 1–2 (Jan. 2018), 53–80. doi:10.3233/FI-2018-1674
- [37] Branislav Hruz and MengChu Zhao. 2007. *Modeling and control of discrete-event dynamic systems*. Springer. doi:10.1007/978-1-84628-877-7

- [38] Petr Jancar. 1990. Decidability of a Temporal Logic Problem for Petri Nets. *Theor. Comput. Sci.* 74, 1 (1990), 71–93. doi:10.1016/0304-3975(90)90006-4
- [39] Addie Jordon, Juri Kolčák, and Daniel Merkle. 2025. Continuous Petri Nets for Fast Yield Computation: Polynomial-Time and MLP Approaches. arXiv:2509.02371 [cs.DM] <https://arxiv.org/abs/2509.02371>
- [40] Eren Keskin and Roland Meyer. 2024. On the Separability Problem of VASS Reachability Languages. In *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2024, Tallinn, Estonia, July 8-11, 2024*, Pawel Sobocinski, Ugo Dal Lago, and Javier Esparza (Eds.). ACM, 49:1–49:14. doi:10.1145/3661814.3662116
- [41] Naoki Kobayashi. 2019. Inclusion between the frontier language of a non-deterministic recursive program scheme and the Dyck language is undecidable. *Theoretical Computer Science* 777 (2019), 409–416. doi:10.1016/J.TCS.2018.09.035
- [42] Ina Koch, Wolfgang Reisig, and Falk Schreiber. 2011. Modeling in Systems Biology—The Petri Net Approach. *Computational Biology* 16 (2011). doi:10.1007/978-1-84996-474-6
- [43] S. Rao Kosaraju. 1982. Decidability of Reachability in Vector Addition Systems (Preliminary Version). In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber (Eds.). ACM, 267–281. doi:10.1145/800070.802201
- [44] Jean-Luc Lambert. 1992. A Structure to Decide Reachability in Petri Nets. *Theor. Comput. Sci.* 99, 1 (1992), 79–104. doi:10.1016/0304-3975(92)90173-D
- [45] Jérôme Leroux. 2011. Vector addition system reachability problem: a short self-contained proof. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, Thomas Ball and Mooly Sagiv (Eds.). ACM, 307–316. doi:10.1145/1926385.1926421
- [46] Jérôme Leroux. 2021. The Reachability Problem for Petri Nets is Not Primitive Recursive. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*. IEEE, 1241–1252. doi:10.1109/FOCS52979.2021.00121
- [47] Jérôme Leroux and Sylvain Schmitz. 2015. Demystifying Reachability in Vector Addition Systems. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*. IEEE Computer Society, 56–67. doi:10.1109/LICS.2015.16
- [48] Jérôme Leroux and Sylvain Schmitz. 2019. Reachability in Vector Addition Systems is Primitive-Recursive in Fixed Dimension. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*. IEEE, 1–13. doi:10.1109/LICS.2019.8785796
- [49] Cristian Mahulea, Laura Recalde, and Manuel Silva. 2006. On performance monotonicity and basic servers semantics of continuous Petri nets. In *2006 8th International Workshop on Discrete Event Systems*. IEEE, 345–351.
- [50] A. N. Maslov. 1974. The Hierarchy of Indexed Languages of an Arbitrary Level. *Soviet Mathematics Doklady* 15 (1974), 1170–1174.
- [51] A. N. Maslov. 1976. Multilevel Stack Automata. *Problems of Information Transmission* 12 (1976), 38–43.
- [52] Ernst W. Mayr. 1984. An Algorithm for the General Petri Net Reachability Problem. *SIAM J. Comput.* 13, 3 (1984), 441–460. doi:10.1137/0213029
- [53] Luke Ong. 2015. Higher-Order Model Checking: An Overview. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*. IEEE Computer Society, 1–15. doi:10.1109/LICS.2015.9
- [54] Jacques Sakarovitch. 2021. Automata and rational expressions. In *Handbook of Automata Theory*, Jean Éric Pin (Ed.). European Mathematical Society, Berlin, Germany, Chapter 2. doi:10.4171/AUTOMATA-1/2
- [55] Sylvain Schmitz. 2017. *Algorithmic Complexity of Well-Quasi-Orders. (Complexité algorithmique des beaux pré-ordres)*. <https://tel.archives-ouvertes.fr/tel-01663266>
- [56] Howard Straubing and Pascal Weil. 2021. Varieties. In *Handbook of Automata Theory*, Jean Éric Pin (Ed.). European Mathematical Society, Berlin, Germany, Chapter 16. doi:10.4171/AUTOMATA-1/16
- [57] Wil van der Aalst and Christian Stahl. 2011. *Modeling business processes: A Petri Net-Oriented Approach*. MIT Press.
- [58] Alex Yakovlev, Luis Gomes, and Luciano Lavagno (Eds.). 2000. *Hardware Design and Petri Nets*. Springer. doi:10.1007/978-1-4757-3143-9
- [59] Georg Zetsche. 2015. An approach to computing downward closures. arXiv:1503.01068 [cs.FL] <https://arxiv.org/abs/1503.01068>
- [60] Georg Zetsche. 2018. Separability by piecewise testable languages and downward closures beyond subwords. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, Anuj Dawar and Erich Grädel (Eds.). ACM, 929–938. doi:10.1145/3209108.3209201

Received 2025-07-10; accepted 2025-11-06