

The Σ -Chain Product: A Succinct Model of Automata (De)Composition

Roberto Borelli  

University of Padua, Italy

Davide Bresolin   

University of Padua, Italy

Luca Geatti   

University of Udine, Italy

Angelo Montanari   

University of Udine, Italy

Matteo Zavatteri   

Fondazione Bruno Kessler, Trento, Italy

Abstract

The cascade product is a fundamental construction in automata theory, enabling hierarchical composition of automata and playing a central role in decomposition results such as the Krohn–Rhodes theorem. However, its use is limited by the exponential size required to represent cascades, which stems from the fact that each component may depend on all preceding ones, leading to exponentially large alphabets.

To address this issue, we introduce the Σ -chain product, a restricted variant in which each component depends only on the input alphabet and the component immediately preceding it. We show that Σ -chains achieve linear-size representations and can be exponentially more succinct than cascades. We prove that Σ -chains and cascades are expressively equivalent even when restricting the components to specific classes of automata, such as permutation-reset automata. As a consequence, we derive that a language is regular if and only if it is recognized by a Σ -chain of permutation-reset automata. Finally, we analyze structural properties of Σ -chains of reset automata, including a relation with well-known subclasses of star-free languages.

2012 ACM Subject Classification Theory of computation $\rightarrow$ Automata extensions; Theory of computation $\rightarrow$ Modal and temporal logics; Theory of computation $\rightarrow$ Regular languages

Keywords and phrases Automata, Cascade Product, Formal Languages, Krohn-Rhodes Theory

Digital Object Identifier 10.4230/OASICS.TIME.2026.3

Related Version *Extended Version*: <https://arxiv.org/abs/2607.16884> [2]

Funding Davide Bresolin and Luca Geatti acknowledge financial support from the INdAM – GNCS Project “Neuro-symbolic semantic contract for rigorous reachability and certificates in CPS” (CUP E53C25002010001). Luca Geatti acknowledges financial support from the *Programma Operativo Complementare* of the *Regione Autonoma Friuli Venezia Giulia*, under CUP G23C25000880002, Operation Code 2025/13846, Project Code 2025/13846/16.

1 Introduction

The *cascade product* of deterministic automata generalises the direct product by arranging the component automata in a linear, hierarchical structure [12, 18, 13]. While in the direct product every component reads symbols from a fixed alphabet Σ , in the cascade product the component at level i reads symbols from Σ and, additionally, observe the state of every component at level $j < i$. Formally, the automaton at level i reads symbols from the Cartesian



© Roberto Borelli, Davide Bresolin, Luca Geatti, Angelo Montanari, and Matteo Zavatteri; licensed under Creative Commons License CC-BY 4.0

33rd International Symposium on Temporal Representation and Reasoning (TIME 2026).

Editors: AndreA Orlandini and Sophie Pinchinat; Article No. 3; pp. 3:1–3:17

OpenAccess Series in Informatics



OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

product $\Sigma \times Q_1 \times \cdots \times Q_{i-1}$, where Q_j denotes the state set of the automaton at level j ; the first component, as in the direct product, reads symbols from Σ alone. A transition labelled $(a, q_1, \dots, q_{i-1})$ is taken if and only if the current input symbol is $a \in \Sigma$ and the automaton at level j is currently in state q_j , for every $1 \leq j < i$. Crucially, the automaton obtained by applying the cascade product is defined over the original alphabet Σ alone.

The cascade product was introduced in the context of semigroup and semiautomaton decomposition theory, and lies at the heart of the Krohn–Rhodes Decomposition Theorem [12]. This theorem states that every deterministic semiautomaton (*i.e.* an automaton without initial and accepting states) can be decomposed into a sequence of permutation-reset semiautomata whose cascade product *covers* the original semiautomaton – that is, the original is a homomorphic image of a sub-semiautomaton of the product.¹ In other words, the cascade product can simulate all the behaviours of the original semiautomaton using only very simple components, namely permutation-reset semiautomata. A semiautomaton over a state set Q is called *permutation-reset* if every symbol in its alphabet induces a transformation $Q \rightarrow Q$ that is either a permutation (*i.e.* a bijection) or a reset (a map whose image has cardinality one). A semiautomaton is called *reset* if every symbol induces either a reset function or the identity (a special case of a permutation). A central insight of the Krohn–Rhodes theorem is that every counter-free semiautomaton – corresponding to star-free regular expressions – can be decomposed into a cascade product of two-state reset semiautomata.

Beyond its theoretical significance in automata theory and semigroup algebra, the Krohn–Rhodes Decomposition Theorem has found applications in the translation of automata to Linear Temporal Logic formulas [14, 13] and in the transformation of finite-word automata into formulas of pure-past Linear Temporal Logic – formulas whose temporal operators refer exclusively to the past – in what is known as the *pastification* problem [3, 1]. An application of the cascade product in formal verification appears in [7], where the approach is to decompose the specification to be model-checked against a system into a sequence of components arranged in a cascade structure, and to execute the model-checking algorithm one component at a time, starting from the first. The rationale is that component-by-component verification is generally more efficient than the monolithic verification of the full system. More recently, the cascade product has been extended from semiautomata to full automata in [3], where the relationship between cascades of a given number of reset automata and the corresponding classes of regular expressions has been studied. It has been shown that the cascade product of permutation-reset automata (respectively, of reset automata) captures exactly the class of regular (respectively, star-free) languages.

A significant drawback that limits the applicability of the cascade product is the size of its representation. On one hand, the automaton obtained by computing the cascade product of n (semi)automata $\mathcal{A}_1, \dots, \mathcal{A}_n$ has a number of states exponential in n , a phenomenon shared with the direct product. On the other hand, and more problematically, even representing the *sequence of components* $\langle \mathcal{A}_1, \dots, \mathcal{A}_n \rangle$ – which we call a *cascade* of (semi)automata – without computing the actual product requires exponential space. This is a consequence of the component alphabets: in particular, the alphabet of the last component $\mathcal{A}_n$ has cardinality $|\Sigma| \times |Q_1| \times \cdots \times |Q_{n-1}|$, which is already exponential in n . Since we consider deterministic automata, the number of transitions of $\mathcal{A}_n$ is also exponential in n . While this represents a significant drawback from the perspective of space complexity, it is also a key source of expressive power, allowing $\mathcal{A}_n$ to distinguish every possible configuration of the preceding

¹ A stronger version of the theorem exists, where the decomposition consists of permutation automata and two-state reset automata. We use the weaker formulation to keep all components of the same type.

components together with the current input symbol, and to select a different successor state for each such configuration. By contrast, representing the sequence of components of a direct product requires only space linear in n and in the size of the largest automaton.

Motivated by the desire to overcome this representational inefficiency while surpassing the expressive power of the direct product (which only captures closure under intersection), we introduce in this paper the notion of Σ -chains and the corresponding Σ -chain product. A Σ -chain is a sequence of automata in which the first component reads symbols from Σ , while each subsequent component depends only on Σ and on the immediately preceding component, that is, where the component at level i reads symbols from the Cartesian product $\Sigma \times Q_{i-1}$, where Q_{i-1} is the state set of the automaton at level $i-1$. The Σ -chain product is the resulting product automaton, defined analogously to the cascade product: the i -th automaton takes a transition labeled (a, q_{i-1}) if and only if the current input symbol is a and the preceding component is in state q_{i-1} . The Σ -chain product is thus a restriction of the cascade product in which each component's dependency is confined to Σ and the component immediately preceding it, and a generalization of the direct product in that it admits a hierarchical dependency structure among components.

We begin by analyzing the representational complexity of Σ -chains. We show that, while the Σ -chain product of n automata is an automaton with a number of states exponential in n (as with both the cascade and the direct product), the Σ -chain itself – *i.e.* the sequence of components – has size linear in n (as with the direct product, but unlike the cascade). Furthermore, we demonstrate that Σ -chains can be exponentially more succinct than cascades: there exists a family of languages recognisable by Σ -chains of n reset automata (thus representable in space polynomial in n) such that any cascade of reset automata for those languages requires at least n components, and hence at least exponential space.

We then turn to the expressive power of Σ -chains. We show that Σ -chains and cascades of automata are expressively equivalent. We first observe that every Σ -chain is, by construction, a special case of a cascade: the dependency restriction imposed on each component is simply a syntactic constraint, and lifting it yields a valid cascade with identical components. The converse direction is less immediate. We show that every cascade can be transformed into an equivalent Σ -chain via a *chainification* technique: intuitively, the dependence of a component – say, the third – on the states of a non-adjacent predecessor – say, the first – can be eliminated by encoding the relevant state information into the state space of the intervening component. We prove, moreover, that for the classes of permutation-reset, pure-reset (all symbols induce a reset), and permutation automata (all symbols induce a permutation), the resulting Σ -chain has the same number of components as the original cascade, and each component belongs to the same class. This comes at the cost that, at level i , the number of states becomes exponential in i . A corollary of this transformation is that Σ -chains of permutation-reset automata recognise exactly the class of regular languages. This means that, for permutation-reset automata, maximal expressive power can be achieved without requiring the i -th level to depend on all previous levels. Rather, it only needs to depend on the immediately preceding level and the input alphabet. For reset automata, the proposed transformation does not preserve the component type: a reset automaton may become a permutation-reset automaton. While the number of components is preserved, the resulting Σ -chain may no longer consist entirely of reset automata. Therefore, we can only establish that languages recognized by Σ -chains of reset automata are included in the star-free languages, leaving the converse open. Table 1 summarizes and compares the expressiveness results for Σ -chains against known results on cascades and direct products.

■ **Table 1** Summary of expressiveness results on direct products, Σ -chains and cascades of automata, for the classes of pure-reset ($\mathcal{R}^-$), reset ($\mathcal{R}$) and permutation-reset ($\mathcal{R}^+$) automata. SF denotes the class of star-free languages and REG denotes the class of all regular languages. [†] Some preliminary results on the expressiveness of Σ -chains of reset automata are discussed in Section 5.

	Direct Product	Σ -chains	Cascades
$\mathcal{R}^-$	$\bigcap_i (\Sigma^* R_i \cup J_i)$ $J_i = \varepsilon \text{ or } J_i = \emptyset$ $R_i, I_i \subseteq \Sigma, R_i \cap I_i = \emptyset$ [3]	$\subseteq$ Lemma 11 <i>open</i>	$=$ Corollary 14 <i>open</i>
$\mathcal{R}$	$\bigcap_i (\Sigma^* R_i I_i^* \cup J_i)$ $J_i = I_i^* \text{ or } J_i = \emptyset$ $R_i, I_i \subseteq \Sigma, R_i \cap I_i = \emptyset$ [3]	$\subseteq$ Lemma 11 <i>open</i> [†]	$\subseteq$ Corollary 14 SF [4]
$\mathcal{R}^+$	<i>open</i>	REG Corollary 14	$=$ Corollary 14 REG [4]

We next investigate Σ -chains of reset automata, showing that they can recognize several natural and well-studied classes of star-free languages, including Piecewise Testable languages [16] and R-trivial languages [5]. We also consider two hierarchies of languages obtained by restricting Σ -chains of reset automata by height (number of components) and by width (number of states per level). We prove that the height hierarchy is strict – for every h , there exist languages recognisable by a Σ -chain of reset automata of height h but not of height $h - 1$ – whereas the width hierarchy collapses: every Σ -chain of reset automata is equivalent to one in which the number of states of each component depends only on the size of the alphabet Σ . Finally, we address the pastification problem for Σ -chains of reset automata, namely the transformation of such structures into equivalent formulas in pure past Linear Temporal Logic. A straightforward approach would consist in first translating a Σ -chain into a cascade and then applying the procedure of [4] to obtain the corresponding formula. Instead, we adapt the construction of [4] so as to directly translate a Σ -chain of reset automata into a formula of quadratic size, thereby avoiding the exponential blow-up caused by the intermediate transformation from Σ -chains to cascades.

The paper is structured as follows. Related work is discussed in Section 2. In Section 3, we provide some background knowledge. In Section 4, we introduce the notion of Σ -chains and Σ -chain product of automata, we state some basic results and we discuss succinctness and expressiveness through the use of the chainification (the transformation of cascades into Σ -chains) technique. In Section 5 we focus on Σ -chains of reset automata, in particular we study height and width hierarchies, we compare Σ -chains of reset automata to piecewise-testable and R-trivial languages and we discuss an efficient pastification procedure for Σ -chains. Finally, in Section 6 we assess the work done, and we outline some directions for future works. Proofs of all results are provided in the Extended Version [2].

2 Related Work

The composition of (semi)automata has been extensively studied in automata theory and semigroup theory. In particular, the cascade product plays a central role in decomposition results such as the Krohn–Rhodes Theorem [12], which shows that every finite semiautomaton can be decomposed into a cascade of permutation-reset semiautomata. Since then, several works have investigated structural, algebraic, and logical properties of cascades of automata and semiautomata, while others have also considered restrictions of the cascade product.

In [9], Gelderie studies structural properties of cascade products and introduces the notion of scope: a cascade has scope k if each transition of each component depends on the states of at most the previous k components. Cascades of scope 1 have a strictly linear dependency structure and can be readily transformed into Σ -chains. Gelderie further shows that piecewise-testable languages are recognized by cascades of reset semiautomata of scope 1, and R-trivial languages by cascades of scope 2.

Our work differs in two ways. First, we consider full automata rather than semiautomata. Second, instead of studying cascades of bounded-scope, we introduce Σ -chains, where the i -th component depends only on the input letter and the state of component $i - 1$. We exhibit Σ -chains of reset automata for generalizations of both piecewise-testable and R-trivial languages. As a byproduct, since R-trivial languages are recognized by finite unions of Σ -chains of reset automata and cascades are closed under union, R-trivial languages are also recognized by cascades of reset semiautomata of scope 1, improving Gelderie's bound.

In [8], Gecseg surveys a wide range of results on homomorphically and isomorphically complete classes of semiautomata with respect to α_j -products [11]. Informally, the α_0 -product corresponds to a cascade product in which the dependency structure among components is acyclic, i.e., no loops are allowed between the dependencies of the automata. In [6], Domosi introduces the notion of v_i -product, where each semiautomaton is allowed to depend on at most i other semiautomata. The α_0 - v_1 product [10] combines these two restrictions: the dependency structure is acyclic and each component depends on at most one other component.

This construction is closely related to the Σ -chain product introduced in this paper. Indeed, both frameworks restrict the dependency structure among components to be essentially linear. In particular, every Σ -chain product defines an α_0 - v_1 product whose dependency graph is connected. Conversely, an α_0 - v_1 product may consist of several independent connected components; once an initial state and a set of final states are fixed, these components can be sequentially concatenated (in any order) to obtain a corresponding Σ -chain. In [10], it is shown that the α_0 - v_1 product is equivalent, in terms of homomorphic representations, to the general product, i.e., a product in which each component may depend on all the others. At first glance, this may seem to contradict the results of the present paper, since general products of reset automata can recognize languages that are not star-free, suggesting that Σ -chain of reset automata products might also exceed the expressive power of the star-free class. However, this apparent discrepancy is resolved by observing that the equivalence result of [10] relies on infinite products. Furthermore, the same work shows that, when restricting to finite products of finite automata, both the α_0 -product and the v_1 -product are strictly more expressive than the α_0 - v_1 product with respect to homomorphic representations. The expressiveness results for Σ -chain products we establish in this paper can thus be seen as an extension of the finite-product results of [10], providing a more refined understanding of the expressive power of finite α_0 - v_1 products.

3 Background

In this section, we introduce the necessary background on automata and cascades of automata.

3.1 Automata

A deterministic *semiautomaton* $\mathcal{A}$ is a tuple (Σ, Q, δ) such that: (i) Σ is the alphabet, i.e. a finite set of symbols; (ii) Q is the set of states; (iii) $\delta : Q \times \Sigma \rightarrow Q$ is the transition function. An deterministic *automaton* $\mathcal{A} = (\Sigma, Q, \delta, s_0, F)$ is a semiautomaton augmented

with an initial state s_0 and a set F of final states. From now on, the terms automaton and semiautomaton refer exclusively to deterministic ones. With δ^* we denote the Kleene closure of δ . The *size* of $\mathcal{A}$, denoted with $|\mathcal{A}|$, is defined as $|\delta|$.

Given an automaton $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ and a (finite) word $\sigma := \langle \sigma_0, \dots, \sigma_n \rangle \in \Sigma^*$, the run $\tau \in Q^+$ induced by σ is a sequence $\langle q_0, q_1, \dots, q_{n+1} \rangle$ starting from the initial state q_0 and such that $\delta(q_i, \sigma_i) = q_{i+1}$, for all $0 \leq i \leq n$. We say that τ is *accepting* iff $q_{n+1} \in F$. A word $\sigma \in \Sigma^*$ is *accepted* by $\mathcal{A}$ iff the run induced by σ is accepting. We define the *language of $\mathcal{A}$* (or *the language recognized by $\mathcal{A}$*), denoted by $\mathcal{L}(\mathcal{A})$, as the set of accepted words. Given a state $q \in Q$, let $\mathcal{L}_q(\mathcal{A})$ be the set of words inducing a run $\tau := \langle q_0, \dots, q_m \rangle$ with $q_m = q$.

Each symbol of the alphabet of an automaton $\mathcal{A}$ (with set of states Q) induces a transformation $Q \rightarrow Q$, defined as follows.

► **Definition 1** (Function induced by a symbol). *Let $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ be an automaton. For each symbol $a \in \Sigma$, we define the function induced by a in $\mathcal{A}$, denoted with $\tau_a^{\mathcal{A}}$ or simply τ_a (when $\mathcal{A}$ is clear from the context), as the function $\tau_a : Q \rightarrow Q$ such that, for all $s \in Q$, we have that $\tau_a(s) = s'$ iff $\delta(s, a) = s'$.*

Let $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ be an automaton and let $a \in \Sigma$ be a symbol in the alphabet. We say that τ_a is a *reset function* (or simply a *reset*) iff there exists a $q' \in Q$ such that, for all $q \in Q$, we have that $\tau_a(q) = q'$; in this case, we say that τ_a is a *reset on q'* . If instead τ_a is bijective, we say that τ_a is a *permutation function* (or simply a *permutation*).

Reset automata and permutation automata are defined as follows.

► **Definition 2** (The classes of Pure-reset, Reset, Permutation, and Permutation-Reset automata). *Let $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ be an automaton. $\mathcal{A}$ is called:*

- a pure-reset automaton iff, for each $a \in \Sigma$, τ_a is a reset function;
- a reset automaton iff, for each symbol $a \in \Sigma$, τ_a is either the identity function or a reset function;
- a permutation automaton iff, for each symbol $a \in \Sigma$, τ_a is a permutation function.
- a permutation-reset automaton iff, for each symbol $a \in \Sigma$, τ_a is either a reset or a permutation function.

We denote with $\mathcal{R}^-$, $\mathcal{R}$, $\mathcal{P}$ and $\mathcal{R}^+$ the classes of pure-reset, reset, permutation, and permutation-reset automata, respectively.

Let $A \in \{\mathcal{R}^-, \mathcal{R}, \mathcal{P}, \mathcal{R}^+\}$ be a class of automata. We denote with $\mathbb{DP}[A]$ the class of languages definable by a direct product of automata in the class A .

► **Proposition 3** (Size of direct products). *Let D be a set of automata $D = \{\mathcal{A}_1, \dots, \mathcal{A}_n\}$ such that $\mathcal{A}_i$ has alphabet Σ and set of states Q_i with $|Q_i| \geq 2$. Let $m = \max\{|Q_i| : 1 \leq i \leq n\}$. It holds that:*

- $|\mathcal{A}_1 \times \dots \times \mathcal{A}_n| \geq |\Sigma| \cdot 2^n$;
- $|D| = \sum_i |\mathcal{A}_i| \leq |\Sigma| \cdot n \cdot m$.

3.2 Cascades and the Cascade Product

The cascade product [14, 13] between two (semi)automata $\mathcal{A}$ and $\mathcal{B}$, is a generalization of the direct product in which $\mathcal{B}$ reads the Cartesian product of the input alphabet with the set of states of $\mathcal{A}$, and where the (semi)automaton $\mathcal{B}$ transitions from state s to state s' reading the symbol (a, q) if the current input symbol is a and the (semi)automaton $\mathcal{A}$ is in state q . The (semi)automaton obtained by applying the cascade product is defined over the Cartesian product of the set of states of $\mathcal{A}$ and $\mathcal{B}$ and the alphabet Σ , and is such that

transition function is applied component-wise. Additionally, for full automata [3], the initial state of the product automaton is the state comprising the initial states of $\mathcal{A}$ and $\mathcal{B}$ and the set of final states is the Cartesian product of the final states of $\mathcal{A}$ and $\mathcal{B}$.

► **Definition 4** (Cascade Product of automata). *Let Σ be a finite alphabet and let $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ and $\mathcal{B} = (\Sigma \times Q, Q', \delta', q'_0, F')$ be two automata over the alphabets Σ and $\Sigma \times Q$, respectively. The cascade product of $\mathcal{A}$ and $\mathcal{B}$, denoted by $\mathcal{A} \circ \mathcal{B}$, is the automaton $(\Sigma, Q \times Q', \delta'', (q_0, q'_0), F \times F')$ where δ'' is defined as:*

$$\delta''((q, q'), a) = (\delta(q, a), \delta'(q', (a, q)))$$

for all $(q, q') \in Q \times Q'$ and for all $a \in \Sigma$.

In the following, we will use the term *cascade* to denote a sequence of automata $\mathcal{C} = \langle \mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n \rangle$ to which the cascade product is applied, and we define the *size* of $\mathcal{C}$, denoted with $|\mathcal{C}|$, as $\sum_{i=1}^n |\mathcal{A}_i|$. Instead, with the term *cascade product*, we will denote the automaton $\mathcal{C} = \mathcal{A}_1 \circ \dots \circ \mathcal{A}_n$ obtained by applying the cascade product to the given sequence of automata. We define the *height* of the cascade $\mathcal{C} = \langle \mathcal{A}_1, \dots, \mathcal{A}_n \rangle$ as n . Given a cascade $\mathcal{C} = \langle \mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n \rangle$, we define the *language* of $\mathcal{C}$, denoted with $\mathcal{L}(\mathcal{C})$, as the language $\mathcal{L}(\mathcal{A}_1 \circ \dots \circ \mathcal{A}_n)$.

The proposition below shows that, contrary to the case of the direct product (cf. Proposition 3), representing a cascade (i.e. the *sequence* of components to which the cascade product is applied) requires no less space than representing the product automaton (i.e. the automaton obtained after applying the cascade product), which is exponential with respect to the number of components. This is due to the fact that, for cascades of automata, the alphabet of each component is the Cartesian product between Σ and the set of states of all previous components. In particular, in a cascade $\langle \mathcal{A}_1, \dots, \mathcal{A}_n \rangle$, the alphabet of the last component $\mathcal{A}_n$ has cardinality $|\Sigma| \times |Q_1| \times \dots \times |Q_{n-1}|$, which is exponential in n .

► **Proposition 5** (Size of cascades [4]). *For each cascade $\mathcal{C} = \langle \mathcal{A}_1, \dots, \mathcal{A}_n \rangle$, where Q_i is the set of states of $\mathcal{A}_i$, for each $1 \leq i \leq n$, it holds that:*

- $|\mathcal{A}_1 \circ \dots \circ \mathcal{A}_n| \geq |\Sigma| \cdot 2^n$;
- $|\mathcal{C}| \geq 2 \cdot |\Sigma| \cdot (2^n - 1)$.

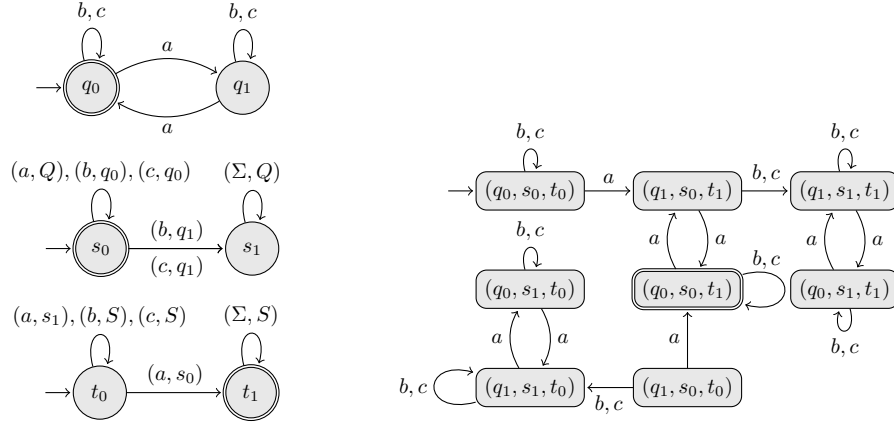
Let $A \in \{\mathcal{R}^-, \mathcal{R}, \mathcal{P}, \mathcal{R}^+\}$ be a class of automata. We denote with $\mathbb{C}[A]$ the class of languages recognized by cascades of automata of class A . A language $\mathcal{L} \subseteq \Sigma^*$ is regular iff it can be defined starting from finite subsets of Σ^* by the applications of union, negation, concatenation, and closure under Kleene star. $\mathcal{L}$ is *star-free* iff it can be generated starting from finite subset of Σ^* only by the application of union, negation, and concatenation. The following results characterizes the expressiveness of $\mathbb{C}[\mathcal{R}]$ and $\mathbb{C}[\mathcal{R}^+]$.

► **Theorem 6** ([4]). *The following characterizations hold:*

- A language $\mathcal{L}$ is regular if and only if $\mathcal{L} \in \mathbb{C}[\mathcal{R}^+]$;
- A language $\mathcal{L}$ is star-free if and only if $\mathcal{L} \in \mathbb{C}[\mathcal{R}]$.

4 Σ -Chains of automata

In this section we define Σ -chains of automata and the associated Σ -chain product. We then study their main properties in terms of succinctness and expressive power.



■ **Figure 1** On the left, starting from the top, the three components $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$ of a Σ -chain $\mathcal{C}$, with $\Sigma = \{a, b, c\}$. On the right, the associated Σ -product. We denoted final states with double circles and with (a, Q) we denote the set of symbols $\{(a, q) \mid q \in Q\}$.

4.1 Definitions, Basic Properties and Succinctness

We define Σ -chains of automata as a restricted variant of cascades of automata, in which the alphabet of the first component is Σ (as in the case of cascades), while the alphabet of each successive component $\mathcal{A}_i$ is the Cartesian product between Σ and the set of states of the previous component $\mathcal{A}_{i-1}$.

► **Definition 7** (Σ -chain Product of automata). *Let Σ be an alphabet and let $\mathcal{A}_1, \dots, \mathcal{A}_n$ be n automata such that:*

- $\mathcal{A}_1 = \langle \Sigma, Q^1, \delta^1, q_0^1, F^1 \rangle$;
- $\mathcal{A}_i = \langle \Sigma \times Q^{i-1}, Q^i, \delta^i, q_0^i, F^i \rangle$, for each $i > 1$.

We define the Σ -chain product between $\mathcal{A}_1, \dots, \mathcal{A}_n$, denoted with $\mathcal{A}_1 \odot \dots \odot \mathcal{A}_n$, as the automaton $(\Sigma, Q^1 \times \dots \times Q^n, \delta, (q_0^1, \dots, q_0^n), F^1 \times \dots \times F^n)$ such that:

$$\delta((q^1, q^2, \dots, q^n), a) = (\delta^1(q^1, a), \delta^2(q^2, (a, q^1)), \dots, \delta^n(q^n, (a, q^{n-1})))$$

for all $(q^1, q^2, \dots, q^n) \in Q^1 \times Q^2 \times \dots \times Q^n$ and for all $a \in \Sigma$.

In the following, we will use the term Σ -chain to denote a sequence of automata to which the Σ -chain product is applied. To distinguish them from cascades, we will denote this sequence as $\mathcal{C} = [\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n]$. As with cascades, we define the *size* of the Σ -chain $\mathcal{C} = [\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n]$, denoted with $|\mathcal{C}|$, as $\sum_{i=1}^n |\mathcal{A}_i|$, and its *height* as n .

Given a Σ -chain $\mathcal{C} = [\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n]$, we define the *language of $\mathcal{C}$* , denoted with $\mathcal{L}(\mathcal{C})$, as the language $\mathcal{L}(\mathcal{A}_1 \odot \dots \odot \mathcal{A}_n)$. From now on, given a cascade (or a Σ -chain) $\mathcal{C}$ and a Σ -chain (or a cascade) $\mathcal{C}'$, we say that $\mathcal{C}$ is *equivalent* to $\mathcal{C}'$ iff $\mathcal{L}(\mathcal{C}) = \mathcal{L}(\mathcal{C}')$.

An example of a Σ -chain and of the corresponding Σ -chain product is shown in Figure 1. The three automata on the left constitute the components of the Σ -chain $\mathcal{C} = [\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3]$. Automaton $\mathcal{A}_1$ is a permutation-reset automaton, whereas $\mathcal{A}_2$ and $\mathcal{A}_3$ are reset automata. The language recognized by $\mathcal{C}$ consists of all words over the alphabet $\Sigma = \{a, b, c\}$ satisfying the following conditions: (i) the number of occurrences of the symbol “a” is even; (ii) for every i^{th} occurrence of “a” with odd index i , the immediately following symbol must also be “a”; and (iii) there is at least one occurrence of “a”. Each component of the Σ -chain enforces one of these conditions. In particular, since τ_a is a permutation in $\mathcal{A}_1$ and q_0 is its only

accepting state, $\mathcal{A}_1$ accepts precisely those inputs containing an even number of occurrences of “a”. The automaton $\mathcal{A}_2$ transitions to the sink state s_1 upon reading either (b, q_1) or (c, q_1) . This implies that if the most recently read “a” corresponds to an odd occurrence (i.e. $\mathcal{A}_1$ is in state q_1) and the current symbol is b or c , then the input word—and all its continuations—must be rejected. The automaton obtained by applying the Σ -chain product to $\mathcal{C}$ is depicted on the right-hand side of Figure 1.

Let $A \in \{\mathcal{R}^-, \mathcal{R}, \mathcal{P}, \mathcal{R}^+\}$ be a class of automata. We denote by $\text{SCH}[A]$ the class of languages recognized by Σ -chains whose components are automata in the class A . The following proposition shows that Σ -chains are a particular case of cascades. In particular, each Σ -chain of automata in a class $A \in \{\mathcal{R}^-, \mathcal{R}, \mathcal{P}, \mathcal{R}^+\}$ can be converted to an equivalent cascade of automata of the same class A with the same height and with the same number of states in each component.

► **Proposition 8.** *Let $\mathcal{C} = [\mathcal{A}_1, \dots, \mathcal{A}_n]$ be a Σ -chain where, for every $i = 1, \dots, n$, $\mathcal{A}_i$ is of class $A_i \in \{\mathcal{R}^-, \mathcal{R}, \mathcal{P}, \mathcal{R}^+\}$ and has m_i states. Then, there exists a cascade $\mathcal{C}' = \langle \mathcal{A}'_1, \dots, \mathcal{A}'_n \rangle$ such that $\mathcal{L}(\mathcal{C}) = \mathcal{L}(\mathcal{C}')$, where each $\mathcal{A}'_i$ is of class A_i and has m_i states (for every $i = 1, \dots, n$).*

From Proposition 8 and Theorem 6, it directly follows that the language of any Σ -chain of reset automata is a star-free language and that the language of any Σ -chain of permutation-reset automata is a regular language.

We analyze the size of Σ -chains with respect to the number of its components and we compare it to the size of its associated Σ -chain product. The next proposition shows that, as for direct products (cf. Proposition 3), the size of a Σ -chain is always linear in the number of its components, while the size of its associated Σ -chain product is exponential (in the number of components). This is a key difference with respect to the case of cascades, in which *both* the size of a cascade and the size of its cascade product are exponential in the number of the components (Proposition 5).

► **Proposition 9** (Size of Σ -chains and Σ -chain products). *Let $\mathcal{C} = [\mathcal{A}_1, \dots, \mathcal{A}_n]$ be a Σ -chain where, for every $i = 1, \dots, n$, Q_i is the set of states of $\mathcal{A}_i$, and let $m = \max\{|Q_i| : 1 \leq i \leq n\}$. The following statements hold:*

- $|\mathcal{A}_1 \odot \dots \odot \mathcal{A}_n| \geq |\Sigma| \cdot 2^n$;
- $|\mathcal{C}| \leq |\Sigma| \cdot n \cdot m^2$.

The preceding proposition raises the following question: *do there exist languages that admit a compact representation via Σ -chains but not via cascades?* The next theorem answers this positively for the class of reset automata, by exhibiting a family of languages $\{\mathcal{L}_n\}_{n=1}^\infty$ witnessing an exponential succinctness gap. Each $\mathcal{L}_n$ can be recognized by a Σ -chain of n reset automata of polynomial size (cf. Proposition 9), while any cascade of reset automata recognizing $\mathcal{L}_n$ must have at least n components, and therefore exponential size (cf. Proposition 5).

► **Theorem 10** (Succinctness of Σ -chains). *Let Σ be a finite alphabet. For every $h > 0$, let $\mathcal{L}_h = \Sigma^h \cdot \Sigma^*$. The following holds:*

- the smallest cascade of reset automata $\mathcal{C}_h$ recognizing $\mathcal{L}_h$ has size $|\mathcal{C}_h| \in \Omega(2^h \cdot |\Sigma|)$;
- there exists a Σ -chain of two-state reset automata $\mathcal{C}'_h$ recognizing $\mathcal{L}_h$ and of size $|\mathcal{C}'_h| \in O(h \cdot |\Sigma|)$.

Proof (sketch). It is possible to construct a Σ -chain $\mathcal{C}'_h = [\mathcal{A}_1, \dots, \mathcal{A}_h]$ recognizing $\mathcal{L}_h$ as follows (for every $1 \leq i \leq h$, $\mathcal{A}_i$ has only two states, the initial state q_0^i and the only final state q_1^i): (1) in $\mathcal{A}_1$, each symbol in Σ induces a reset on q_1^1 ; (2) for every $1 \leq i \leq h$, $\mathcal{A}_i$

transitions from its initial state to its final state only when the preceding component is in its final state (regardless of the symbol read from Σ), *i.e.* each symbol (a, q_1^{i-1}) induces a reset on q_1^i and all the others induce resets on q_0^i . This ensures that the automaton $\mathcal{A}_1 \odot \dots \odot \mathcal{A}_h$ reaches its final state $(q_1^1, \dots, q_1^h)$ only after consuming at least h symbols from Σ . The lower bound on the size of the smallest cascade for $\mathcal{L}_h$ is proved by Theorem 5.8 of [4]. ◀

Theorem 10 shows that, for certain languages, the use of Σ -chains can yield representations that are exponentially more succinct than those based on cascades. This motivates the study of the expressive power of Σ -chains, that is, the characterization of the languages recognizable via Σ -chains over a given class A of automata.

4.2 Expressiveness

We now investigate the expressive power of Σ -chains, focusing in particular on the case where the components are automata drawn from a fixed class $A \in \{\mathcal{R}^-, \mathcal{R}, \mathcal{P}, \mathcal{R}^+\}$ (*i.e.* pure-reset, reset, permutation, and permutation-reset automata).

First, we compare the Σ -chain product with the direct product, showing that the former is strictly more expressive than the latter.

Then, we prove that Σ -chains of permutation-reset automata capture exactly the class of regular languages. The result is established via a *chainification method*, namely, a construction that transforms any cascade of permutation-reset automata into an equivalent Σ -chain of permutation-reset automata. The expressive equivalence with regular languages then follows from the known equivalence between regular languages and cascades of permutation-reset automata (*cf.* Theorem 6).

We further show that the chainification method extends to the class $\mathcal{R}^-$, yielding that Σ -chains of pure-reset automata are expressively equivalent to cascades of pure-reset automata.

For the class $\mathcal{R}$ of reset automata, we show that Σ -chains of reset automata recognize only star-free languages. Whether the converse holds remains open. The natural proof strategy, namely applying chainification to cascades of reset automata, which are known to recognize all star-free languages (*cf.* Theorem 6), fails: chainification produces Σ -chains of permutation-reset automata rather than reset automata.

Comparison with the direct product

As already discussed in the introduction, the Σ -chain product can be viewed both as a restricted variant of the cascade product and as a generalization of the direct product. In fact, Σ -chains retain a key feature of direct products, namely their linear size with respect to the number of components (*cf.* Proposition 9). This naturally raises the question of their relative expressive power. We show that any direct product can be simulated by a Σ -chain product ($\text{DP}[A] \subseteq \text{SCH}[A]$, for each $A \in \{\mathcal{R}^-, \mathcal{R}, \mathcal{P}, \mathcal{R}^+\}$) and that the opposite is *not* possible: we exhibit two simple languages that cannot be recognized by any direct product of reset (resp. pure-reset) automata, while they are definable by a Σ -chain of reset (resp. pure-reset) automata.

► **Lemma 11.** *The following inclusions hold:*

- $\text{DP}[A] \subseteq \text{SCH}[A]$, for each $A \in \{\mathcal{R}^-, \mathcal{R}, \mathcal{P}, \mathcal{R}^+\}$;
- $\text{DP}[\mathcal{R}] \subsetneq \text{SCH}[\mathcal{R}]$ and $\text{DP}[\mathcal{R}^-] \subsetneq \text{SCH}[\mathcal{R}^-]$.

In particular, the strict inclusions are witnessed by the languages $\mathcal{L}_1 = \Sigma^ a \Sigma^* a \Sigma^*$ and $\mathcal{L}_2 = \Sigma^* a a$, over $\Sigma = \{a, b\}$, as we have that $\mathcal{L}_1 \in \text{SCH}[\mathcal{R}] \setminus \text{DP}[\mathcal{R}]$ and $\mathcal{L}_2 \in \text{SCH}[\mathcal{R}^-] \setminus \text{DP}[\mathcal{R}^-]$.*

Transformation of cascades into Σ -chains (Chainification)

We describe the transformation of cascades into equivalent Σ -chains starting from the simplest case, that is, the case of cascades of three automata, since every cascade of height two is trivially a Σ -chain.

Let $\mathcal{C} = \langle \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3 \rangle$ be a cascade of three automata: we shall construct an equivalent Σ -chain $\mathcal{C}' = [\mathcal{A}'_1, \mathcal{A}'_2, \mathcal{A}'_3]$ such that $\mathcal{L}(\mathcal{C}) = \mathcal{L}(\mathcal{C}')$ and if $\mathcal{A}_i$ is a permutation-reset (resp., permutation, pure-reset) automaton, then also $\mathcal{A}'_i$ is a permutation-reset (resp., permutation, pure-reset) automaton, for each $1 \leq i \leq 3$.

The main challenge in this translation lies in handling the dependence of the third component $\mathcal{A}_3$ on the first component $\mathcal{A}_1$, which is not permitted in a Σ -chain. To eliminate this dependence, we encode the required information into the states of the second component $\mathcal{A}'_2$. The construction proceeds as follows: (i) the second component $\mathcal{A}'_2$ is defined so as to combine the state information of $\mathcal{A}_1$ and $\mathcal{A}_2$; (ii) the third component $\mathcal{A}'_3$ then reads the state of $\mathcal{A}'_2$, from which it can recover the information about both $\mathcal{A}_1$ and $\mathcal{A}_2$. As stated by the following Lemma, if the second component $\mathcal{A}_2$ is permutation-reset (resp. permutation, pure-reset), the new component $\mathcal{A}'_2$ can be defined to be permutation-reset (resp. permutation, pure-reset).

► **Lemma 12.** *Let $\mathcal{C} = \langle \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3 \rangle$ be a cascade of permutation-reset (resp. permutation, pure-reset) automata. There exists a Σ -chain $\mathcal{C}' = [\mathcal{A}'_1, \mathcal{A}'_2, \mathcal{A}'_3]$ of permutation-reset (resp. permutation, pure-reset) automata such that $\mathcal{L}(\mathcal{C}) = \mathcal{L}(\mathcal{C}')$.*

Proof (sketch). Define $\mathcal{A}'_1$ as $\mathcal{A}_1$. To ensure that $\mathcal{A}'_2$ carries all the information needed by $\mathcal{A}'_3$, which reads only symbols in Σ and states of $\mathcal{A}'_2$, we define the state set S' of $\mathcal{A}'_2$ as the Cartesian product of the state sets of $\mathcal{A}_1$ and $\mathcal{A}_2$. Furthermore, $\mathcal{A}'_2$ must satisfy the following condition: for every state $(q, s) \in S'$ and every transition labeled (a, q) , where q matches the first component of the current state, the transition leads to a state (q', s') such that $\delta_1(q, a) = q'$ and $\delta_2(s, (a, q)) = s'$, i.e.

$$\forall (a, q) \in \Sigma \times Q. (\delta'_2((q, s), (a, q)) = (q', s') \Leftrightarrow \delta_1(q, a) = q' \wedge \delta_2(s, (a, q)) = s')$$

Moreover, it is possible to extend the definition of δ'_2 on all other input pairs (i.e. the pairs (a, q'') with $q'' \neq q$) in order to guarantee that $\tau_{(a, q)}^{\mathcal{A}'_2}$ is a permutation (when $\tau_{(a, q)}^{\mathcal{A}_2}$ is a permutation) or a reset (when $\tau_{(a, q)}^{\mathcal{A}_2}$ is a reset). The automaton $\mathcal{A}'_3$ is defined as $\mathcal{A}_3$ by transforming each transition $\tau_{(a, q, s)}$ into $\tau_{(a, (q, s))}$. Crucially, all states of $\mathcal{A}'_1 \odot \mathcal{A}'_2 \odot \mathcal{A}'_3$ of type $(q, (q', s), t)$ with $q \neq q'$ are not reachable from the initial state, having that the language of $\mathcal{A}'_1 \odot \mathcal{A}'_2 \odot \mathcal{A}'_3$ is equal to the language of $\mathcal{A}_1 \odot \mathcal{A}_2 \odot \mathcal{A}_3$. ◀

Figure 2 illustrates an example of chainification for a cascade $\langle \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3 \rangle$ over the alphabet $\Sigma = \{a, b\}$ (left side of Figure 2). The components $\mathcal{A}'_2$ and $\mathcal{A}'_3$ of the equivalent Σ -chain are shown on the right side of Figure 2. It is important to note that all transitions (a, q) in $\mathcal{A}'_2$ can be defined as either permutations or reset functions. Furthermore, we observe that symbols acting as identities in $\mathcal{A}_2$ may become non-identity permutations in $\mathcal{A}'_2$; this is the case, for instance, for the symbol (a, q_0) . Indeed, while $\mathcal{A}_2$ is a reset automaton, $\mathcal{A}'_2$ is a permutation-reset automaton.

Lemma 12 can be generalized to deal with cascades of unbounded height as stated by the following theorem.

► **Theorem 13 (Chainification Theorem).** *Let $\mathcal{C} = \langle \mathcal{A}_1, \dots, \mathcal{A}_n \rangle$ be a cascade of permutation-reset (resp. permutation, pure-reset) automata. There exists a Σ -chain $\mathcal{C}' = [\mathcal{A}'_1, \dots, \mathcal{A}'_n]$ of permutation-reset (resp. permutation, pure-reset) automata such that $\mathcal{L}(\mathcal{C}) = \mathcal{L}(\mathcal{C}')$ and $|\mathcal{C}'| \in \mathcal{O}(|\mathcal{C}|)$.*

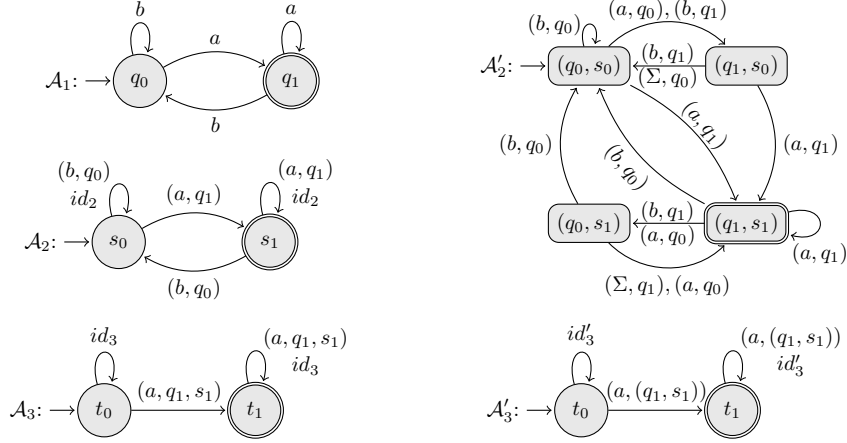


Figure 2 On the left, the cascade $\langle \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3 \rangle$ over $\Sigma = \{a, b\}$, where Q, S, T are the sets of states of $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$, respectively, $id_2 = \{(a, q_0), (b, q_1)\}$ and $id_3 = (\Sigma \times Q \times S) \setminus \{(a, q_1, s_1)\}$. On the right, the automata $\mathcal{A}'_2$ (with set of states S') and $\mathcal{A}'_3$ of the Σ -chain obtained through the chainification technique of Lemma 12, where $id'_3 = (Q \times S') \setminus \{(a, (q_1, s_1))\}$. The cascade on the left is equivalent to the Σ -chain $[\mathcal{A}_1, \mathcal{A}'_2, \mathcal{A}'_3]$.

Implications on the expressive power of Σ -chains

Proposition 8 and Theorem 13 imply that Σ -chains are expressively equivalent to cascades for the class $\mathcal{R}^-$ of pure-reset automata, for the class $\mathcal{P}$ of permutation automata and for the class $\mathcal{R}^+$ of permutation-reset automata. Moreover, Σ -chains of permutation-reset automata characterize the class of regular languages (cf. Theorem 6).

As for the class $\mathcal{R}$, we have that if a language $\mathcal{L}$ is recognized by a Σ -chain of reset automata then $\mathcal{L}$ is also recognized by a cascade of reset automata (Proposition 8) and $\mathcal{L}$ is star-free (Theorem 6).

Corollary 14. *It holds that:*

- $\text{SCH}[\mathcal{R}^+] = \mathbb{C}[\mathcal{R}^+]$ and thus a language $\mathcal{L}$ is regular if and only if $\mathcal{L} \in \text{SCH}[\mathcal{R}^+]$;
- $\text{SCH}[\mathcal{R}^-] = \mathbb{C}[\mathcal{R}^-]$;
- $\text{SCH}[\mathcal{R}] \subseteq \mathbb{C}[\mathcal{R}]$ and thus if $\mathcal{L} \in \text{SCH}[\mathcal{R}]$ then $\mathcal{L}$ is star-free;
- $\text{SCH}[\mathcal{P}] = \mathbb{C}[\mathcal{P}]$.

5 Σ -Chains of Reset Automata

In this section, we investigate Σ -chains of reset automata and the languages they recognize ($\text{SCH}[\mathcal{R}]$). First, we analyze how imposing structural constraints on Σ -chains – specifically, bounding their *height* (i.e. the number of components) and *width* (i.e. the number of states per component) – affects the class of recognizable languages.

Second, we compare Σ -chains of reset automata with Piecewise Testable [16] and R-trivial languages [5], showing that every language in these classes can be expressed as a Boolean combination and as a finite union of Σ -chains of reset automata, respectively. Moreover, we establish linear upper bounds on the size of such Σ -chain representations.

Finally, we address the *pastification* problem for Σ -chains of reset automata, namely, the transformation of these structures into equivalent formulas in pure past Linear Temporal Logic. We show a direct transformation yielding a formula of quadratic size when represented as a DAG (cf. dag-complexity [4]).

5.1 Height and Width Hierarchies

First, we show that increasing the height of a Σ -chain of reset automata (*i.e.* the number of its components) yields an infinite hierarchy of languages: for every level h , there exists a language at level h that is definable by a Σ -chain of h reset automata but is not recognizable by any Σ -chain of height $h - 1$. This result is analogous to what is known for cascades of reset automata (*cf.* Height-Hierarchy Lemma of [4]).

- **Proposition 15.** *Let Σ be a finite alphabet. For each $h > 0$, let $\mathcal{L}_h = \Sigma^h \cdot \Sigma^*$. It holds that:*
- $\mathcal{L}_h$ is definable by a Σ -chain of (two-state) reset automata $\mathcal{C}_h$ of height h ;
 - $\mathcal{L}_h$ cannot be defined by any Σ -chain of reset automata of height less than h .

As shown by Proposition 15 and by the Height-Hierarchy Lemma of [4], both Σ -chains and cascades of reset automata admit a height hierarchy: for every $h \geq 2$ there exist languages definable by a product of height h that cannot be defined by any product of smaller height. In contrast, such a hierarchy collapses for direct products, as formalized by the following lemma.

- **Lemma 16.** *Let $\Sigma = \{a, b\}$. Every language definable by a direct product of reset automata over Σ is also definable by a direct product of two reset automata over Σ .*

We now investigate the effect of increasing the number of states in each component of a Σ -chain of reset automata. The following proposition shows that this does not yield an infinite hierarchy of languages. In particular, every Σ -chain of reset automata is equivalent to a Σ -chain of the same height in which the number of states in each component depends only on the size of the alphabet Σ .

- **Proposition 17.** *Let Σ be a finite alphabet with $|\Sigma| = k \geq 2$ and let $\mathcal{L} \subseteq \Sigma^*$ be a language accepted by a Σ -chain of reset automata $\mathcal{C} = [\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_h]$, for some $h > 0$. It holds that $\mathcal{L}$ is accepted by some Σ -chain of reset automata $\mathcal{C}' = [\mathcal{A}'_1, \mathcal{A}'_2, \dots, \mathcal{A}'_h]$ where $\mathcal{A}'_i$ has at most $f(i) = \frac{k^{i+1}-1}{k-1}$ states (for $i = 1, \dots, h$).*

The $f(i) = \frac{k^{i+1}-1}{k-1}$ bound on the number of states in the i -th component of a Σ -chain, established in the previous proposition, is – owing to the restriction that each component depends only on its immediate predecessor – significantly smaller than the corresponding bound for cascades of reset automata (*cf.* Width-Collapse Lemma in [4]), namely $g^{i-1}(k+1)$ with $g(x) = x^2 - x + 1$.

5.2 Comparison with Piecewise Testable Languages

Piecewise testable languages were introduced by Simon [16] as a natural class of formal languages whose membership can be decided by inspecting only the scattered subsequences of bounded length occurring in a given word, regardless of the positions at which they appear. Intuitively, two words are indistinguishable with respect to a piecewise testable language whenever they share the same set of scattered subsequences up to a fixed length. This class has since attracted considerable attention in the theory of formal languages, owing in part to Simon's elegant algebraic characterization, which identifies piecewise testable languages with precisely those regular languages whose syntactic monoid is $\mathcal{J}$ -trivial [16], and in part to their natural logical characterization as exactly the languages definable by Boolean combinations of existential first-order sentences over words [17].

► **Definition 18** (Piecewise Testable Languages [16, 15]). Let Σ be a finite alphabet and let $\sigma = \langle a_1, \dots, a_n \rangle \in \Sigma^n$ be a word of length n , for some $n > 0$. We define $\mathcal{L}_\sigma \subseteq \Sigma^*$ as the language $\mathcal{L}_\sigma = \Sigma^* a_1 \Sigma^* \dots \Sigma^* a_n \Sigma^*$. A language $\mathcal{L}$ is k -piecewise testable if $\mathcal{L}$ is a Boolean combination of languages $\mathcal{L}_{\sigma_i}$ with $\sigma_i \in \Sigma^{\leq k}$. A language $\mathcal{L}$ is piecewise testable if it is k -piecewise testable for some $k > 0$.

To show that k -piecewise testable languages are recognizable by Boolean combinations of (languages recognized by) Σ -chains of k reset automata, we first establish a generalization of the fact that, for every $n > 0$ and every $\sigma \in \Sigma^+$, there exists a Σ -chain of n reset automata recognizing $\mathcal{L}_\sigma$. In particular, the following lemma shows that Σ -chains of reset automata can recognize languages of the form $\Sigma^* \sigma_1 \Sigma^* \dots \Sigma^* \sigma_n \Sigma^*$, where each σ_i is not restricted to a single symbol of the alphabet Σ (as in Definition 18), but may be an arbitrary word. This is achieved using a number of components equal to the sum of the lengths of all the σ_i .

► **Lemma 19.** Let Σ be a finite alphabet and let $\sigma_1, \dots, \sigma_m$ be m words in Σ^+ , for some $m > 0$. Let $\mathcal{L}_{\sigma_1, \dots, \sigma_m}$ be the language $\Sigma^* \sigma_1 \Sigma^* \dots \Sigma^* \sigma_m \Sigma^*$. There exists a Σ -chain $\mathcal{C}$ of (two-states) reset automata such that:

- $\mathcal{L}(\mathcal{C}) = \mathcal{L}_{\sigma_1, \dots, \sigma_m}$;
- the height of $\mathcal{C}$ is $|\sigma_1| + \dots + |\sigma_m|$.

Proof (sketch). We first construct, for each $1 \leq i \leq m$, a Σ -chain for the language $\Sigma^* \cdot \sigma_i \cdot \Sigma^*$. Let $\sigma_i = a_1 a_2 \dots a_n$. The Σ -chain consists of n two-state reset automata: for each $1 \leq j \leq n$, the component $\mathcal{A}_j$ transitions from its initial to its noninitial state (by means of a reset function) if and only if the current symbol is a_j and the previous component (if any) is in its noninitial state. Moreover, the last component ($\mathcal{A}_n$) must remain in its current state (by means of an identity function) for all combinations of symbols in $\Sigma \setminus \{a_n\}$ and states of $\mathcal{A}_{n-1}$, in order to ensure that the last Σ^* in $\Sigma^* \cdot \sigma_i \cdot \Sigma^*$. Finally, we set the final states of $\mathcal{A}_j$ to be the singleton containing only its noninitial state if $j = n$, otherwise we define it as the set of all the two states of $\mathcal{A}_j$.

We can then compose the Σ -chains for each σ_i obtained in this fashion to obtain a Σ -chain of $\sum_{i=1}^m |\sigma_i|$ components recognizing $\Sigma^* \sigma_1 \Sigma^* \dots \Sigma^* \sigma_m \Sigma^*$. ◀

As a straightforward corollary of Lemma 19, applied to the case where $|\sigma_i| = 1$ for every $1 \leq i \leq n$, we obtain the following result.

► **Corollary 20.** Let Σ be a finite alphabet and let $\mathcal{L} \subseteq \Sigma^*$. If $\mathcal{L}$ is a k -piecewise testable language, then $\mathcal{L}$ is a Boolean combination of languages, each of which is definable by a Σ -chain of (two-states) reset automata of height at most k .

5.3 Comparison with R-trivial Languages

R-trivial languages are those regular languages whose syntactic monoid is $\mathcal{R}$ -trivial and are dual to $\mathcal{L}$ -trivial languages [5]. Beyond their algebraic characterizations, R-trivial languages are deeply connected to automata theory. In particular, it holds that a language $\mathcal{L}$ is R-trivial if and only if the minimum automaton accepting $\mathcal{L}$ is partially ordered [5]. Surprisingly, there is also a characterization in terms of the cascade product. A language $\mathcal{L}$ is R-trivial if and only if the minimum automaton accepting $\mathcal{L}$ is covered by a cascade product of half-resets where an half-reset is a two-state semiautomaton $\mathcal{A} = \langle \Sigma, \{q_0, q_1\}, \delta \rangle$ such that Σ can be partitioned in two non-empty sets B_0 and B_1 such that $\delta(q_0, a_0) = q_0$ for every $a_0 \in B_0$, $\delta(q_0, a_1) = q_1$ for every $a_1 \in B_1$ and $\delta(a, q_1) = q_1$ for every $a \in \Sigma$ [5].

As with piecewise-testable languages, we start by defining R-trivial languages in terms of regular expressions.

► **Definition 21** (R-trivial Languages [5, 15]). Let Σ be a finite alphabet. A language $\mathcal{L} \subseteq \Sigma^*$ is R-trivial if $\mathcal{L}$ is a disjoint union of languages of the form $I_0^* a_1 I_1^* a_2 I_2^* \dots a_n I_n^*$ where: (i) $n \geq 0$; (ii) $a_i \in \Sigma$, for each $1 \leq i \leq n$; (iii) $I_i \subseteq \Sigma \setminus \{a_{i+1}\}$, for each $0 \leq i < n$; and (iv) $I_n \subseteq \Sigma$.

To show that R-trivial languages are recognizable by finite unions of (languages recognized by) Σ -chains of reset automata, we first establish a more general result. In particular, we show that Σ -chains of reset automata can recognize languages of the form $I_0^* B_1 I_1^* B_2 \dots B_n I_n^*$ where each I_i and each B_j is a subset of Σ without further restrictions. Moreover, the obtained Σ -chain is of height $n + 1$.

► **Lemma 22.** Let Σ be a finite alphabet. For some $m > 0$, let $I_0, \dots, I_m, B_1, \dots, B_m$ be subsets of Σ . Let $\mathcal{L}_{B_1, \dots, B_m}^{I_0, \dots, I_m} = I_0^* B_1 I_1^* \dots I_{m-1}^* B_m I_m^*$. There exists a Σ -chain of two-states reset automata $\mathcal{C}$ such that:

- $\mathcal{L}(\mathcal{C}) = \mathcal{L}_{B_1, \dots, B_m}^{I_0, \dots, I_m}$;
- $\mathcal{C}$ has height $m + 1$.

Proof (sketch). The proof follows an approach similar to the proof of Lemma 19. ◀

Applying Lemma 22 to the case where $|B_i| = 1$ (for every $1 \leq i \leq n$) and $I_i \subseteq \Sigma \setminus \{a_{i+1}\}$ (for $0 \leq i \leq n - 1$) leads to the following result.

► **Corollary 23.** Let Σ be a finite alphabet. Let $\mathcal{L} \subseteq \Sigma^*$ be an R-trivial language. $\mathcal{L}$ is recognized by a finite union of Σ -chains of two-states reset automata.

5.4 Σ -Chains of reset automata and ppLTL

In [14], Maler proposed a procedure to compile cascades of reset semi-automata into equivalent formulae of ppLTL, namely formulae of LTL with only past temporal operators. We call such a procedure a *pastification* procedure. In [4], this pastification procedure was applied to cascades of reset automata, providing bounds on the complexity of the resulting formula $\text{past}(\mathcal{C})$ (where $\mathcal{C}$ is a cascade of reset automata) under two measures:

- The tree-complexity (denoted $|\text{past}(\mathcal{C})|_t$): the size of the formula $\text{past}(\mathcal{C})$;
- The dag-complexity (denoted $|\text{past}(\mathcal{C})|_d$): the number of syntactically distinct sub-formulas of $\text{past}(\mathcal{C})$. The dag-complexity measures the space needed to represent the formula using the sub-term sharing technique.

Theorem 8.4 of [4] provides bounds on both measures: while the tree-complexity of the obtained ppLTL formula may be exponential in the size of $\mathcal{C}$, the dag-complexity has only a linear bound.

Given a Σ -chain $\mathcal{C}$ of n reset automata, Proposition 8 allows us to first transform it into an equivalent cascade $\mathcal{C}'$ of n reset automata, whose size may however be exponential in n . Applying Theorem 8.4 of [4] then yields an equivalent ppLTL formula, but whose dag-complexity is exponential in n . A direct translation, however, achieves a significantly better bound. Indeed, a straightforward extension of the techniques of [4] yields a transformation from a Σ -chain of n reset automata into an equivalent ppLTL formula with only quadratic dag-complexity.

► **Theorem 24.** Let $\mathcal{AP}$ be a set of atomic propositions and let $\mathcal{C} = [\mathcal{A}_1, \dots, \mathcal{A}_n]$ be a Σ -chain of reset automata over $\Sigma = 2^{\mathcal{AP}}$. There exists a ppLTL formula $\text{past}(\mathcal{C})$ such that:

- $\mathcal{L}(\text{past}(\mathcal{C})) = \mathcal{L}(\mathcal{C}) \setminus \{\epsilon\}$;
- $|\text{past}(\mathcal{C})|_d \in \mathcal{O}(|\mathcal{C}| \cdot (|\mathcal{AP}| + n)) \in \mathcal{O}(|\mathcal{C}| \cdot (|\mathcal{AP}| + |\mathcal{C}|))$.

By Theorem 10 and Theorem 24, the language $\Sigma^h \Sigma^*$ admits a pastification of dag-complexity $\mathcal{O}(h \cdot |\Sigma| \cdot (|\mathcal{AP}| + h))$ after a prior transformation into a Σ -chain of reset automata. In contrast, translating the same language into a cascade of reset automata yields a ppLTL formula of dag-complexity $\mathcal{O}(2^h \cdot |\Sigma| \cdot |\mathcal{AP}|)$. Therefore, Σ -chains can produce exponentially more succinct pastifications than cascades of reset automata.

6 Conclusions and Future Work

In this paper, we have introduced the notion of Σ -chain and the associated Σ -chain product operation, in which the component automata are combined in a feed-forward fashion, with each component depending solely on the initial alphabet Σ and the states of the immediately preceding component. We have investigated the expressive power of Σ -chains, establishing that every regular language can be decomposed into a Σ -chain of permutation-reset automata, and that every cascade of permutation-reset (pure-reset, and permutation) automata can be translated into an equivalent Σ -chain of the same class and height. While an analogous result for reset automata remains open, we have shown that (Boolean combinations of) Σ -chains of reset automata capture piecewise testable and R-trivial languages. Finally, we addressed the pastification problem, providing a translation of Σ -chains of reset automata into ppLTL formulas of quadratic dag-complexity.

A natural direction for future work is to determine whether every star-free language – that is, a language recognizable by a cascade of reset automata – is also recognizable by some Σ -chain of reset automata, which requires a complete characterization of the languages recognizable by Σ -chains. A further open problem is whether, as in the case of cascades of reset automata, every Σ -chain of reset automata can be translated into an equivalent one in which each component has only two states, possibly at the cost of increasing the height.

Several questions remain open regarding the pastification problem. It would be interesting to investigate whether the quadratic dag-complexity bound is tight, and whether the structural properties of Σ -chains can be exploited to obtain more efficient pastification procedures for LTL formulas in general.

Finally, the cascade product admits the wreath product as its algebraic counterpart. Identifying and studying the algebraic structure corresponding to the Σ -chain product is an interesting direction for future research.

References

- 1 Alessandro Artale, Luca Geatti, Nicola Gigante, Andrea Mazzullo, and Angelo Montanari. A Singly Exponential Transformation of LTL[X, F] into Pure Past LTL. In Pierre Marquis, Tran Cao Son, and Gabriele Kern-Isberner, editors, *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning, KR 2023, Rhodes, Greece, September 2-8, 2023*, pages 65–74, 2023. doi:10.24963/KR.2023/7.
- 2 Roberto Borelli, Davide Bresolin, Luca Geatti, Angelo Montanari, and Matteo Zavatterì. The σ -chain product: A succinct model of automata (de)composition (extended version), 2026. arXiv:2607.16884.
- 3 Roberto Borelli, Luca Geatti, Marco Montali, and Angelo Montanari. On Cascades of Reset Automata. In Olaf Beyersdorff, Michal Pilipczuk, Elaine Pimentel, and Kim Thang Nguyen, editors, *42nd International Symposium on Theoretical Aspects of Computer Science, STACS 2025, Jena, Germany, March 4-7, 2025*, LIPIcs, pages 20:1–20:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.STACS.2025.20.
- 4 Roberto Borelli, Luca Geatti, Marco Montali, and Angelo Montanari. Cascades of Reset Automata, 2026. Submitted to ACM Transactions on Computational Logic.

- 5 Janusz A. Brzozowski and Faith E Fich. Languages of R-trivial monoids. *Journal of Computer and System Sciences*, 20(1):32–49, 1980. doi:10.1016/0022-0000(80)90003-3.
- 6 Pál Dömösi and Balázs Imreh. On v_i -products of automata. *Acta Cybernetica*, 6(2):149–162, 1983.
- 7 Luca Geatti. Automata Cascades for Model Checking. In Angelo Montanari, Andrea Orlandini, Nicola Saccomanno, and Stefano Tonetta, editors, *Short Paper Proceedings of the 7th International Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis, OVERLAY 2025, Bologna, Italy, October 26, 2025*, CEUR Workshop Proceedings, pages 49–57. CEUR-WS.org, 2025. URL: <https://ceur-ws.org/Vol-4142/paper6.pdf>.
- 8 Ferenc Gécseg. *Products of automata*, volume 7. Springer Science & Business Media, 2012.
- 9 Marcus Gelderie. Classifying regular languages via cascade products of automata. In *International Conference on Language and Automata Theory and Applications*, pages 286–297. Springer, 2011. doi:10.1007/978-3-642-21254-3_22.
- 10 F. Gécseg and H. Jürgensen. On α_0 - v_1 -products of automata. *Theoretical Computer Science*, 80(1):35–51, 1991. doi:10.1016/0304-3975(91)90204-F.
- 11 Balázs Imreh. On α_i -products of automata. *Acta Cybernetica*, 3(4):301–307, 1977. URL: <https://cyber.bibl.u-szeged.hu/index.php/actcybern/article/view/3156>.
- 12 Kenneth Krohn and John Rhodes. Algebraic theory of machines. I. Prime decomposition theorem for finite semigroups and machines. *Transactions of the American Mathematical Society*, 116:450–464, 1965.
- 13 Oded Maler. On the Krohn-Rhodes Cascaded Decomposition Theorem. In Zohar Manna and Doron A. Peled, editors, *Time for Verification, Essays in Memory of Amir Pnueli*, volume 6200 of *Lecture Notes in Computer Science*, pages 260–278. Springer, 2010. doi:10.1007/978-3-642-13754-9_12.
- 14 Oded Maler and Amir Pnueli. Tight Bounds on the Complexity of Cascaded Decomposition of Automata. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II*, pages 672–682. IEEE Computer Society, 1990. doi:10.1109/FSCS.1990.89589.
- 15 Jean Eric Pin. *Varieties of formal languages*. Plenum Publishing Co., 1986.
- 16 Imre Simon. Piecewise testable events. In *Automata Theory and Formal Languages: 2nd GI Conference Kaiserslautern, May 20–23, 1975*, pages 214–222. Springer, 2005.
- 17 Wolfgang Thomas. Classifying regular events in symbolic logic. *Journal of Computer and System Sciences*, 25(3):360–376, 1982. doi:10.1016/0022-0000(82)90016-2.
- 18 Karl-Heinz Zimmermann. On Krohn-Rhodes theory for semiautomata. *CoRR*, abs/2010.16235, 2020. arXiv:2010.16235.