On Cascades of Reset Automata

Roberto Borelli
University of Udine, Italy

Luca Geatti
University of Udine, Italy

Marco Montali
Free University of Bozen-Bolzano, Italy

Angelo Montanari
University of Udine, Italy

Abstract

The Krohn-Rhodes decomposition theorem is a pivotal result in automata theory. It introduces the concept of *cascade product*, where two semiautomata, that is, automata devoid of initial and final states, are combined in a feed-forward fashion. The theorem states that any semiautomaton can be decomposed into a sequence of permutation-reset semiautomata. For the counter-free case, this decomposition consists entirely of reset components with two states each. This decomposition has significantly impacted recent research in various areas of computer science, including the identification of a class of transformer encoders equivalent to star-free languages and the conversion of Linear Temporal Logic formulas into past-only expressions (pastification).

The paper revisits the cascade product in the context of *reset automata*, thus considering each component of the cascade as a language acceptor. First, we give regular expression counterparts of cascades of reset automata. We then establish several expressiveness results, identifying hierarchies of languages based on the restriction of the height (number of components) of the cascade or of the number of states in each level. We also show that any cascade of reset automata can be transformed, with a quadratic increase in height, into a cascade that only includes two-state components. Finally, we show that some fundamental operations on cascades, like intersection, union, negation, and concatenation with a symbol to the left, can be directly and efficiently computed by adding a two-state component.

2012 ACM Subject Classification Theory of computation → Regular languages; Theory of computation → Automata extensions

Keywords and phrases Automata, Cascade products, Regular expressions, Krohn-Rhodes theory

Digital Object Identifier 10.4230/LIPIcs.STACS.2025.20

Funding Luca Geatti and Angelo Montanari acknowledge the support from the 2024 Italian INdAM-GNCS project "Certificazione, monitoraggio, ed interpretabilità in sistemi di intelligenza artificiale", ref. no. CUP E53C23001670001 and the support from the Interconnected Nord-Est Innovation Ecosystem (iNEST), which received funding from the European Union Next-GenerationEU (PIANO NAZIONALE DI RIPRESA E RESILIENZA (PNRR) – MISSIONE 4 COMPONENTE 2, INVESTIMENTO 1.5 -– D.D. 1058 23/06/2022, ECS00000043). In addition, Marco Montali and Angelo Montanari acknowledges the support from the MUR PNRR project FAIR - Future AI Research (PE00000013) also funded by the European Union Next-GenerationEU.

Acknowledgements The authors would like to thank Alessio Mansutti for his valuable comments during the preparation of this paper.

1 Introduction

The Krohn-Rhodes decomposition theorem is a fundamental result both in automata theory and in semigroup algebra [12]. It relies on the concept of *cascade product* of two semiautomata, i.e., automata devoid of initial and final states, and thus, ultimately, edge-labeled graphs.

In this setup, the first semiautomaton operates on an alphabet Σ, while the second one reads symbols belonging to the Cartesian product of Σ and the set of states of the first semiautomaton. The key feature of the cascade product, which extends the notion of direct product, is that the second semiautomaton transitions from state s to state s' by reading the pair (σ, q) if and only if the input symbol is σ and the first semiautomaton is in state q.

The Krohn-Rhodes theorem states that any semiautomaton can be decomposed into a *cascade* (i.e., a sequence of cascade products) of *permutation-reset semiautomata*.[1] In such semiautomata, each symbol of the alphabet induces a function on the set of states that is either a *permutation*, i.e., a bijective function, or a *reset*, that is, there is a specific state to which all other states are mapped into when reading that symbol. Crucially, if the semiautomaton is *counter-free*, that is, it does not contain non-trivial cycles [18], the Krohn-Rhodes theorem guarantees the existence of a decomposition that consists of *reset automata* only, i.e., automata where all symbols induce reset functions (as described above) or the identity function.

The Krohn-Rhodes theorem, in particular the decomposition of counter-free automata into reset automata, had a significant impact on some meaningful problems of current research in computer science. A notable example comes from Angluin et al. [1], who employ the Krohn-Rhodes decomposition theorem to prove that Linear Temporal Logic (LTL [19]) is equivalent to transformer encoders with hard attention and strict future masking (see also [13]). Specifically, they show how reset semiautomata can be encoded in B-RASP, a minimal programming language that compiles into transformers. Similarly, studies such as [21, 10, 11] utilized this theorem to analyze the sample complexity of cascades and the expressiveness of Recurrent Neural Networks without circular dependencies. Another example is provided by Maler [14, 15], who used the decomposition theorem to transform any formula of LTL, interpreted over finite words, into an equivalent formula using only past operators (see also [20]), a problem now known as *pastification* [2].

In this paper, we revisit the cascade product in the *reset automata* setting, i.e., language acceptors whose underlying semiautomaton is a reset. We address various expressiveness issues for cascade products by themselves and in relation to regular expressions. These results represent a necessary step towards a more efficient exploitation of Krohn-Rhodes decomposition in pastification, with the ultimate goal of lowering its current, triply exponential upper bound, which is far away from the know, singly exponential lower bound.

The paper consists of three main parts. In the first part, we address the question: given a cascade of reset automata, which is its corresponding regular expression? We begin by focusing on cascades of height 1, proving that the language corresponding to a reset automaton over the alphabet Σ is always of the form $J \cup (\Sigma^* \cdot R \cdot I^*)$, for some $I, R \subseteq \Sigma$, such that $I \cap R = \varnothing$ and either $J = I^*$ or $J = \varnothing$. Then, we extend the analysis to cascades of reset automata of arbitrary height. As a first step, we show that the last level can always be transformed into a two-state automaton, and then, by exploiting such a result, we derive the regular expression corresponding to a generic cascade of reset automata.

In the second part, we build on the previously obtained results and establish several expressiveness results about cascades of reset automata. We structure the analysis into three types of cascades:

- **(i)** short cascades (whose height is bounded by 2),
- **(ii)** narrow cascades (where each component has two states, but there is not a height limitation), and
- **(iii)** general cascades (with no limitations on the height or on the number of states per level).

[1] Formally, the decomposition is guaranteed to preserve a homomorphism from the cascade to the initial semiautomaton.

As for short cascades, we prove that any language $\mathcal{L}$ over an alphabet Σ of cardinality k that is definable by a reset cascade of height 2 can also be defined by one where the first component has at most $k+1$ states. Additionally, we show that increasing the number of states in the first component results in a strict increase in expressiveness: there exists a family of languages which are definable by a two-reset cascade whose first component has n states, but are not definable if the first component is restricted to $n-1$ states. Similarly, for narrow cascades, we show that increasing the height results in an increase in expressiveness. These two results – the increase in the number of states in the first component for short cascades and the increase in height for narrow cascades – lead to two hierarchies (with infinitely many levels) of languages that are not definable at previous levels. Finally, we show that any general cascade can be transformed into one whose components have all 2 states, with an increase in height of at most a quadratic factor (relative to the height of the original cascade).

In the last part, we deal with closure properties of the languages recognized by reset cascades, and show that some fundamental operations can be computed in an efficient way. More precisely, we prove that the operations of intersection, union, negation, and concatenation with Σ to the left ("next operation") all require the addition of one component with 2 states only.

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 cascades of reset automata, we state some basic results about them, and we provide a characterization of the languages that they recognize in terms of regular expressions. In Section 5, we present some expressiveness results for short, narrow, and general cascades of reset automata. Section 6 focus on closure properties and the efficient computation of some basic operations. Finally, Section 7 provides an assessment of the work done, and it outlines some directions for future research.

2 Related Work

The Krohn-Rhodes theorem and the cascade product turn out be quite useful in understanding the structure and the expressiveness of finite-state systems, in particular in the context of automata and neural networks, and their connection to logic. Various recent contributions have leveraged this foundational theory to explore the expressiveness, modularity, and learning potential of automata in such a context.

A pivotal contribution in this area is the work by Maler on the cascade decomposition of semiautomata [14, 15]. It revisits the Eilenberg's variant of the Krohn-Rhodes theorem [6] and offers a constructive proof that any semiautomaton can be decomposed into a cascade of elementary (permutation and reset) semiautomata. The paper introduces the *holonomy tree* as a data structure to represent cascade decompositions and an algorithm to build such a tree. Crucially, the algorithm carefully maps the permutations of the obtained cascade product to non-trivial cycles of the starting semiautomaton: this guarantees that, whenever the starting semiautomaton is counter-free, that is, devoid of non-trivial cycles, the generated cascade decomposition only consists of reset components. An exponential bound on the size of the cascade decomposition in terms of the size of the starting semiautomaton is given. This algorithm can be used to actually translate counter-free automata to temporal logic. More precisely, Maler shows how to translate any cascade product of reset semiautomata into a pure past LTL formula, that is, a formula featuring only past temporal modalities.

Together with the transformation of the future fragment of LTL, interpreted over finite words, into counter-free automata, this leads to a triply exponential upper bound to the problem of transforming pure-future LTL over finite words into pure-past LTL (pastification problem). Equivalently, in the case of LTL interpreted over infinite words, Maler shows how to use the proposed algorithm to normalize every LTL formula by mapping it into one belonging to the *Reactivity* class [16], at a cost of a triply exponential blowup. For both problems, that is, pastification and normalization, the best known lower bounds are singly exponential [3, 17].

The Krohn-Rhodes theorem has also been applied to analyze the complexity of semigroups, as shown in [9]. This study examines semigroups of upper triangular matrices over finite fields and establishes that the Krohn-Rhodes complexity of these semigroups corresponds to $n-1$, where n is the matrix dimension. These results underline the deep connection between the algebraic structure of semigroups and their matrix representations, providing a measure of how intricate the cascade product representation needs to be for such semigroups.

In [8], the Krohn-Rhodes theorem is used to characterize piecewise testable and commutative languages. The authors define biased reset semiautomata, where the current state changes at most once, and characterize cascades $\mathcal{A} \circ \mathcal{B}$, where $\mathcal{B}$ is a biased reset semiautomaton. Theorem 4.12 in Section 4 can be seen as a simplification and a generalization (to cascades of unbounded height) of such a characterization. Finally, the authors propose the notion of scope of a cascade, which is used to analyze the dot-depth of star-free languages.

In [21], Ronca builds on the Krohn-Rhodes theorem, proposing automata cascades as a structured and modular framework to describe complex systems. The resulting framework allows automata to be decomposed into components with specific functionalities, enabling fine-grained control of their expressiveness. By focusing on component-based decomposition, the study demonstrates that the sample complexity of learning automata cascades is linear in the number of components and their individual complexities, up to logarithmic factors. This contrasts with traditional state-centric perspectives, where sample complexity scales with the number of states, often limiting the feasibility of learning large systems. The relationships between the cascade product and neural networks are investigated in [10]. Recurrent Neural Cascades (RNCs) are a class of networks with acyclic connections, which naturally align with the cascade product of automata. By exploiting the Krohn-Rhodes theorem, the authors prove that RNCs capture star-free regular languages.

The Krohn-Rhodes theorem also underpins the exploration of transformer models in [13]. While transformers lack recurrence, the paper demonstrates that their layered architecture can simulate the cascade decomposition of finite automata. Leveraging Krohn-Rhodes theory, the authors show that shallow transformers can hierarchically approximate automata computations, enabling polynomial-sized and constant-depth shortcuts for specific automata.

In [1], Angluin et al. draws direct parallels between the expressive power of masked hard-attention transformers and star-free regular languages. These models, constrained by strict future masking, are shown to be equivalent to LTL and counter-free automata – both closely tied to the Krohn-Rhodes cascade framework. The study underscores how the structured limitations of these transformers, akin to a cascade decomposition, yield expressive yet computationally efficient models.

Together, these contributions extend the applicability of the Krohn-Rhodes theory to neural networks, transformers, and beyond, demonstrating the versatiliy of the cascade framework as a powerful principle in computation.

3 Background

A *semiautomaton* $\mathcal{A}$ is a tuple (Σ, Q, δ) such that:

(i) Σ is a (finite) alphabet;
(ii) Q is a set of states;
(iii) $\delta : Q \times \Sigma \to Q$ is a transition function.

An *automaton* $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ is a semiautomaton extended with an initial state $q_0 \in Q$ and a set $F \subseteq Q$ of final states. With δ^* we denote the Kleene's closure of δ. We say that $\mathcal{A}$ is *two-state* iff $|Q| = 2$.

Given an automaton $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ and a (finite) word $\sigma \coloneqq \langle \sigma_0, \dots, \sigma_n \rangle \in \Sigma^*$, *the run* $\tau \in Q^+$ *induced by* σ is a sequence $\langle q_0, q_1, \dots, q_{n+1} \rangle$ 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* $\mathcal{A}$, 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 $\tau \coloneqq \langle q_0, \dots, q_m \rangle$ with $q_m = q$. The classic *direct product* of automata is defined as follows.

▶ **Definition 3.1** (Direct product of automata). *Let $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ and $\mathcal{A}' = (\Sigma, Q', \delta', q_0', F')$ be two automata. The* direct product of $\mathcal{A}$ and $\mathcal{A}'$*, denoted by $\mathcal{A} \times \mathcal{A}'$, is the automaton $(\Sigma, Q \times Q', \delta'', (q_0, q_0'), F \times F')$ such that, for all $(q, q') \in Q \times Q'$ and for all $a \in \Sigma$, it holds that $\delta''((q, q'), a) = (\delta(q, a), \delta'(q', a))$.*

The cascade product of semiautomata is defined as follows.

▶ **Definition 3.2** (Cascade Product of semiautomata [15, 22]). *Let Σ be a finite alphabet and let $\mathcal{A} = (\Sigma, Q, \delta)$ and $\mathcal{A}' = (\Sigma \times Q, Q', \delta')$ be two semiautomata over the alphabets Σ and $\Sigma \times Q$, respectively. We define the* cascade product between $\mathcal{A}$ and $\mathcal{A}'$*, denoted with $\mathcal{A} \circ \mathcal{A}'$, as the semiautomaton $(\Sigma, Q \times Q', \delta'')$ such that, for all $(q, q') \in Q \times Q'$ and for all $a \in \Sigma$:*

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

We will often simply use *"cascade"* for *"cascade product"*.

It is worth noticing that the cascade product of semiautomata is a generalization of the classic direct product: the latter can be recovered by imposing the alphabet of the second semiautomaton to be Σ (i.e., the alphabet of the first one) instead of $\Sigma \times Q$.

The cascade product is an *associative* operation, meaning that $(\mathcal{A} \circ \mathcal{A}') \circ \mathcal{A}''$ is the same semiautomaton as $\mathcal{A} \circ (\mathcal{A}' \circ \mathcal{A}'')$. We define the *height* of the product $\mathcal{A}_1 \circ \cdots \circ \mathcal{A}_n$ as n.

We now introduce two classes of semiautomata, *reset* and *permutation*, depending on the form of their transitions. We first define the notion of *function induced by a symbol*.

▶ **Definition 3.3** (Function induced by a symbol). *Let $\mathcal{A} = (\Sigma, Q, \delta)$ be a semiautomaton. For each symbol $a \in \Sigma$, we define the* function induced by a in $\mathcal{A}$*, denoted by $\tau_a^{\mathcal{A}}$ (or simply τ_a when $\mathcal{A}$ is clear from the context), as the* transformation $\tau_a : Q \to Q$ *such that, for all $q \in Q$, it holds $\tau_a(q) = q'$ iff $\delta(q, a) = q'$.*

Reset and permutation functions are defined as follows.

▶ **Definition 3.4** (Reset and permutation functions). *Let $\tau : Q \to Q$. We say that τ is a* reset function *iff there exists $q' \in Q$ such that $\tau(q) = q'$, for all $q \in Q$. In this case, we say that τ is a* reset on q'*. If $\tau : Q \to Q$ is a bijection, then it is called a* permutation.

On the basis of the functions induced by the symbols of their alphabet, we define the following classes of semiautomata.

▶ **Definition 3.5** (Classes of semiautomata). *Let $\mathcal{A} = (\Sigma, Q, \delta)$ be a semiautomaton. We say that $\mathcal{A}$ is:*

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

We now introduce counter-free semiautomata [18]. Let $\sigma \in \Sigma^*$. From now on, we denote by $(\sigma)^i$ the word generated by concatenating i times the word σ to itself. A word $\sigma \in \Sigma^*$, with $\sigma \neq \varepsilon$, defines a *nontrivial cycle* in a semiautomaton $\mathcal{A} = (\Sigma, Q, \delta)$ if there exists a state $q \in Q$ such that:

(i) $\delta^*(q, \sigma) \neq q$
(ii) $\delta^*(q, (\sigma)^i) = q$, for some $i > 1$.

We say that a semiautomaton $\mathcal{A}$ is *counter-free* if there are no words that define a nontrivial cycle. Counter-free automata recognize exactly the set of languages definable by *star-free* regular expressions, i.e., expressions devoid of Kleene'star. We denote this set by $\mathcal{SF}$.

A fundamental result in the field is the Krohn-Rhodes Cascade Decomposition Theorem. The theorem's initial formulation was expressed in the context of semigroups [12], and its automata-theoretic counterpart [14] can be articulated as follows.

▶ **Theorem 3.6** (The Krohn-Rhodes Cascade Decomposition Theorem [12, 14]). *For each semiautomaton $\mathcal{A} = (\Sigma, Q, \delta)$, there exists a cascade product of semiautomata $\mathcal{C} := \mathcal{A}_1 \circ \mathcal{A}_2 \circ \cdots \circ \mathcal{A}_n$ such that:*

(i) *$\mathcal{A}_i$ is a permutation-reset semiautomaton, for each $1 \leq i \leq n$;*
(ii) *there is an homomorphism[2] from $\mathcal{C}$ to $\mathcal{A}$;*
(iii) *if $\mathcal{A}$ is* counter-free*, then $\mathcal{A}_i$ is a two-state reset semiautomaton, for each $1 \leq i \leq n$.*

4 Cascades of automata

In this section, we begin our study of the languages recognized by cascades of automata. We start by formally defining them and stating some basic properties. Then, we focus on cascades of reset automata, and provide a characterization of the languages they recognize in terms of regular expressions.

4.1 Definitions and basic properties

To begin with, we generalize the notion of cascade product of semiautomata (Definition 3.2) to automata.

▶ **Definition 4.1** (Cascade product of automata). *Let Σ be a finite alphabet and let $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ and $\mathcal{A}' = (\Sigma \times Q, Q', \delta', q_0', F')$ be two automata over the alphabets Σ and $\Sigma \times Q$, respectively. We define the* cascade product of $\mathcal{A}$ and $\mathcal{A}'$*, denoted by $\mathcal{A} \circ \mathcal{A}'$, as the automaton $(\Sigma, Q \times Q', \delta'', (q_0, q_0'), F \times F')$ where δ'' is defined as in Definition 3.2.*

We say that a language $\mathcal{L}$ is *definable by* a cascade $\mathcal{C}$ iff $\mathcal{L} = \mathcal{L}(\mathcal{C})$. Figure 1 shows the cascade product of two reset automata defining the language $a \cdot \Sigma^*$.

[2] We refer to [12, 14] for a formal definition of homomorphism between semiautomata.

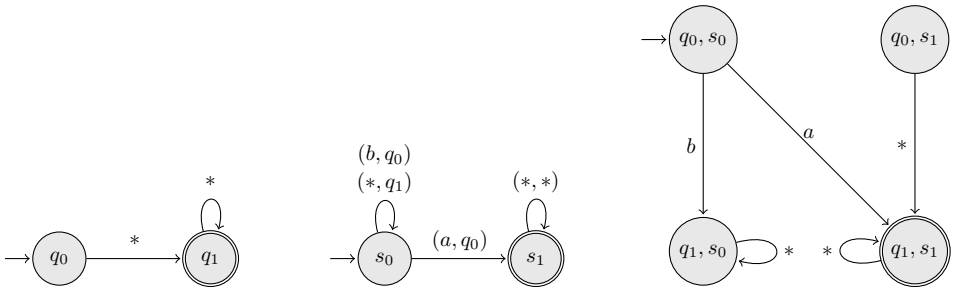

Figure 1 The reset automaton $\mathcal{A}_1$ with set of states $Q = \{q_0, q_1\}$ over the alphabet $\Sigma = \{a, b\}$ (left). The reset automaton $\mathcal{A}_2$ over the alphabet $\Sigma \times Q$ (middle). The cascade product $\mathcal{A}_1 \circ \mathcal{A}_2$ over the alphabet Σ that recognizes the languages $a \cdot \Sigma^*$ (right).

In the following, we will use the term cascade to refer both to the component automata and to the resulting automaton.

We now show how to compute the language recognized by a cascade of automata on the basis of the languages recognized by its components. Let Σ_1 and Σ_2 be two alphabets. Let $\sigma^1 = \sigma^1_1 \dots \sigma^1_n \in (\Sigma_1)^n$ and $\sigma^2 = \sigma^2_1 \dots \sigma^2_n \in (\Sigma_2)^n$ be two words of length n. We define $\mathsf{aug}(\sigma^1, \sigma^2) \in (\Sigma_1 \times \Sigma_2)^n$ as the word $(\sigma^1_1, \sigma^2_1) \dots (\sigma^1_n, \sigma^2_n)$.

▶ **Definition 4.2** (Language of $\mathcal{B}$ at a state s over $\mathcal{A}$). *Let $\mathcal{A} = \langle \Sigma, Q, \delta_A, q_0, F_A \rangle$ and $\mathcal{B} = \langle \Sigma \times Q, S, \delta_B, s_0, F_B \rangle$ be two automata. The language of $\mathcal{B}$ at state $s \in S$ over $\mathcal{A}$, denoted by $\mathcal{L}_s(\mathcal{B})[\mathcal{A}]$, is defined as follows: the empty word only belongs to $\mathcal{L}_{s_0}(\mathcal{B})[\mathcal{A}]$; a word $\sigma = \sigma_1 \dots \sigma_k$, with $k \geq 1$, belongs to $\mathcal{L}_s(\mathcal{B})[\mathcal{A}]$ if*

(i) *$\sigma_1 \dots \sigma_{k-1}$ induces a run $\tau = \langle q_0, q_1 \dots, q_{k-1} \rangle$ on $\mathcal{A}$; and*
(ii) *$\mathsf{aug}(\sigma, \tau) \in \mathcal{L}_s(\mathcal{B})$.*

The language of a cascade can be computed from those of its components as follows. Let $\mathcal{C} = \mathcal{A} \circ \mathcal{B}$ be a cascade. The words forcing $\mathcal{C}$ to reach a state (q, s) are exactly those words such that:

(i) they force $\mathcal{A}$ to reach state q; and
(ii) they force $\mathcal{B}$ to reach state s, when augmented with the run of $\mathcal{A}$.

▶ **Proposition 4.3** (Language of a cascade in terms of its components). *Let $\mathcal{A} = \langle \Sigma, Q, \delta_A, q_0, F_A \rangle$ and $\mathcal{B} = \langle \Sigma \times Q, S, \delta_B, s_0, F_B \rangle$ be two automata. It holds that:*

1. *$\mathcal{L}_{(q,s)}(\mathcal{A} \circ \mathcal{B}) = \mathcal{L}_q(\mathcal{A}) \cap \mathcal{L}_s(\mathcal{B})[\mathcal{A}]$, for all states $q \in Q$ and $s \in S$;*
2. *$\mathcal{L}(\mathcal{A} \circ \mathcal{B}) = \bigcup_{(q,s) \in F} \mathcal{L}_{(q,s)}(\mathcal{A} \circ \mathcal{B})$.*

We now show that, as it happens with semiautomata, the direct product of automata is just a special case of the cascade product. To this end, we first define the notion of augmentation of an automaton.

▶ **Definition 4.4** (Augmentation). *Let $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, F \rangle$ and $\mathcal{A}' = (\Sigma', Q', \delta', q'_0, F')$ be two automata such that either $\Sigma' = \Sigma$ or $\Sigma' = \Sigma \times S$, for an arbitrary finite set S. We define the* augmentation of $\mathcal{A}'$ relative to $\mathcal{A}$, *denoted by $\mathsf{aug}(\mathcal{A}, \mathcal{A}')$, as the automaton $(\Sigma'', Q', \delta'', q'_0, F')$ such that:*

- *if $\Sigma' = \Sigma$, then $\Sigma'' := \Sigma \times Q$ and for all $q \in Q'$ and all $a \in \Sigma$, $\delta''(q, (a, *)) = \delta'(q, a)$;*
- *if $\Sigma' = \Sigma \times S$, then $\Sigma'' := \Sigma \times Q \times S$ and, for all $q \in Q'$ and for all $(a, s) \in \Sigma \times S$, it holds that $\delta''(q, (a, *, s)) = \delta'(q, (a, s))$.*

Given a cascade $\mathcal{C} = \mathcal{A}'_1 \circ \dots \circ \mathcal{A}'_n$ over Σ, we define the augmentation of $\mathcal{C}$ relative to $\mathcal{A}$, *denoted by $\mathsf{aug}(\mathcal{A}, \mathcal{C})$, as the cascade $\mathsf{aug}(\mathcal{A}, \mathcal{A}'_1) \circ \dots \circ \mathsf{aug}(\mathcal{A}, \mathcal{A}'_n)$.*

The notion of augmentation can be generalized to a pair of cascades $\mathcal{C}$ and $\mathcal{C}'$ by treating $\mathcal{C}$ as a single automaton: from now on, when we will refer to the cascade product of $\mathcal{C}$ and $\mathcal{C}'$, we will interpret it as the cascade product of the automaton $\mathcal{A}$ generated by $\mathcal{C}$ and $\mathcal{C}'$.

The next proposition shows that direct product can be simulated by means of augmentation and cascade product.

▶ **Proposition 4.5** (Direct product by means of cascade product). *Let $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, F \rangle$ be an automaton and let $\mathcal{C}$ be a cascade over Σ. It holds that $\mathcal{A} \circ \mathsf{aug}(\mathcal{A}, \mathcal{C}) = \mathcal{A} \times \mathcal{C}$.*

Furthermore, augmenting an automaton does not affect its property of being *reset* (or *permutation*), as stated by the following Proposition 4.6.

▶ **Proposition 4.6.** *Let Σ be a finite alphabet and let $\mathcal{A}$ be an automaton over Σ or $\Sigma \times S$, for an arbitrary finite set S. If $\mathcal{A}$ is a reset (resp., permutation) automaton, then, for any automaton $\mathcal{A}'$ over Σ, $\mathsf{aug}(\mathcal{A}', \mathcal{A})$ is a reset (resp., permutation) automaton.*

It follows that, in particular, if $\mathcal{C}$ is a cascade of reset (resp., permutation) automata, then $\mathsf{aug}(\mathcal{A}, \mathcal{C})$ is a cascade of reset (resp., permutation) automata. From Propositions 4.5 and 4.6, it follows directly that, given two cascades $\mathcal{C}$ and $\mathcal{C}'$ of height m and n of reset (resp., permutation) automata, there exists a cascade of height $m + n$ of reset (resp., permutation) automata for $\mathcal{L}(\mathcal{C}) \cap \mathcal{L}(\mathcal{C}')$. In Section 6, we will show how to compute other basic operations on cascades of resets.

4.2 Languages of cascades of resets

In this part, we characterize the language recognized by a cascade of reset automata in terms of regular expressions. We begin with the case of cascades of height 1 and then we move to cascades of unbounded height.

4.2.1 Cascades of height 1

The study of which regular expressions characterize height-1 cascades of resets coincides with the study of the languages recognized by *reset automata*. The following theorem gives a characterization of reset automata in terms of regular expressions.

▶ **Theorem 4.7** (The languages of reset automata). *Let Σ be a finite alphabet. A language $\mathcal{L} \subseteq \Sigma^*$ is recognized by a reset automaton if and only if $\mathcal{L} = J \cup (\Sigma^* \cdot R \cdot I^*)$ for some $I, R \subseteq \Sigma$ such that $I \cap R = \varnothing$ and either $J = I^*$ or $J = \varnothing$.*

Intuitively, an automaton reading a symbol that induces a reset function on a final state is forced to end up in that state, regardless of which state it was in before. Furthermore, it remains in that state if all subsequent symbols induce identity functions. In the case of words containing multiple resets on a final state, only the last of these symbols matters, resulting in words of the form $\Sigma^* \cdot R \cdot I^*$. The case of $J = I^*$ arises when the initial state is also final. In this scenario, to accept a word, the automaton does not need to read a symbol that induces a reset on a final state (since it is already there), but only needs to stay in the initial state.

A by-product of Theorem 4.7 is that any reset automaton is equivalent to one with two states, only one of which is final. The rationale is as follows:

(i) the symbols in R induce a reset on the single final state;
(ii) the symbols in I act as identities; and
(iii) the symbols neither in R nor in I induce resets on the single non-final state.

Figure 2 The reset automaton corresponding to a language of the form $J \cup (\Sigma^* \cdot R \cdot I^*)$ in the case $J = \varnothing$ (on the left) and in the case $J = I^*$ (on the right).

Moreover, the initial state is also the final state if and only if $J = I^*$. A graphical account is given in Figure 2.

▶ **Proposition 4.8.** *For every reset automaton, there exists an equivalent one with two states, exactly one of which is final.*

Theorem 4.7 allows us to establish a first connection between *Linear Temporal Logic on finite traces* (LTLf [5]) formulas and equivalent reset cascades. As highlighted in the introduction, the languages expressible in LTLf are exactly the star-free languages, that is, those languages that can be represented by regular expressions that do not use the Kleene star, or equivalently, by languages whose minimal automaton is counter-free [18]. By Krohn-Rhodes' theorem (Theorem 3.6), it follows that the languages definable in LTLf are precisely those expressible through cascades of resets. Given the relevance of reset cascade decomposition in problems such as *pastification* [2, 4] and *normalization* [7] of temporal logic formulas, it is crucial to understand which LTLf formulas can be expressed with cascades of a specific height. The following result shows that even simple formulas like p (the proposition letter "p" holds at the initial time point) or $p \,\mathsf{U}\, q$ (there is a future point where "q" holds, and until then, "p" remains true) cannot be expressed with cascades of resets of height 1. In fact, the languages they recognize[3] are respectively $\vec{p} \cdot \Sigma^*$ and $(\vec{p})^* \cdot \vec{q} \cdot \Sigma^*$, which are not of the form $J \cup (\Sigma^* \cdot R \cdot I^*)$, for any choice of R, I, and J. However, the formula $\mathsf{F}p$ (there exists a point in the future where "p" holds) can be expressed with reset cascades of height 1, as its language is of the form $J \cup (\Sigma^* \cdot R \cdot I^*)$, choosing $R \coloneqq \vec{p}$, $I \coloneqq \Sigma \setminus \vec{p}$, and $J = \varnothing$.

▶ **Corollary 4.9.** *The languages defined by the* LTLf *formulas p and $p \,\mathsf{U}\, q$ are not definable with height-1 cascades of resets.*

4.2.2 Cascades of unbounded-height

In this section we derive regular expressions for cascades of arbitrary height. As a first step, we show that a cascade of height h of reset automata, say $\mathcal{A}_1 \circ \cdots \circ \mathcal{A}_h$, can be transformed into an equivalent cascade of the same height, still consisting of reset automata, where the last automaton ($\mathcal{A}_h$) has exactly two states, one of which is the only accepting state. This result forms the basis for Theorem 4.12, which provides the characterization of cascades of arbitrary height.

▶ **Lemma 4.10.** *Let $\mathcal{A}$ be an automaton with set of states Q over the alphabet Σ, and let $\mathcal{B}$ be a* reset *automaton over the alphabet $\Sigma \times Q$. There exists a 2-states reset automaton $\mathcal{B}'$, with exactly one final state, such that $\mathcal{L}(\mathcal{A} \circ \mathcal{B}) = \mathcal{L}(\mathcal{A} \circ \mathcal{B}')$.*

[3] Here, assuming a set of atomic propositions $\mathcal{AP} \coloneqq \{p, q, r, \ldots\}$, the languages of formulas over $\mathcal{AP}$ are defined over the alphabet $\Sigma \coloneqq 2^{\mathcal{AP}}$. Moreover, given any $p \in \mathcal{AP}$, we indicate with $\vec{p}$ all the letters $a \in \Sigma$ such that $p \in a$.

The proof of Lemma 4.10 heavily relies on the characterization of cascades of height 1. More precisely, since $\mathcal{B}$ is a reset automaton over the alphabet $\Sigma \times Q$, by Proposition 4.8, there exists an equivalent reset automaton with two states (one of which is the only accepting state) over the same alphabet. Since $\mathcal{B}$ is at the bottom of the cascade, the language of the cascade $\mathcal{A} \circ \mathcal{B}'$ is the same as the language of $\mathcal{A} \circ \mathcal{B}$. This is because there are no other automata below $\mathcal{B}$ in the cascade that can exploit information about $\mathcal{B}$'s current state. As a matter of fact, in Section 5, we will prove that this no longer true when applying the same procedure to $\mathcal{A}$: there exist languages definable by a cascade $\mathcal{A} \circ \mathcal{B}$, where $\mathcal{A}$ has 3 states and $\mathcal{B}$ has 2 states, that cannot be expressed if the number of states of $\mathcal{A}$ is limited to 2.

Let us now introduce the notion of *filtered automaton*, which is obtained from a given automaton by removing (filtering) certain outgoing transitions and possibly changing its initial state.

▶ **Definition 4.11** (Filtered Automaton). *Let $\mathcal{A} = \langle Q, \Sigma, q_0, \delta, F \rangle$ be an automaton. A filter is pair (q, H), where $q \in Q$ and $H \subseteq \Sigma \times Q$. The partial automaton $\mathcal{A}$, filtered by (q, H), denoted as $\mathcal{A} \downarrow_H^q$, is the automaton $(Q, \Sigma, q, \delta', F)$ where $\delta'(q', \sigma) \coloneqq \delta(q', \sigma)$ if $(\sigma, q') \in H$, or is undefined otherwise.*

Before formally stating Theorem 4.12, that characterizes the languages of cascades of unbounded-height, we give an intuitive account of it. Let $\mathcal{A} \circ \mathcal{B}$ be a cascade, with $\mathcal{A}$ an automaton and $\mathcal{B}$ a reset automaton, where, w.l.o.g. (Lemma 4.10), $\mathcal{B}$ has only two states and exactly one final state. Any word accepted by $\mathcal{A} \circ \mathcal{B}$ must drive both $\mathcal{A}$ and $\mathcal{B}$ to an accepting state. Its language can be captured by analyzing the symbols inducing a reset function that leads to a final state of $\mathcal{B}$, and the symbols inducing identities in $\mathcal{B}$. The words in the language of $\mathcal{A} \circ \mathcal{B}$ are precisely those consisting of

- **(i)** a *prefix* that, for any symbol (σ, q) inducing a reset on a final state of $\mathcal{B}$, drives automaton $\mathcal{A}$ to state q;
- **(ii)** followed by the symbol $\sigma \in \Sigma$ (let q_σ be the state reached by $\mathcal{A}$ after reading it);
- **(iii)** a *suffix* that forces $\mathcal{B}$ to remain in its accepting state through its identity functions I_B, and forces $\mathcal{A}$ to reach a final state starting from q_σ.

In addition, $\mathcal{A}$ *cannot* transition from state q' when reading a symbol σ' if the pair (σ', q') does not belong to $\mathcal{B}$'s identity functions, as this would cause $\mathcal{B}$ to leave its accepting state. Therefore, the suffix corresponds to the language of automaton $\mathcal{A}$, filtered by $(\delta_A(q, \sigma), I_B)$. This is formally expressed by the following theorem.

▶ **Theorem 4.12** (Languages of cascades of unbounded-height). *Let $\mathcal{A} = \langle \Sigma, Q = \{q_0, \ldots, q_n\}, \delta_A, q_0, F_A \rangle$ be an automaton and let $\mathcal{B} = \langle \Sigma \times Q, \{s_0, s_1\}, \delta, s_0, \{s_f\} \rangle$, with $s_f \in \{s_0, s_1\}$, be a two-state reset automaton with one final state. It holds that:*

$$\mathcal{L}(\mathcal{A} \circ \mathcal{B}) = M \cup \bigcup_{(\sigma,q) \in R_{s_f}} \mathcal{L}_q(\mathcal{A}) \cdot \sigma \cdot \mathcal{L}\left(\mathcal{A} \downarrow_{I_B}^{\delta_A(q,\sigma)}\right)$$

where R_{s_f} is the set of symbols in $\Sigma \times Q$ that induce a reset function on state s_f, and $M \coloneqq \mathcal{L}(\mathcal{A} \downarrow_{I_B}^{q_0})$ if $s_0 = s_f$ or $M \coloneqq \varnothing$ otherwise.

Figure 3 gives an example of application of Theorem 4.12 to a cascade over the alphabet $\Sigma \coloneqq \{a, b\}$ of two reset automata, $\mathcal{A}$ (on the left) and $\mathcal{B}$ (on the center), with two states each, recognizing the language $b^* \cdot a^+$. Using Theorem 4.12, we have that $\mathcal{L}(\mathcal{A} \circ \mathcal{B}) = \mathcal{L}_{q_0}(\mathcal{A}) \cdot a \cdot \mathcal{L}(\mathcal{A} \downarrow_{I_B}^{q_1})$, where $\mathcal{A} \downarrow_{I_B}^{q_1}$ is the automaton obtained from $\mathcal{A}$ filtered by the identities $I_B = \{(a, q_1), (b, q_0)\}$ of automaton $\mathcal{B}$. Since $\mathcal{L}_{q_0}(\mathcal{A}) = b^*$ and $\mathcal{L}(\mathcal{A} \downarrow_{I_B}^{q_1}) = a^*$, we obtain $\mathcal{L}(\mathcal{A} \circ \mathcal{B}) = b^* \cdot a^+$. The following is a corollary of Theorem 4.12 in the case in which $\mathcal{B}$ is a pure-reset automaton.

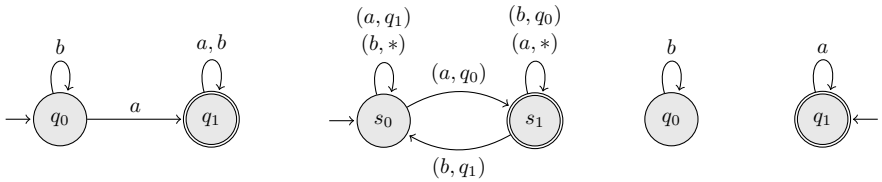

Figure 3 On the left and on the center, the reset automata $\mathcal{A}$ and $\mathcal{B}$, respectively, for the cascade $\mathcal{A} \circ \mathcal{B}$ recognizing the language $b^* \cdot a^+$. On the right, the automaton $\mathcal{A} \downarrow_{I_B}^{s_1}$, where $I_B = \{(a, q_1), (b, q_0)\}$ are the identities of automaton $\mathcal{B}$.

▶ **Corollary 4.13.** *Let $\mathcal{A} = \langle \Sigma, \{q_0, \ldots, q_n\}, \delta_A, q_0, F_A \rangle$ be an automaton and let $\mathcal{B} = \langle \Sigma \times Q, \{s_0, s_1\}, \delta, s_0, \{s_f\} \rangle$ be a two-state pure-reset automaton with one final state. It holds that*

$$\mathcal{L}(\mathcal{A} \circ \mathcal{B}) = M \cup \bigcup_{\substack{(\sigma,q) \in R_{s_f} \\ \delta_A(q,\sigma) \in F_A}} \mathcal{L}_q(\mathcal{A}) \cdot \sigma$$

where $M \coloneqq \epsilon$ if $s_0 = s_f$ and $q_0 \in F_A$, or $M \coloneqq \varnothing$ otherwise.

It is worth noticing that the regular expressions in Theorem 4.12 and Corollary 4.13 refer to states of the cascade under consideration. This is a major difference with the characterization of reset cascades of height 1 (Theorem 4.7). In order to prove some undefinability results in the next section, we provide a characterization of the languages recognized by cascades of two-states resets of height 2 where the second component is pure-reset, based on regular expressions that do not refer to states of the cascade.

▶ **Lemma 4.14.** *Let $\mathcal{L} \subseteq \Sigma^*$ be a language. $\mathcal{L}$ is definable by a cascade of height 2 in which the second component is pure-reset* if and only if *$\mathcal{L} = M \cup \bigcup_{i=1}^{n} K_i \cdot \sigma_i$ for some M, n, σ_i and K_i such that:*

- **(i)** *M is either ϕ or ϵ;*
- **(ii)** *$0 \leq n \leq 2 \cdot |\Sigma|$;*
- **(iii)** *for all $i = 1 \ldots n$ it holds $\sigma_i \in \Sigma$;*
- **(iv)** *there exists a language L recognizable by a two-state reset automaton such that for all $i = 1 \ldots n$, either K_i is L or K_i is $\overline{L} = \Sigma^* \setminus L$.*

Notice that the Lemma above can be easily extended to the case in which the first automaton in the cascade has k states, for some $k \geq 2$. This is done by relaxing (iv) and imposing that K_i can be chosen between k languages $K_1, \ldots, K_k$ such that $\{K_1, \ldots, K_k\}$ is a partition of Σ^* and each K_i is definable by a cascade of resets of height 1. Constraint (ii) is also relaxed to $0 \leq n \leq k \cdot |\Sigma|$.

We will use Theorem 4.12, Corollary 4.13, and Lemma 4.14 in the next section to prove undefinability results of certain languages by cascades of a given height and with a specified number of states at each level.

5 Expressiveness results

In this section, we analyze the expressive power of various types of reset automaton cascades. We begin by defining several language classes, and subsequently structure our analysis into *short cascades* (where the height is constrained to at most two), *narrow cascades* (where the height is unbounded but each component contains two states), and *general cascades* (with no restrictions on either the height or the number of states).

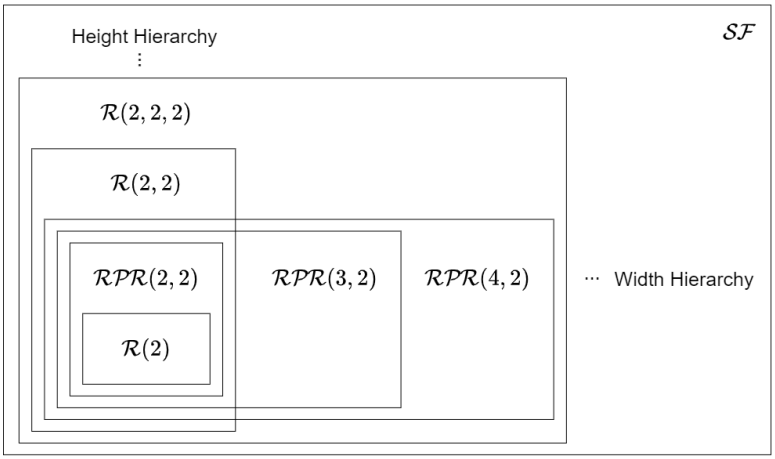

Figure 4 Summary of (some of) the results in Section 5.

▶ **Definition 5.1** (Classes $\mathcal{R}$ and $\mathcal{RPR}$). *Let $h \in \mathbb{N}^{>0}$ and let $k_1, \ldots, k_h \in \mathbb{N}^{>1}$. We denote by $\mathcal{R}(k_1, \ldots, k_h)$ the class of languages definable by a cascade $\mathcal{A}_1 \circ \cdots \circ \mathcal{A}_h$ of reset automata such that $\mathcal{A}_i$ has k_i states, for each $1 \leq i \leq h$. We denote by $\mathcal{RPR}(k_1, \ldots, k_h)$ the subclass of $\mathcal{R}(k_1, \ldots, k_h)$ where the last automaton ($\mathcal{A}_h$) is required to be pure-reset. For $h > 0$ and $k > 1$, we define $\mathcal{R}_k^h \coloneqq \bigcup_{2 \leq k_1, \ldots, k_h \leq k} \mathcal{R}(k_1, \ldots, k_h)$ as the set of languages definable by a cascade of height h, where each component has at most k states. We define $\mathcal{R} \coloneqq \bigcup_{h>0, k>1} \mathcal{R}_k^h$ as the set of languages definable by any cascade of reset automata. The classes $\mathcal{RPR}_k^h$ and $\mathcal{RPR}$ are defined analogously.*

Figure 4 provides an overview of (some of) the results presented in this section. Specifically, it illustrates that increasing the cascade height and increasing the number of states at the first level lead to two distinct language hierarchies.

5.1 Short Cascades

We begin by considering *short* cascades, i.e. cascades of reset automata of height 2. As a first step, we start by comparing the classes $\mathcal{R}(2), \mathcal{RPR}(2,2)$ and $\mathcal{R}(2,2)$, and then we focus on $\mathcal{RPR}(k,2)$ and $\mathcal{R}(k,2)$ for every $k > 2$.

We already know that with a single pure-reset automaton we can recognize the set of all words ending with a certain symbol of the alphabet (this follows from Theorem 4.7 in the special case in which $I = \varnothing$). As an example, it holds that $\Sigma^* a \in \mathcal{RPR}(2)$. Now, if we introduce an additional pure-reset layer, we can effectively recognize the set of words ending with a *two-character suffix*. However, we also demonstrate that this is impossible using a single reset automaton.

▶ **Lemma 5.2.** *Let $L = \Sigma^* aa$. It holds that:*
(i) *$L \in \mathcal{RPR}(2,2)$;*
(ii) *$L \notin \mathcal{R}(2)$.*

Lemma 5.2 shows that increasing the height of a cascade, even of height 1 and even with a pure-reset automaton, results into a gain of expressive power.

In the upcoming lemma, we demonstrate that, at the same height, prohibiting identities in the final layer results in a loss of expressive power. To illustrate this, let us consider the language $a \cdot \Sigma^*$. As shown in Figure 1, this language can be defined using a cascade of two reset

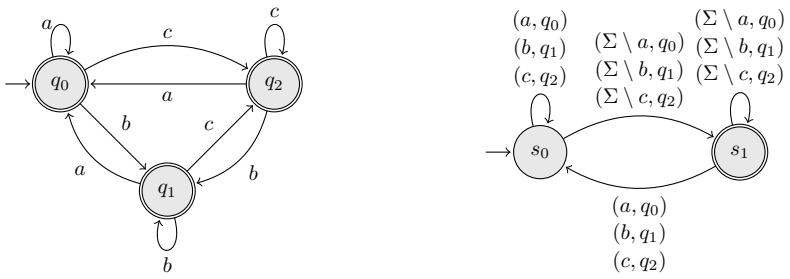

Figure 5 On the left, the reset automaton $\mathcal{A}$ and on the right the reset automaton $\mathcal{B}$ such that, for $\Sigma = \{a, b, c\}$, the cascade $C \coloneqq \mathcal{A} \circ \mathcal{B}$ accepts the language $\Sigma^* \left(\Sigma^2 \setminus \{aa, bb, cc\}\right) \cup \{b, c\}$, which precisely corresponds to the language L_3 described in Lemma 5.4. When viewed as a single automaton, C is also the minimal automaton for the language L_3.

automata, with the last one specifically containing identities. Building upon Lemma 4.14, we further demonstrate that achieving the same language recognition is not possible when prohibiting identities in the final layer, no matter of the number of states of the first automaton.

▶ **Lemma 5.3.** *Let Σ be an alphabet with at least two symbols, let $L = a\Sigma^*$. It holds that:*
(i) *$L \in \mathcal{R}(2, 2)$;*
(ii) *$L \notin \mathcal{RPR}(k, 2)$, for every $k \geq 2$.*

In Lemma 4.10, we have shown that the final component of a cascade can always be restricted to two states. A natural question arises: *Can the first component also be limited to just two states?* The answer is negative, as illustrated by the following example. Consider an alphabet of three symbols and the language L consisting of all words that end with two distinct symbols (e.g. $cb \in L$ but $aa \notin L$). As demonstrated in Figure 5, this language can be recognized by a cascade of two reset automata where the first component has three states. However, we will prove that it cannot be recognized if the first component has only two states. Intuitively, the first component's role is to remember the second-to-last symbol, but with an alphabet of three symbols and only two states, this task becomes impossible. The following *Width-Hierarchy Lemma* formalizes this intuition, demonstrating the existence of an infinite hierarchy of languages that can be defined using cascades of two resets where the first component contains k states, but cannot be defined when the first component is restricted to $k - 1$ states.

▶ **Lemma 5.4** (Width-Hierarchy Lemma)**.** *For each $k > 2$, let $\Sigma = \{\sigma_0, \ldots, \sigma_{k-1}\}$. Let $L_k = \Sigma^* \left(\Sigma^2 \setminus \bigcup_{0 \leq i < k} \sigma_i \sigma_i\right) \cup (\Sigma \setminus \sigma_0)$, it holds that:*
(i) *$L_k \in \mathcal{RPR}(k, 2)$;*
(ii) *$L_k \notin \mathcal{R}(k - 1, 2)$.*

The following corollary (depicted in Figure 4) follows from Lemmas 5.2–5.4.

▶ **Corollary 5.5.** *It holds that:*
- $\varnothing \subsetneq \mathcal{R}(2) \subsetneq \mathcal{RPR}(2, 2) \subsetneq \mathcal{R}(2, 2)$;
- $\mathcal{RPR}(3, 2) \not\subseteq \mathcal{R}(2, 2)$ *and* $\mathcal{R}(2, 2) \not\subseteq \mathcal{RPR}(3, 2)$.

Additionally, we prove that for a fixed alphabet Σ of cardinality k, any language over Σ expressible by a cascade of height 2 can also be expressed by a cascade of height 2 where the first component has at most $k + 1$ states. The intuition behind this is that, due to

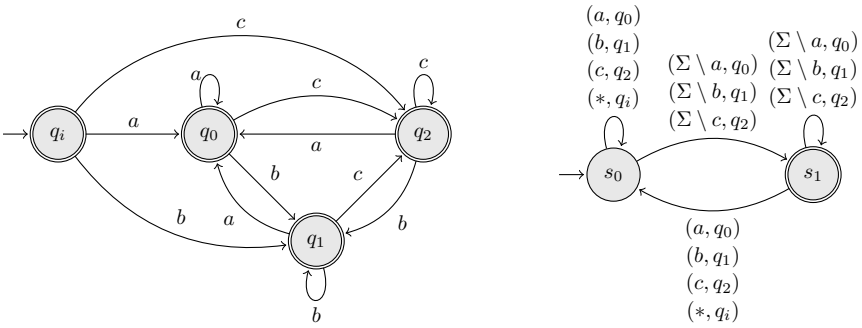

Figure 6 The cascade $C \coloneqq \mathcal{A} \circ \mathcal{B}$ accepts the language $L'_3 \coloneqq \Sigma^* \left(\Sigma^2 \setminus \{aa, bb, cc\}\right)$. When treated as a single automaton, C consists of 8 states, in contrast to the minimal automaton for L'_3, which has only 7 states.

the restriction of transitions to resets or identities, only a finite number of states can be reached from the initial state. This is formalized in the next lemma, which also establishes the optimality of the bound on the number of states of the first component.

▶ **Lemma 5.6.** *Let Σ be a finite alphabet with size $|\Sigma| = k$. Let L be a language such that $L \subseteq \Sigma^*$. For every $m \in \mathbb{N}^{>0}$, if $L \in \mathcal{R}(m, 2)$, then $L \in \mathcal{R}(k+1, 2)$. Furthermore, there exists a language $L'_k \subseteq \Sigma^*$ such that $L'_k \in \mathcal{R}(k+1, 2)$ but $L'_k \notin \mathcal{R}(k, 2)$.*

The language L'_k used to prove the optimality of the bound in Lemma 5.6 is defined as $\Sigma^* \left(\Sigma^2 \setminus \bigcup_{0 \leq i < k} \sigma_i \sigma_i\right)$. As an example, Figure 6 shows the case of L'_3.

5.2 Narrow Cascades

Thus far, our discussion has centered around cascades composed of one or two components. Now, we shift our focus to *narrow cascades*, i.e. cascades of greater height but in which each components is restricted to have two states (i.e. $\mathcal{R}_2^h$). Just as we have seen that some languages cannot be expressed by cascades of height 1, we will demonstrate that for any given height, there exists a language that cannot be captured at that height, provided the components of the cascade are restricted to two states. We call this the *Height-Hierarchy Lemma*, and is a counterpart of the Width-Hierarchy Lemma (Lemma 5.4) focused on the height of cascades. It is based on the following family of languages: for each $h \geq 2$, we consider the language $L_h = \Sigma^{h-2} a \Sigma^*$, that is all words that contain symbol "a" precisely at position $h-1$. The Height-Hierarchy Lemma below proves that, for any $h \geq 2$, the language L_h is *not* definable by cascades of two states reset automata of height less than h.

▶ **Lemma 5.7** (Height-Hierarchy Lemma)**.** *For each $h \geq 2$, let $L_h = \Sigma^{h-2} a \Sigma^*$. It holds that:*

(i) $L_h \in \mathcal{R}_2^h$;

(ii) $L_h \notin \mathcal{R}_2^{h-1}$.

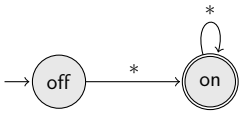

Figure 7 Switch automaton.

We briefly explain the intuition behind Lemma 5.7. Regarding point *(i)* $L_h \in \mathcal{R}_2^h$, the construction of the two-state reset cascade proceeds as follows. The base case corresponds to Figure 1, while the inductive step for height h involves the use of the two-state reset automaton $\mathcal{A}_{switch}$ (illustrated in Figure 7) in cascade with the augmentation of the cascade for the case $h-1$. Intuitively, $\mathcal{A}_{switch}$ recognizes all words containing at least one symbol. Using in cascade $h-2$ copies of $\mathcal{A}_{switch}$ together with the cascade in Figure 1, corresponds exactly to the language $\Sigma^{h-2}a\Sigma^*$. In Figure 8, we provide an example of the construction for the case $h = 3$.

The proof that $L_h \notin \mathcal{R}_2^{h-1}$ is more involved and proceeds by induction on h. For the base case ($h = 2$), we have $L_2 = a\Sigma^*$. This case is verified by Lemma 5.3. For the inductive step, assume that the statement holds for every $i \leq h$. We need to prove that it also holds for $i = h+1$. Suppose, by contradiction, that $L_{h+1} \in \mathcal{R}_2^h$. By definition, this would imply the existence of a cascade $C = \mathcal{A}_1 \circ \cdots \circ \mathcal{A}_h$ of two-state reset automata such that $\mathcal{L}(C) = L_{h+1} = \Sigma^{h-1}a\Sigma^*$. However, starting from C, the proof shows how to construct a new cascade $C' = \mathcal{B}_1 \circ \cdots \circ \mathcal{B}_{h-1}$ of two-state reset automata such that $\mathcal{L}(C') = \Sigma^{h-2}a\Sigma^* = L_h$. The existence of C' contradicts the inductive hypothesis, which states that $L_h \notin \mathcal{R}_2^{h-1}$. Therefore, the assumption that $L_{h+1} \in \mathcal{R}_2^h$ must be false, and the cascade C cannot exist.

5.3 General Cascades

In this subsection, we examine cascades of reset automata without imposing restrictions on the number of states in each component or on the total number of components. In the previous part, Lemma 5.6 demonstrates that, for cascade of two resets over the alphabet Σ, the maximum expressiveness is achieved when the first component has $|\Sigma| + 1$ states. Here, we extend this result to cascades of unbounded height in the *Width-Collapse Lemma*, that provides lower bounds on the number of states in each component, for which adding states at certain levels does not affect expressiveness.

▶ **Lemma 5.8** (Width-Collapse Lemma). *Let Σ be a finite alphabet with $|\Sigma| = k \geq 2$. Let $L \subseteq \Sigma^*$ be a language. For any positive integers h and $k_1, \ldots, k_h$, if $L \in \mathcal{R}(k_1, \ldots, k_h, 2)$, then $L \in \mathcal{R}(f(1), \ldots, f(h), 2)$, where $f(i) = \frac{k^{i+1}-1}{k-1}$.*

We now demonstrate how to transform *general cascades* into narrow cascades. Specifically, we show how any cascade of reset automata (of height h and with k_i states at level i, for each $i \in \{1, \ldots, h\}$) can be transformed into an equivalent *narrow cascade* (i.e. made of two-state resets), at the cost of increasing its height at most by a factor of $2 + \sum_{i=1}^{h-1} \lceil \log_2(k_i) \rceil$. This result is based on two key points:

1. Given a general cascade, we can always append a pure-reset automaton at the end without altering its language;
2. the *Narrowing Lemma*, which we prove below, demonstrates that any cascade of reset automata, whose final component is pure-reset and containing a component $\mathcal{A}_j$ with k_j states (and $k_j > 2$), can be transformed into a new cascade where $\mathcal{A}_j$ is replaced by two new automata, with 2 and $\lceil \frac{k_j}{2} \rceil$ states each.

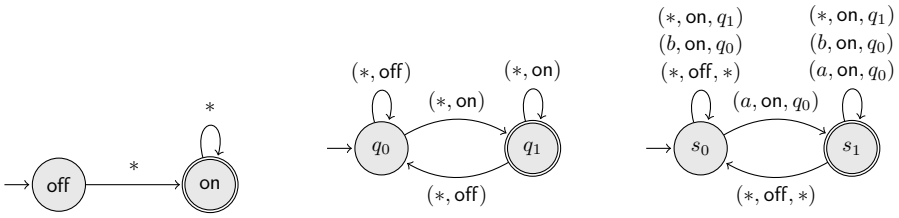

Figure 8 A cascade $C_3 = \mathcal{A}_1 \circ \mathcal{A}_2 \circ \mathcal{A}_3$ that recognizes the language $\Sigma a \Sigma^*$. The first two components enforce that any accepted word contains at least two symbols, as $\mathcal{L}(\mathcal{A}_1 \circ \mathcal{A}_2) = \Sigma\Sigma\Sigma^*$.

Instrumental to the Narrowing Lemma, the following result demonstrates that, given a general cascade whose last component is a pure-reset automaton, we can modify this last component to make all the states of the preceding components final, without altering the recognized language.

▶ **Lemma 5.9.** *Consider a cascade $\mathcal{A} \circ \mathcal{B}$ of automata, where $\mathcal{B}$ is a two-state pure-reset automaton. Let $\mathcal{A}'$ be the automaton obtained from $\mathcal{A}$ by making all states final. Then, there exists a two-state pure-reset automaton $\mathcal{B}'$ such that $\mathcal{L}(\mathcal{A} \circ \mathcal{B}) = \mathcal{L}(\mathcal{A}' \circ \mathcal{B}')$.*

The Narrowing Lemma is stated as follows.

▶ **Lemma 5.10** (Narrowing Lemma)**.** *Let $C = \mathcal{A}_1 \circ \cdots \circ \mathcal{A}_h$ be a cascade where $\mathcal{A}_i$ is a reset automaton with k_i states for each $1 \leq i \leq h-1$, and $\mathcal{A}_h$ is a pure-reset automaton. Let j be an index such that $1 \leq j \leq h-1$. Then, there exists a cascade C' of reset automata*

$$C' = \mathcal{A}'_1 \circ \cdots \circ \mathcal{A}'_{j-1} \circ \mathcal{B}_1 \circ \mathcal{B}_2 \circ \mathcal{A}'_{j+1} \circ \cdots \circ \mathcal{A}'_h$$

such that:

- **(i)** *each $\mathcal{A}'_i$ has k_i states for $i \neq j$;*
- **(ii)** *if $\mathcal{A}_i$ is pure-reset (resp., reset), then also $\mathcal{A}'_i$ is pure-reset (resp., reset), for $i \neq j$;*
- **(iii)** *$\mathcal{B}_1$ has 2 states and $\mathcal{B}_2$ has $\lceil \frac{k_j}{2} \rceil$ states; and*
- **(iv)** *$\mathcal{L}(C) = \mathcal{L}(C')$.*

By iteratively applying the Narrowing Lemma to every component with more than two states, we obtain a procedure that, given a cascade of reset automata, produces an equivalent cascade where all components are two-state reset automata. Moreover, it is worth noticing that:

- **(i)** by Lemma 4.10, *w.l.o.g.* the last component of any cascades of reset (or pure-resets) has two states, and therefore the Narrowing Lemma does not need to be applied at the last level;
- **(ii)** if the final component of a cascade is not a pure-reset, a new pure-reset level can always be added without affecting the language of the cascade.

This leads to the following inclusions.

▶ **Corollary 5.11.** *For each positive $h, k_1, \ldots, k_h$ it holds that*

1. $\mathcal{RPR}(k_1, \ldots, k_h) \subseteq \mathcal{RPR}_2^{H+1}$
2. $\mathcal{R}(k_1, \ldots, k_h) \subseteq \mathcal{R}_2^{H+2}$

where $H = \lceil \log_2 k_1 \rceil + \cdots + \lceil \log_2 k_{h-1} \rceil$.

Combining Lemma 5.8 and Corollary 5.11, we conclude that if a language L is recognized by a cascade of resets of height h, it can also be recognized by a cascade of height $\Theta(h^2)$ composed entirely of two-state resets.

▶ **Corollary 5.12.** *Let Σ be an alphabet such that $|\Sigma| = k \geq 2$ and let $L \subseteq \Sigma^*$ be a language. If L admits a cascade of reset automata of height h, then $L \in \mathcal{R}_2^H$ where $H \in \Theta(h^2)$. If $k = 2$, then $H = \frac{h^2+h+2}{2}$.*

Exploiting the bound for the case $|\Sigma| = 2$, we can prove undefinability of certain languages by *general cascades*, i.e. without any bound on their height nor on the number of states of its component. As an example, by Lemma 5.7, we know that the language $L = \Sigma^6 a \Sigma^*$ over the alphabet $\Sigma = \{a, b\}$ does not belong to the class $\mathcal{R}_2^7$. If L could be recognized by a cascade of height $h = 3$, then it would also be recognized by a two-state cascade of height $H = \frac{h^2+h+2}{2} = 7$, leading to the following conclusion: with $\Sigma = \{a, b\}$, the language $\Sigma^6 a \Sigma^*$ does not admit any cascade of resets of height 3.

Building upon this reasoning, we can formulate the Generalized Height-Hierarchy Lemma. Unlike the original Height-Hierarchy Lemma, which focuses solely on two-state cascades, the generalized version addresses the undefinability of cascades in a broader context, encompassing general cascades.

▶ **Lemma 5.13** (Generalized Height-Hierarchy Lemma)**.** *Let h be a positive integer, and define $H = \frac{h^2+h+2}{2} + 1$. Consider the language $L_H \subseteq \Sigma^*$, where $L_H = \Sigma^{H-2} a \Sigma^*$ and Σ is a two-symbol alphabet. The language L_H cannot be recognized by any cascade of reset automata of height h, but it holds that $L_H \in \mathcal{R}_2^H$.*

6 Efficient closure properties of cascades of reset automata

In this section, we present an efficient method for computing specific closure properties of reset cascades. For instance, for the case in which the operation $\otimes$ is binary, given two cascades of resets $\mathcal{C}$ and $\mathcal{C}'$ (made of only two-states components), we show how it is possible to compute a cascade of two-states resets that recognizes $\mathcal{L}(\mathcal{C}) \otimes \mathcal{L}(\mathcal{C}')$ by adding *at most one* two-state reset automaton (that, in this context, we call *brick*). We show this for the following operations:

(i) intersection;
(ii) complementation;
(iii) union; and
(iv) left-concatenation of Σ, i.e. given a language $\mathcal{L}$ to compute $\Sigma \cdot \mathcal{L}$.[4]

Proposition 4.5 already shows that *intersection* can be implemented efficiently for cascades of resets: given two reset cascades $\mathcal{C}$ and $\mathcal{C}'$ (with m and n two-states components, respectively), there exists a cascade for $\mathcal{L}(\mathcal{C}) \cap \mathcal{L}(\mathcal{C}')$ with $m + n$ two-state resets.

Before showing the construction for the remaining operations, we give the following key definitions. We define the *finalized version* of an automaton $\mathcal{A}$, denoted with $\mathsf{finv}(\mathcal{A})$, as the automaton obtained from $\mathcal{A}$ by setting all its states as final. The definition naturally extends to cascades: the finalized version of $\mathcal{C}$, denoted with $\mathsf{finv}(\mathcal{C})$, is defined as $\mathsf{finv}(\mathcal{A}_1) \circ \cdots \circ \mathsf{finv}(\mathcal{A}_n)$. Clearly, if $\mathcal{C}$ is a cascade of reset automata, $\mathsf{finv}(\mathcal{C})$ is still a cascade of reset automata. We define the *reachability set* of an automaton relative to a set of states as follows.

▶ **Definition 6.1** (Reachability Set)**.** *Let $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, F \rangle$ be an automaton. Let $P \subseteq Q$ be a set of states. The* reachability set of $\mathcal{A}$ with respect to P*, denoted with $\mathsf{RS}(\mathcal{A}, P)$, is the set $\{(\sigma, q) \in (\Sigma \times Q) : \delta(q, \sigma) \in P\}$. We denote with $\overline{\mathsf{RS}(\mathcal{A}, P)}$ the set $(\Sigma \times Q) \setminus \mathsf{RS}(\mathcal{A}, P)$. We write $\mathsf{RS}(\mathcal{A})$ to refer to $\mathsf{RS}(\mathcal{A}, F)$.*

[4] It is worth noticing that this operation corresponds to compute the closure under the LTL *next* modality.

Figure 9 The *negation brick* $\mathsf{negb}(\mathcal{A})$ in the two cases: (a) $\epsilon \in \mathcal{L}(\mathcal{A})$ (b) $\epsilon \notin \mathcal{L}(\mathcal{A})$.

We now show how to efficiently compute the remaining closure properties.

Complementation

To compute complementation, we introduce the *negation brick*, whose structure is illustrated in Figure 9 and is formally defined here below.

▶ **Definition 6.2** (Negation brick). *Let $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, F\rangle$ be an automaton. The* negation brick for $\mathcal{A}$, *denoted with* $\mathsf{negb}(\mathcal{A})$, *is the two-state pure-reset automaton $\langle \Sigma \times Q, \{n_0, n_1\}, \delta, n_0, \{n_f\}\rangle$ such that:*

- **(i)** *the final state n_f is n_1 if and only if $\epsilon \in \mathcal{L}(\mathcal{A})$;*
- **(ii)** *the function τ induced by symbols in $\mathsf{RS}(\mathcal{A})$ maps all states in the non-final state, i.e. $\tau : \{n_0, n_1\} \mapsto \{n_0, n_1\} \setminus \{n_f\}$;*
- **(iii)** *the function τ' induced by symbols in $\overline{\mathsf{RS}(\mathcal{A})}$ maps all states in the final one, i.e. $\tau : \{n_0, n_1\} \mapsto \{n_f\}$.*

The intuition is that the negation brick, when appended to the end of a cascade $\mathcal{C}$, reaches its final state if and only if the underlying cascade $\mathcal{C}$ is not in a final state. Consequently, by setting all the states of $\mathcal{C}$ as final, we obtain a cascade that recognizes the complement of $\mathcal{L}(\mathcal{C})$, as proved by the following lemma.

▶ **Lemma 6.3.** *Let $\mathcal{C}$ be a cascade of automata. The cascade $\mathcal{C}' \coloneqq \mathsf{finv}(\mathcal{C}) \circ \mathsf{negb}(\mathcal{C})$ recognizes the language $\overline{\mathcal{L}(\mathcal{C})}$. Moreover, if $\mathcal{C}$ is a cascade of reset automata, then so is $\mathcal{C}'$.*

Interestingly, if the cascade terminates with a pure-reset layer $\mathcal{A}$, this automaton can itself serve the function of the negation brick, without the need of an additional component.

▶ **Lemma 6.4.** *Let $\mathcal{C} = \mathcal{A}_1 \circ \cdots \circ \mathcal{A}_n$ be a cascade of automata such that $\mathcal{A}_n$ is a pure-reset automaton. There exists a cascade $\mathcal{C}' = \mathcal{A}'_1 \circ \cdots \circ \mathcal{A}'_n$ such that:*

- **(i)** *$\mathcal{L}(\mathcal{C}') = \overline{\mathcal{L}(\mathcal{C})}$;*
- **(ii)** *each automaton $\mathcal{A}'_i$ has the same number of states as $\mathcal{A}_i$;*
- **(iii)** *if $\mathcal{A}_i$ is a reset (resp., pure-reset), then $\mathcal{A}'_i$ is also a reset (resp., pure-reset).*

Union

Given two cascades $\mathcal{C}$ and $\mathcal{C}'$ of height m and n, respectively, since $\mathcal{L}(\mathcal{C}) \cup \mathcal{L}(\mathcal{C}') = \overline{\overline{\mathcal{L}(\mathcal{C})} \cap \overline{\mathcal{L}(\mathcal{C}')}}$, it is possible to build cascade for $\mathcal{L}(\mathcal{C}) \cup \mathcal{L}(\mathcal{C}')$ of height $m + n + 3$, using the previously discussed constructions. In this section, we present a more efficient construction that introduces only one additional component, referred to as the *union brick*, resulting in a cascade for $\mathcal{L}(\mathcal{C}) \cup \mathcal{L}(\mathcal{C}')$ of height $n + m + 1$.

▶ **Definition 6.5** (Union brick). *Let $\mathcal{A} = \langle \Sigma, Q_A, \delta_A, q_{0A}, F_A\rangle$ and $\mathcal{B} = \langle \Sigma, Q_B, \delta_B, q_{0B}, F_B\rangle$ be two automata. Let $U \subseteq Q_A \times Q_B$ the set of states $\{(q_A, q_B) : q_A \in F_A \vee q_B \in F_B\}$. Let $\mathcal{C} = \mathcal{A} \circ \mathsf{aug}(\mathcal{A}, \mathcal{B})$. The* union brick of $\mathcal{A}$ and $\mathcal{B}$, *denoted with* $\mathsf{unionb}(\mathcal{A}, \mathcal{B})$, *is the two-state pure-reset automaton $\langle \Sigma \times Q, \{u_0, u_1\}, \delta, u_0, \{u_f\}\rangle$ such that:*

(i) *the final state u_f is u_0 if and only if $\epsilon \in \mathcal{L}(\mathcal{A}) \cup \mathcal{L}(\mathcal{B})$;*
(ii) *the function τ induced by symbols in $\mathsf{RS}(\mathcal{C}, U)$ maps all states in the final one, i.e. $\tau : \{n_0, n_1\} \mapsto \{n_f\}$;*
(iii) *the function τ' induced by symbols in $\overline{\mathsf{RS}(\mathcal{C}, U)}$ maps all states in the non-final state, i.e. $\tau : \{n_0, n_1\} \mapsto \{n_0, n_1\} \setminus \{n_f\}$.*

Similarly to the case of complementation, when appended to the end of a cascade $\mathcal{C} \circ \mathsf{aug}(\mathcal{C}, \mathcal{C}')$, the union brick reaches its final state if and only if either $\mathcal{C}$ is in a final state or $\mathsf{aug}(\mathcal{C}, \mathcal{C}')$ is in a final state. This leads to the following lemma.

▶ **Lemma 6.6.** *Let $\mathcal{C}$ and $\mathcal{C}'$ be two cascade of automata. The cascade $\mathcal{C}'' := \mathsf{finv}(\mathcal{C} \circ \mathsf{aug}(\mathcal{C}, \mathcal{C}')) \circ \mathsf{unionb}(\mathcal{C}, \mathcal{C}')$ recognizes the language $\mathcal{L}(\mathcal{C}) \cup \mathcal{L}(\mathcal{C}')$. Moreover, if $\mathcal{C}$ and $\mathcal{C}'$ are cascades of reset automata, then so is $\mathcal{C}''$.*

Also in this case, if one of the two automata corresponds to a cascade terminating with a pure-reset component $\mathcal{A}$, the union can be performed without the need for additional layers: the automaton $\mathcal{A}$ effectively serves as the union brick.

Left-concatenation of Σ

Given a cascade $\mathcal{C}$, we demonstrate how to construct a cascade that recognizes the language $\Sigma \cdot \mathcal{L}(\mathcal{C})$, adding only one brick and guaranteeing that the property of being a reset cascade is preserved. As a by-product of this construction, we obtain that, given a cascade of resets of height h equivalent to an LTL formula ϕ (interpreted over finite words), it is possible to construct a cascades of resets for $\mathsf{X}(\phi)$ of height $h + 1$, where X is the *next* modality of LTL.

We first define the *next version* of an automaton. The next version of an automaton $\mathcal{A}$, denoted as $\mathsf{nextv}(\mathcal{A})$, is defined considering the Cartesian product between the alphabet of $\mathcal{A}$ and the set $\{\mathsf{off}, \mathsf{on}\}$. Intuitively, if $\mathcal{A}$ transitions from q to q' with a symbols σ, so does $\mathsf{nextv}(\mathcal{A})$ with the symbol (σ, on). On the contrary, all symbols (σ, off) force $\mathsf{nextv}(\mathcal{A})$ to transition to the initial state. The formal definition of $\mathsf{nextv}(\mathcal{A})$ is given here below.

▶ **Definition 6.7** (Next version of an automaton)**.** *Let $\mathcal{A} = \langle \Sigma', Q, \delta, q_0, F \rangle$ be an automaton such that either $\Sigma' = \Sigma$ or $\Sigma' = \Sigma \times S$, for an arbitrary finite set S. We define the* next version of $\mathcal{A}$*, denoted as $\mathsf{nextv}(\mathcal{A})$, as the automaton $(\Sigma'', Q, \delta', q_0, F)$ such that:*

- *if $\Sigma' = \Sigma$, then $\Sigma'' := \Sigma \times \{\mathsf{off}, \mathsf{on}\}$ and, for all $q \in Q$ and for all $a \in \Sigma$, it holds: $\delta'(q, (a, \mathsf{on})) = \delta(q, a)$ and $\delta'(q, (*, \mathsf{off})) = q_0$.*
- *if $\Sigma' = \Sigma \times S$, then $\Sigma'' := \Sigma \times \{\mathsf{off}, \mathsf{on}\} \times S$ and, for all $q \in Q$ and for all $(a, s) \in \Sigma \times S$, it holds that: $\delta'(q, (a, \mathsf{on}, s)) = \delta(q, (a, s))$ and $\delta'(q, (*, \mathsf{off}, *)) = q_0$.*

Given a cascade $\mathcal{C} = \mathcal{A}_1 \circ \cdots \circ \mathcal{A}_n$ over Σ, we define the next version of $\mathcal{C}$*, denoted with $\mathsf{nextv}(C)$, as the cascade $\mathsf{nextv}(\mathcal{A}_1) \circ \cdots \circ \mathsf{nextv}(\mathcal{A}_n)$ over $\Sigma \times \{\mathsf{off}, \mathsf{on}\}$.*

Figure 10 shows the next versions of the automata in Figure 1. Crucially, computing the next version of an automaton does not alter its property of being a reset automaton.

▶ **Lemma 6.8.** *Let Σ be a finite alphabet and let $\mathcal{A}$ be an automaton over Σ or over $\Sigma \times S$ for an arbitrary finite set S. If $\mathcal{A}$ is reset automaton, then also $\mathsf{nextv}(\mathcal{A})$ is a reset automaton.*

Now, given any cascade $\mathcal{C}$, to capture the language $\Sigma \cdot \mathcal{L}(\mathcal{C})$, it suffices to consider the automaton $\mathcal{A}_{switch}$ (depicted in Figure 7) with the next version of $\mathcal{C}$. In fact, considering that initially both $\mathcal{A}_{switch}$ and $\mathsf{nextv}(\mathcal{C})$ are in their initial states (which, for $\mathcal{A}_{switch}$, is state off), reading the first input symbol σ forces:

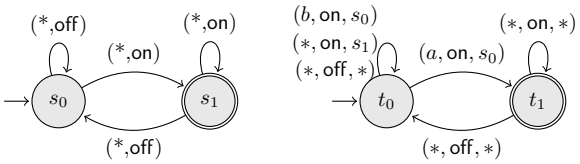

Figure 10 The next version of the two automata in the cascade of Figure 1.

(i) $\mathcal{A}_{switch}$ to transition to state on; and
(ii) $\mathsf{nextv}(\mathcal{C})$ to remain in its initial state, because the symbol it reads is (σ, off).

After the first symbol and for all the rest of the input word, $\mathcal{A}_{switch}$ remains in state on, while $\mathsf{nextv}(\mathcal{C})$ operates like $\mathcal{C}$ because it reads symbols of the form (σ', on). As shown by the following lemma, this captures exactly $\Sigma \cdot \mathcal{L}(\mathcal{C})$.

▶ **Lemma 6.9.** *Let $\mathcal{C}$ be a cascade of automata. The cascade $\mathcal{C}' := \mathcal{A}_{switch} \circ \mathsf{nextv}(\mathcal{C})$ recognizes the language $\Sigma \cdot \mathcal{L}(\mathcal{C})$. Moreover, if $\mathcal{C}$ is a cascade of reset automata, then so is $\mathcal{C}'$.*

From Lemma 5.7, it follows the optimality of the construction outlined in Lemma 6.9.

7 Conclusions and Future Work

In this paper, we investigated some fundamental properties of cascades of reset automata. Unlike the approach commonly followed in the literature, where the cascade product is restricted to semi-automata, we focused on the case of automata. This allowed us to study the properties of the recognized languages. As an initial step, we showed how to compute regular expressions equivalent to a cascade. Then, on the basis of such a transformation, we established some meaningful expressiveness results, in particular lower bounds to the height and to the minimum number of states per level of a cascade of resets for specific families of languages. Finally, we showed how to compute the closure of reset cascades under certain basic operations by adding at most one brick to the end of the cascade.

As for the future developments of the work, finding an efficient construction for the closure of reset cascades under the concatenation operation is undoubtedly a crucial direction. This would enable the design of an efficient approach to handling the *eventually* and *until* operators of LTL, providing, together with the results given in the last section of the paper, an efficient decomposition into reset cascades for full LTL. This would improve the triply-exponential upper bound to such a decomposition achieved by Maler's algorithm [4, 15]. Last but not least, giving analogous expressiveness and closure results for permutation automata appears to be another promising avenue for further investigation.

References

1 Dana Angluin, David Chiang, and Andy Yang. Masked hard-attention transformers and boolean RASP recognize exactly the star-free languages. *CoRR*, abs/2310.13897, 2023. doi:10.48550/arXiv.2310.13897.

2 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.

3 Alessandro Artale, Luca Geatti, Nicola Gigante, Andrea Mazzullo, and Angelo Montanari. Succinctness issues for LTLf and safety and cosafety fragments of LTL. *Information and Computation*, 302:105262, 2025. doi:10.1016/j.ic.2024.105262.

4 Giuseppe De Giacomo, Antonio Di Stasio, Francesco Fuggitti, and Sasha Rubin. Pure-past linear temporal and dynamic logic on finite traces. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 4959–4965, 2021.

5 Giuseppe De Giacomo and Moshe Y. Vardi. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In Francesca Rossi, editor, *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, pages 854–860. IJCAI/AAAI, 2013. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6997.

6 Samuel Eilenberg. *Automata, languages, and machines., B.* Pure and applied mathematics. Academic Press, 1976. URL: https://www.worldcat.org/oclc/310535259.

7 Javier Esparza, Rubén Rubio, and Salomon Sickert. Efficient Normalization of Linear Temporal Logic. *J. ACM*, 71(2):16:1–16:42, 2024. doi:10.1145/3651152.

8 Marcus Gelderie. Classifying regular languages via cascade products of automata. In *Language and Automata Theory and Applications: 5th International Conference, LATA 2011, Tarragona, Spain, May 26-31, 2011. Proceedings 5*, pages 286–297. Springer, 2011. doi:10.1007/978-3-642-21254-3_22.

9 Mark Kambites. On the krohn-rhodes complexity of semigroups of upper triangular matrices. *Int. J. Algebra Comput.*, 17(1):187–201, 2007. doi:10.1142/S0218196707003548.

10 Nadezda Alexandrovna Knorozova and Alessandro Ronca. On the expressivity of recurrent neural cascades. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan, editors, *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 10589–10596. AAAI Press, 2024. doi:10.1609/AAAI.V38I9.28929.

11 Nadezda Alexandrovna Knorozova and Alessandro Ronca. On the expressivity of recurrent neural cascades with identity. In Pierre Marquis, Magdalena Ortiz, and Maurice Pagnucco, editors, *Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning, KR 2024, Hanoi, Vietnam. November 2-8, 2024*, 2024. doi:10.24963/KR.2024/82.

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 Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL: https://openreview.net/forum?id=De4FYqjFueZ.

14 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.

15 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.

16 Zohar Manna and Amir Pnueli. A hierarchy of temporal properties (invited paper, 1989). In *Proceedings of the 9th annual ACM symposium on Principles of distributed computing*, pages 377–410, 1990. doi:10.1145/93385.93442.

17 Nicolas Markey. Temporal logic with past is exponentially more succinct, concurrency column. *Bull. EATCS*, 79:122–128, 2003.

18 Robert McNaughton and Seymour A Papert. *Counter-Free Automata (MIT research monograph no. 65)*. The MIT Press, 1971.

19 Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57. IEEE, 1977. doi:10.1109/SFCS.1977.32.

20 Alessandro Ronca. The transformation logics. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024*, pages 3549–3557. ijcai.org, 2024. URL: https://www.ijcai.org/proceedings/2024/393.

21 Alessandro Ronca, Nadezda Alexandrovna Knorozova, and Giuseppe De Giacomo. Automata cascades: Expressivity and sample complexity. In Brian Williams, Yiling Chen, and Jennifer Neville, editors, *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, pages 9588–9595. AAAI Press, 2023. doi:10.1609/AAAI.V37I8.26147.

22 Karl-Heinz Zimmermann. On Krohn-Rhodes theory for semiautomata. *CoRR*, abs/2010.16235, 2020. arXiv:2010.16235.