Layered automata: A canonical model for automata over infinite words

Antonio Casares
University Kaiserslautern-Landau, Germany

Christof Löding
RWTH Aachen University, Germany

Igor Walukiewicz
CNRS, Bordeaux University, France

Abstract

We introduce layered automata, a subclass of alternating parity automata that generalises deterministic automata. Assuming a consistency property, these automata are history deterministic and 0-1 probabilistic. We show that every omega-regular language is recognised by a unique minimal consistent layered automaton, and that this canonical form can be computed in polynomial time from every layered or deterministic automaton. We further establish that for layered automata both consistency checking and inclusion testing can be performed in polynomial time. Much like deterministic finite automata, minimal consistent layered automata admit a characterisation based on congruences.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Omega automata, history determinism, canonical automata

Funding *Antonio Casares*: Partially supported by the European Research Council under the European Union's Horizon 2020 research and innovation programme, number 101089343. Part of this work was done while Casares was at the University of Warsaw, Poland, supported by the Polish National Science Centre (NCN) grant "Polynomial finite state computation" (2022/46/A/ST6/00072).

Acknowledgements We thank Thomas Colcombet, Aditya Prakash and Pierre Ohlmann for interesting discussions on the subject.

This document contains hyperlinks. Each occurrence of a notion is linked to its *definition*. On an electronic device, the reader can click on words or symbols (or just hover over them on some PDF readers) to see their definition.

Contents

1 Introduction

Automata over infinite words (ω-automata in the following) were first introduced by Büchi to show the decidability of monadic second-order logic over $(\mathbb{N}, <)$ [Büc62]. Since then, they have found numerous applications in the verification and synthesis of reactive systems [CES86, PR89, Kup18].

A central limitation of current ω-automata theory is the lack of minimal canonical models. Minimisation is instrumental for practical applications, particularly when automata are derived from specifications, like LTL formulas, through hierarchical translations. In this process, each constructor in the specification corresponds to an operation on the automaton — some of which necessitate a powerset construction. We know from the case of finite words and trees that minimisation after each operation is key to efficient translations [KMS02]. Minimisation is also important for theory. One immediate example is learning algorithms for ω-regular languages, but more broadly, it indicates that we have gained insights into the structural properties of acceptors. Such insights may eventually allow us to revisit long-standing open problems, such as the index problem for tree automata [CL08, IL25].

Contrary to the case of automata over finite words, deterministic ω-automata do not have minimal canonical models, as a language may admit several non-isomorphic deterministic automata with a minimal number of states. Moreover, the minimisation problem is NP-hard for all kinds of deterministic ω-automata expressive enough to recognise all ω-regular languages [Sch10, Cas22, AE25]. While there are some canonical ways to represent ω-regular languages, such as ω-semigroups or congruences, these representations are potentially exponentially larger than deterministic parity automata, and furthermore, they cannot be used directly in applications from verification and synthesis.

History deterministic (HD) automata are a subclass of non-deterministic automata for which the non-determinism can be resolved on the fly. They were originally introduced under the name of *good for games* [HP06][1], as they are exactly those automata that can be composed with any game while preserving the winner. This property allows their use in applications in verification and synthesis instead of deterministic automata. Every deterministic automaton is history deterministic, but a history deterministic automaton can be exponentially smaller than the smallest deterministic one [KS15]. History determinism has been generalised to alternating ω-automata [Col13, BL19]. History deterministic alternating automata have the same good compositionality properties making them suitable for applications in reactive synthesis, while being even smaller than non-deterministic HD automata [BKLS20, Lemma 6].

Other types of automata that are actively studied recently are those in which the non-determinism can be resolved randomly [AK15, HPT25, PPS$^+$25], or automata that compose faithfully with MPDs [HPS$^+$20]. In particular, *0-1 probabilistic automata* are automata in which a random walk over an infinite word is accepting with probability 1 if and only if the word belongs to the semantics of the automaton; otherwise, it is rejecting with probability 1. Such automata are in particular suitable for the study of MDPs [HPT25].

In this work, we propose a new model of acceptors for ω-regular languages that we call layered automata. Our starting point for this model is the result of Abu Radi and Kupferman [AK22] stating that (transition-based) history deterministic coBüchi automata can be minimised in polynomial time and admit a canonical minimal representation. In [LW25] it has furthermore been shown that this canonical automaton can be described in a natural way using a congruence over pairs of finite words. However, HD coBüchi automata

[1] The related notion of *good for trees automata* was previously introduced in [KSV96].

can only recognise a fragment of ω-regular languages: those that are in the Σ_2^0 level in the Borel hierarchy, or equivalently, persistent properties in the Manna-Pnueli hierarchy [MP90].

The model of layered automata that we propose in this work can be seen as an extension of the minimisation result of Abu Radi and Kupferman to all ω-regular languages. This is in line with some recent works such as *chains of coBüchi automata* [ES22] and *rerailing automata* [Ehl25] to which we compare in Section 3.6. Orthogonally, layered automata can also be seen as a generalisation of the Zielonka tree of a Muller language [Zie98] and of signature automata for positional languages [CO24]. We provide ample evidence that layered automata have good structural and algorithmic properties.

Contributions

We introduce *layered automata*, a formalism to represent all ω-regular languages. One can think of a layered automaton as a representation of an alternating parity automaton with some good structural properties. If a layered automaton satisfies an additional property of *consistency*, then the associated alternating automaton is history deterministic.

Consistent layered automata combine the structure of both minimal history deterministic coBüchi automata (Proposition 3.9) and Zielonka trees. They also encompass deterministic parity automata (Proposition 3.8). They enjoy remarkable good properties. Most notably, they admit a canonical minimal form that is computable in polynomial time. This canonicity result can be stated as follows:

▶ **Theorem 1.1.** *Every ω-regular language L can be recognised by a consistent layered automaton $\mathcal{A}_L$ such that every equivalent consistent layered automaton contains an equivalent subautomaton admitting a surjective morphism to $\mathcal{A}_L$. Moreover, $\mathcal{A}_L$ can be computed in polynomial time from any given consistent layered automaton recognising L.*

We detail further properties of layered automata.

Applicability. Consistent layered automata are representations of alternating parity automata that are guaranteed to be both history deterministic and 0-1 probabilistic (Propositions 3.19 and 3.20). Therefore, consistent layered automata have the key properties that make them suitable for applications in verification and reactive synthesis. Moreover, they provide an easy-to-check sufficient condition for HDness and being 0-1 probabilistic, as layered automata are defined syntactically and checking consistency can be done in polynomial time (Proposition 3.26).

As layered automata generalise minimal coBüchi automata, they can be exponentially smaller than minimal deterministic parity automata [KS15].

We also show that emptiness and inclusion of consistent layered automaton can be done in polynomial time (Propositions 3.27 and 3.28).

Canonicity, morphisms, and congruences. We show that every ω-regular language can be recognised by a unique minimal consistent layered automaton. More precisely, we show that for every ω-regular language L there is a consistent layered automaton $\mathcal{A}_L$ that is normal, centralised and safe minimal and any layered automaton with these three properties is isomorphic to $\mathcal{A}_L$ (Theorem 4.1). Moreover, every consistent layered automaton recognising L contains a subautomaton that admits a surjective morphism to $\mathcal{A}_L$ (Corollary 5.2). Furthermore, we provide a characterisation in terms of congruences for the minimal layered automaton $\mathcal{A}_L$ (Theorem 6.14). This characterisation is purely based on the language L, and does not depend on a given representation as an automaton. To the best of our knowledge, these are the first results of this type for an automaton model capable to recognise all ω-regular languages and having the minimal form never bigger than deterministic parity automata.

Minimisation in PTIME. We show that given a consistent layered automaton $\mathcal{A}$, we can build in polynomial time the minimal layered automaton for $L(\mathcal{A})$ (Theorem 5.1). As every deterministic parity automaton is trivially a consistent layered automaton, this gives an efficient minimisation procedure for deterministic parity automata into layered automata. This does not contradict the above-mentioned lower bounds on minimisation, as the result is an alternating automaton.

Relation to other models. We provide some examples and indicate some results comparing and relating layered automata to other models like chains of coBüchi automata and Zielonka trees. We also discuss how we can use layered automata to characterise important properties of ω-regular languages, such as positionality, following [CO24]. While we leave a more detailed study of all these relations as future work, our discussion shows that layered automata have many interesting connections to various other well-established and more recent formalisms from the theory of ω-regular languages.

Related work

The starting point of our work is the minimisation procedure for HD coBüchi automata from Abu Radi and Kupferman [AK22]. By duality, this entails a minimisation procedure for universal HD Büchi automata, and the notion of layered automata can be seen as a natural extension to arbitrary parity indices that mixes nondeterministic and universal branching. They are also inspired by signature and ε-complete automata from [CO24, Sect 3.3], as well as the minimisation procedure for them. The proof that all layered automata are history deterministic generalises ideas from [KS15, Lemma 3]. The congruence-based characterisation is an extension of the congruences for coBüchi automata from [LW25, Sect 3], using conditions closely related to the priority mapping on finite words defined in [BL24, Def 3.1,3.5].

Various representations of ω-regular languages admitting – in some sense – canonical models exist in the literature. These include some algebraic approaches based on congruences, as Arnold's congruence [Arn85], Wilke-algebras and ω-semigroups [Wil91, PP95], families of right congruences [MS97, Def. 5] and families of DFAs [Kla94, AF16]. These objects have the disadvantage that they can be exponentially larger than deterministic parity automata (it is folklore that already for regular languages of finite words, the syntactic monoid can be of size n^n for the minimal DFA being of size n, see [HK02, Thm. 3] for a concrete example; this exponential gap transfers to all the aforementioned formalisms). While their size is incomparable to the size of a smallest DPA for a language, they cannot directly be used in applications like synthesis.

Another representation, taking the form of a specific deterministic parity automata (DPA), is proposed in [BL24, Sect. 3]. There, the "precise DPA" of a language is defined as a product of deterministic Mealy machines that are extracted directly from the language in terms of a coloring function. However, the precise DPA can be exponentially larger than a smallest DPA.

More recently, some canonical representations based on HD automata have been introduced. In 2022, Ehlers and Schewe proposed to represent an ω-regular language as a decreasing chain of coBüchi-recognisable languages [ES22]. Each of these languages can be given by the minimal HD coBüchi automaton from Abu Radi and Kupferman, leading to a canonical representation. This formalism has subsequently been studied under the name *COCOA* [EK24a, EK24b]. Minimal layered automata are clearly related to COCOA, but there are some important differences. A key difference is that layered automata bring a direct connection with alternating history deterministic automata – which are widely accepted as the class of ω-automata suitable for verification and synthesis. Notably, we conjecture that canonical layered automata are minimal amongst all history deterministic alternating parity automata (Conjecture 7.1). The size comparison between both models is a bit subtle. While COCOA can be smaller than layered automata, in order to obtain an HD automaton from a COCOA, one needs to perform a product of all its components [EK24a]. We give a more detailed comparison with COCOA in Section 3.6.

In 2025, Ehlers introduced *rerailing automata* [Ehl25, Def. 1] and shows that they can be minimised efficiently. These are automata where each run is associated with a priority (the minimal appearing infinitely often), and a word is accepted if the maximal priority of its runs is even. Moreover, they must satisfy a semantic "rerailing property", inspired from the properties of minimal HD coBüchi automata. In [Ehl25, Sect. VI], rerailing automata are shown to be usable in reactive synthesis, but their exact relation with HD automata is not yet established.

2 Preliminaries

We let $[d] = \{1, \dots, d\}$, and we call the elements of $[d]$ *priorities*. We use the variables x, y to denote priorities. The *parity condition* is the language over $[d]^\omega$ given by:

$$\mathsf{Parity} = \{x_1 x_2 \cdots \in [d]^\omega \mid \liminf x_i \text{ is even}\}. \text{ (Note the use of min-parity in this paper.)}$$

Transition systems. A *transition system* over a set Σ is an edge-labelled directed graph $\mathcal{T} = (Q, \Delta)$, where $\Delta \subseteq Q \times \Sigma \times Q$. If $(q, a, q') \in \Delta$, we also write $q \xrightarrow{a} q'$. It is *complete* if for all $q \in Q$ and $a \in \Sigma$, there is some $(q, a, q') \in \Delta$. It is *deterministic* if for every $q \in Q$ and $a \in \Sigma$ there is at most one q' such that $(q, a, q') \in \Delta$. We prefer to represent the transitions of deterministic transition systems by a function $\delta \colon Q \times \Sigma \to Q \cup \{\bot\}$, where $\delta(q, a) = q'$ if $(q, a, q') \in \Delta$, and $\delta(q, a) = \bot$ if no such transition exists.

For a word $w \in \Sigma^*$ we write $q \xrightarrow{w} q'$ if there is a path from q to q' whose label is w, which we call a *run*. For $w \in \Sigma^* \cup \Sigma^\omega$ we write $q \xrightarrow{w}$ if there is a path with label w starting in q, and we write $q \xrightarrow{w} \bot$ it there is no path with label w from q.

If $\mathcal{T}$ is a transition system over a product alphabet $\Sigma \times [d]$, by an abuse of notation we call a *run over* $w \in \Sigma^\omega$ any path having w as labels in the Σ-component. In this case, we also say that a run *is accepting* if its sequence of $[d]$-labels satisfies the parity condition. We write $q \xrightarrow{a:x} q'$ if $(q, (a, x), q') \in \mathcal{T}$, and $q \xrightarrow{a:\geq x} q'$ if $(q, (a, y), q') \in \mathcal{T}$ for some $y \geq x$, similarly for $> x$, etc. We write $q \xrightarrow{w:\geq x} q'$ if the minimal priority over the path is $\geq x$, similarly for $> x$, etc.

A *morphism* between two transition systems $\mathcal{T} = (Q, \Delta)$, $\mathcal{T}' = (Q', \Delta')$ over the same set Σ is a function $\mu \colon Q \to Q'$ such that $(p, a, q) \in \Delta \implies (\mu(p), a, \mu(q)) \in \Delta'$. Note that any run over w in $\mathcal{T}$ from q maps to a run over w in $\mathcal{T}'$ from $\mu(q)$.

Deterministic parity automata. A *deterministic parity automaton* is a complete deterministic transition system $\mathcal{A} = (Q, \Delta)$ over the set $\Sigma \times [d]$ with a designated initial state $q_{\mathsf{init}} \in Q$.

A word $w \in \Sigma^\omega$ is accepted by the automaton if the unique run over w satisfies the parity condition. The language of the automaton is the set of words it accepts. A language is *ω-regular* if it can be recognised by a deterministic parity automaton. The class of ω-regular languages admits many equivalent definitions, for instance, through definability in MSO logic, or recognition by non-deterministic Büchi automata, see for instance [GTW02].

Alternating parity automata. In full generality, an *alternating parity automaton* is given by a set of states together with a transition function of the form $\delta\colon Q \times \Sigma \to \mathsf{Bool}^+(Q \times [d])$. In this work, all alternating automata are of a simpler form, which we call *simple by priorities alternating automata*. These correspond to automata where all transitions are either

$$\delta(q,a) = \bigwedge_{p \in Q'} (p,x) \text{ for some even } x, \quad \text{or} \quad \delta(q,a) = \bigvee_{p \in Q'} (p,x) \text{ for some odd } x.$$

We refer to [BL19] for definitions in the general case. We give next a formal, alternative presentation of the subclass of simple by priorities alternating automata.

A *simple by priorities* automaton is a complete transition system $\mathcal{A} = (Q, \Delta)$ over the set $\Sigma \times [d]$ with a designated *initial state* $q_{\mathsf{init}} \in Q$, satisfying that for every $q \in Q$ and $a \in \Sigma$ all a-transitions from q have the same priority. So there is x a *priority of action a* in q, meaning that for all $(q,(a,y),q') \in \Delta$ we have $y = x$.

The semantics of a simple by priorities automaton is defined as follows. Given an input word $w = a_1 a_2 \cdots \in \Sigma^\omega$, we define the game $\mathcal{G}(\mathcal{A}, w)$ where two players, called Eve and Adam, build a run over w in $\mathcal{A}$ by turns:

- The initial position is the state $q_1 = q_{\mathsf{init}}$.
- Suppose the run constructed in a play after reading first $i-1$ letters ends in a state q_i, and x_i is the priority of a_i in q_i. If x_i is odd then Eve picks a transition $q_i \xrightarrow{a_i:x_i} q_{i+1}$, otherwise Adam makes the choice.
- Eve wins if the constructed path is accepting.

We say that w is *accepted* by $\mathcal{A}$ if Eve has a winning strategy in the game $\mathcal{G}(\mathcal{A}, w)$. Note that, by determinacy of parity games [Mar75], on the contrary Adam has a winning strategy. The *language recognised* by $\mathcal{A}$, written $L(\mathcal{A})$, is the set of words it accepts. The *language of a state* $q \in Q$ is the language recognised by the automaton obtained by fixing q as the initial state. We denote it $L(\mathcal{A}, q)$, or just $L(q)$ if $\mathcal{A}$ is clear from the context.

Semantic determinism. An automaton $\mathcal{A}$ is *semantically deterministic* if whenever we have $p \xrightarrow{a} q_1$ and $p \xrightarrow{a} q_2$ then $L(\mathcal{A}, q_1) = L(\mathcal{A}, q_2)$. Equivalently, if $p \xrightarrow{a} q$ then $L(\mathcal{A}, q) = a^{-1}L(\mathcal{A}, p)$.

▶ Remark 2.1. Let $\mathcal{A}$ be a semantically deterministic automaton recognising a language L. Let u be a word labelling a path from the initial state to q. Then, the language of q is:

$$u^{-1}L = \{w \in \Sigma^\omega \mid uw \in L\}.$$

Resolvers and history determinism. Consider a simple by priorities automaton $\mathcal{A}$. A *resolver* for Eve in $\mathcal{A}$ is a function $\sigma\colon (\{q_{\mathsf{init}}\} \cup \Delta^+) \times \Sigma \to Q$, satisfying that whenever a run $\rho \in (\{q_{\mathsf{init}}\} \cup \Delta^+)$ ends in a state q, and the priority of a in q is odd then $q' = \sigma(\rho, a)$ is an a-successor of q, namely $q \xrightarrow{a} q'$. A run is *consistent with* σ (or a *σ-run*) if for every prefix

$\rho \cdot (q, (a, x), q')$ with x odd, we have $\sigma(\rho, a) = q'$. The *language accepted by an Eve-resolver* σ is

$$L(\mathcal{A}, \sigma) = \{w \in \Sigma^\omega \mid \text{ every } \sigma\text{-run over } w \text{ is accepting}\}.$$

Note that we always have $L(\mathcal{A}, \sigma) \subseteq L(\mathcal{A})$. We say that the Eve-resolver σ is *valid* if $L(\mathcal{A}, \sigma) = L(\mathcal{A})$. We define symmetrically *resolvers for Adam* and their languages (note that if τ is an Adam-resolver, then $L(\mathcal{A}, \tau) \supseteq L(\mathcal{A})$). For two fixed resolvers σ and τ for Eve and Adam, respectively, every word admits a unique run that is consistent with both σ and τ. We write $L(\mathcal{A}, \sigma, \tau)$ for the set of words for which this run is accepting.

An alternating automaton is *history deterministic* (HD) if both Eve and Adam have valid resolvers. That is, if there are resolvers σ and τ for Eve and Adam such that

$$L(\mathcal{A}) = L(\mathcal{A}, \sigma) = L(\mathcal{A}, \tau) = L(\mathcal{A}, \sigma, \tau).$$

3 Layered automata

In this section we introduce layered automata, the main notion of this paper. We first define them syntactically. Then, we propose two equivalent ways of defining the language they recognise. One of them consists in associating to a layered automaton an alternating parity automaton (Section 3.2). Actually, the structure of a layered automaton imposes the additional constraints of simple by priorities alternating parity automata; so another way to define the same class of alternating automata is to add some more constraints on their transition functions, as we describe in Section 3.2.3. We introduce uniform semantic determinism, a slightly stronger notion than semantic determinism. We show that for our class of alternating automata, uniform semantic determinism implies that automata are history deterministic and 0-1 probabilistic (Propositions 3.19 and 3.20) For this, we give in Section 3.3 a characterisation of acceptance by layered automata that does not use alternation. We also introduce a notion of consistency, that is a syntactic equivalent of uniform semantic determinism. Testing consistency of a layered automaton can be done in polynomial time. Moreover, checking emptiness and language-inclusion of consistent layered automata can be also done in polynomial time (Propositions 3.27 and 3.28).

3.1 Syntactic definition of layered automata

A *layered automaton* $\mathcal{A}$ is a tuple of deterministic transition systems $\mathcal{T}_1, \ldots, \mathcal{T}_d$ together with morphisms $\mu_x : \mathcal{T}_{x+1} \to \mathcal{T}_x$, for $1 \leq x < d$. Additionally, we require that $\mathcal{T}_1$ is complete, has a distinguished initial state q_{init} in $\mathcal{T}_1$, and all states of $\mathcal{T}_1$ are reachable from q_{init}.[2]

We say that $\mathcal{T}_x$ is the *x-th layer* of the automaton. We use Q_x to denote the set of states of $\mathcal{T}_x$, which we refer to as *x-states*. We assume $Q_1, \ldots, Q_d$ pairwise disjoint. We let $\delta_x : Q_x \times \Sigma \to Q_x \cup \{\bot\}$ be the (partial) transition function of $\mathcal{T}_x$ (which is required to be total for $\mathcal{T}_1$). We write $p \xrightarrow{a}_x q$ to emphasize that a transition is taken in the x-th layer of the automaton. Sometimes we write Q_{x+1} for x some layer; we assume that this is the empty set when $x = d$.

We write $Q_{\geq x}$ for the union of $Q_x, \ldots, Q_d$, so $Q_{\geq 1}$ is the union of all the sets of states.

[2] The choice of starting at index 1 (instead of for instance 0) is arbitrary. It has been chosen so that layered automata with two layers correspond to the well understood class of HD coBüchi automata.

For every x, we define $\widehat{\mu}_x\colon Q_{\geq x} \to Q_x$ by

$$\widehat{\mu}_x(q) = \mu_x(\mu_{x+1}(\dots \mu_{y-1}(q)\dots)),\ \text{ for } q \in Q_y,\ y > x \ \ ; \quad \text{and} \quad \widehat{\mu}_x(q) = q,\ \text{ for } q \in Q_x.$$

For a state q, let *layer*(q) be the layer x such that $q \in Q_x$. For a pair (q, a) of a state and a letter, let *layer*(q, a) be the biggest layer x such that $\delta_x(\widehat{\mu}_x(q), a)$ is defined. Observe that *layer*(q, a) is always defined as $\mathcal{T}_1$ is complete.

We say that p is an *ancestor* of q, and write $p \sqsubseteq_{\mathcal{A}} q$, if $\widehat{\mu}_x(q) = p$, for $x =$ *layer*(p). Given $q, p \in Q_{\geq x}$, we write $q \sim_x p$ if $\widehat{\mu}_x(q) = \widehat{\mu}_x(p)$ and we let $[q]_x = \widehat{\mu}_x^{-1}(\widehat{\mu}_x(q))$. A state $q \in Q_{\geq 1}$ is a *leaf state* if it is not an image of any μ_x.

The forest of a layered automaton. We can view a layered automaton as a *forest* with nodes $Q_{\geq 1}$. Each state in $\mathcal{T}_1$ is the root of a tree, a node $q \in Q_{x+1}$ has $\mu_x(q) \in Q_x$ as the parent. The levels of this forest are labelled from 1 (top level) to d.

In this way, *layer*(q) is the level of q in this forest and $\sqsubseteq_{\mathcal{A}}$ is the ancestor relation. The function $\widehat{\mu}_x(q)$ designates the ancestor at level x of a node $q \in Q_{\geq x}$. We have that $q \sim_x p$ if p and q have the same ancestor at level x. Equivalently, $[q]_x$ is the set of nodes of the subtree rooted at the ancestor of q at level x. As expected, leaf states are the states that are leaves in this forest. Observe that more than one level can have leave states.

▶ **Example 3.1.** In Figure 1, we show two layered automata, represented as trees. Each transition system constitutes a level of the tree. The morphisms μ_x, corresponding to the parent relation, are represented as dotted arrows.

In these two examples, all leaf states are on the same layer. This needs not to be the case. Figure 7 (Section 6, Example 6.1) shows a layered automaton with leaf states at different layers.

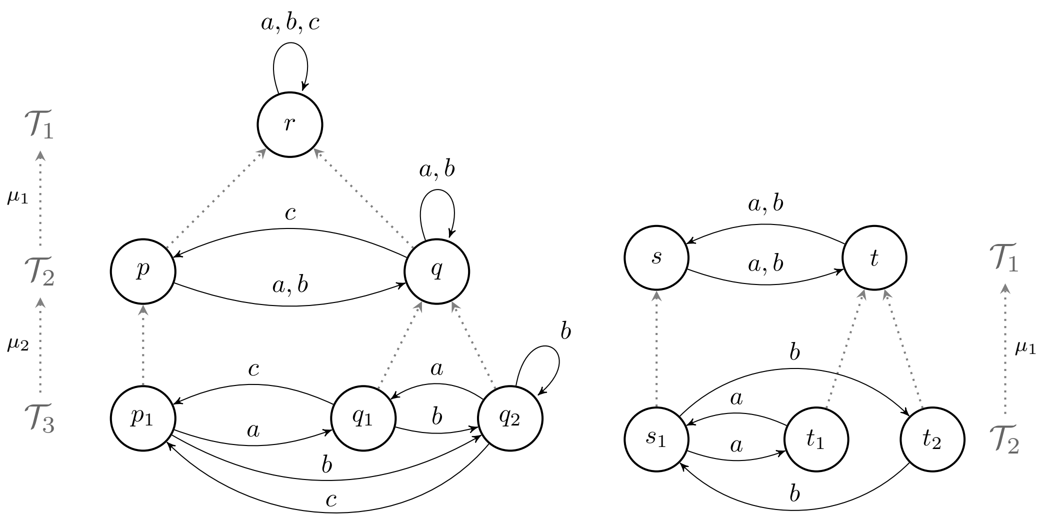

Figure 1 Two layered automata, represented as trees.

Morphisms of layered automata. A *morphism* between two layered automata $\mathcal{A}$ and $\mathcal{A}'$ is a function $\varphi\colon Q_{\geq 1} \to Q'_{\geq 1}$ that preserves the structure of the transition systems and the ancestor relation in the tree representation. That is:

- $\varphi(q_{\mathsf{init}}) = q'_{\mathsf{init}}$,
- if $q \xrightarrow{a}_x q'$ in $\mathcal{T}_x$, then $\varphi(q) \xrightarrow{a}_y \varphi(q')$ in $\mathcal{T}'_y$, for $y = layer(\varphi(q))$, and
- if $p \sqsubseteq_{\mathcal{A}} q$, then $\varphi(p) \sqsubseteq_{\mathcal{A}'} \varphi(q)$.

Note that, while states may change layer via φ, some rigidity is imposed by the conditions. Notably, if $q \in Q_x$ is sent to $\mathcal{T}'_y$, then all states reachable from q in $\mathcal{T}_x$ are also sent to the layer y. In particular, due to our reachability assumption, all states in $\mathcal{T}_1$ are sent to $\mathcal{T}'_1$. See Section 5.1 for a notion of strong morphism that moreover preserves the semantics.

A morphism of layered automata φ is an *isomorphism* if it is bijective and φ^{-1} is also a morphism. Note that in this case it preserves layers, that is, $\varphi|_{Q_x}$ is an isomorphism of transition systems from $\mathcal{T}_x$ to $\mathcal{T}'_x$.

Subautomata. A layered automaton $\mathcal{A}'$ is a *subautomaton* of $\mathcal{A}$ if there exists an injective morphism $\varphi\colon \mathcal{A}' \to \mathcal{A}$.

Note that $\mathcal{A}'$ may be obtained not just by removing states from some layers in $\mathcal{A}$, but also by changing some SCCs from one layer to another.

3.2 Semantics via alternating parity automata

We are ready to define a notion of acceptance for layered automata. We think of a layered automaton as defining an alternating automaton $[\![\mathcal{A}]\!]$ that we call the semantics of $\mathcal{A}$. The language of $\mathcal{A}$ is then simply the language of $[\![\mathcal{A}]\!]$. The idea is that the extra tree structure in $\mathcal{A}$ puts some restrictions on the transitions in $[\![\mathcal{A}]\!]$. We explain this view later in Section 3.2.3. Then in Section 3.3 we characterise acceptance by $[\![\mathcal{A}]\!]$ by a property that does not use alternation. This further shows the usefulness of the layered structure.

3.2.1 Definition

We associate with a layered automaton $\mathcal{A}$ its *semantics automaton* $[\![\mathcal{A}]\!]$, which is a simple by priorities alternating parity automaton defined as follows. The states Q are the leaf states of $\mathcal{A}$. The initial state is any leaf state q with $\widehat{\mu}_1(q) = q_{\mathsf{init}}$ the initial state in $\mathcal{T}_1$.[3]

The transitions are

$$\Delta = \left\{ (q, a, x, q') \in Q \times \Sigma \times [d] \times Q \quad \middle| \quad \begin{array}{l} x = layer(q, a), \text{ and} \\ \widehat{\mu}_x(q) \xrightarrow{a} \widehat{\mu}_x(q') \text{ in } \mathcal{T}_x \end{array} \right\}.$$

Intuitively, when in a leaf q and given a letter a we consider $layer(q, a) = x$, that is the biggest x such that the transition $\widehat{\mu}_x(q) \xrightarrow{a}_x p$ is defined. There is a transition from q to every leaf $p' \in \widehat{\mu}_x^{-1}(p)$. All these transitions have priority x. If x is odd, Eve chooses a leaf under p, otherwise it is Adam.

▶ Remark 3.2. If $p \xrightarrow{a:x} q$ in $[\![\mathcal{A}]\!]$, then there is a transition $p \xrightarrow{a:x} q'$ for every leaf state $q' \in [q]_x$, and to no other states. This justifies writing $p \xrightarrow{a:x} [q]_x$.

In the following, we focus almost exclusively on layered automata such that $[\![\mathcal{A}]\!]$ is semantically deterministic. We actually work with a slightly stronger property.

---

[3] Formally, this defines a family of automata. In general, these automata are not necessarily equivalent. However, for the automata considered in this paper (see notion of uniformly SD and consistency below), the choice of initial state is irrelevant.

▶ **Definition 3.3.** *A layered automaton $\mathcal{A}$ is uniformly semantically deterministic if for every pair of leaf states $p \sim_1 q$ we have $L([\![\mathcal{A}]\!], p) = L([\![\mathcal{A}]\!], q)$.*

As the name suggests we have the following.

▶ **Lemma 3.4.** *If a layered automaton $\mathcal{A}$ is uniformly semantically deterministic then it is semantically deterministic.*

Proof. If $q \xrightarrow{a} p_1$ and $q \xrightarrow{a} p_2$ in $[\![\mathcal{A}]\!]$, then $\widehat{\mu}_1(p_1) = \widehat{\mu}_1(p_2)$, hence the result. ◀

In Proposition 3.26 we show that we can check in polynomial time if a given layered automaton is uniformly semantically deterministic.

▶ Remark 3.5. If $\mathcal{A}$ is uniformly semantically deterministic then $\mathcal{T}_1$ admits a morphism into the *automaton of residuals* of $L = L([\![\mathcal{A}]\!])$: the transition system with states the sets $u^{-1}L$ and transitions $u^{-1}L \xrightarrow{a} (ua)^{-1}L$. This is thanks to our assumption that all states in $\mathcal{T}_1$ are reachable from q_{init}.

3.2.2 Examples

▶ **Example 3.6.** In Figure 2, we show the alternating parity automaton corresponding to the layered automaton on the left of Figure 1. It recognises the language

$L =$ Words with finitely many cc and infinitely many aa.

The states are the leaves of the tree. Whenever a letter c is seen from state p_1, priority 1 is produced, and Eve can choose to go to any state. Whenever a letter a is seen from state q_1, priority 2 is produced, and Adam can choose to go to q_1 or q_2. The rest of transitions produce priority 3, and correspond to the transitions at level 3 in the layered automaton.

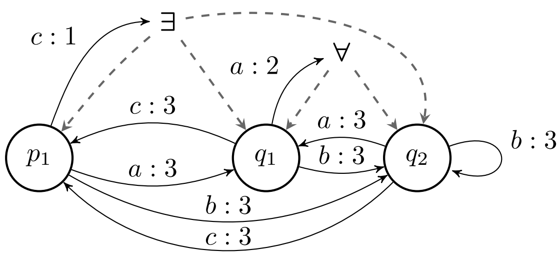

Figure 2 Alternating parity automaton for the layered automaton on the left of Figure 1.

▶ **Example 3.7.** In Figure 3, we show the alternating parity automaton corresponding to the layered automaton on the right of Figure 1. It recognises the language

$L = (\Sigma^2)^*(aa + bb)^\omega$.

In this case, the automaton turns out to be deterministic. The transition system $\mathcal{T}_1$ is the automaton of residuals of this language, which counts the parity (even or odd) of positions in the word.

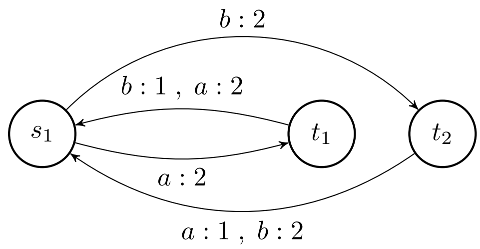

Figure 3 Alternating parity automaton for the layered automaton on the right of Figure 1.

Deterministic parity automata as layered automata. Deterministic automata can be seen as a special case of layered automata, as shown in the next proposition.

▶ **Proposition 3.8.** *Let $\mathcal{B}$ be a deterministic parity automaton. There exists a layered automaton $\mathcal{A}$ such that $[\![\mathcal{A}]\!] = \mathcal{B}$.*

Proof. Let $\mathcal{B}_{\geq x}$ be the transition system obtained by restricting $\mathcal{B}$ to transitions with priority $\geq x$. Let $\mathcal{A}$ be the layered automaton with transition systems $\mathcal{T}_x = \{x\} \times \mathcal{B}_{\geq x}$ and morphisms $\mu_x((x+1, q)) = (x, q)$ for all priorities x and states q. (Note that the x-component is just added for obtaining disjointness of the sets Q_x.) It is direct to check that $[\![\mathcal{A}]\!] = \mathcal{B}$. ◀

Minimal HD coBüchi automata. A (nondeterministic) *coBüchi automaton* is a $(\Sigma \times [1,2])$-transition system. Kuperberg and Skrzypczak showed that history deterministic coBüchi automata can be pruned not only into semantically deterministic, but also into *safe deterministic* automata at the same time. The later property meaning that the non-deterministic choices only appear in transitions carrying priority 1 [KS15]. Such automata are simple by priorities automata with priorities $[1, 2]$ as for every $q \in Q$ and $a \in \Sigma$ there is at most one transition $q \xrightarrow{a:2} q'$.

Building on this work, Abu Radi and Kupferman showed that HD coBüchi automata can be minimised in polynomial time, and that the minimal automaton is unique if it is assumed 1-saturated [AK22]. An automaton is 1-*saturated* if whenever $q \xrightarrow{a:1} p$, then $q \xrightarrow{a:1} p'$ for all p' with $L(p') = L(p)$. This minimisation is a special of the minimisation of layered automata presented here.

▶ **Proposition 3.9.** *Let $\mathcal{B}$ be a semantically deterministic, safe deterministic, 1-saturated coBüchi automaton. There exists a layered automaton $\mathcal{A}$ such that $[\![\mathcal{A}]\!] = \mathcal{B}$.*

Proof. Let $\mathcal{T}_1$ be the automaton of residuals of $\mathcal{B}$ and $\mathcal{T}_2 = \mathcal{B}_{\geq 2}$, with $\mu_1(q) = L(q)$. It is immediate to check that $[\![\mathcal{A}]\!] = \mathcal{B}$. ◀

3.2.3 From alternating parity automata to layered automata

We have introduced layered automata as a tree shaped structure that is interpreted as an alternating parity automaton. Here we offer another perspective on this class. We characterise directly alternating parity automata coming from layered automata in terms of equivalence relations on states that should be well-behaved.

Consider a simple by priorities alternating parity automaton together with a family of d nested equivalence relations on its states, that is:

$$Q \times Q \supseteq \sim_1 \supseteq \sim_2 \supseteq \cdots \supseteq \sim_d .$$

We consider the following properties. For all $a \in \Sigma$, $x \in [d]$ and $p, q \in Q$:

- (*x-determinism*) If $p \xrightarrow{a:x} q$ and $p \xrightarrow{a:x} q'$, then $q' \sim_x q$.
- (*x-coherence*) If $p \sim_x p'$ and $p \xrightarrow{a:\geq x} q$ then $p' \xrightarrow{a:\geq x} q'$ for some $q' \sim_x q$.
- (*x-saturation*) If $p \xrightarrow{a:x} q$ then for all $q' \sim_x q$ there is a transition $p \xrightarrow{a:x} q'$.

▶ **Proposition 3.10.** *Let $\mathcal{B}$ be a simple by priorities automaton. Then, there is a layered automaton $\mathcal{A}$ such that $\mathcal{B} = [\![\mathcal{A}]\!]$ if and only if $\mathcal{B}$ can be equipped with d nested equivalence relations $\sim_1, \dots, \sim_d$ satisfying x-determinism, x-coherence and x-saturation.*

Proof. Assume that $\mathcal{B} = [\![\mathcal{A}]\!]$ for a layered automaton $\mathcal{A}$. Then, we can define $q \sim_x p$ if $\widehat{\mu}_x(q) = \widehat{\mu}_x(p)$ and $q, p \in Q_{\geq x}$. By definition of $[\![\mathcal{A}]\!]$, it is immediate that the three properties hold.

Assume that $\mathcal{B}$ can be equipped with d nested equivalence relations $\sim_1, \dots, \sim_d$ satisfying the three properties above. Let $\mathcal{T}_x$ be the transition system having as states the $\sim_x$-classes of $\mathcal{B}$, and $[p]_x \xrightarrow{a}_x [q]_x$ if $p \xrightarrow{a:\geq x} q'$ in $\mathcal{B}$ for some $q' \in [q]_x$ (which, by x-saturation, implies that $p \xrightarrow{a:\geq x} q'$ for all $q' \in [q]_x$). These transitions are well-defined by x-coherence. The transition system $\mathcal{T}_x$ is deterministic by x-determinism. We define $\mu_{x+1}([q]_{x+1}) = [q]_x$, which is a morphism by definition of the transitions of $\mathcal{T}_x$. The fact that $\mathcal{B} = [\![\mathcal{A}]\!]$ is a routine check. ◀

We note, however, that it may be possible to equip $\mathcal{B}$ with different equivalence relations satisfying the three properties above. In that case, the proof above provides two different layered automata $\mathcal{A}, \mathcal{A}'$ such that $[\![\mathcal{A}]\!] = [\![\mathcal{A}']\!] = \mathcal{B}$.

3.3 Semantics via safe languages

Our goal in this section is characterising the language of a layered automaton without referring to alternation. The definition above is useful, as it uses the classical concept of alternating automata and allows introducing uniform semantic determinism, a variant of another well-known concept. Here we propose another route. We introduce a restriction on layered automata that we call consistency. We show that consistency is equivalent to uniform semantic determinism. Moreover, for consistent automata the language of $[\![\mathcal{A}]\!]$ can be characterised by the following notion of acceptance that does not use alternation. We note that this notion of acceptance is closely related to the natural color of a word [ES22] and the acceptance condition of rerailing automata [Ehl25] (see Section 3.6 for further details).

Safe languages. The *x-safe language* of a state $q \in Q_{\geq x}$ is defined as

$$L_x(q) = \{w \in \Sigma^* \cup \Sigma^\omega \mid \text{ the run over } w \text{ from } \widehat{\mu}_x(q) \text{ is defined in } \mathcal{T}_x\}.$$

That is, $L_x(q)$ are the words on which there is a path from the ancestor of q at level x. Note that by completeness of $\mathcal{T}_1$ we have $L_1(q) = \Sigma^* \cup \Sigma^\omega$ for every state q. When we need to specify the automaton $\mathcal{A}$, we write $L_x^{\mathcal{A}}(q)$.

▶ **Example 3.11.** For the layered automaton on the left of Figure 1, we have

$$L_2(q) = \text{Words without the factor } cc,$$
$$L_3(q_2) = \text{Words without the factors } cc \text{ and } aa.$$

We have the following equalities relating the words labelling paths with no priority $< x$ in $[\![\mathcal{A}]\!]$ and x-safe languages. Recall that we write $\xrightarrow{w:\geq x}$ if the minimal priority on this path is $\geq x$.

$$\begin{aligned} L_x(q) &= \{w \in \Sigma^* \cup \Sigma^\omega \mid \text{there is a path } \widehat{\mu}_x(q) \xrightarrow{w}_x \text{ in } \mathcal{T}_x\} \\ &= \{w \in \Sigma^* \cup \Sigma^\omega \mid \text{there is a path } [q]_x \xrightarrow{w:\geq x} \text{ in } [\![\mathcal{A}]\!]\} \\ &= \{w \in \Sigma^* \cup \Sigma^\omega \mid \text{no path } [q]_x \xrightarrow{w} \text{ in } [\![\mathcal{A}]\!] \text{ sees a priority } < x\}. \end{aligned}$$

▶ **Definition 3.12.** *Consider a layered automaton $\mathcal{A}$. A word $w \in \Sigma^\omega$ is ultimately safe in layer x if there is a decomposition $uw' = w$ and a state p on layer x such that $q_{\mathsf{init}} \xrightarrow{u}_1 \widehat{\mu}_1(p)$ and $w' \in L_x(p)$. We say that w is accepted by $\mathcal{A}$ if the maximal layer in which w is ultimately safe is even. If it is odd then we say that w is rejected.*

We also introduce a related notion of strong acceptance that later plays a central role in our minimisation procedure.

Strong acceptance. We say that $w \in \Sigma^\omega$ is *strongly accepted by p* if $w \in L_x(p)$, p is on some even layer x, and for every decomposition $w = uw'$, there is no $p' \in Q_{x+1}$ with $\widehat{\mu}_x(p) \xrightarrow{u}_x \widehat{\mu}_x(p')$ and $w' \in L_{x+1}(p')$. The notion of w being *strongly rejected* is defined similarly but for x odd.

Intuitively, strong acceptance implies that on every run from $[p]_x$ over w in $[\![\mathcal{A}]\!]$, only priorities $\geq x$ occur and x occurs infinitely often, regardless the choices of the players. As x is even, all these runs are accepting.

A priori, a word can be both strongly accepted by some state q, and strongly rejected by a different state. We forbid this by imposing an extra condition called consistency, defined below. We show that under the consistency assumption, the acceptance mechanisms in the following proposition are equivalent.

▶ **Proposition 3.13.** *Let $\mathcal{A}$ be a consistent layered automaton and let $w \in \Sigma^\omega$. The following are equivalent:*

- *$w \in L([\![\mathcal{A}]\!])$,*
- *w is accepted by $\mathcal{A}$,*
- *There is a decomposition $uw' = w$ and a state p such that $q_{\mathsf{init}} \xrightarrow{u}_1 \widehat{\mu}_1(p)$ and p strongly accepts w, and*
- *There is no decomposition $uw' = w$ and a state p such that $q_{\mathsf{init}} \xrightarrow{u}_1 \widehat{\mu}_1(p)$ and p strongly rejects w.*

▶ **Example 3.14.** In the layered automaton on the left of Figure 1, any word containing infinitely many factors cc's is strongly rejected by the state $r \in Q_1$. The state $q \in Q_2$ strongly accepts the words that contain no factor cc and infinitely many factors aa.

The next two lemmas link strong acceptance in $\mathcal{A}$ and the acceptance by $[\![\mathcal{A}]\!]$.

▶ **Lemma 3.15.** *Let $\mathcal{A}$ be a layered automaton, p a leaf state. If a word w is strongly accepted by $\widehat{\mu}_x(p)$, for some even x, then $w \in L([\![\mathcal{A}]\!], p)$. Analogously, if w is strongly rejected by $\widehat{\mu}_x(p)$, for some odd x, then $w \notin L([\![\mathcal{A}]\!], p)$*

Proof. We show that any run over w from p produces the priority x infinitely often (regardless the strategies of Eve and Adam). Since $w \in L_x(p)$, only priorities $\geq x$ are produced by any run over w from p. If such a run does not produce priority x infinitely often, then it is of the form $p \xrightarrow{u} q \xrightarrow{w':>x}$, a contradiction. ◀

▶ **Lemma 3.16.** *If $\mathcal{A}$ accepts w then there is a decomposition $uw' = w$ and a state p on some even layer x such that $q_{\mathsf{init}} \xrightarrow{u}_1 p$ and w' is strongly accepted from p.*

Proof. Consider the maximal layer x where w is ultimately safe, which is even as w is accepted. By definition, we have a decomposition $uw' = w$, with a state p, a path $q_{\mathsf{init}} \xrightarrow{u}_1 \widehat{\mu}_1(p)$, and $w' \in L_x(p)$. It is direct to check that w' is strongly accepted from p, because otherwise x would not be the maximal layer where w is ultimately safe. ◀

To get the desired characterisation we need one more condition that corresponds to uniform semantic determinism.

Consistency We say that a layered automaton $\mathcal{A}$ is *consistent* if there is no pair of states $p, p' \in Q_{\geq 1}$ with $\widehat{\mu}_1(p) = \widehat{\mu}_1(p')$, and a word $w \in \Sigma^\omega$ such that w is strongly accepted by p and w is strongly rejected by p'.

As stated below, this notion is equivalent to uniform semantic determinism of $[\![\mathcal{A}]\!]$. We show in Proposition 3.26 that this property can be checked in polynomial time.

We state the key result saying that the semantics via alternating parity automata and via strong acceptance coincide for consistent layered automata. The proof, which requires introducing longest suffix resolvers in $[\![\mathcal{A}]\!]$, is relegated to Section 3.4 below.

▶ **Theorem 3.17.** *Let $\mathcal{A}$ be a consistent layered automaton. A word $w \in \Sigma^\omega$ is accepted by $\mathcal{A}$ if and only if $w \in L([\![\mathcal{A}]\!])$. Moreover, $[\![\mathcal{A}]\!]$ is uniformly semantically deterministic.*

Proposition 3.13 follows.

▶ **Corollary 3.18.** *A layered automaton is consistent if and only if it is uniformly semantic deterministic.*

Proof. The left to right implication is given by the theorem above. The other direction follows directly from the definitions and Lemma 3.15. ◀

3.4 Layered automata are history deterministic and 0-1 probabilistic

In this section we prove Theorem 3.17 and the following propositions.

▶ **Proposition 3.19.** *For every consistent layered automaton $\mathcal{A}$, the alternating parity automaton $[\![\mathcal{A}]\!]$ is history deterministic.*

We say that a simple by priorities parity automaton $\mathcal{A}$ is a *0-1 probabilistic automaton* if for every random walk ρ over a word $w \in \Sigma^\omega$ starting in q_{init}, it holds:

- $\Pr(\rho$ is accepting $) = 1 \iff w \in L(\mathcal{A})$, and
- $\Pr(\rho$ is accepting $) = 0 \iff w \notin L(\mathcal{A})$.

We note that 0-1 probabilistic automata are *good for MDPs* in the sense of [HPS+20] See [BGB12, HPT25] for related notions.

▶ **Proposition 3.20.** *For every consistent layered automaton $\mathcal{A}$, the alternating parity automaton $[\![\mathcal{A}]\!]$ is a 0-1 probabilistic automaton.*

We start by introducing *longest suffix resolvers*. This is a class of resolvers witnessing HDness of layered automata and enjoying some useful properties.

Longest suffix resolvers. For a priority x, a finite word u, and an x-state q, we define the *longest x-suffix of u to q* to be the longest suffix v of u such that $p \xrightarrow{v}_x q$ in $\mathcal{T}_x$, for some p. Observe that this suffix can be empty.

We let Q and Δ be the sets of states and transitions of $[\![\mathcal{A}]\!]$, respectively.

The *longest suffix resolver*[4] for Eve is a resolver $\sigma : (\{q_{\mathsf{init}}\} \cup \Delta^+) \times \Sigma \to Q$ such that whenever we have a play $p_0 \xrightarrow{v} p \xrightarrow{a:x} [q]_x$ in $[\![\mathcal{A}]\!]$ with x odd, Eve chooses any state $q' \in [q]_x$ admitting the longest $(x+1)$-suffix of va to $\widehat{\mu}_{x+1}(q')$. The longest suffix resolver for Adam is defined symmetrically.

The next lemma states the main property of longest suffix resolvers. It guarantees that a play leaves an odd layer whenever there is a run with this property. The statement uses a more general notion of a *longest suffix resolver initialised* to $u \in \Sigma^*$. It is a function $\sigma_u : (\{q_{\mathsf{init}}\} \cup \Delta^+) \times \Sigma \to Q$ such that whenever we have a play $p_0 \xrightarrow{v} p \xrightarrow{a:x} [q]_x$ in $[\![\mathcal{A}]\!]$ with x odd, Eve chooses any state $q' \in [q]_x$ admitting the longest $(x+1)$-suffix of uva to $\widehat{\mu}_{x+1}(q')$; instead of the longest suffix of va, as a non-initialised resolver would. So the previous notion amounts to initialisation with the empty word.

▶ **Lemma 3.21.** *Let x be an odd priority and p a leaf state in a layered automaton $\mathcal{A}$. Suppose $w = u\tilde{w} \in \Sigma^\omega$ admits a run $p \xrightarrow{u:\geq x} q \xrightarrow{\tilde{w}:\geq x+1}$ in $[\![\mathcal{A}]\!]$ (equivalently, $w \in L_x(p)$ but not strongly rejected by $\widehat{\mu}_x(p)$). Then, for every u_0, any longest suffix resolver σ for Eve initialised to u_0 ensures that on any σ-run on the word w from p, eventually all priorities are $\geq x+1$.*

Proof. Consider a play from p according a longest suffix resolver for Eve, initialised to some u_0, that sees priority x after some prefix uw' of $u\tilde{w}$:

$$p \xrightarrow{u:\geq x} q \xrightarrow{w':\geq x} q' \xrightarrow{a:x} q'' .$$

In this case, the longest suffix resolver initialised to u_0 finds some suffix v of u_0uw' with $p_{va} \xrightarrow{va}_{x+1} \widehat{\mu}_{x+1}(q'')$ in $\mathcal{T}_{x+1}$ (with p_{va} some state in this automaton). Observe that by assumption of the lemma, w' is a suffix of v since $\widehat{\mu}_{x+1}(q) \xrightarrow{w'a}_{x+1} \widehat{\mu}_{x+1}(q'')$. So this path on v can be presented as $p_{va} \xrightarrow{v'}_{x+1} q_{w'a} \xrightarrow{w'a}_{x+1} \widehat{\mu}_{x+1}(q'')$, for some v' and $q_{w'a} \in \mathcal{T}_{x+1}$. We call $q_{w'a}$ a support of $w'a$. Now consider the next time when the longest suffix run sees priority x, namely

$$p \xrightarrow{u:\geq x} q \xrightarrow{w':\geq x} q' \xrightarrow{a:x} q'' \xrightarrow{w'':>x} q^{(3)} \xrightarrow{b:x} .$$

This means that we have $q_{w'a} \xrightarrow{w'a}_{x+1} \widehat{\mu}_{x+1}(q'') \xrightarrow{w''}_{x+1} \widehat{\mu}_{x+1}(q^{(3)})$ with no b-transition from $\widehat{\mu}_{x+1}(q^{(3)})$ in $\mathcal{T}_{x+1}$. Hence, $q_{w'a}$ cannot be a support for $w'aw''b$ and a new support is found by the reasoning above. This argument shows that each time a play following the longest suffix resolver initialised to u_0 meets priority x, one $(x+1)$-state is eliminated as a potential support. Hence, the number of times such a play can see priority x is bounded by the number of $(x+1)$-states. ◀

For the statement of the next lemma recall that $(\mathcal{A}, q)$ stands for $\mathcal{A}$ where q is taken to be the initial state. For layered automata, the initial state should come from layer 1, while the initial state of $[\![\mathcal{A}]\!]$ is a leaf of $\mathcal{A}$.

[4] Formally, we define a family of resolvers, as there are some non-deterministic choices in the definition. By a slight abuse of notation, we call *the* longest suffix resolver any of them.

▶ **Lemma 3.22.** *Consider $\mathcal{A}$ a consistent layered automaton, q a state of $[\![\mathcal{A}]\!]$, and σ and τ longest suffix resolvers for Eve and Adam in $[\![\mathcal{A}]\!]$, respectively. If a word w is accepted by $(\mathcal{A}, \widehat{\mu}_1(q))$ then every σ-run of $([\![\mathcal{A}]\!], q)$ over w is accepting. Symmetrically, if a word w is rejected by $(\mathcal{A}, \widehat{\mu}_1(q))$ then every τ-run of $([\![\mathcal{A}]\!], q)$ over w is rejecting.*

Proof. We start by recalling Lemma 3.16 saying that if w is accepted by $(\mathcal{A}, q)$ then there is a decomposition $u'w' = w$ and a state p' on some even layer y such that $q \xrightarrow{u'}_1 \widehat{\mu}_1(p')$, and w' is strongly accepted from p'.

Suppose the longest suffix resolver of Eve rejects when Eve plays on w in $([\![\mathcal{A}]\!], q)$. Take a non-accepting run consistent with this resolver,

$$q \xrightarrow{u''} p'' \xrightarrow{w'':\geq x} \quad \text{in } [\![\mathcal{A}]\!],$$

where for some odd x, eventually only priorities $\geq x$ appear and x appears infinitely often. We can assume that u' is a prefix of u''.

First, we show that w'' is strongly rejected from $\widehat{\mu}_x(p'')$ in $\mathcal{A}$. Since the run in $[\![\mathcal{A}]\!]$ avoids priorities $< x$, we have $w'' \in L_x(p'')$. If w'' is not strongly rejected, then by Lemma 3.21 the longest suffix resolver for Eve initialised to u'' eventually sees only priorities $\geq x+1$ from p''; a contradiction with the fact that x appears infinitely often in the run in $[\![\mathcal{A}]\!]$.

Now let $u'' = u'v$ (u' is assumed to be a prefix of u''). We have a path $p' \xrightarrow{v}_y p_y$ in the even layer $\mathcal{T}_y$, and w'' is strongly accepted from p_y. Moreover, $\widehat{\mu}_1(p'') = \widehat{\mu}_1(p_y)$. This contradicts the consistency of $\mathcal{A}$. ◀

The above lemma implies the first part of Theorem 3.17, stating that for any consistent layered automaton $\mathcal{A}$, acceptance by $\mathcal{A}$ is the same as acceptance by $[\![\mathcal{A}]\!]$. It also proves HDness of $[\![\mathcal{A}]\!]$ (Proposition 3.19).

▶ **Corollary 3.23.** *Let $\mathcal{A}$ be a consistent layered automaton. Then, $[\![\mathcal{A}]\!]$ is history deterministic, and longest suffix resolvers are valid resolvers.*

It remains to see that a consistent $[\![\mathcal{A}]\!]$ is uniformly semantically deterministic.

▶ **Lemma 3.24.** *If $\mathcal{A}$ is a consistent automaton then it is uniformly semantically deterministic.*

Proof. We note that the definition of acceptance (by safe languages) in $(\mathcal{A}, q)$ only depends on $\widehat{\mu}_1(q)$. Since Lemma 3.22 holds for any choice of initial state in $\mathcal{A}$, we have that for any $p \sim_1 q$

$$L([\![\mathcal{A}]\!], p) = L(\mathcal{A}, p) = L(\mathcal{A}, q) = L([\![\mathcal{A}]\!], q).$$ ◀

Random strategies. Proposition 3.20 admits an almost identical proof.

▶ **Lemma 3.25.** *Let p be a leaf state in a layered automaton $\mathcal{A}$. Let $uw \in \Sigma^\omega$ admitting a run $p \xrightarrow{u:\geq x} q \xrightarrow{w:\geq x+1}$ in $[\![\mathcal{A}]\!]$. Then, any random walk on the word uw from p eventually produces only priorities $\geq x+1$.*

Proof. Let n be the number of leaf states. Let $w = uw'a\tilde{w}$ and let

$$p \xrightarrow{u:\geq x} q' \xrightarrow{w':\geq x} q'' \xrightarrow{a:x} [\tilde{q}]_x$$

be a random walk from p on a prefix of uw, producing priority x. By assumption, there is at least one state $\tilde{q}'$ in $[\tilde{q}]_x$ such that $\tilde{w} \in L_{x+1}(\tilde{q}')$. Therefore, the random walk goes to this state with probability at least $1/n$. The probability of seeing k transitions with priority x after the prefix u is therefore less than $(1 - 1/n)^k \xrightarrow[k\to\infty]{} 0$. ◀

Proof of Proposition 3.20. Let $w \notin L(\mathcal{A})$ (the case $w \in L(\mathcal{A})$ is symmetric), and let x be an even priority. We show that the probability that a random walk over w produces x as minimal priority infinitely often is 0. The probability of this event is the probability of obtaining a path of the form:

$$q_{\mathsf{init}} \xrightarrow{u} p \xrightarrow{w':\geq x} \quad \text{producing } x \text{ infinitely often, with } w = uw'.$$

For every such path $q_{\mathsf{init}} \xrightarrow{u} p$, since $w' \in L_x(p)$ and is not strongly rejected by $\widehat{\mu}_x(p)$, there is a run $p \xrightarrow{u':\geq x} q \xrightarrow{w'':\geq x+1}$, with $w' = u'w''$. Therefore, by Lemma 3.25, the probability of obtaining a path $p \xrightarrow{w':\geq x}$ producing x infinitely often is 0. ◀

3.5 Some decision procedures in PTIME

We show that we can check whether a given layered automata $\mathcal{A}$ is consistent in polynomial time. We can also check emptiness and language inclusion of consistent layered automata in polynomial time.

▶ **Proposition 3.26.** *It is decidable in polynomial time whether a given layered automaton $\mathcal{A}$ is consistent.*

Proof. We check for a pair $p, p' \in Q_{\geq 1}$ conflicting with the definition of consistency, that is, $\widehat{\mu}_1(p) = \widehat{\mu}_1(p')$ and there is a word w strongly accepted by p and strongly rejected by p'. There are polynomially many candidates, so we can iterate through all of them. Given $p \in Q_x$ and $p' \in Q_{x'}$, with x even and x' odd, we can check whether there exists a conflicting word $w \in \Sigma^\omega$ by a game. In this game, a player *Prover* tries to find a word witnessing an inconsistency, and *Refuter* tries to show that the constructed word is not a witness.

The game starts in (p, p'). In each round, Prover plays the next letter, and Refuter chooses the next pair of states according to the transition function on the chosen letters. Prover wins if on the run from p only priorities $\geq x$ occur and x occurs infinitely often, and on every run from p' on w, only priorities $\geq x'$ occur and x' occurs infinitely often. This is a generalised Büchi game which can be solved in polynomial time. We claim that Prover has a winning strategy if and only if w is strongly accepted by p and strongly rejected by p'. Clearly, if this condition is satisfied, then Prover can play the word $w \in \Sigma^\omega$, and all the runs that Refuter can play are such that Prover wins the game.

For the other direction, assume that Prover has a winning strategy in the game and let her play according to the strategy. Any word produced by this strategy has the property that all priorities visited from p are $\geq x$, and all priorities visited from p' are $\geq x'$. Let Refuter use a longest suffix resolver in both components for selecting the runs, and let $w \in \Sigma^\omega$ be the word produced by Prover's strategy against this strategy of Refuter. If there is a run on w from p that does not produce x infinitely often, then this run produces only priorities $\geq x + 1$ from some point onwards. The longest suffix resolver used by Refuter for selecting the run would then result in a run that sees only priorities $\geq x + 1$ from some point onwards (see Lemma 3.21). Similarly for p'. ◀

▶ **Proposition 3.27.** *Given a consistent layered automaton $\mathcal{A}$, we can check in polynomial time whether $L(\mathcal{A}) = \emptyset$.*

Proof. Since we assume that all states in $\mathcal{T}_1$ are reachable, $L(\mathcal{A}) \neq \emptyset$ if and only if there is some state $p_0 \in Q_{\geq 1}$ in an even layer x that strongly accepts some word.

We can check if this is the case via a Büchi game similar to the one considered above. Two players play over pairs $(p, q) \in Q_x \times Q_{x+1}$. The starting position is (p_0, q_0), for q_0 any state in $\mu_x^{-1}(p_0)$. At each step, Prover gives the next letter such that the obtained word is safe from p_0. The next state $p \xrightarrow{a}_x p'$ is given deterministically. If transition $q \xrightarrow{a}_{x+1} q'$ exists, this transition is taken. If not, Refuter can chose to reset to a state in $\mu_x^{-1}(p')$. Prover wins if there are infinitely many resets at level $x+1$. It is clear that Prover wins this game if and only if p_0 strongly accepts some word. Indeed, Refuter can use the longest suffix resolver to win whenever the produced word is not strongly accepted. ◀

▶ **Proposition 3.28.** *Given two consistent layered automaton $\mathcal{A}$ and $\mathcal{A}'$, we can check in polynomial time whether $L(\mathcal{A}) \subseteq L(\mathcal{A}')$.*

Proof. We show that we can reduce this problem to solving a Rabin game with 3 Rabin pairs of polynomial size. This allows to conclude, as solving Rabin games with a fix number of Rabin pairs can be done in polynomial time (see e.g. [FAA+25]. The winning condition is in fact slightly simpler: if the game stays in the first phase Prover loses, otherwise Prover wins in phase 2 if two Büchi conditions are satisfied.

In this game, Prover tries to show that $L(\mathcal{A}) \not\subseteq L(\mathcal{A}')$ by giving a word in $L(\mathcal{A}) \setminus L(\mathcal{A}')$. Prover gives letters one by one. In a first phase, these letters determine two runs:

$$q_{\mathsf{init}} \xrightarrow{a_1}_1 q_2 \xrightarrow{a_2}_1 q_3 \xrightarrow{a_3}_1 \dots \text{ in } \mathcal{T}_1 \quad \text{ and } \quad q'_{\mathsf{init}} \xrightarrow{a_1}_1 q'_2 \xrightarrow{a_2}_1 q'_3 \xrightarrow{a_3}_1 \dots \text{ in } \mathcal{T}'_1.$$

At any point, Prover can decide to enter a phase 2 in $\mathcal{A}$: he goes to some pair (p_i, s_i), with $p_i \in \widehat{\mu}_1^{-1}(q_i)$ in some even layer $\mathcal{T}_x$ and $s_i \in \mu_x^{-1}(p_i)$. Similar, at any point Prover can enter a phase 2 in $\mathcal{A}'$, going to (p'_i, s'_i), with $p'_i \in \widehat{\mu}_1^{-1}(q'_i)$ in some odd layer $\mathcal{T}'_{x'}$ and $s'_i \in \mu_{x'}^{-1}(p'_i)$.

If for $\mathcal{A}$ or $\mathcal{A}'$, Prover never enters phase 2, then he loses.

When automaton $\mathcal{A}$ is in phase 2 and in a pair (p_i, q_i), the game is as in the previous proposition. The transition $p_i \xrightarrow{a_i}_x p_{i+1}$ must exist (else, Prover loses automatically). If the transition $q_i \xrightarrow{a_i}_{x+1} q_{i+1}$ exists in $\mathcal{T}_{x+1}$, then we go to q_{i+1} in the second component. If this transition does not exist, then this is an accepting edge for the first Büchi condition, and Refuter resets the second component to any $q_{i+1} \in \mu_x^{-1}(p_{i+1})$.

Phase 2 in $\mathcal{A}'$ is identical. We follow the run in $\mathcal{T}'_{x'}$ in the first component, which must exist (else, Prover loses). Refuter can choose how to reset the second component in $\mathcal{T}'_{x'+1}$ whenever necessary. Each time a reset happens, we see an accepting edge for the second Büchi condition.

It is clear that there exists a word in $L(\mathcal{A}) \setminus L(\mathcal{A}')$ if and only if Prover can see infinitely often edges in both Büchi conditions. ◀

3.6 Connections and comparisons with other models

In this section, we discuss how layered automata relate to other models from the literature. Sometimes we refer to the minimal layered automaton for a language that is introduced in Section 4. We leave for future work a thorough analysis on the size of layered automata compared to other representations of ω-regular languages; we include here just some remarks on this matter.

Deterministic and alternating automata. First, we notice that consistent layered automata can be exponentially smaller than deterministic automata, as this is already the case for HD coBüchi automata [KS15, Thm. 1]. On the other hand, they can be doubly exponentially larger than alternating parity automata, as this double exponential gap already occurs in automata over finite words [CKS81, Thm. 5.3], because consistent layered automata on the first layer need to have at least as many states as there are residuals in the language.

Minimal HD coBüchi automata. As discussed in Proposition 3.9, minimal HD coBüchi automata can be seen as a special case of layered automata. The properties characterising minimal layered automata that we introduce in Section 4 are a generalisation of the notions of safe minimality and safe centrality introduced by Abu Radi and Kupferman [AK22].

Muller languages. The reader familiar with Zielonka trees [Zie98] and their parity automata [DJW97, CCFL24] may have noticed close resemblances between layered automata and that model. Indeed, it is not difficult to see that the Zielonka tree of a Muller language L can be taken as the tree of a layered automaton recognising L. In this case, the associated automaton $[\![\mathcal{A}]\!]$ is deterministic by pruning. Any such pruning produces the deterministic parity automaton of the Zielonka tree. We refer to [CCFL24, Section 4] for definitions on the parity automaton of a Zielonka tree.

Positional languages. In 2024, Casares and Ohlmann characterised which ω-regular languages are *positional*, that is, languages L such that the existential player can always play optimally in games with winning condition L [CO24]. In this characterisation, they use so-called *signature automata*. It turns out that signature automata are just layered automata with extra properties that ensure positionality.

More precisely, we can restate they characterisation (point (2) in [CO24, Thm. 3.1]) as follows: An ω-regular language L is positional if and only if the minimal layered automaton $\mathcal{A}_L$ is such that:

1. Nodes at even layers have at most one child,
2. For each x, the states in every SCC of $\mathcal{T}_x$ are totally ordered by inclusion of x-safe languages,
3. The residuals are totally ordered by inclusion (this can be seen as a special case of the previous condition), and
4. Each even layer x is *progress consistent*. That is, if $p \xrightarrow{u}_x q$ and $L_x(p) \subsetneq L_x(q)$, then u^ω is strongly accepted by p.

Chains of coBüchi automata (COCOA). In [ES22], Ehlers and Schewe propose to represent an ω-regular language by a decreasing sequence of coBüchi languages $L_1 \supseteq L_2 \supseteq \cdots \supseteq L_d$,[5] where L_x are the words that are *not at home at level* $x-1$ (we refer to [ES22, Def. 1, Thm. 7] for definitions[6]). A word w is in the language L represented by the chain if the maximal index x such that $w \in L_x$ is even.

It is straightforward to obtain HD coBüchi automata for the languages L_x from a consistent layered automaton for L: take the transition system $\mathcal{T}_x$ on layer x, interpret the existing transitions as 2-transitions of the coBüchi automaton, and for all missing transitions, add 1-transitions to all states of the corresponding residual class. Therefore, the size of a COCOA representation of a language is never bigger than the size of the minimal layered automaton for that language. But the minimal layered automaton may be exponentially larger than the minimal COCOA for a language, as illustrated by Example 3.29 below.

Since COCOA represents a language as a Boolean combination of coBüchi automata, it is therefore not surprising that it is smaller than a representation by a single automaton. In

[5] The definitions in [ES22] use 0 as minimal color, whereas we use 1 as minimal layer. As mentioned in [ES22], the definitions can easily be adapted to the setting with 1 as minimal color.
[6] We believe that in [ES22, Thm. 7] it should read "i.e., w is *not* at home in color $i-1$".

order to use COCOA in applications like synthesis, one needs to obtain a history deterministic automaton from it. This requires to make a product of all the coBüchi automata that constitute it [EK24a]. Therefore, the number of states of the final alternating parity automaton coming from a COCOA may be exponentially larger than the alternating parity automaton given by a layered automaton. In Example 3.30 we give such an example.[7] Moreover, the languages of this example is prefix-independent (it has a single residual), so this blow-up is not due exclusively to taking a product of all residuals of the language.

One of the central definitions in [ES22] is the *natural color* of a word w with respect to a language L. We believe that the natural color is visible in the minimal layered automaton for L, namely it coincides with the maximal layer x in which w is ultimately safe.

▶ **Example 3.29.** The language in this example is a minor modification of a language used in [ABF18] for showing an exponential gap between saturated families of DFAs and DPAs. Let $\Sigma = \{1, \dots, k\}$ and consider the language L of all $w \in \Sigma^\omega$ satisfying the following two conditions:

- w contains an even number of different letters infinitely often,
- from some point onwards, for every two-letter infix ij, we have $j \in \{1, \dots, i+1\}$.

The second condition ensures that for all words in L, the set of symbols that occur infinitely often forms an interval inside $\{1, \dots, k\}$. For $k = 4$, the minimal layered automaton for this language is shown in Figure 4. The automaton has $k+1$ layers. Since the language is prefix independent, the first layer has only one state looping on all letters. For $x \geq 1$, the SCCs on layer x correspond to the intervals of size $k - x + 2$. However, an interval might appear several times. For example, on layer 4, the interval $[2, 3]$ appears twice because it is contained in $[1, 3]$ and in $[2, 4]$. And on layer 5 the singleton interval $[3, 3]$ appears three times because it is contained in $[3, 4]$ and in $[2, 3]$ which itself appears already twice on the previous layer. Because of the binary tree structure starting from layer 2, there are 2^{k-1} states on layer $k + 1$. These are the states used by the corresponding alternating HD automaton. If one builds the COCOA for this language, then each interval appears only once on each level of the COCOA, so the number of states of each coBüchi automaton in the COCOA is at most quadratic in k.

As a further remark, for those readers familiar with ω-semigroups, the size of the syntactic ω-semigroup of this language is of order k^4 because an element of the semigroup only needs to store the first, last, minimal, and maximal number occurring in a word. The first and the last number in a word are needed to keep track of the second condition when concatenating two words. And assuming that the second condition is satisfied, the number of different letters occurring infinitely often can be derived from the minimal and maximal number occurring infinitely often, because letters can increase by at most one in each step.

▶ **Example 3.30.** Let $\Sigma = \{a_1, \dots, a_k, b_1, \dots, b_k, c\}$ and consider the language containing all words satisfying one of the following two conditions:

- $\{b_1, \dots, b_k\}$ occur finitely often, and $\min(\{k+1\} \cup \{x \mid a_x \text{ occurs infinitely often}\}$ is even, or
- $\{a_1, \dots, a_k\}$ occur finitely often, and $\min(\{k+1\} \cup \{x \mid b_x \text{ occurs infinitely often}\}$ is even.

[7] We consider here the naive product of [EK24a]. They also define a reduced and an optimised product that ignores some product states. A more fine-grained comparison of these products with layered automata is left as future work.

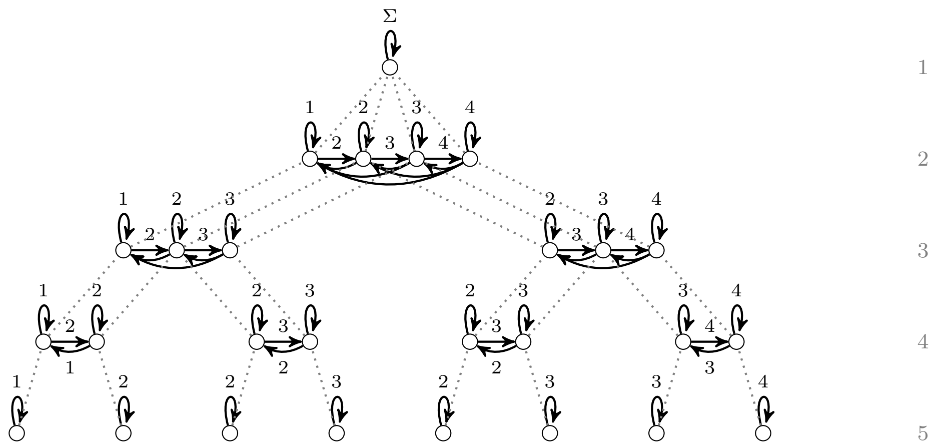

Figure 4 The layered automaton for $k = 4$ in Example 3.29. In each SCC on each layer, all edges going to the same state have the same label. For readability, we have omitted the labels for the edges from right to left on layers 2 and 3.

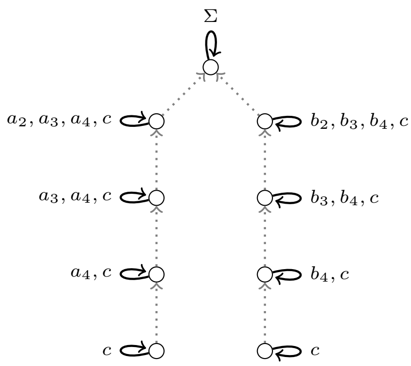

Figure 5 The layered automaton for $k = 4$ in Example 3.30.

That is, this is the language of words where one sort of the numbered symbols occurs finitely often, while the parity condition is satisfied on the other sort of symbols (the $k + 1$ in the minimum takes care of the case that only c occurs infinitely often). The minimal layered automaton needs two strands for checking the two parity conditions, as shown in Figure 5 for $k = 4$. The minimal COCOA is obtained by taking each layer x, interpreting the loops on the states as 2-transitions, and saturating it with 1-transitions for the missing letters on that layer. Taking the (naive) product of these coBüchi automata, results in exponentially many states, most of which only have a c-loop.

Rerailing automata. In 2025, Ehlers introduced *rerailing automata* [Ehl25], a kind of automata with specific semantics: a word is accepted if the maximal priority of its runs is even.

We believe that a consistent layered automaton $\mathcal{A}$ can be transformed into a rerailing automaton by further saturating $[\![\mathcal{A}]\!]$. More precisely, whenever there is a transition $p \xrightarrow{a:x} [q]_x$, we can add transitions $p \xrightarrow{a:y} [q]_y$ for all $y < x$ without changing the language recognised by the automaton. The automaton obtained in this way has the rerailing property. Moreover, a word is accepted by $\mathcal{A}$ if and only if it is accepted with the semantics of rerailing automata after saturation.

For the other direction, it is unclear to us if rerailing automata are layered, or even simply HD alternating. However, it may be the case that the minimal rerailing automaton of a language and its minimal layered automaton are essentially the same object.

4 A canonical minimal layered automaton

In this section, we show that each ω-regular language admits a unique minimal consistent layered automaton. This automaton is characterised by three properties: normality, centrality and safe minimality (extending the properties of minimal HD coBüchi automata [AK22]).

▶ **Theorem 4.1.** *Let $\mathcal{A}$ and $\mathcal{B}$ be two equivalent consistent layered automata that are normal, central and safe minimal. Then, $\mathcal{A}$ and $\mathcal{B}$ are isomorphic (as layered automata).*

4.1 Normality, centrality and safe minimality

We define the three properties that identify minimal consistent layered automata.

▶ **Definition 4.2.** *A layered automaton is in normal form if for all $x \geq 2$ and $p \in Q_x$ the following two properties hold in $\mathcal{T}_x$:*

N1 *If $p \xrightarrow{u}_x q$, for some $u \in \Sigma^*$ then $q \xrightarrow{v}_x p$ for some $v \in \Sigma^+$.*
N2 *There is some u for which $\mu_{x-1}(p) \xrightarrow{u}_{x-1}$ is defined in $\mathcal{T}_{x-1}$ but $p \xrightarrow{u}_x \perp$ in $\mathcal{T}_x$.*

Being in normal form ensures that there are no evidently useless transitions, and moreover the priorities of transitions in $[\![\mathcal{A}]\!]$ cannot be lowered without modifying the recognised language. The intuitive meaning of the two properties is easier to see by looking at their negations. A transition between two SCCs in $\mathcal{T}_x$ can be removed, leading to its priority being lowered to $x-1$ in $[\![\mathcal{A}]\!]$. This does not change the accepted language, as such a transition cannot appear infinitely often on a run without another transition of priority $\leq x-1$ appearing infinitely often too. So after establishing N1 each $\mathcal{T}_x$ is a union of non-trivial SCCs. The second property says that μ_x does not completely cover all edges of some SCC of $\mathcal{T}_x$. In the semantics automaton this means that if $p \xrightarrow{a:\geq x}$, then there is u and a run $p \xrightarrow{u:x}$ from p producing x as minimal priority.

An *x-SCC* of a layered automaton is just an SCC of $\mathcal{T}_x$. The next property requires that no x-SCC can be simulated by another one. We define this formally below.

▶ **Definition 4.3.** *For $x \geq 2$, and $p, q \in Q_x$ we define $p \preccurlyeq_x q$ if $p \sim_{x-1} q$ (that is, $\mu_{x-1}(q) = \mu_{x-1}(p)$) and $L_x(p) \subseteq L_x(q)$.*

▶ **Definition 4.4.** *A layered automaton is centralised if for every $x \geq 2$ and for all states $p, q \in Q_x$, $p \preccurlyeq_x q$ implies p, q are in the same x-SCC.*

Finally, on each layer we can perform a similar minimisation construction as for finite automata by merging states with the same safe language.

We define by induction on x the relation $p \approx_x q$, for $p, q \in Q_{\geq x}$:

$$p \approx_1 q \quad \text{iff} \quad L(p) = L(q), \quad \text{and}$$

$$p \approx_x q \quad \text{iff} \quad p \approx_{x-1} q \quad \text{and} \quad L_x(p) = L_x(q) \quad (\text{for } x > 1).$$

Recall that $q \sim_x p$ if $q, p \in Q_{\geq x}$ have the same ancestor at layer x in $\mathcal{A}$.

▶ **Definition 4.5.** *A layered automaton is safe minimal if $\sim_x = \approx_x$, for all layers x.*

▶ Remark 4.6. In a consistent layered automaton $\sim_x \subseteq \approx_x$ for all x. This is because $p \sim_x q$ means $\widehat{\mu}_x(p) = \widehat{\mu}_x(q)$. By the consistency assumption, $L(p) = L(q)$, and for $1 < y \leq x$ the equality $L_y(p) = L_y(q)$ follows trivially. Safe minimality says that the safe language equivalence implies $\sim_x$. Namely, if $p, q \in Q_x$ with $L(p) = L(q)$ and $L_y(p) = L_y(q)$ for all $y \leq x$ then $p = q$.

4.2 Central sequences

Intuitively, a central sequence for a state p at some level x is a finite word that identifies the state p uniquely. The properties listed in previous subsection ensure that every state has a central sequence. This is an important step towards showing Theorem 4.1.

Let $\mathcal{A}$ be a layered automaton and $p \in Q_x$ an x-state. We say that $z_p \in \Sigma^+$ is a *central sequence* for p if:

- $p \xrightarrow{z_p}_x p$ in $\mathcal{T}_x$;
- for all $q \in Q_x$ with $q \sim_{x-1} p$, either $q \xrightarrow{z_p}_x p$ or $q \xrightarrow{z_p}_x \perp$;
- for all $q' \in Q_{x+1}$ with $q' \sim_{x-1} p$ we have $q' \xrightarrow{z_p}_{x+1} \perp$.

In $[\![\mathcal{A}]\!]$, this means that $p \xrightarrow{z_p:x} [p]_x$, and that from any other state in $[p]_{x-1}$, either $q \xrightarrow{z_p:x} [p]_x$ or $q \xrightarrow{z_p:<x}$, meaning z_p produces a priority $< x$. Observe that the third condition is needed to ensure that the runs in question do not have priority $> x$.

▶ **Lemma 4.7.** *Let $\mathcal{A}$ be a normal, centralised and safe minimal layered automaton. Then, for every $x \geq 2$, every state $p \in Q_x$ admits a central sequence.*

Proof. Let $P_x = \{p_1, \dots, p_k\}$ be the x-states in $[p]_{x-1}$.

Assume first that p is $\preccurlyeq_x$-maximal in P_x. By safe minimality of $\mathcal{A}$, there is no $p' \in P_x \setminus \{p\}$ with $L_x(p') = L_x(p)$. Since p is $\preccurlyeq_x$-maximal, for every $p' \in P_x \setminus \{p\}$ we can find $v_{p'} \in L_x(p) \setminus L_x(p')$. By normality, we can assume that $p \xrightarrow{v_{p'}}_x p$.

Let $u_0 \in \Sigma^+$ such that $p \xrightarrow{u_0}_x p$ and $\mu_x^{-1}(p) \xrightarrow{u_0}_{x+1} \perp$ (which exists by normality). Define u_i by induction:

- If $p_i \xrightarrow{u_{i-1}}_x \perp$ or $p_i \xrightarrow{u_{i-1}}_x p$ then $u_i = u_{i-1}$.
- If $p_i \xrightarrow{u_{i-1}}_x p' \neq p$, then we take the $v_{p'}$ chosen above and define $u_i = u_{i-1}v_{p'}$.

It is easy to check that u_k is a central sequence for p.

Assume now that p is not $\preccurlyeq_x$-maximal. Let p' be a $\preccurlyeq_x$-maximal state in P with $p \preccurlyeq_x p'$. As the automaton is centralised, p and p' are in the same SCC of $\mathcal{T}_x$. Let u, v be words such that $p \xrightarrow{u}_x p' \xrightarrow{v}_x p$. By the previous case, p' admits a central sequence $z_{p'}$. It is easy to check that $uz_{p'}v$ is a central sequence for p. ◀

4.3 Uniqueness of minimal layered automata

We are ready to prove Theorem 4.1.

▶ **Theorem 4.1.** *Let $\mathcal{A}$ and $\mathcal{B}$ be two equivalent consistent layered automata that are normal, central and safe minimal. Then, $\mathcal{A}$ and $\mathcal{B}$ are isomorphic (as layered automata).*

Let $\mathcal{A}$ and $\mathcal{B}$ be two consistent layered automata for L that are normal, central and safe minimal. We should construct an isomorphism $f : \mathcal{A} \to \mathcal{B}$. Such an isomorphism can be equivalently written as a sequence of isomorphisms of transition systems $f_x : \mathcal{T}_x^{\mathcal{A}} \to \mathcal{T}_x^{\mathcal{B}}$ compatible with the μ_x-functions. We build this sequence inductively on x.

The first layers $\mathcal{T}_1^{\mathcal{A}}$ and $\mathcal{T}_1^{\mathcal{B}}$ both consist of the residual classes of L, and therefore there is an isomorphism $f_1 : \mathcal{T}_1^{\mathcal{A}} \to \mathcal{T}_1^{\mathcal{B}}$. Now assume that we have constructed isomorphisms $f_1, \dots, f_{x-1}$ up to layer $x-1$ and want to extend it to layer x. First, we show that for each x-state p of $\mathcal{A}$ with parent $\mu_{x-1}^{\mathcal{A}}(p) = r$, there is an x-state q of $\mathcal{B}$ with parent $f_{x-1}(r)$, such that q and p have the same safe language (Claim 4.8). By safe minimality, such a state q is unique. We then define $f_x(p) = q$. The roles of the two automata are symmetric in the proof, so we get that both automata have states for the same safe languages on layer x. This implies that f_x is an ismorphism between $\mathcal{T}_x^{\mathcal{A}}$ and $\mathcal{T}_x^{\mathcal{B}}$ (Claim 4.9). Moreover, by definition, f_x is compatible with the μ_x-functions: $f_{x-1}(\mu_{x-1}^{\mathcal{A}}(p)) = \mu_{x-1}^{\mathcal{B}}(f_x(p))$. This concludes the proof of Theorem 4.1.

▷ Claim 4.8. For each x-state p of $\mathcal{A}$ with parent $\mu_{x-1}^{\mathcal{A}}(p) = r$, there is a unique x-state q of $\mathcal{B}$ with parent $f_{x-1}(r)$ such that ${L_x}^{\mathcal{A}}(p) = {L_x}^{\mathcal{B}}(q)$.

Proof. Unicity of q directly follows from safe minimality. We show existence via the following construction, which is illustrated in Figure 6. Let p be a x-state of $\mathcal{A}$ with parent r, and let $f_{x-1}(r) = s$ be the state on layer $x-1$ of $\mathcal{B}$ that r is mapped to. Let z_p be a central sequence for p. Then z_p^ω generates minimal priority x when read in $[\![\mathcal{A}]\!]$ starting from any state in $p' \sim_x p$. Moreover, we have that z_p loops on r since it loops on p, and thus, z_p also loops on s in $\mathcal{B}$ because layers $x-1$ of $\mathcal{A}$ and $\mathcal{B}$ are isomorphic. Therefore, z_p^ω generates minimal priority at least $x-1$ when read from s in $\mathcal{B}$. Since r and s are language equivalent, there needs to be some child q' of s such that z_p^ω is x-safe from q'. Let q be such that $q' \xrightarrow{z_p}_x q$. Note that q also has parent s because z_p loops on s.

Let z_q be a central sequence for q, and w be such that $q \xrightarrow{w}_x q'$ (which exists because $\mathcal{B}$ is normal). Note that z_q and w loop on s and hence also on r. The ω-word $(z_q w z_p)^\omega$ produces priority x when read starting from q in $\mathcal{B}$. Since p and q are language equivalent (because their parents are), it has to be possible in $\mathcal{A}$ to produce a priority $\geq x$ on $(z_q w z_p)^\omega$ from a child of r. Since z_p is central for p, such a run is in p after the first $z_q w z_p$, and hence $(z_q w z_p)^\omega$ must be safe from p. Let p_1, p_2 such that $p \xrightarrow{z_q} p_1 \xrightarrow{w} p_2$. Since z_p is central for p, we get that $p_2 \xrightarrow{z_p : x} p$.

We claim that ${L_x}^{\mathcal{A}}(p) = {L_x}^{\mathcal{B}}(q)$. Assume the contrary. The first case is that there is $u \in L_x(p) \setminus L_x(q)$. Since $\mathcal{A}$ is normal, we can choose u such that it loops on p. Hence it also loops on r and s. Then the ω-word $(z_q w z_p u)^\omega$ generates priority x from p and priority $x-1$ from q. The latter holds because either priority $x-1$ is generated on z_q, and if not, then the state after z_q is q and hence also after $z_q w z_p$, and thus priority $x-1$ is produced on u. Since p and q are language equivalent, this is a contradiction.

The second case is that there is $v \in L_x(q) \setminus L_x(p)$. Then we get that $(z_q w z_p v)^\omega$ generates priority x from q and priority $x-1$ from p using the same type of argument as in the first case. ◁

We let $f_x(p) = q$, for q the unique x-state in $\mathcal{B}$ satisfying $f_{x-1}(\mu_{x-1}^{\mathcal{A}}(p)) = \mu_{x-1}^{\mathcal{B}}(q)$ and ${L_x}^{\mathcal{A}}(p) = {L_x}^{\mathcal{B}}(q)$.

▷ Claim 4.9. The function f_x is an isomorphism of transition systems.

Proof. We first show that f_x is a morphism. Let $p \xrightarrow{a}_x p'$ in $\mathcal{A}$ and $q = f(p)$. Since ${L_x}^{\mathcal{A}}(p) = {L_x}^{\mathcal{B}}(q)$, there is a transition $q \xrightarrow{a}_x q'$ in $\mathcal{B}$, and ${L_x}^{\mathcal{A}}(p') = {L_x}^{\mathcal{B}}(q')$. If r is the

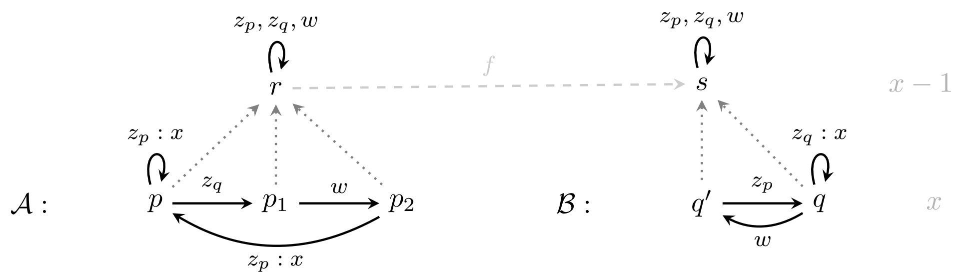

Figure 6 Illustration for the construction of the isomorphism in the proof of Claim 4.8.

parent of p we have $r \xrightarrow{a} r'$ for some r' in $\mathcal{A}$. By definition of $f(p)$, the parent of $q = f(p)$ is $f_{x-1}(r)$. By isomorphism, $f_{x-1}(r) \xrightarrow{a} f_{x-1}(r')$ thus $f_{x-1}(r')$ is the parent of q'. So by definition, we have $f(p') = q'$. Thus, f_x is a morphism, as desired.

By Claim 4.8, f_x is a bijection, and the function f_x^{-1} is also a morphism by symmetry. ◁

5 Minimisation in polynomial time

In this section, we show that we can obtain the canonical consistent layered automaton of an ω-regular language in polynomial time, which is also minimal.

▶ **Theorem 5.1.** *Let $\mathcal{A}$ be a consistent layered automaton. We can build in polynomial time an equivalent consistent layered automaton $\mathcal{A}_L$ that is normal, central and safe minimal.*

Moreover, $\mathcal{A}$ contains an equivalent subautomaton that admits a surjective morphism to $\mathcal{A}_L$. In particular, $\mathcal{A}_L$ has no more states nor leaf states than $\mathcal{A}$.

Together with Theorem 4.1, this gives us minimality and canonicity of normal, central and safe minimal consistent layered automata.

▶ **Corollary 5.2.** *Let $\mathcal{A}_L$ be a consistent layered automaton that is normal, central and safe minimal. Then, any equivalent consistent layered automaton $\mathcal{A}$ contains an equivalent subautomaton that admits a surjective morphism to $\mathcal{A}_L$. Therefore, $\mathcal{A}$ has at least as many states as $\mathcal{A}_L$, and $[\![\mathcal{A}]\!]$ has at least as many states as $[\![\mathcal{A}_L]\!]$.*

The rest of the section is devoted to the proof of Theorem 5.1. We start by introducing the notion of strong morphism and proving some technical lemmas that are used to show correctness of the minimisation procedure.

5.1 Strong morphisms

In the upcoming sections, we start with a consistent layered automaton $\mathcal{A}$ and tweak it to obtain a modified automaton $\mathcal{A}'$. In all modifications, except the last one, we obtain $\mathcal{A}'$ by *removing* states or transitions of $\mathcal{A}$ in such a way that there is an injection $\mathcal{A}' \hookrightarrow \mathcal{A}$. The correctness of this operation is ensured by Lemma 5.3 below. In the last step (Section 5.4), $\mathcal{A}'$ is obtained by *merging* states of $\mathcal{A}$ and we obtain a surjection $\mathcal{A} \twoheadrightarrow \mathcal{A}'$.

In general, morphisms of layered automata do not need to preserve the language accepted. Strong morphisms, as defined next, do preserve it.

We say that a morphism of layered automata $\varphi : \mathcal{A} \to \mathcal{A}'$ is a *strong morphism* if for every $p \in Q_{\geq 1}$ and every $w \in \Sigma^\omega$ we have:

- if w is strongly accepted by p then w is strongly accepted by $\varphi(p)$,
- if w is strongly rejected by p then w is strongly rejected by $\varphi(p)$.

We note that a strong morphism may not be layer-preserving.

▶ **Lemma 5.3.** *Let $\varphi : \mathcal{A}' \to \mathcal{A}$ be a strong morphism. If $\mathcal{A}$ is consistent then $\mathcal{A}'$ is consistent and $L(\mathcal{A}') = L(\mathcal{A})$.*

Proof. Take $p'_1 \sim_1 p'_2$ in $\mathcal{A}'$. Towards a contradiction suppose w is strongly accepted by p'_1 and strongly rejected by p'_2. By the morphism property we have $\varphi(p_1) \sim_1 \varphi(p_2)$. Then by the definition of a strong morphism, w is strongly accepted by $\varphi(p'_1)$ and strongly rejected by $\varphi(p'_2)$. A contradiction with consistency of $\mathcal{A}$.

If w is accepted by $\mathcal{A}'$ then by Lemma 3.16 there is a state p together with a decomposition $uw' = w$ such that $q'_{\mathsf{init}} \xrightarrow{u} \widehat{\mu}_1(p)$ and w' strongly accepted from p. Then in $\mathcal{A}$ we have $q_{\mathsf{init}} \xrightarrow{u} \widehat{\mu}_1(\varphi(p))$ and w' is strongly accepted from $\varphi(p)$. By consistency of $\mathcal{A}'$ and Lemma 3.16, w is accepted by $\mathcal{A}'$. The argument when w is rejected by $\mathcal{A}'$ is analogous. ◀

5.2 Normalisation

We need to build an equivalent automaton having the properties N1 and N2.

5.2.1 Separating SCCs

In order to ensure property N1 we separate every $\mathcal{T}_x$ into disconnected, non-trivial SCCs. Let us fix $x > 1$ and suppose that in $\mathcal{T}_x$ there is a transition $p \xrightarrow{a}_x q$ of $\mathcal{T}_x$, such that the two states are not in the same SCC. The operation is simply to remove $p \xrightarrow{a}_x q$ from $\mathcal{T}_x$ and all transitions mapping to it via $\widehat{\mu}_x$ in $\mathcal{T}_y$ for $y > x$. We also remove trivial SCCs that may have been generated (states in $\mathcal{T}_y$ with no outgoing transition). Let us denote the resulting automaton $\mathcal{A}^{sep}$.

▶ **Lemma 5.4.** *Let $\mathcal{A}$ be a consistent layered automaton. Then, $\mathcal{A}^{sep}$ is consistent and $L(\mathcal{A}^{sep}) = L(\mathcal{A})$. Moreover, $\mathcal{A}^{sep}$ is a subautomaton of $\mathcal{A}$.*

Proof. We show that the mapping $\varphi \colon \mathcal{A}^{sep} \to \mathcal{A}$ given by $\varphi(p) = p$ is an injective strong morphism. The result then follows by Lemma 5.3. The fact that φ is a morphism and injective is straightforward.

Let w strongly accepted by p in $\mathcal{A}^{sep}$ (the case strongly rejected is symmetric). In particular, $w \in L_x^{\mathcal{A}^{sep}}(p)$, so $w \in L_x^{\mathcal{A}}(p)$ by the morphism property. Assume by contradiction that w is not strongly accepted by p in $\mathcal{A}$, that is, there is a decomposition $w = uw'$ and state $q \in Q_{x+1}$ with $p \xrightarrow{u}_x \widehat{\mu}_x(q)$ in $\mathcal{A}$ and $w' \in L_{x+1}^{\mathcal{A}}(q)$. Let $w' = u'w''$ such that the run $q \xrightarrow{u'}_{x+1} q' \xrightarrow{w''}_{x+1}$ does not change of SCC during the suffix w''. Then, $w'' \in L_{x+1}^{\mathcal{A}^{sep}}(q')$. Therefore, the decomposition $w = uu' \cdot w''$ witnesses that w is not strongly accepted by p in $\mathcal{A}^{sep}$, a contradiction. ◀

After applying this operation we remove one transition. So repeating this operation we finally obtain an equivalent consistent subautomaton satisfying the property N1.

5.2.2 Lowering

We suppose that $\mathcal{A}$ satisfies the property N1, and we make it satisfy N2 without affecting N1. Let us suppose that N2 does not hold for a state p at level $x + 1$. This means that for every $u \in \Sigma^*$ such that $\mu_x(p) \xrightarrow{u}_x$ we also have $p \xrightarrow{u}_{x+1}$. Take S_{x+1}, the SCC of p in $\mathcal{T}_{x+1}$, as well as S_x, the SCC of $\mu_x(p)$ in $\mathcal{T}_x$. We have that S_x is *covered* by S_{x+1}, meaning that for every $q \in S_{x+1}$ such that there is a transition $\mu_x(q) \xrightarrow{a}_x$ we also have $q \xrightarrow{a}_{x+1}$. In $[\![\mathcal{A}]\!]$ this means that states mapping to S_{x+1} do not have outgoing transitions of priority x; if $\widehat{\mu}_x(q) \in S_x$ then $q \xrightarrow{a:x}$ is not possible. We show how to eliminate a covered SCC at level x.

Here is a small lemma saying that SCCs in higher levels are included in SCCs of lower levels.

▶ **Lemma 5.5.** *Let $S \subseteq Q_x$ be an SCC in $\mathcal{T}_x$, and $S' \subseteq Q_{x+1}$ be an SCC in $\mathcal{T}_{x+1}$. If $\mu_x(S') \cap S \neq \emptyset$ then $\mu_x(S') \subseteq S$.*

Proof. This is just because μ_x preserves paths. ◀

Suppose $\mathcal{A}$ satisfies N1 and fix S_x, S_{x+1} such that S_x is covered by S_{x+1} as described above. For every $y \geq x+2$ let $S_y = \{p \in Q_y : \widehat{\mu}_x(p) \in S_x\}$ be the states mapping to S_x. Observe that by Lemma 5.5, each S_y is a set of SCCs in $\mathcal{T}_y$.

A new layered automaton $\mathcal{A}^{low}$ is obtained by lowering some components. For $y \geq x+2$:

- $\mathcal{T}_{y-2}^{low} = (\mathcal{T}_{y-2} - S_{y-2}) \cup S_y$.
- $\mu_{y-3}^{low}(p) = \begin{cases} \mu_{y-1}(p) & \text{if } p \in S_y, \\ \mu_{y-3}(p) & \text{otherwise.} \end{cases}$

That is, the SCCs in $\mathcal{T}_y$ that have S_x as ancestor (that are mapped to S_x by $\widehat{\mu}_x$) are overwritten by the one in $\mathcal{T}_{y+2}$. In particular, we erase the "redundant" SCCs in $\mathcal{T}_x$ and $\mathcal{T}_{x+1}$. In terms of $[\![\mathcal{A}^{low}]\!]$ this means that priorities of transitions coming from S_{x+2} are lowered by 2. Moreover, the transitions that come from S_{x+1} in $[\![\mathcal{A}]\!]$ have their priority lowered by 1, changed to x. Finally, also some transitions carrying priority $x-1$ may be added as S_x may have fewer transitions than $\mu_{x-1}(S_x)$.

▶ **Lemma 5.6.** *Let $\mathcal{A}$ be a consistent layered automaton satisfying N1. Then, $\mathcal{A}^{low}$ is consistent, satisfies N1 and $L(\mathcal{A}^{low}) = L(\mathcal{A})$. Moreover, $\mathcal{A}^{low}$ is a subautomaton of $\mathcal{A}$.*

Proof. The automaton $\mathcal{A}^{low}$ satisfies N1, as no transitions between SCCs are added. We show that the mapping $\varphi\colon \mathcal{A}^{low} \to \mathcal{A}$ given by $\varphi(p) = p$ is an injective strong morphism. The result then follows by Lemma 5.3 (recall that, by definition, $\mathcal{A}^{low}$ is a subautomaton of $\mathcal{A}$ if there exists an injective morphism to $\mathcal{A}$). The fact that φ is a morphism and injective is straightforward.

Let w be strongly accepted (resp. rejected) by p in $\mathcal{A}^{low}$. Note that the parity of the layers of p and $\varphi(p)$ coincide, as $\mathit{layer}(\varphi(p)) \in \{\mathit{layer}(p), \mathit{layer}(p)+2\}$. Also, the SCCs of p and $\varphi(p)$ in their corresponding layers are isomorphic, as well as the SCCs of their children. Therefore, w is strongly accepted (resp. rejected) by $\varphi(p)$. ◀

After each lowering operation we remove some states. When there are no covered SCCs we obtain an equivalent subautomaton that in normal form, namely, satisfying both N1 and N2.

5.3 Centralisation

We now show how to make a normalised automaton centralised (more precisely, it is enough to assume that the automaton $\mathcal{A}$ satisfies N1). The following lemma shows that we can extend the $\preccurlyeq_x$ relation to compare x-SCCs.

▶ **Lemma 5.7.** *Let $\mathcal{A}$ be a consistent layered automaton satisfying N1. Let S_1, S_2 be two x-SCCs in $\mathcal{T}_x$. If there are $p_1 \in S_1$ and $p_2 \in S_2$ such that $p_1 \preccurlyeq_x p_2$ then for every $q_1 \in S_1$ there is $q_2 \in S_2$ with $q_1 \preccurlyeq_x q_2$.*

Proof. Take a path $p_1 \xrightarrow{u}_x q_1$ in $\mathcal{T}_x$. Since $p_1 \preccurlyeq_x p_2$, $u \in L_x p_2$. Let q_2 be the state reached in $\mathcal{T}_x$ by the path $p_2 \xrightarrow{u}_x q_2$. As $\mathcal{A}$ satisfies N1, $q_2 \in S_2$. Clearly, $q_1 \sim_x q_2$. Finally, $L^x(q_1) = u^{-1}L^x(p_1) \subseteq u^{-1}L^x(p_2) = L^x(q_2)$, so $p_2 \preccurlyeq_x q_2$.. ◀

We describe a *centralisation operation* removing x-SCCs that are not $\preccurlyeq_x$-maximal. Fix $x \geq 2$ odd and two components $S \preccurlyeq_x S'$. Let $\mathcal{A}^{ctl}$ be the layered automaton with S removed. This means that for every $y \geq x$ we remove all $p \in Q_y$ such that $\widehat{\mu}_x(p) \in S$.

Let us see what this operation means in terms of the semantics of $\mathcal{A}^{ctl}$. In $[\![\mathcal{A}^{ctl}]\!]$, we may have removed some states (leafs below S are potentially merged). More precisely, all states in $\mu_{x-1}^{-1}(S)$ become leaf states. Transitions between these states that had priority $\geq x$ in $[\![\mathcal{A}]\!]$ are lowered to priority $x-1$. Thanks to the property N1, we know that the priority of transitions between these new states and the exterior of S are not modified.

▶ **Lemma 5.8.** *Let $\mathcal{A}$ be a consistent, normal layered automaton. Then, $\mathcal{A}^{ctl}$ is consistent, normal and $L(\mathcal{A}^{ctl}) = L(\mathcal{A})$. Moreover, $\mathcal{A}^{ctl}$ is a subautomaton of $\mathcal{A}$.*

Proof. Since $\mathcal{A}^{ctl}$ has been obtained by removing whole x-SCCs, no transitions between existent SCCs are added, nor any x-SCC is covered in $\mathcal{A}^{ctl}$. Therefore, $\mathcal{A}^{ctl}$ is also in normal form.

We show that the mapping $\varphi\colon \mathcal{A}^{ctl} \to \mathcal{A}$ given by $\varphi(p) = p$ is an injective strong morphism. The result then follows by Lemma 5.3. The fact that φ is a morphism and injective is straightforward. Let w be strongly accepted (resp. rejected) by p in $\mathcal{A}^{ctl}$. As the x-SCCs of p and all its children are the same in $\mathcal{A}$ and in $\mathcal{A}^{ctl}$, w is also strongly accepted (resp. rejected) by p in $\mathcal{A}$. ◀

Every application of the centralisation operation removes some states. When the operation cannot be applied anymore, we get an equivalent subautomaton that is normal and centralised.

5.4 Safe minimisation

We show how we can safe minimise a layered automaton while maintaining normalisation and centralisation. First, we obtain directly from the definition that the $\approx_x$ relation is a bisimulation in T_x.

▶ **Lemma 5.9.** *Let $p, p' \in Q_x$ such that $p \approx_x p'$. If $p \xrightarrow{a}_x q$ then $p' \xrightarrow{a}_x q'$ and $q' \approx_x q$.*

The desired safe minimal automaton $\mathcal{A}^{\mathrm{smin}}$ is the automaton obtained by using $\approx_x$ equivalence classes as x-states. Formally, $\mathcal{A}^{\mathrm{smin}}$ is given by:

- $\mathcal{T}_x^{\mathrm{smin}} = (Q_x^{\mathrm{smin}}, \delta_x^{\mathrm{smin}})$ with
  - $Q_x^{\mathrm{smin}} = \{[q]_{\approx_x} \mid q \in Q_x\}$,
  - $\delta_x^{\mathrm{smin}}([q]_{\approx_x}, a) = \begin{cases} [\delta_x(q,a)]_{\approx_x} & \text{if } \delta_x(q,a) \text{ defined,} \\ \bot & \text{else.} \end{cases}$
- The initial state is $[q_0]_{\approx_1}$, for q_0 the initial state in $\mathcal{T}_1$.
- $\mu_x^{\mathrm{smin}}([q]_{\approx_{x+1}}) = [\mu_x(q)]_{\approx_x}$.

The transition function δ^{smin} is well-defined by Lemma 5.9. A small calculation is needed to check that μ_x^{smin} is a morphism. Lemma 5.9 implies that ${L_x}^{\mathcal{A}}(p) = {L_x}^{\mathcal{A}^{\mathrm{smin}}}([p]_{\approx_x})$. Thus, $\mathcal{A}^{\mathrm{smin}}$ is safe minimal. Moreover, the other properties we have already established are not invalidated by this construction.

▶ **Lemma 5.10.** *If $\mathcal{A}$ is normal and centralised then so is $\mathcal{A}^{\mathrm{smin}}$.*

Proof. We check the properties N1, N2 and centralisation.

N1. Let $[p]_{\approx_x} \xrightarrow{u}_x [q]_{\approx_x}$ in $\mathcal{T}_x^{\mathrm{smin}}$. Then, $p \xrightarrow{u}_x q$ in $\mathcal{T}_x$, so $q \xrightarrow{v}_x p$ for some v, as $\mathcal{A}$ satisfies N1. Then, $[q]_{\approx_x} \xrightarrow{v}_x [p]_{\approx_x}$.

N2. Let u such that $\mu_{x-1}(p) \xrightarrow{u}_{x-1}$ in $\mathcal{T}_{x-1}$ but not $p \xrightarrow{u}_x$ in $\mathcal{T}_x$. Then, ${\mu^{\mathrm{smin}}}_{x-1}([p]_{\approx_x}) \xrightarrow{u}_{x-1}$ in ${\mathcal{T}^{\mathrm{smin}}}_{x-1}$ but not $[p]_{\approx_x} \xrightarrow{u}_x$ in ${\mathcal{T}^{\mathrm{smin}}}_x$.

Centralisation. Let $[p]_{\approx_x} \preccurlyeq_x [q]_{\approx_x}$. As $L_x(p) = L_x^{\mathcal{A}^{\mathrm{smin}}}([p]_{\approx_x})$, it also holds $p \preccurlyeq_x^{\mathcal{A}} q$, so p and q are in the same x-SCC in $\mathcal{A}$. Therefore $[p]_{\approx_x}$ and $[q]_{\approx_x}$ are in the same x-SCC in $\mathcal{A}^{\mathrm{smin}}$. ◀

The following lemma shows the key property to obtain correctness of $\mathcal{A}^{\mathrm{smin}}$.

▶ **Lemma 5.11.** *Let $\mathcal{A}$ be a consistent layered automaton and $p \in Q_x$ an x-state. If a word $w \in \Sigma^\omega$ is strongly accepted (resp. rejected) by $[p]_{\approx_x}$ in $\mathcal{A}^{\mathrm{smin}}$, then it is strongly accepted (resp. rejected) by p in $\mathcal{A}$.*

Proof. Assume that w is strongly accepted by $[p]_{\approx_x}$ in $\mathcal{A}^{\mathrm{smin}}$(so x is even). In particular, $w \in L_x^{\mathcal{A}^{\mathrm{smin}}}([p]_{\approx_x})$, and therefore $w \in L_x^{\mathcal{A}}(p)$. Assume by contradiction that w is not strongly accepted by p. Then, there is a decomposition $w = uw'$ and a state $q' \in Q_{x+1}$ such that $p \xrightarrow{u}_x q$, where $q = \widehat{\mu}_x(q')$, and $w' \in L_{x+1}^{\mathcal{A}}(q')$. We claim that the decomposition $w = uw'$ witnesses that w is not strongly accepted by $[p]_{\approx_x}$ in $\mathcal{A}^{\mathrm{smin}}$, a contradiction. Indeed, by definition of δ_x^{smin} we have that $[p]_{\approx_x} \xrightarrow{u}_x [q]_{\approx_x}$, and also $w' \in L_{x+1}^{\mathcal{A}^{\mathrm{smin}}}([q']_{\approx_{x+1}})$. As $\mu_x^{\mathrm{smin}}([q']_{\approx_{x+1}}) = [q]_{\approx_x}$, this concludes the proof. ◀

We are ready to prove the correctness of the construction of $\mathcal{A}^{\mathrm{smin}}$.

▶ **Lemma 5.12.** *Let $\mathcal{A}$ be a consistent layered automaton. Then, $\mathcal{A}^{\mathrm{smin}}$ is consistent, safe minimal and $L(\mathcal{A}) = L(\mathcal{A}^{\mathrm{smin}})$. Moreover, $\mathcal{A}$ admits a surjective morphism to $\mathcal{A}^{\mathrm{smin}}$.*

Proof. By Lemma 5.9, the mapping $\varphi\colon \mathcal{A} \to \mathcal{A}^{\mathrm{smin}}$ is a morphism, which is clearly surjective.

We show consistency of $\mathcal{A}^{\mathrm{smin}}$. Let $[p]_{\approx_x}$ and $[p']_{\approx_{x'}}$ be $\sim_1$-equivalent in $\mathcal{A}^{\mathrm{smin}}$, that is, $L(\mathcal{A}, p) = L(\mathcal{A}, p')$. Assume by contradiction that a word w is strongly accepted by $[p]_{\approx_x}$ and strongly rejected by $[p']_{\approx_{x'}}$ in $\mathcal{A}^{\mathrm{smin}}$. By Lemma 5.11, w is strongly accepted by p and strongly rejected by p' in $\mathcal{A}$. Therefore, by Lemma 3.15, $w \in L(\mathcal{A}, p)$ and $w \notin L(\mathcal{A}, p')$, contradicting language equivalence of these states.

We prove that $L(\mathcal{A}^{\mathrm{smin}}) = L(\mathcal{A})$. Let w be a word accepted by $\mathcal{A}^{\mathrm{smin}}$ (similar proof for w rejected), that is, there is x even and a decomposition $w = uw'$ such that $[q_{\mathsf{init}}]_{\approx_1} \xrightarrow{u}_1 [q]_{\approx_1}$ and w' is strongly accepted by $[q']_{\approx_x}$, with $\widehat{\mu}_1(q') \approx_1 q$. By Lemma 5.11, w' is strongly accepted by q' in $\mathcal{A}$, and therefore by Lemma 3.15, $w' \in L(\mathcal{A}, q') = u^{-1}L(\mathcal{A})$, where the last equality follows from uniformly semantic determinism of $\mathcal{A}$.

Finally, safe minimality of $\mathcal{A}^{\mathrm{smin}}$ follows from language equivalence and the fact that $L_x(p) = L_x^{\mathcal{A}^{\mathrm{smin}}}([p]_{\approx_x})$. ◀

This finishes the proof of Theorem 5.1. Indeed, the automaton $\mathcal{A}'$ obtained after normalising and centralising $\mathcal{A}$ is an equivalent subautomaton. This automaton $\mathcal{A}'$ admits a surjective morphism to $\mathcal{A}_L$, obtained by safe minimising $\mathcal{A}'$.

6 Congruence-based characterisation

In this section, we provide a definition based on congruences of the canonical layered automaton of an ω-regular language.

6.1 Definitions and basic properties of congruences

Let $L \subseteq \Sigma^\omega$ (the definitions can be applied to every language of infinite words, but the results concerning layered automata only work for regular ω-languages). In our congruence-based description of the minimal layered automaton for L, the states on layer x are classes of an equivalence relation $\equiv_L$ over x-tuples $\overline{u} = (u_1, \ldots, u_x) \in (\Sigma^*)^x$ of finite words. The relation $\equiv_L$ can be seen as one equivalence relation over the set of all finite tuples (of length at least one) of finite words over Σ, in which equivalent tuples must have the same length. For defining the transition relations on the individual layers and the morphisms between the layers, we consider two operations on such finite tuples:

Concatenation: We write $\overline{u} \cdot u_{x+1}$ for concatenation with a word $u_{x+1} \in \Sigma^*$ in the last component, that is $\overline{u} \cdot u_{x+1} = (u_1, \ldots, u_{x-1}, u_x u_{x+1})$ is the tuple that has the same first $x - 1$ components as $\overline{u}$, and $u_x u_{x+1}$ as last component (if $x = 1$, then the resulting tuple is just $(u_x u_{x+1})$). We use this operation for defining the transition relation on the individual layers.

Merging: We write $(\overline{u}, u_{x+1})$ for the tuple $(u_1, \ldots, u_x, u_{x+1})$ obtained by adding u_{x+1} as additional component. Merging the last two components of the tuple $(\overline{u}, u_{x+1})$ of length at least 2, is the operation of removing the last component and concatenating it to the second last component, resulting in the tuple $\overline{u} \cdot u_{x+1}$. We use this operation for the morphism from layer $x + 1$ to layer x.

Before going into the formal definitions, let us look an example to get an intuition for the notions that we are going to define.

▶ **Example 6.1.** Consider the language L of all words over $\Sigma = \{a, b\}$ in which aba occurs finitely often, and a and b both occur infinitely often. The congruence automaton for L is shown in Figure 7. The language is prefix independent, so there is only one class on layer 1, and all the representatives of classes in the other layers can use ε as first component. **Layer** 2 ensures that all words that do not contain aba are safe from some state. In order to capture this, we introduce the notion of a tuple being (equivalent to) $\perp$. In the example, $(u_1, u_2) \equiv_L \perp$ iff u_2 contains aba as infix. This is captured by looking at possible extensions $u_2 w$ of u_2 into words $u_1(u_2 w)^\omega \in L$. Such an extension exists iff u_2 does not contain aba. And the tuples for which no such extension exists are declared $\perp$ (when we discuss layer 3 of the example further below, we give some more details on the definition of $\perp$). This notion of tuples being $\perp$ defines a safe language for each tuple: the set of finite words that can be appended to the last component without making the tuple $\perp$. On layer x our equivalence merges x-tuples that are equivalent w.r.t. this safe language. This results in three classes on layer 2 in the example that track the prefixes of aba, namely $[(\varepsilon, \varepsilon)]_{\equiv_L}, [(\varepsilon, a)]_{\equiv_L}, [(\varepsilon, ab)]_{\equiv_L}$. In the picture, the representative (ε, bb) is used for the class $[(\varepsilon, \varepsilon)]_{\equiv_L}$. The intuitive reason is that bb is a word that ensures the following: concatenating it to any tuple (on layer 2) results in a tuple that is either $\perp$ or equivalent to (ε, bb). So in that sense, bb in the second component points to a unique class. This is captured by our definition of pointed tuples further below. We use this notion for selecting those classes that are used for the construction of the automaton: only classes that contain a pointed tuple are used. There can be classes

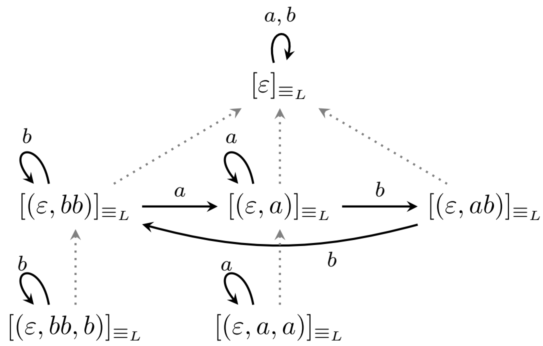

Figure 7 Congruence automaton for the language of words with finitely many aba and infinitely many a and b from Example 6.1.

that contain only pointed tuples or both, pointed and non-pointed tuples (as $(\varepsilon, \varepsilon)$ and (ε, bb) in the example), but there can also be classes that do not contain any pointed tuple, and these are not used as states. The latter case does not happen in this example, we refer to Example 6.6 for this. **Layer 3** is responsible for checking that infinitely many a and b occur, so words that contain only one of the two letters have to be safe on layer 3 (and are thus rejected because 3 is odd). Note that there is no state on layer 3 that maps to $[(\varepsilon, ab)]_{\equiv_L}$ on layer 2. The reason is that if $[(\varepsilon, ab)]_{\equiv_L}$ occurs infinitely often in a safe run on layer 2, then the word has to contain infinitely many a and b, because all words looping on that class on layer 2 contain a and b. So there is nothing to check on layer 3 in this case. This is captured by our definition of $\perp$ as follows. We say that a tuple (u_1, u_2, u_3) on layer 3 is $\perp$ if either (u_1, u_2u_3) is $\perp$ on layer 2, or every extension $w \in \Sigma^+$ that closes a loop on the class of (u_1, u_2), written as $(u_1, u_2u_3w) \equiv_L (u_1, u_2)$, is such that the resulting word $u_1u_2(u_3w)^\omega$ is in L. Intuitively, this means that we do not have to track anything on layer 3 because there is nothing to reject if the class on layer 2 occurs infinitely often. Consider the tuple $(\varepsilon, ab, \varepsilon)$ in the example. Merging the last two components results in (ε, ab). Our definition ensures that $(\varepsilon, ab, \varepsilon) \equiv_L \perp$ because every nonempty w such that $(\varepsilon, abw) \equiv_L (\varepsilon, ab)$ must be such that abw does not contain aba (otherwise (ε, abw) would be $\perp$ and thus not equivalent to (ε, ab)), and w ends in ab. Then clearly $ab(w)^\omega \in L$ because w contains both a and b, and $ab(w)^\omega$ does not contain aba, which follows from the facts that abw does not contain aba and that w ends in ab.

We now proceed with the formal definitions, which generalise the definitions for the co-Büchi case on pairs of words from [LW25].[8]

For tuples $\overline{u} = (u_1, \ldots, u_x)$, $\overline{v} = (v_1, \ldots, v_x)$ of finite words, we define inductively over the length of the tuples the notions $\overline{u} \equiv_L \perp$, $\overline{u} \equiv_L \overline{v}$, and $\overline{u}$ is pointed.

For the induction base $x = 1$, define

- $(u_1) \not\equiv_L \perp$ for all u_1,
- $(u_1) \equiv_L (v_1)$ iff $u_1 \sim_L v_1$,

[8] There is a small difference. In [LW25] for the co-Büchi case all states of the congruence automaton are classes of tuples of length 2. With our definitions here applied to the co-Büchi case, there might also be states that are classes of tuples of length 1 in some cases. This flexibility of tuple length actually makes the definition simpler because it avoids a special treatment of ε as in [LW25].

- (u_1) pointed for all u_1.

Bottom and safe language. For the induction step, let $(\overline{u}, u_{x+1}) \equiv_L \perp$ if

$\overline{u} \cdot u_{x+1} \equiv_L \perp$, or

$\forall w \in \Sigma^+ : \overline{u} \cdot u_{x+1} w \equiv_L \overline{u} \Rightarrow [u_1 \cdots u_x (u_{x+1} w)^\omega \in L \iff x \text{ even}]$.

Note that we require w to be non-empty in order to ensure that the looping part $u_{x+1}w$ is non-empty.

▶ Remark 6.2. **1.** Let $(\overline{u}, u_{x+1})$ be such that there is no $w \in \Sigma^+$ so that $\overline{u} \cdot u_{x+1} w \equiv_L \overline{u}$. Then, $(\overline{u}, u_{x+1}) \equiv_L \perp$, as the quantification "$\forall w$ [...]" is vacuous.

2. In proofs we often show that a tuple is $\not\equiv_L \perp$ by finding a $w \in \Sigma^+$ with $\overline{u} \cdot u_{x+1} w \equiv_L \overline{u}$ and $[u_1 \cdots u_x (u_{x+1} w)^\omega \in L \iff x+1 \text{ even}]$ (note the change to "$x+1$ even" here). We sometimes call such a w a *witness for the tuple not being* $\perp$.

The next lemma states that $\perp$ is invariant under right concatenation in the last component, and thus we can use it for defining a safety language.

▶ **Lemma 6.3.** *Let $\overline{u} \in (\Sigma^*)^x$ and $a \in \Sigma$. Then $\overline{u} \equiv_L \perp \Rightarrow \overline{u} \cdot a \equiv_L \perp$.*

Proof. Induction on x. For $x = 1$ the claim is true because no tuple of length 1 is $\perp$. For $x+1$, assume that $(\overline{u}, u_{x+1}a) \not\equiv_L \perp$. Then $\overline{u} \cdot u_{x+1}a \not\equiv_L \perp$, and hence by induction $\overline{u} \cdot u_{x+1} \not\equiv_L \perp$. Further, there is $w \in \Sigma^+$ such that $\overline{u} \cdot u_{x+1}aw \equiv_L \overline{u}$ and $u_1 \cdots u_x (u_{x+1}aw)^\omega \in L$ iff $x+1$ even. Then aw witnesses that $(\overline{u}, u_{x+1}) \not\equiv_L \perp$. ◀

For an infinite word $w \in \Sigma^\omega$ we define $(\overline{u}, w) \not\equiv_L \perp$ if $(\overline{u}, v) \not\equiv_L \perp$ for all prefixes v of w. The *safe language* of a tuple is then the set of finite and infinite words that can be concatenated to the tuple without making the tuple $\perp$:

$$L^s((\overline{u}, u_{x+1})) := \{w \in \Sigma^* \cup \Sigma^\omega \mid (\overline{u}, u_{x+1}w) \not\equiv_L \perp\}.$$

Note that the safe language is completely determined by the finite word it contains (the infinite words in the safe language are those for which all finite prefixes are in the safe language).

Equivalence. Based on that, let

$$(\overline{u}, u_{x+1}) \equiv_L (\overline{v}, v_{x+1}) \quad \text{if} \quad \overline{u} \cdot u_{x+1} \equiv_L \overline{v} \cdot v_{x+1}, \text{ and } L^s((\overline{u}, u_{x+1})) = L^s((\overline{v}, v_{x+1}))$$

This is residual equivalence w.r.t. the safe language together with the condition that the tuples map to the same class on the previous layer. The class of a tuple is written $[\overline{u}]_{\equiv_L}$.

▶ **Example 6.4.** Let $L = \mathsf{parity}(1, 2, \ldots, d)$. Then, the tuples that are $\equiv_L \perp$ are those that have length $> d$ or that contain x in some component of index $> x$. Therefore, for every length $x \in \{1, \ldots, d\}$, there are exactly two equivalence classes: $[(\varepsilon, 1, 2, \ldots, x-1)]_{\equiv_L} \equiv_L \perp$, and $[(\varepsilon, \ldots, \varepsilon)]_{\equiv_L} \not\equiv_L \perp$.

The following lemma shows that $\equiv_L$ is a congruence w.r.t. to the operations that we consider on tuples of words, namely right concatenation in the last component, merging the last two components, and extension with a new last component. Note that this implies that $\equiv_L$ is preserved under arbitrary sequences of such operations, for example first extending the tuple by a new last component, and then filling this new component with a word by repeated right concatenation.

▶ **Lemma 6.5.** *Let $\overline{u}, \overline{v} \in (\Sigma^*)^x$, $a \in \Sigma$, $u_{x+1}, v_{x+1} \in \Sigma^*$. The following properties hold.*

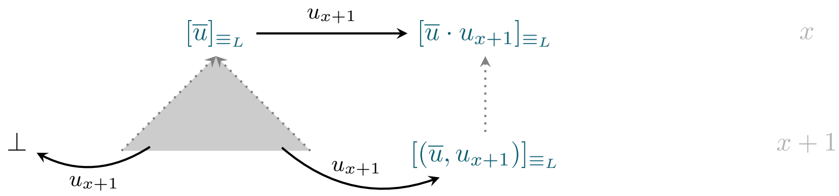

Figure 8 Illustration of the definition of $(\overline{u}, u_{x+1})$ being pointed. Basically, from every child of $[\overline{u}]_{\equiv_L}$, the word u_{x+1} leads to $\perp$ or to $(\overline{u}, u_{x+1})$.

1. $\overline{u} \equiv_L \overline{v} \Rightarrow \overline{u} \cdot a \equiv_L \overline{v} \cdot a$.
2. $(\overline{u}, u_{x+1}) \equiv_L (\overline{v}, v_{x+1}) \Rightarrow \overline{u} \cdot u_{x+1} \equiv_L \overline{v} \cdot v_{x+1}$.
3. $\overline{u} \equiv_L \overline{v} \Rightarrow (\overline{u}, \varepsilon) \equiv_L (\overline{v}, \varepsilon)$.

Proof. The proof is a simple application of the definitions.

1. Clearly, if $L^s(\overline{u}) = L^s(\overline{v})$, then also $L^s(\overline{u} \cdot a) = L^s(\overline{v} \cdot a)$. Based on this observation, the proof is a straightforward induction on x.
2. This property is part of the inductive definition of $\equiv_L$.
3. Assume that $\overline{u} \equiv_L \overline{v}$. Then clearly $\overline{u} \cdot \varepsilon \equiv_L \overline{v} \cdot \varepsilon$. For showing that $L^s(\overline{u}, \varepsilon) = L^s(\overline{v}, \varepsilon)$, let $u_{x+1} \in L^s(\overline{u}, \varepsilon)$. Then there is w such that $\overline{u} \cdot u_{x+1}w \equiv_L \overline{u}$ and $u_1 \cdots u_x(u_{x+1}w)^\omega \in L$ iff $x+1$ even. By assumption we have $\overline{u} \equiv_L \overline{v}$, which by 1 implies that $\overline{u} \cdot u_{x+1}w \equiv_L \overline{v} \cdot u_{x+1}w$, and thus $\overline{v} \cdot u_{x+1}w \equiv_L \overline{v}$. Further, $v_1 \cdots v_x(u_{x+1}w)^\omega \in L$ iff $u_1 \cdots u_x(u_{x+1}w)^\omega \in L$ because $v_1 \cdots v_x \sim_L u_1 \cdots u_x$. Therefore, $u_{x+1} \in L^s(\overline{v}, v_{x+1})$. By symmetry we obtain that $L^s(\overline{u}, \varepsilon) = L^s(\overline{v}, \varepsilon)$. This shows that $(\overline{u}, \varepsilon) \equiv_L (\overline{v}, \varepsilon)$.

◀

Pointed. We can use all classes of $\equiv_L$ (except $\perp$) for the construction of a layered automaton that accepts L. However, this automaton is not necessarily minimal (see Example 6.6 below). We now proceed with the definition of pointed tuples, which are used to select those classes of $\equiv_L$ that are actually used in the definition of the minimal automaton $\mathcal{A}_{\equiv_L}$. The definition of pointed is made such that it allows us to construct central sequences for the classes that contain pointed tuples. Consider a tuple $(\overline{u}, u_{x+1}) \not\equiv_L \perp$. It can be seen as the tuple that is reached from $(\overline{u}, \varepsilon)$ by reading u_{x+1}. The parent of $(\overline{u}, \varepsilon)$ on level x is $\overline{u}$. We now say that $(\overline{u}, u_{x+1})$ is pointed, if reading u_{x+1} from any $x+1$-tuple $(\overline{v}, v_{x+1})$ that has $\overline{u}$ as parent on layer x, results in a tuple $(\overline{v}, v_{x+1}u_{x+1})$ that is either $\perp$ or equivalent to $(\overline{u}, u_{x+1})$. In other words, u_{x+1} uniquely points to the class of $(\overline{u}, u_{x+1})$ when starting from a child of $\overline{u}$.

Formally, $(\overline{u}, u_{x+1}) \not\equiv_L \perp$ is *pointed* if $\overline{u}$ is pointed and for all pointed $\overline{v}$ and all v_{x+1} with $\overline{v} \cdot v_{x+1} \equiv_L \overline{u}$:

$$(\overline{v}, v_{x+1}u_{x+1}) \not\equiv_L \perp \Rightarrow (\overline{v}, v_{x+1}u_{x+1}) \equiv_L (\overline{u}, u_{x+1}).$$

This is illustrated in Figure 8, where the gray triangle represents all the children of $[\overline{u}]_{\equiv_L}$ such as $(\overline{v}, v_{x+1})$ in the definition.

▶ **Example 6.6.** Consider the alphabet $\Sigma := \{a, b, c\}$ and the language of all ω-words that contain at least one of the letters only finitely often. This language is prefix independent, so the first layer has only one class, and we only need to consider ε in the first component. Further, $(\varepsilon, u_2) \equiv_L \perp$ iff u_2 contains all letters, and $(\varepsilon, u_2) \equiv_L (\varepsilon, v_2)$ if the same letters

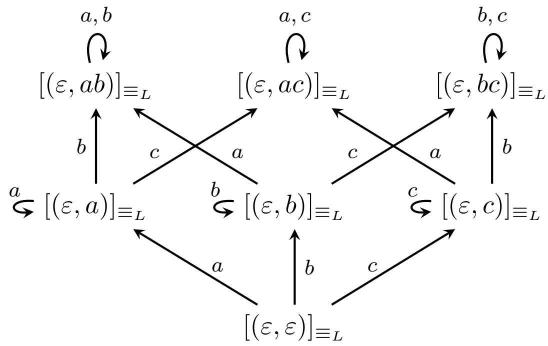

Figure 9 Diagram of all the $\equiv_L$-classes on layer 2 for the language L from Example 6.6 (at least one letter finitely often).

occur in u_2 and v_2. All tuples of length more than 2 are $\perp$. Otherwise, if $(\varepsilon, u_2, u_3) \not\equiv_L \perp$, then $(\varepsilon, u_2u_3) \not\equiv_L \perp$ and there is $w \in \Sigma^+$ with $(\varepsilon, u_2u_3w) \equiv_L (\varepsilon, u_2)$ and $u_2(u_3w)^\omega \notin L$. From $(\varepsilon, u_2u_3w) \equiv_L (\varepsilon, u_2)$ we know that u_2 and u_2u_3w contain the same letters. From $(\varepsilon, u_2u_3) \not\equiv_L \perp$ we conclude that u_2u_3 does not contain all the letters. Hence u_3w does not contain all the letters, so $u_2(u_3w)^\omega \in L$, a contradiction. The full diagram of $\equiv_L$ on layer 2 is shown in Figure 9 (except the class $\perp$, where the missing transitions are going). The parent of all these classes is the unique class on level 1. For the automaton, we only need the classes $[(\varepsilon, ab)]_{\equiv_L}$, $[(\varepsilon, ac)]_{\equiv_L}$, and $[(\varepsilon, bc)]_{\equiv_L}$, which respectively check that c, b, or a occur finitely often. And these are precisely the classes that contain pointed tuples. Consider, for example, (ε, ab). Reading ab starting from any class in the diagram either goes to $\perp$ or ends in $[(\varepsilon, ab)]_{\equiv_L}$, so (ε, ab) is pointed. But for example, the class of (ε, a), consisting of tuples (ε, a^i), $i \geq 1$, does not contain any pointed tuple. If (ε, a^i) were pointed, then with $(\overline{v}, v_{x+1}) = (\varepsilon, ab)$ in the definition of pointed, we would have $(\varepsilon, aba^i) \equiv_L \{\perp, (\varepsilon, a)\}$ but we have $(\varepsilon, aba^i) \equiv_L (\varepsilon, ab)$. In other words, reading a^i starting from (ε, ab) leads back to (ε, ab) and not to (ε, a) or $\perp$.

The following lemma asserts that being pointed is a property that is preserved under right concatenation in the last component and under merging the last two components.

▶ **Lemma 6.7.** *Let $\overline{u} \in (\Sigma^*)^x$, $a \in \Sigma$, $u_{x+1} \in \Sigma^*$. The following properties hold.*

1. *If $\overline{u}$ is pointed and $\overline{u} \cdot a \not\equiv_L \perp$, then $\overline{u} \cdot a$ is pointed.*
2. *If $(\overline{u}, u_{x+1})$ is pointed, then $\overline{u} \cdot u_{x+1}$ is pointed.*

Proof. Note that the property 2 follows from property 1: If $(\overline{u}, u_{x+1})$ is pointed, then by definition $\overline{u}$ is pointed. By repeated application of property 1 we get that $\overline{u} \cdot u_{x+1}$ is pointed.

Property 1 clearly holds for $x = 1$ because all tuples of length 1 are pointed. We now show that claim for tuples of length at least two, so assume that $(\overline{u}, u_{x+1})$ is pointed. For showing that $(\overline{u}, u_{x+1}a)$ is pointed, let $\overline{v} \in (\Sigma^*)^x$, $v_{x+1} \in \Sigma^*$ such that $\overline{v} \cdot v_{x+1} \equiv_L \overline{u}$. Then $(\overline{v}, v_{x+1}u_{x+1}) \equiv_L \{\perp, (\overline{u}, u_{x+1})\}$ because $(\overline{u}, u_{x+1})$ is pointed.

If $(\overline{v}, v_{x+1}u_{x+1}) \equiv_L \perp$, then $(\overline{v}, v_{x+1}u_{x+1}a) \equiv_L \perp$ by Lemma 6.3. If $(\overline{v}, v_{x+1}u_{x+1}) \equiv_L (\overline{u}, u_{x+1})$, then $(\overline{v}, v_{x+1}u_{x+1}a) \equiv_L (\overline{u}, u_{x+1}a)$ by Lemma 6.5(1). ◀

Regularity and Finiteness.

We show that for ω-regular languages, the set of pointed classes is finite. The proof consists of two parts, the first part (Lemma 6.8) shows that on each layer (for each fixed length of the tuples), there are finitely many pointed classes. The second part (Lemma 6.9) shows that there are only finitely many layers that contain classes that are different from $\perp$.

▶ **Lemma 6.8.** *If L is ω-regular, then $\equiv_L$ has finite index over $(\Sigma^*)^x$ for all $x \geq 1$.*

Proof. Let $L = L(\mathcal{A})$ for a nondeterministic Büchi automaton (NBA) $\mathcal{A}$. For a finite word $u \in \Sigma^*$, let the transition profile of u contain for each pair p, q of states of $\mathcal{A}$ the information, whether there is a run from p to q on u, and whether there is such a run passing through an accepting state. It is not difficult to show by induction on x that if $\overline{u}, \overline{v} \in (\Sigma^*)^x$ are such that u_i, v_i have the same transition profile for all i, then $\overline{u} \equiv_L \overline{v}$. Since there are only finitely many possible transition profiles, the claim follows. So let $\overline{u}, \overline{v} \in (\Sigma^*)^x$ be such that u_i, v_i have the same transition profile for each i. If u_1, v_1 have the same transition profile, then they also are in the same residual class, because the same states are reachable from the initial state of $\mathcal{A}$ via u_i and via v_i. So the claim follows for $x = 1$. For the step, consider $u_{x+1}, v_{x+1} \in \Sigma^*$ with the same transition profile, and show that $(\overline{u}, u_{x+1}) \equiv_L (\overline{v}, v_{x+1})$. Since u_x, v_x and u_{x+1}, v_{x+1} have, respectively, the same transition profile, also $u_x u_{x+1}, v_x v_{x+1}$ have the same transition profile, so $\overline{u} \cdot u_{x+1} \equiv_L \overline{v} \cdot v_{x+1}$ by induction. This implies that $\overline{u} \cdot u_{x+1} w' \equiv_L \overline{v} \cdot v_{x+1} w'$ for all $w' \in \Sigma^*$ by Lemma 6.5(1). Thus, $(\overline{u}, u_{x+1} w') \equiv_L \overline{u}$ iff $(\overline{v}, v_{x+1} w') \equiv_L \overline{v}$ because also $\overline{u} \equiv_L \overline{v}$ by induction. Further, since all pairs u_i, v_i have the same transition profiles, we get that $u_1 \cdots u_x (u_{x+1} w')^\omega \in L$ iff $v_1 \cdots v_x (v_{x+1} w')^\omega \in L$. This means that for every $w \in L^s(\overline{u}, u_{x+1})$, the same words witnessing that $(\overline{u}, u_{x+1} w) \not\equiv_L \perp$ also witness that $(\overline{v}, v_{x+1} w) \not\equiv_L \perp$, and vice versa. Hence $L^s(\overline{u}, u_{x+1}) = L^s(\overline{v}, v_{x+1})$. ◀

The following lemma shows that $\equiv_L$ has "finite depth" if L is ω-regular, in the sense that the length of tuples that are not $\perp$ is bounded by the parity index of the language.

▶ **Lemma 6.9.** *If L is ω-regular and accepted by a DPA with priorities in $[1, d]$, then $\overline{u} \equiv_L \perp$ for all $\overline{u} \in (\Sigma^*)^{d+1}$ (and hence also for all longer tuples).*

Proof. Let $\mathcal{A}$ be a DPA for L and let n be bigger than the number of states of $\mathcal{A}$. Assume by contradiction that there is $\overline{u} = (u_1, \ldots, u_{d+1}) \in (\Sigma^*)^{d+1}$ with $\overline{u} \not\equiv_L \perp$. We inductively define words $v_x \in \Sigma^*$ for $x \in \{2, \ldots, d+2\}$ and show that the loop that the run of $\mathcal{A}$ on $u_1 v_2^n$ enters, must visit d different priorities with the smallest one being even. This is not possible if $\mathcal{A}$ only uses priorities $1, \ldots, d$.

We start with the definition of $v_{d+2} := \varepsilon$, which is only needed for a simple induction base.

Assume that v_{x+1} has already been defined for $x \in \{2, \ldots, d+1\}$, and that $(u_1, \ldots, u_x) \cdot v_{x+1} \equiv_L (u_1, \ldots, u_x)$. Since $v_{d+2} := \varepsilon$, this condition is clearly satisfied for the start of the sequence with $x = d+1$. From $(u_1, \ldots, u_x) \cdot v_{x+1} \equiv_L (u_1, \ldots, u_x)$ we also get that $(u_1, \ldots, u_x) \cdot v_{x+1}^i \equiv_L (u_1, \ldots, u_x)$ for all i by Lemma 6.5(1). In particular $(u_1, \ldots, u_x) \cdot v_{x+1}^n \not\equiv_L \perp$. Hence, there is a word $w_x \in \Sigma^+$, such that $(u_1, \ldots, u_{x-1}) \cdot u_x v_{x+1}^n w_x \equiv_L (u_1, \ldots, u_{x-1})$ and $[u_1 \cdots u_{x-1} (u_x v_{x+1}^n w_x)^\omega \in L \Leftrightarrow x \text{ is even}]$. Let $v_x := u_x v_{x+1}^n w_x$. Note that

$$(u_1, \ldots, u_{x-1}) \cdot v_x = (u_1, \ldots, u_{x-1}) \cdot u_x v_{x+1}^n w_x \equiv_L (u_1, \ldots, u_{x-1}),$$

so v_x satisfies the property that was assumed for the inductive definition.

Our goal is to show that v_x induces $(d+1) - x$ many nested loops of different parity in $\mathcal{A}$ whose acceptance status alternates. For this we need to read v_x starting from the right type of state. Let q_0 be the initial state of $\mathcal{A}$, and q_i be the state reached in $\mathcal{A}$ after reading $u_1 \cdots u_i$:

$$q_0 \xrightarrow{u_1} q_1 \xrightarrow{u_2} q_2 \xrightarrow{u_3} \cdots \xrightarrow{u_{d+1}} q_{d+1}$$

We now show that for each $q \sim q_{x-1}$ such that a non-empty power of v_x loops on q, this loop contains $d+2-x$ many different priorities with the least one having the parity of x. So assume that $q \xrightarrow{v_x^+} q$ where the plus represents some non-empty power. By the choice of v_x, we have that $u_1 \cdots u_{x-1} v_x^\omega \in L$ iff x is even. Hence, v_x^ω is accepted from q iff x is even, which means that the least priority on the loop has the same parity as x.

We show that the loop contains at least $d+2-x$ many priorities by a downward induction starting with $x = d+1$. The induction base is trivial because the loop must contain at least 1 priority.

If $x < d+1$, then consider the states reached after the factors of the first $v_x = u_x v_{x+1}^n w_x$:

$$q \xrightarrow{u_x} p_0 \xrightarrow{v_{x+1}} p_1 \xrightarrow{v_{x+1}} p_2 \cdots \xrightarrow{v_{x+1}} p_n \xrightarrow{w_x} q'$$

Since $q \sim q_{x-1}$ and p_0 is reached from q via u_x, we get that $p_0 \sim q_x$. Furthermore, $(u_1, \cdots, u_x) \cdot v_{x+1}^i \equiv_L (u_1, \cdots, u_x)$ as already noted above. So we get that $p_i \sim q_x$ for all $i \in \{0, \ldots, n\}$. By the choice of n, there must be $i \neq j$ with $p_i = p_j =: p$, so we have found a state $p \sim q_x$ such that a non-empty power of v_{x+1} loops on p. By induction, we know that on this loop at least $d+2-(x+1)$ many priorities are visited with the least one having the parity of $x+1$. So the least priority visited on the loop $q \xrightarrow{v_x^+} q$ must be strictly smaller than the minimal one on the loop $p \xrightarrow{v_{x+1}^+} q$, which gives the desired claim.

From this we immediately get that the loop that is entered on $u_1 v_2^n$ contains at least $d = d+2-2$ many priorities with the least one being even. ◀

6.2 The automaton $\mathcal{A}_{\equiv_L}$

For the definition of $\mathcal{A}_{\equiv_L}$ we assume that L is ω-regular. So in all statements that involve $\mathcal{A}_{\equiv_L}$, we assume that L is an ω-regular language (in the main results stated as theorems we mention this again explicitly, but not in all auxiliary lemmas). Let d be maximal such that there exists $\overline{u} \in (\Sigma^*)^d$ with $\overline{u} \not\equiv_L \bot$. By Lemma 6.9 this is the minimal d such that L is accepted by a $[1,d]$-DPA.

The x-th layer of $\mathcal{A}_{\equiv_L}$ consist of the equivalence classes of the pointed tuples of dimension x, the transitions corresponding to concatenation in the last component. The morphism from layer $x+1$ to layer x is obtained by merging the last two components. And the initial state is as usual the class of the empty word. Formally, we define the layered automaton $\mathcal{A}_{\equiv_L} = (q_0^{\equiv_L}, \mathcal{T}_1^{\equiv_L}, \mu_1^{\equiv_L}, \ldots, \mathcal{T}_{d-1}^{\equiv_L}, \mu_{d-1}^{\equiv_L}, \mathcal{T}_d^{\equiv_L})$ by

- $\mathcal{T}_x^{\equiv_L} = (Q_x^{\equiv_L}, \delta_x^{\equiv_L})$ with
  - $Q_x^{\equiv_L} = \{[\overline{u}]_{\equiv_L} \mid \overline{u} \in (\Sigma^*)^x \text{ pointed}\}$
  - $\delta_x^{\equiv_L}([\overline{u}]_{\equiv_L}, a) = \begin{cases} [\overline{u} \cdot a]_{\equiv_L} & \text{if } \overline{u} \cdot a \not\equiv_L \bot \\ \text{undefined} & \text{else.} \end{cases}$
- $q_0^{\equiv_L} = [\varepsilon]_{\equiv_L}$
- $\mu_x^{\equiv_L} : Q_{x+1}^{\equiv_L} \to Q_x^{\equiv_L}$ with $\mu_x^{\equiv_L}([(\overline{u}, u_{x+1})]_{\equiv_L}) = [\overline{u} \cdot u_{x+1}]_{\equiv_L}$.

Note that in the definition of $\delta_x^{\equiv_L}$ and $\mu_x^{\equiv_L}$ the result is independent of the chosen representative by Lemma 6.5(1) and (2). Further, $\mu_x^{\equiv_L}$ is a morphism because it does not matter whether we first concatenate and then merge, or first merge and then concatenate:

$$\begin{aligned}
\mu_x^{\equiv_L}(\delta_{x+1,\equiv_L}([(\overline{u}, u_{x+1}]_{\equiv_L}), a)) &= \mu_x^{\equiv_L}([(\overline{u}, u_{x+1}a)]_{\equiv_L}) \\
&= [\overline{u} \cdot u_{x+1}a]_{\equiv_L} \\
&= \delta_x^{\equiv_L}([\overline{u} \cdot u_{x+1}]_{\equiv_L}, a) \\
&= \delta_x^{\equiv_L}(\mu_x^{\equiv_L}([(\overline{u}, u_{x+1})]_{\equiv_L}, a))
\end{aligned}$$

So $\mathcal{A}_{\equiv_L}$ is a layered automaton.

6.3 Correctness of $\mathcal{A}_{\equiv_L}$

We show that $\mathcal{A}_{\equiv_L}$ is a consistent layered automaton that accepts L (Theorem 6.14). We first need some auxiliary definitions and lemmas.

Our first goal is to show that the safe languages of tuples are covered by the pointed tuples. This is expressed for finite and infinite words respectively in Lemmas 6.11 and 6.12. For constructing corresponding tuples, we start with the auxiliary Lemma 6.10 that is applied in the proof of Lemma 6.11 (so the reader may skip Lemma 6.10 now and come back to it when reading the proof of Lemma 6.11).

▶ **Lemma 6.10.** *Let $\overline{u}, \overline{v} \in (\Sigma^*)^x$ and $u_{x+1}, v_{x+1} \in \Sigma^*$. If $\overline{u} \equiv_L \overline{v} \cdot v_{x+1}$, then $L^s(\overline{u}, u_{x+1}) \supseteq L^s(\overline{v}, v_{x+1}u_{x+1})$.*

Proof. Let $w \in L^s(\overline{v}, v_{x+1}u_{x+1})$, that is, $(\overline{v}, v_{x+1}u_{x+1}w) \not\equiv_L \bot$. We show that $w \in L^s(\overline{u}, u_{x+1})$. It is sufficient to show this for finite words w because the infinite words in the safe languages are determined by the finite ones.

If $(\overline{v}, v_{x+1}u_{x+1}w) \not\equiv_L \bot$, then there is $w' \in \Sigma^+$ with $\overline{v} \cdot v_{x+1}u_{x+1}ww' \equiv_L \overline{v}$ and $v_1 \cdots v_x(v_{x+1}u_{x+1}ww')^\omega \in L$ iff $x+1$ is even. Observe that $\overline{u} \equiv_L \overline{v} \cdot v_{x+1}$ in particular implies $u_1 \cdots u_x \equiv_L v_1 \cdots v_x v_{x+1}$. Since $v_1 \cdots v_x(v_{x+1}u_{x+1}ww')^\omega = v_1 \cdots v_x v_{x+1}(u_{x+1}ww'v_{x+1})^\omega$, we get $u_1 \cdots u_x(u_{x+1}ww'v_{x+1})^\omega \in L$ iff $x+1$ is even. Further

$$\overline{u} \cdot u_{x+1}ww'v_{x+1} \equiv_L \underbrace{\overline{v} \cdot v_{x+1}u_{x+1}ww'}_{\equiv_L \overline{v}} v_{x+1} \equiv_L \overline{v} \cdot v_{x+1} \equiv_L \overline{u}$$

So the suffix $w'v_{x+1}$ witnesses that $(\overline{u}, u_{x+1}w) \not\equiv_L \bot$ and hence $w \in L^s(\overline{u}, u_{x+1})$. ◀

The following lemma states that if we start from a pointed tuple $\overline{u}$ of length x, then the safe language of the $(x+1)$-length tuple $(\overline{u}, \varepsilon)$ with empty $(x+1)$-st component is covered by pointed tuples $(\overline{v}, v_{x+1})$ of length $x+1$ with $\mu_x^{\equiv_L}([(\overline{v}, v_{x+1})]_{\equiv_L}) = [\overline{u}]_{\equiv_L}$. Recall that $\mu_x^{\equiv_L}([(\overline{v}, v_{x+1})]_{\equiv_L}) = [\overline{u}]_{\equiv_L}$ is equivalent to $\overline{v} \cdot v_{x+1} \equiv_L \overline{u}$.

▶ **Lemma 6.11.** *Consider $(\overline{u}, u_{x+1}) \not\equiv_L \bot$ such that $\overline{u}$ is pointed. Then there is pointed $(\overline{v}, v_{x+1})$ such that $\overline{v} \cdot v_{x+1} = \overline{u}$ and $(\overline{v}, v_{x+1}u_{x+1}) \not\equiv_L \bot$.*

Proof. Assume that we can show that there is are $\overline{v} \in (\Sigma^*)^x$, $v_{x+1} \in \Sigma^*$ with $\overline{v} \cdot v_{x+1} \equiv_L \overline{u}$ and $(\overline{v}, v_{x+1}u_{x+1})$ is pointed (note that the lemma claims that the tuple without u_{x+1} at the end has to be pointed). From that we can construct the desired tuple as follows. Since $(\overline{v}, v_{x+1}u_{x+1}) \not\equiv_L \bot$ (because it is pointed), there is $w \in \Sigma^+$ such that $\overline{v} \cdot v_{x+1}u_{x+1}w \equiv_L \overline{v}$ and $v_1 \cdots v_x(v_{x+1}u_{x+1}w)^\omega \in L$ iff $x+1$ is even. Then also $(\overline{v}, v_{x+1}u_{x+1}wv_{x+1}u_{x+1}) \not\equiv_L \bot$. Further $\underbrace{\overline{v} \cdot v_{x+1}u_{x+1}w}_{\equiv_L \overline{v}} v_{x+1} \equiv_L \overline{v} \cdot v_{x+1} \equiv_L \overline{u}$ by Lemma 6.5 and $(\overline{v}, v_{x+1}u_{x+1}wv_{x+1})$ is pointed as an extension of a pointed element by Lemma 6.7. So with $v'_{x+1} = v_{x+1}u_{x+1}wv_{x+1}$ we obtain a tuple $(\overline{v}, v'_{x+1})$ as claimed in the lemma.

It remains to show that we can find $\overline{v} \in (\Sigma^*)^x$, $v_{x+1} \in \Sigma^*$ with $\overline{v} \cdot v_{x+1} \equiv_L \overline{u}$ and $(\overline{v}, v_{x+1}u_{x+1})$ pointed. The idea is quite simple. We start with $(\overline{u}, u_{x+1})$ as current tuple. As long as the current tuple is not pointed, there is a witness for that: a new tuple whose last component ends in u_{x+1}, and whose safe language is different from the safe language of the current tuple. By Lemma 6.10, the safe language of the new tuple must be strictly included in the safe language of the previous tuple. This can happen only finitely often, since the safe language of a tuple only depends on its $\equiv_L$-class, and there are only finitely many $\equiv_L$-classes. We formalise this idea below.

Starting with $i = 0$ we define a sequence $(\overline{v}^{(i)}, v^{(i)}_{x+1})$ of tuples with $\overline{v}^{(i)} \in (\Sigma^*)^x$ and $v^{(i)}_{x+1} \in \Sigma^*$ such that

(1) $\overline{v}^{(i)}$ is pointed,
(2) $\overline{v}^{(i)} \cdot v^{(i)}_{x+1} \equiv_L \overline{u}$,
(3) $(\overline{v}^{(i)}, v^{(i)}_{x+1}u_{x+1}) \not\equiv_L \perp$, and
(4) if $i > 0$, then $L^s(\overline{v}^{(i)}, v^{(i)}_{x+1}u_{x+1}) \subsetneq L^s(\overline{v}^{(i-1)}, v^{(i-1)}_{x+1}u_{x+1})$.

We show that we can extend the sequence by one more tuple if $(\overline{v}^{(i)}, v^{(i)}_{x+1}u_{x+1})$ is not pointed. Because of the last property, each such sequence must be finite because the safe language of a tuple only depends on its $\equiv_L$-class, and by Lemma 6.8 there are only finitely many $\equiv_L$ classes for each tuple length (we are assuming that L is regular).

We start with $\overline{v}^{(0)} = \overline{u}$ and $v^{(0)}_{x+1} = \varepsilon$, which is easily seen to have all the properties. If $(\overline{v}^{(i)}, v^{(i)}_{x+1}u_{x+1})$ is pointed, then we have found the desired tuple with $\overline{v} := \overline{v}^{(i)}$ and $v_{x+1} := v^{(i)}_{x+1}$. Otherwise, we show how to obtain the next tuple in the sequence.

So assume that $(\overline{v}^{(i)}, v^{(i)}_{x+1}u_{x+1})$ is not pointed. Since $\overline{v}^{(i)}$ is pointed, there must be $\overline{v}' \in (\Sigma^*)^x$ and $v'_{x+1} \in \Sigma^*$ such that

(i) $\overline{v}'$ is pointed,
(ii) $\overline{v}' \cdot v'_{x+1} \equiv_L \overline{v}^{(i)}$,
(iii) $(\overline{v}', v'_{x+1}v^{(i)}_{x+1}u_{x+1}) \not\equiv_L \perp$, and
(iv) $(\overline{v}', v'_{x+1}v^{(i)}_{x+1}u_{x+1}) \not\equiv_L (\overline{v}^{(i)}, v^{(i)}_{x+1}u_{x+1})$.

If we define $\overline{v}^{(i+1)} = \overline{v}'$ and $v^{(i+1)}_{x+1} := v'_{x+1}v^{(i)}_{x+1}$, then properties (1) and (3) for the new tuple are immediate from (i) and (iii). Property (2) follows from (ii) by $\overline{v}^{(i+1)} \cdot v^{(i+1)}_{x+1} = \overline{v}' \cdot v'_{x+1}v^{(i)}_{x+1} \equiv_L \overline{v}^{(i)}v^{(i)}_{x+1} \equiv_L \overline{u}$, where the second last equivalence is from (ii), and the last equivalence from (2) for i. Property (4) follows from (iv) and Lemma 6.10 because $(\overline{v}', v'_{x+1}v^{(i)}_{x+1}u_{x+1}) \not\equiv_L (\overline{v}^{(i)}, v^{(i)}_{x+1}u_{x+1})$ is the same as $L^s(\overline{v}', v'_{x+1}v^{(i)}_{x+1}u_{x+1}) \neq L^s(\overline{v}^{(i)}, v^{(i)}_{x+1}u_{x+1})$, and thus by Lemma 6.10 $L^s(\overline{v}', v'_{x+1}v^{(i)}_{x+1}u_{x+1}) \subsetneq L^s(\overline{v}^{(i)}, v^{(i)}_{x+1}u_{x+1})$. That we can apply Lemma 6.10 is guaranteed by (ii). ◀

The following is an analogue of Lemma 6.11 for infinite words.

▶ **Lemma 6.12.** *Let $\overline{u} \in (\Sigma^*)^x$ be pointed and $w \in \Sigma^\omega$ such that $(\overline{u}, w) \not\equiv_L \perp$. Then there are $\overline{v} \in (\Sigma^*)^x$, $v_{x+1} \in \Sigma^*$ with $\overline{v} \cdot v_{x+1} \equiv_L \overline{u}$, $(\overline{v}, v_{x+1})$ is pointed and $(\overline{v}, v_{x+1}w) \not\equiv_L \perp$.*

Proof. We can apply Lemma 6.11 to each finite prefix $u^{(i)} = u_{x+1}$ of w, obtaining $\overline{v}^{(i)} \in (\Sigma^*)^x$, $v^{(i)}_{x+1} \in \Sigma^*$ with $\overline{v}^{(i)} \cdot v^{(i)}_{x+1} \equiv_L \overline{u}$, $(\overline{v}^{(i)}, v^{(i)}_{x+1})$ is pointed and $(\overline{v}^{(i)}, v^{(i)}_{x+1}u^{(i)}) \not\equiv_L \perp$. We can assume that each $(\overline{v}^{(i)}, v^{(i)}_{x+1})$ is the shortest pointed tuple from its $\equiv_L$-class, because the

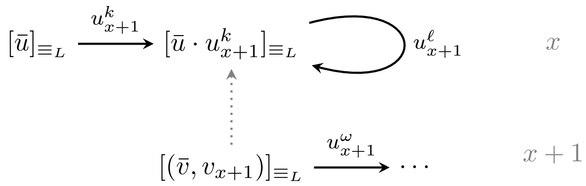

Figure 10 Illustration of the contradiction in the proof of Lemma 6.13 under the assumption that $(\overline{u} \cdot v^k, v^\omega) \not\equiv_L \bot$.

required properties hold for all pointed tuples of a $\equiv_L$-class or for none. Hence, there are only finitely many candidates for the $(\overline{v}^{(i)}, v^{(i)}_{x+1})$, and thus there must be one tuple $(\overline{v}, v_{x+1})$ that equals $(\overline{v}^{(i)}, v^{(i)}_{x+1})$ for infinitely many prefixes $u^{(i)}$. This tuple then satisfies all the required properties. ◀

The next lemma is used in the proof of Theorem 6.14 for showing that $\mathcal{A}_{\equiv_L}$ is consistent and accepts the language L.

▶ **Lemma 6.13.** *Let x be even and $\overline{u} \in (\Sigma^*)^x$ be pointed such that u^ω_{x+1} is strongly accepted from $[\overline{u}]_{\equiv_L}$ in $\mathcal{A}_{\equiv_L}$. Then $u_1 \cdots u_x u^\omega_{x+1} \in L$. The corresponding statement holds for x odd, u^ω_{x+1} strongly rejected and $u_1 \cdots u_x u^\omega_{x+1} \notin L$.*

Proof. We show the case x even. Since we never refer to layer $x-1$, the proof for x odd is symmetric.

Since u^ω_{x+1} is strongly accepted from $[\overline{u}]_{\equiv_L}$, we have $\overline{u} \cdot u^\omega_{x+1} \not\equiv_L \bot$. Let $k \geq 0, \ell \geq 1$ be such that $\overline{u} \cdot u^k_{x+1} \equiv_L \overline{u} \cdot u^{k+\ell}_{x+1}$. Assume that $(\overline{u} \cdot u^k_{x+1}, u^\omega_{x+1}) \not\equiv_L \bot$. Then, by Lemma 6.12, there would be $\overline{v} \in (\Sigma^*)^x$, $v_{x+1} \in \Sigma^*$ with $\overline{v} \cdot v_{x+1} \equiv_L \overline{u} \cdot u^k_{x+1}$, $(\overline{v}, v_{x+1})$ pointed and $(\overline{v}, v_{x+1} u^\omega_{x+1}) \not\equiv_L \bot$. This would witness that u^ω_{x+1} is not strongly accepted from $[\overline{u}]_{\equiv_L}$, as illustrated in Figure 10. Hence, we must have $(\overline{u} \cdot u^k_{x+1}, u^\omega_{x+1}) \equiv_L \bot$, which means that $(\overline{u} \cdot u^k_{x+1}, u^m_{x+1}) \equiv_L \bot$ for some m. By iterated application of $\overline{u} \cdot u^k_{x+1} \equiv_L \overline{u} \cdot u^{k+\ell}_{x+1}$, we obtain $\overline{u} \cdot u^{k+\ell \cdot m}_{x+1} \equiv_L \overline{u} \cdot u^k_{x+1}$. By definition of $(\overline{u} \cdot u^k_{x+1}, u^m_{x+1}) \equiv_L \bot$, this implies that $u_1 \cdots u_x u^\omega_{x+1} = u_1 \cdots u_x u^k_{x+1} (u^{\ell \cdot m}_{x+1})^\omega \in L$ because x is even. ◀

▶ **Theorem 6.14.** *Let L be an ω-regular language. Then $\mathcal{A}_{\equiv_L}$ is a consistent layered automaton with $L(\mathcal{A}_{\equiv_L}) = L$.*

Proof. Assume that $\mathcal{A}_{\equiv_L}$ is not uniformly semantically deterministic and thus not consistent by Corollary 3.18. Then there is an even x and an odd y and pointed tuples $\overline{u} \in (\Sigma^*)^x$, $\overline{v} \in (\Sigma^*)^y$ with $u_1 \cdots u_x \sim_L v_1 \cdots v_y$ and such that some $w \in \Sigma^\omega$ is strongly accepted from $[\overline{u}]_{\equiv_L}$ and strongly rejected from $[\overline{v}]_{\equiv_L}$. Clearly, the set of all these w is ω-regular, so there is also an ultimately periodic $w = uv^\omega$ with this property. Then v^ω is strongly accepted from $[\overline{u} \cdot u]_{\equiv_L}$ and strongly rejected from $[\overline{v} \cdot u]_{\equiv_L}$. By Lemma 6.13 we get that $u_1 \cdots u_x uv^\omega \in L$ and $v_1 \cdots v_y uv^\omega \notin L$, which contradicts $u_1 \cdots u_x \sim_L v_1 \cdots v_y$. We conclude that $\mathcal{A}_{\equiv_L}$ is consistent.

For showing that $L(\mathcal{A}_{\equiv_L}) = L$, let uv^ω be an ultimately periodic word. By Proposition 3.13, uv^ω contains a suffix that is strongly accepted or strongly rejected by some state in $\mathcal{A}_{\equiv_L}$, and not both because we have already shown that $\mathcal{A}_{\equiv_L}$ is consistent. Then v^ω is strongly accepted or strongly rejected from some $[\overline{u}]_{\equiv_L}$ for a pointed tuple $\overline{u} = (u_1, \ldots, u_x) \in \Sigma^x$ such that $(u_1 \cdots u_x)$ is reachable on layer 1 from (ε) by a prefix uv^k of uv^ω. Since all transitions in $\mathcal{A}_{\equiv_L}$ respect the $\sim_L$ class, we get that $u_1 \cdots u_x \sim_L uv^k$. By Lemma 6.13 we get that $uv^k v^\omega \in L$ iff x is even. Hence, $L(\mathcal{A}_{\equiv_L})$ and L contain the same ultimately periodic words and are thus equal (this follows from the closure properties of ω-regular languages, see e.g. [CNP93, Fact 1]). ◀

6.4 Minimality of $\mathcal{A}_{\equiv_L}$

We show that $\mathcal{A}_{\equiv_L}$ is safe minimal, normal, and centralised (Lemma 6.18) and then use Theorem 4.1 to conclude that $\mathcal{A}_{\equiv_L}$ is the minimal layered automaton for L.

We start by showing that $\mathcal{A}_{\equiv_L}$ has central sequences. The proof is split into three lemmas: Lemma 6.15 shows how to obtain a central sequence for each pointed class that has a child that is $\perp$. Lemma 6.16 shows that all children of leafs of $\mathcal{A}_{\equiv_L}$ are $\perp$, and in the proof of Lemma 6.17 we show that Lemma 6.15 can also be applied to the inner states of $\mathcal{A}_{\equiv_L}$.

▶ **Lemma 6.15.** *Let $x \geq 2$, $\overline{u} \in (\Sigma^*)^{x-1}$ and $u_x, u_{x+1} \in \Sigma^*$ such that $(\overline{u}, u_x)$ is pointed, $(\overline{u}, u_x, u_{x+1}) \equiv_L \perp$ and $(\overline{u}, u_x u_{x+1}) \not\equiv_L \perp$. Then $[(\overline{u}, u_x u_{x+1})]_{\equiv_L}$ has a central sequence.*

Proof. Since $(\overline{u}, u_x u_{x+1}) \not\equiv_L \perp$, there is w with $\overline{u} \cdot u_x u_{x+1} w \equiv_L \overline{u}$ and $u_1 \cdots u_{x-1}(u_x u_{x+1} w)^\omega \in L$ iff x even. We show that $z := w u_x u_{x+1}$ is a central sequence for $[(\overline{u}, u_x u_{x+1})]_{\equiv_L}$.

First, every child of $[\overline{u} \cdot u_x u_{x+1}]_{\equiv_L}$ is taken to $[(\overline{u}, u_x u_{x+1})]_{\equiv_L}$ or $\perp$ by z, which follows from $(\overline{u}, u_x u_{x+1})$ being pointed: Let $(\overline{v}, v_x)$ be pointed such that $\overline{v} \cdot v_x \equiv_L \overline{u} \cdot u_x u_{x+1}$. Then by Lemma 6.5, $\overline{v} \cdot v_x w \equiv_L \overline{u} \cdot u_x u_{x+1} w \equiv_L \overline{u}$ and thus

$$(\overline{v}, v_x w u_x u_{x+1}) \equiv_L \{(\overline{u}, u_x u_{x+1}), \perp\}.$$

Second, by choice of w, $(\overline{u}, u_x u_{x+1} w u_x u_{x+1}) \not\equiv_L \perp$, and hence z loops on $[(\overline{u}, u_x u_{x+1})]_{\equiv_L}$.

Third, every grand child of $[\overline{u} \cdot u_x u_{x+1}]_{\equiv_L}$ is taken to $\perp$ by z: Let $(\overline{v}, v_x, v_{x+1})$ be pointed such that $\overline{v} \cdot v_x v_{x+1} \equiv_L \overline{u} \cdot u_x u_{x+1}$. Then again by Lemma 6.5, $\overline{v} \cdot v_x v_{x+1} w \equiv_L \overline{u} \cdot u_x u_{x+1} w \equiv_L \overline{u}$. Since $(\overline{u}, u_x)$ is pointed, we get $(\overline{v}, v_x v_{x+1} w u_x) \equiv_L \{(\overline{u}, u_x), \perp\}$. If $(\overline{v}, v_x v_{x+1} w u_x) \equiv_L \perp$, then clearly $(\overline{v}, v_x v_{x+1} z) = (\overline{v}, v_x v_{x+1} u_x u_{x+1} w) \equiv_L \perp$ by Lemma 6.3. If $(\overline{v}, v_x v_{x+1} w u_x) \equiv_L (\overline{u}, u_x)$, then by Lemma 6.10

$$L^s((\overline{v}, v_x, v_{x+1} z)) = L^s((\overline{v}, v_x, v_{x+1} w u_x u_{x+1})) \subseteq L^s((\overline{u}, u_x, u_{x+1})) = \emptyset.$$

This means that $(\overline{v}, v_x, v_{x+1} z) \equiv_L \perp$, and thus z satisfies all properties of a central sequence for $[(\overline{u}, u_x u_{x+1})]_{\equiv_L}$. ◀

We now give that characterisation of leaf states by showing that all tuples that extend a pointed tuple in a leaf class by another component are $\perp$. The proof is a simple instantiation of Lemma 6.11.

▶ **Lemma 6.16.** *Let $x \geq 1$ and $\overline{u} \in (\Sigma^*)^x$ be pointed. Then $[\overline{u}]_{\equiv_L}$ is a leaf of $\mathcal{A}_{\equiv_L}$ iff $(\overline{u}, \varepsilon) \equiv_L \perp$.*

Proof. Let $\overline{u} \in (\Sigma^*)^x$ be pointed. If $(\overline{u}, \varepsilon) \not\equiv_L \perp$, then we apply Lemma 6.11 with $u_{x+1} = \varepsilon$. We obtain $\overline{v} \in (\Sigma^*)^x$, $v_{x+1} \in \Sigma^*$ such that $(\overline{v}, v_{x+1})$ is pointed and $\overline{v} \cdot v_{x+1} \equiv_L \overline{u}$. So $[(\overline{v}, v_{x+1})]_{\equiv_L}$ is a state in $\mathcal{T}_{x+1}^{\equiv_L}$ with $\mu_x^{\equiv_L}([(\overline{v}, v_{x+1})]_{\equiv_L}) = [\overline{v} \cdot v_{x+1}]_{\equiv_L} = [\overline{u}]_{\equiv_L}$. Hence $[\overline{u}]_{\equiv_L}$ is not a leaf.

For the other direction, assume that $[\overline{u}]_{\equiv_L}$ is not a leaf and let $\overline{v} \in (\Sigma^*)^x$, $v_{x+1} \in \Sigma^*$ such that $(\overline{v}, v_{x+1})$ is pointed and $\overline{v} \cdot v_{x+1} \equiv_L \overline{u}$, that is, $\mu_x^{\equiv_L}([(\overline{v}, v_{x+1})]_{\equiv_L}) = [\overline{v} \cdot v_{x+1}]_{\equiv_L} = [\overline{u}]_{\equiv_L}$.

Let $w \in \Sigma^+$ be such that $\overline{v} \cdot v_{x+1}w \equiv_L \overline{v}$, and $v_1 \cdots v_x(v_{x+1}w)^\omega \in L$ iff $x+1$ is even. Since $\overline{v} \cdot v_{x+1} \equiv_L \overline{u}$, we get $u_1 \cdots u_x(wv_{x+1})^\omega \in L$ iff $x+1$ is even. In combination with

$$\overline{u} \cdot wv_{x+1} \equiv_L \overbrace{\underbrace{\overline{v} \cdot v_{x+1}}_{\equiv_L \overline{u}} w}^{\equiv_L \overline{v}} v_{x+1} \equiv_L \overline{v} \cdot v_{x+1} \equiv_L \overline{u}$$

we see that wv_{x+1} is a witness for $(\overline{u}, \varepsilon) \not\equiv_L \perp$. ◀

▶ **Lemma 6.17.** *Every state in $\mathcal{A}_{\equiv_L}$ has a central sequence.*

Proof. If $[(\overline{u}, u_x)]_{\equiv_L}$ is a leaf of $\mathcal{A}_{\equiv_L}$, then $(\overline{u}, u_x, \varepsilon) \equiv_L \perp$ by Lemma 6.16, and thus $[(\overline{u}, u_x)]_{\equiv_L}$ has a central sequence by Lemma 6.15.

For showing that internal states have a central sequence, let $(\overline{v}, v_x, v_{x+1})$ be pointed and show that its parent $[(\overline{v}, v_x v_{x+1})]_{\equiv_L}$ has a central sequence, assuming that $[(\overline{v}, v_x, v_{x+1})]_{\equiv_L}$ has a central sequence z. We want to show that we can apply Lemma 6.15, so we construct a word z' that loops on $[(\overline{v}, v_x v_{x+1})]_{\equiv_L}$ and takes $(\overline{v}, v_x, v_{x+1})$ to $\perp$, so it has the following properties:

(i) $(\overline{v}, v_x v_{x+1} z') \equiv_L (\overline{v}, v_x v_{x+1})$ and
(ii) $(\overline{v}, v_x, v_{x+1} z') \equiv_L \perp$.

We can apply Lemma 6.15 to $(\overline{v}, v_x, v_{x+1} z')$ and obtain a central sequence for $[(\overline{v}, v_x v_{x+1} z')]_{\equiv_L} = [(\overline{v}, v_x v_{x+1})]_{\equiv_L}$ as desired.

The following arguments for establishing (i) and (ii) are not very difficult but might be confusing because they involve several tuples. Figure 11 gives an overview of the situation that is constructed below. The tuple on the left-hand side on layer x is the one for which we want to construct the word z'.

We start by explaining the z-edges in Figure 11. Since z is central for $[(\overline{v}, v_x, v_{x+1})]_{\equiv_L}$, every child of $[(\overline{v}, v_x, v_{x+1})]_{\equiv_L}$ is taken to $\perp$ by z (this is indicated by $[(\cdots)]_{\equiv_L}$ on layer $x+2$ in Figure 11). Further, we have $(\overline{v}, v_x, v_{x+1} z) \equiv_L (\overline{v}, v_x, v_{x+1})$ and thus also $(\overline{v}, v_x v_{x+1} z) \equiv_L (\overline{v}, v_x v_{x+1})$ and $\overline{v} \cdot v_x v_{x+1} z \equiv_L \overline{v} \cdot v_x v_{x+1}$ by Lemma 6.5 (these are the z-loops in Figure 11). Hence, $(\overline{v}, v_x v_{x+1} z) \not\equiv_L \perp$, and there is w such that $(\overline{v}, v_x v_{x+1} z w) \equiv_L \overline{v}$ and $v_1 \cdots v_{x-1}(v_x v_{x+1} z w)^\omega \in L$ iff x is even. This is the w-edge on layer $x-1$ in Figure 11. Further, the choice of w ensures that $(\overline{v}, v_x v_{x+1}(z w v_x v_{x+1})^\omega) \not\equiv_L \perp$.

We let $z' := z w v_x v_{x+1}$. The paths induced by z' on the layers $x-1$, x, and $x+1$ are shown in Figure 11. Most edges shown on the layers are just obtained by appending the corresponding word to the last component of the corresponding tuple. The parent edges are obtained by the morphism property. There are two thick edges on the layers for which the target tuple is determined by the definition of pointed. First, consider the thick v_x-edge on layer x. Since $(\overline{v}, v_x)$ is pointed and $\overline{v} \cdot v_x v_{x+1} z w \equiv_L \overline{v}$ (as can be seen in Figure 11), we get that $(\overline{v}, v_x v_{x+1} z w v_x) \equiv_L \{(\overline{v}, v_x), \perp\}$. And as noted earlier, the choice of w ensures that $(\overline{v}, v_x v_{x+1}(z w v_x v_{x+1})^\omega) \not\equiv_L \perp$, so in particular $(\overline{v}, v_x v_{x+1} z w v_x) \not\equiv_L \perp$. Hence, we obtain that $(\overline{v}, v_x v_{x+1} z w v_x) \equiv_L (\overline{v}, v_x)$ and thus also

$$(\overline{v}, v_x v_{x+1} z') = (\overline{v}, v_x v_{x+1} z w v_x v_{x+1}) \equiv_L (\overline{v}, v_x v_{x+1}).$$

This explains the edges on layer x in Figure 11 and shows (i).

We now show (ii). Actually, the thick v_{x+1}-edge on layer $x+1$ is drawn under the assumption that (ii) does not hold, and is then used for a contradiction. Since $(\overline{v}, v_x, v_{x+1})$ is pointed and $(\overline{v}, v_x v_{x+1} z w v_x) \equiv_L (\overline{v}, v_x)$ (as established before), we get that

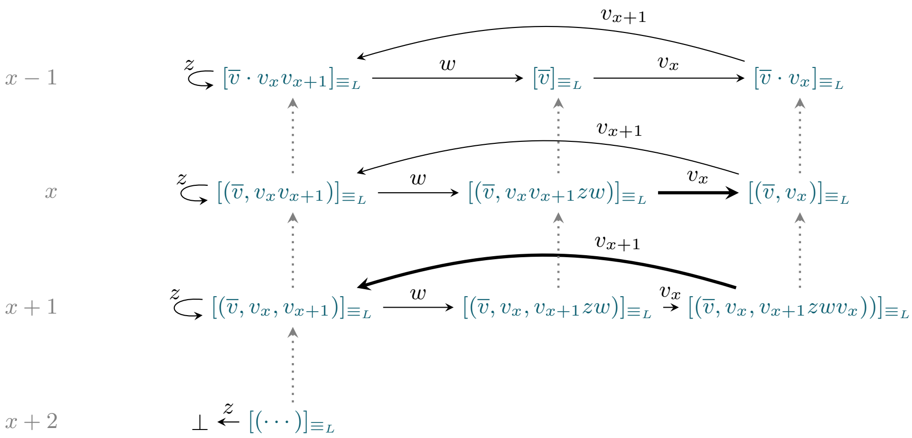

Figure 11 Illustration supporting the reasoning in the proof of Lemma 6.17. Dotted edges represent the morphism to the parent layer. The v_{x+1}-edge on layer $x+1$ is drawn under an assumption that leads to a condtradiction, so the wv_xv_{x+1}-loop on layer $x+1$ does not exist.

$(\overline{v}, v_x, v_{x+1}zwv_xv_{x+1}) \equiv_L \{(\overline{v}, v_x, v_{x+1}), \bot\}$. If $(\overline{v}, v_x, v_{x+1}zwv_xv_{x+1}) \equiv_L \bot$, then (ii) holds. Otherwise, we get $(\overline{v}, v_x, v_{x+1}zwv_xv_{x+1}) \equiv_L (\overline{v}, v_x, v_{x+1})$, which is indicated by the thick v_{x+1}-edge on layer $x+1$. Now it can be seen from Figure 11, that in this case, $(zwv_xv_{x+1})^\omega$ is strongly accepted/rejected from $[(\overline{v}, v_x, v_{x+1})]_{\equiv_L}$ iff $x+1$ is even/odd. By Lemma 6.13, this would mean that $v_1 \cdots v_{x+1}(zwv_xv_{x+1})^\omega \in L$ iff $x+1$ is even. But we have chosen w such that $v_1 \cdots v_{x-1}(v_xv_{x+1}zw)^\omega = v_1 \cdots v_{x+1}(zwv_xv_{x+1})^\omega \in L$ iff x is even, a contradiction. ◀

▶ **Lemma 6.18.** *Let L be an ω-regular language. Then $\mathcal{A}_{\equiv_L}$ is safe minimal, normal, and centralised.*

Proof. A layered automaton is safe minimal if for all x and for all states p, q on layer x that have the same parent, the safe language of p and q are different. So let $(\overline{u}, u_x)$ and $(\overline{v}, v_x)$ be two pointed tuples with $\overline{u} \cdot u_x \equiv_L \overline{v} \cdot v_x$. Then, by definition of $\equiv_L$, we get $(\overline{u}, u_x) \equiv_L (\overline{v}, v_x)$ iff $L^s(\overline{u}, u_x) = L^s(\overline{v}, v_x)$. So two pointed tuples on layer x correspond to different states iff they have a different safe language.

For showing that $\mathcal{A}_{\equiv_L}$ is normal, we first need to show property N1 from Definition 4.2, namely that each layer $x \geq 2$ is a union of non-trivial SCCs. So let $(\overline{u}, u_x)$ be some pointed tuple on level x, and let $v \in \Sigma^*$ be such that $(\overline{u}, u_xv) \not\equiv_L \bot$, that is, in $\mathcal{A}_{\equiv_L}$ there is the path $[(\overline{u}, u_x)]_{\equiv_L} \xrightarrow{v} [(\overline{u}, u_xv)]_{\equiv_L}$. Since $(\overline{u}, u_xv) \not\equiv_L \bot$, there is $w \in \Sigma^+$ such that $\overline{u} \cdot uvw \equiv_L \overline{u}$ and $u_1 \cdots u_{x-1}(u_xvw)^\omega \in L$ iff x is even. Then $(\overline{u}, u_xvwu_x) \not\equiv_L \bot$ and since $(\overline{u}, u_x)$ is pointed, we get $(\overline{u}, u_xvwu_x) \equiv_L (\overline{u}, u_x)$, so in $\mathcal{A}_{\equiv_L}$ we get the following path $[(\overline{u}, u_xv)]_{\equiv_L} \xrightarrow{wu_x} [(\overline{u}, u_x)]_{\equiv_L}$.

The property N2 required for normal follows from the existence of central sequences (the central sequence of a state has the properties required for N2). That $\mathcal{A}_{\equiv_L}$ is centralised also follows from the existence of central sequences: If $[\overline{u}]_{\equiv_L} \preccurlyeq_x [\overline{v}]_{\equiv_L}$ for two tuples of length $x \geq 2$, then a central sequence z for $[\overline{u}]_{\equiv_L}$ cannot send $[\overline{v}]_{\equiv_L}$ to $\bot$, and hence $[\overline{v} \cdot z]_{\equiv_L} = [\overline{u}]_{\equiv_L}$, which means $[\overline{v}]_{\equiv_L} \xrightarrow{z}_x [\overline{u}]_{\equiv_L}$. By property N1 of normal, $[\overline{u}]_{\equiv_L}$ and $[\overline{v}]_{\equiv_L}$ are in the same x-SCC. ◀

Combining Lemma 6.18, Theorem 6.14, and Theorem 4.1 yields the main result of this section.

▶ **Theorem 6.19.** *Let L be an ω-regular language. Then $\mathcal{A}_{\equiv_L}$ is the minimal layered automaton for L.*

7 Conclusions

In this paper, we have introduced layered automata, a model that merges and generalises minimal HD coBüchi automata [AK22] and Zielonka trees [Zie98]. We have shown that they provide a canonical representation for ω-regular languages that can be captured via congruences that only depend on the given language.

We have established minimality of canonical layered automata within the class of consistent layered automata (Corollary 5.2). However, we believe that they are minimal among the full class of alternating history deterministic automata.

▶ **Conjecture 7.1.** *Let $\mathcal{B}$ be a history deterministic alternating parity automaton. Then, there is an equivalent layered automaton $\mathcal{A}$ such that $[\![\mathcal{A}]\!]$ has no more states than $\mathcal{B}$.*

In particular, the minimal layered automaton of a language is minimal among all history deterministic alternating parity automaton.[9]

Other future directions. We believe that layered automata are a promising model for both practical and theoretical goals. There are many aspects of the theory of layered automata that are yet to be developed.

A central question is to provide efficient constructions for boolean operations of languages, or give direct translations from logical formalisms such as *Linear Temporal Logic* to layered automata. Similarly, we expect that we could build the minimal layered automaton of a language L directly from its representation as a COCOA, based on the techniques from [EK24a] and avoiding the full product construction discussed in Section 3.6.

Another exciting direction is the use of layered automata for the passive and active learning of ω-regular languages, extending the results on passive learning for the coBüchi case [LW25].

[9] We have a draft of a proof for Conjecture 7.1 for the case of 2 priorities (Büchi and coBüchi automata). That is, the minimal HD coBüchi automaton from Abu Radi and Kupferman [AK22] is minimal among all HD alternating coBüchi automata.

References

ABF18 Dana Angluin, Udi Boker, and Dana Fisman. Families of DFAs as acceptors of ω-regular languages. *Log. Methods Comput. Sci.*, volume 14(1), 2018. doi: http://doi.org/10.23638/LMCS-14(1:15)2018.

AE25 Bader Abu Radi and Rüdiger Ehlers. Characterizing the polynomial-time minimizable ω-automata. *CoRR*, volume abs/2504.20553, 2025. doi: http://doi.org/10.48550/ARXIV.2504.20553.

AF16 Dana Angluin and Dana Fisman. Learning regular omega languages. *Theor. Comput. Sci.*, volume 650:57–72, 2016. doi: http://doi.org/10.1016/J.TCS.2016.07.031.

AK15 Guy Avni and Orna Kupferman. Stochastization of weighted automata. In *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part I*, volume 9234 of *Lecture Notes in Computer Science*, pages 89–102. Springer, 2015. doi: http://doi.org/10.1007/978-3-662-48057-1_7.

AK22 Bader Abu Radi and Orna Kupferman. Minimization and canonization of GFG transition-based automata. *Logical Methods in Computer Science*, volume Volume 18, Issue 3:7587, 2022. doi: http://doi.org/10.46298/lmcs-18(3:16)2022.

Arn85 André Arnold. A syntactic congruence for rational omega-languages. *Theor. Comput. Sci.*, volume 39:333–335, 1985. doi: http://doi.org/10.1016/0304-3975(85)90148-3.

BGB12 Christel Baier, Marcus Grösser, and Nathalie Bertrand. Probabilistic ω–automata. *J. ACM*, volume 59(1), 2012. doi: http://doi.org/10.1145/2108242.2108243.

BKLS20 Udi Boker, Denis Kuperberg, Karoliina Lehtinen, and Michal Skrzypczak. On the succinctness of alternating parity good-for-games automata. In Nitin Saxena and Sunil Simon, editors, *FSTTCS*, volume 182 of *LIPIcs*, pages 41:1–41:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi: http://doi.org/10.4230/LIPICS.FSTTCS.2020.41.

BL19 Udi Boker and Karoliina Lehtinen. Good for games automata: From nondeterminism to alternation. In Wan J. Fokkink and Rob van Glabbeek, editors, *CONCUR*, volume 140 of *LIPIcs*, pages 19:1–19:16. 2019. doi: http://doi.org/10.4230/LIPICS.CONCUR.2019.19.

BL24 León Bohn and Christof Löding. Constructing deterministic parity automata from positive and negative examples. *TheoretiCS*, volume 3, 2024. doi: http://doi.org/10.46298/THEORETICS.24.17.

Büc62 J. Richard Büchi. On a decision method in restricted second order arithmetic. *Proc. Internat. Congr. on Logic, Methodology and Philosophy of Science*, pages 1–11, 1962.

Cas22 Antonio Casares. On the minimisation of transition-based Rabin automata and the chromatic memory requirements of Muller conditions. In *CSL*, volume 216, pages 12:1–12:17. 2022. doi: http://doi.org/10.4230/LIPIcs.CSL.2022.12.

CCFL24 Antonio Casares, Thomas Colcombet, Nathanaël Fijalkow, and Karoliina Lehtinen. From Muller to parity and Rabin automata: Optimal transformations preserving (history) determinism. *TheoretiCS*, volume 3, 2024. doi: http://doi.org/10.46298/THEORETICS.24.12.

CES86 E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, volume 8(2):244–263, 1986. doi: http://doi.org/10.1145/5397.5399.

CKS81 Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, volume 28(1):114–133, 1981. doi: http://doi.org/10.1145/322234.322243.

CL08 Thomas Colcombet and Christof Löding. The non-deterministic Mostowski hierarchy and distance-parity automata. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008*, volume 5126 of *Lecture Notes in Computer Science*, pages 398–409. Springer, 2008. doi: http://doi.org/10.1007/978-3-540-70583-3_33.

CNP93 Hugues Calbrix, Maurice Nivat, and Andreas Podelski. Ultimately periodic words of rational w-languages. In Stephen D. Brookes, Michael G. Main, Austin Melton, Michael W.

Mislove, and David A. Schmidt, editors, *Mathematical Foundations of Programming Semantics, 9th International Conference, New Orleans, LA, USA, April 7-10, 1993, Proceedings*, volume 802 of *Lecture Notes in Computer Science*, pages 554–566. Springer, 1993. doi: http://doi.org/10.1007/3-540-58027-1_27.

CO24 Antonio Casares and Pierre Ohlmann. Positional ω-regular languages. In *LICS*, pages 21:1–21:14. ACM, 2024. doi: http://doi.org/10.1145/3661814.3662087.

Col13 Thomas Colcombet. *Fonctions régulières de coût*. Habilitation (hdr), Université Paris Diderot – Paris 7, 2013.

DJW97 Stefan Dziembowski, Marcin Jurdziński, and Igor Walukiewicz. How much memory is needed to win infinite games? In *LICS*, pages 99–110. 1997. doi: http://doi.org/10.1109/LICS.1997.614939.

Ehl25 Rüdiger Ehlers. Rerailing automata. *CoRR*, volume abs/2503.08438, 2025. doi: http://doi.org/10.48550/ARXIV.2503.08438.

EK24a Rüdiger Ehlers and Ayrat Khalimov. Fully Generalized Reactivity(1) Synthesis. In Bernd Finkbeiner and Laura Kovács, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 14570, pages 83–102. 2024. doi: http://doi.org/10.1007/978-3-031-57246-3_6.

EK24b Rüdiger Ehlers and Ayrat Khalimov. A Naturally-Colored Translation from LTL to Parity and COCOA. (arXiv:2410.01021), 2024. doi: http://doi.org/10.48550/ARXIV.2410.01021.

ES22 Rüdiger Ehlers and Sven Schewe. Natural Colors of Infinite Words. *LIPIcs, Volume 250, FSTTCS 2022*, volume 250:36:1–36:17, 2022. doi: http://doi.org/10.4230/LIPICS.FSTTCS.2022.36.

FAA+25 Nathanaël Fijalkow, C. Aiswarya, Guy Avni, Nathalie Bertrand, Patricia Bouyer, Romain Brenguier, Arnaud Carayol, Antonio Casares, John Fearnley, Paul Gastin, Hugo Gimbert, Thomas A. Henzinger, Florian Horn, Rasmus Ibsen-Jensen, Nicolas Markey, Benjamin Monmege, Petr Novotný, Pierre Ohlmann, Mickael Randour, Ocan Sankur, Sylvain Schmitz, Olivier Serre, Mateusz Skomra, Nathalie Sznajder, and Pierre Vandenhove. Games on graphs: From logic and automata to algorithms, 2025. doi: https://arxiv.org/abs/2305.10546. To appear in Cambridge University Press.

GTW02 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata Logics, and Infinite Games*. Springer, Berlin, Heidelberg, 2002. doi: http://doi.org/10.1007/3-540-36387-4.

HK02 Markus Holzer and Barbara König. On deterministic finite automata and syntactic monoid size. In *Developments in Language Theory, DLT 2002*, volume 2450 of *Lecture Notes in Computer Science*, pages 258–269. Springer, 2002. doi: http://doi.org/10.1007/3-540-45005-X_22.

HP06 Thomas A. Henzinger and Nir Piterman. Solving games without determinization. In *Computer Science Logic*, pages 395–410. 2006. doi: http://doi.org/10.1007/11874683_26.

HPS+20 Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. Good-for-mdps automata for probabilistic analysis and reinforcement learning. In Armin Biere and David Parker, editors, *TACAS*, volume 12078 of *Lecture Notes in Computer Science*, pages 306–323. 2020. doi: http://doi.org/10.1007/978-3-030-45190-5_17.

HPT25 Thomas A. Henzinger, Aditya Prakash, and K. S. Thejaswini. Resolving nondeterminism with randomness. In Pawel Gawrychowski, Filip Mazowiecki, and Michal Skrzypczak, editors, *MFCS*, volume 345 of *LIPIcs*, pages 57:1–57:18. 2025. doi: http://doi.org/10.4230/LIPICS.MFCS.2025.57.

IL25 Olivier Idir and Karoliina Lehtinen. Using games and universal trees to characterise the nondeterministic index of tree languages. In *52nd International Colloquium on Automata, Languages, and Programming, ICALP 2025*, volume 334 of *LIPIcs*, pages 160:1–160:19.

Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025. doi: http://doi.org/10.4230/LIPICS.ICALP.2025.160.

Kla94 Nils Klarlund. A homomorphism concepts for omega-regularity. In *Computer Science Logic, 8th International Workshop, CSL '94, Kazimierz, Poland, September 25-30, 1994, Selected Papers*, volume 933 of *Lecture Notes in Computer Science*, pages 471–485. Springer, 1994. doi: http://doi.org/10.1007/BFb0022276.

KMS02 Nils Klarlund, Anders Møller, and Michael I. Schwartzbach. MONA implementation secrets. *Int. J. Found. Comput. Sci.*, volume 13(4):571–586, 2002. doi: http://doi.org/10.1142/S012905410200128X.

KS15 Denis Kuperberg and Michał Skrzypczak. On determinisation of good-for-games automata. In *ICALP*, pages 299–310. 2015. doi: http://doi.org/10.1007/978-3-662-47666-6_24.

KSV96 Orna Kupferman, Shmuel Safra, and Moshe Y. Vardi. Relating word and tree automata. In *LICS*, pages 322–332. 1996. doi: http://doi.org/10.1109/LICS.1996.561360.

Kup18 Orna Kupferman. Automata theory and model checking. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 107–151. Springer International Publishing, 2018. doi: http://doi.org/10.1007/978-3-319-10575-8_4.

LW25 Christof Löding and Igor Walukiewicz. Minimal history-deterministic co-Büchi automata: Congruences and passive learning. In *LICS*, pages 431–443. IEEE, 2025. doi: http://doi.org/10.1109/LICS65433.2025.00039.

Mar75 Donald A. Martin. Borel determinacy. *Annals of Mathematics*, volume 102(2):363–371, 1975.

MP90 Zohar Manna and Amir Pnueli. A hierarchy of temporal properties. In Cynthia Dwork, editor, *PODC*, pages 377–410. ACM, 1990. doi: http://doi.org/10.1145/93385.93442.

MS97 Oded Maler and Ludwig Staiger. On syntactic congruences for ω-languages. *Theoretical Computer Science*, volume 183(1):93–112, 1997. doi: http://doi.org/10.1016/S0304-3975(96)00312-X. Formal Language Theory.

PP95 Dominique Perrin and Jean-Éric Pin. Semigroups and automata on infinite words. In *Semigroups, formal languages and groups (York, 1993)*, volume 933, pages 49–72. Kluwer Acad. Publ., 1995.

PPS+25 Soumyajit Paul, David Purser, Sven Schewe, Qiyi Tang, Patrick Totzke, and Di-De Yen. Resolving nondeterminism by chance. In Patricia Bouyer and Jaco van de Pol, editors, *CONCUR*, volume 348 of *LIPIcs*, pages 32:1–32:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025. doi: http://doi.org/10.4230/LIPICS.CONCUR.2025.32.

PR89 Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL*, page 179–190. 1989. doi: http://doi.org/10.1145/75277.75293.

Sch10 Sven Schewe. Beyond hyper-minimisation—minimising DBAs and DPAs is NP-complete. In *FSTTCS*, volume 8, pages 400–411. 2010. doi: http://doi.org/10.4230/LIPIcs.FSTTCS.2010.400.

Wil91 Thomas Wilke. An Eilenberg theorem for infinity-languages. In *ICALP*, volume 510, pages 588–599. 1991. doi: http://doi.org/10.1007/3-540-54233-7_166.

Zie98 Wiesław Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, volume 200(1-2):135–183, 1998. doi: http://doi.org/10.1016/S0304-3975(98)00009-7.