Acta Informatica manuscript No.
(will be inserted by the editor)

Davide Bresolin · Angelo Montanari · Gabriele Puppis

A Theory of Ultimately Periodic Languages and Automata with an Application to Time Granularity

Abstract In this paper, we develop a theory of regular ω-languages that consist of ultimately periodic words only and we provide it with an automaton-based characterization. The resulting class of automata, called Ultimately Periodic Automata (UPA), is a subclass of the class of Büchi automata and inherits some properties of automata over finite words (NFA). Taking advantage of the similarities among UPA, Büchi automata, and NFA, we devise efficient solutions to a number of basic problems for UPA, such as the inclusion, the equivalence, and the size optimization problems. The original motivation for developing a theory of ultimately periodic languages and automata was to represent and to reason about sets of time granularities in knowledge-based and database systems. In the last part of the paper, we show that UPA actually allow one to represent (possibly infinite) sets of granularities, instead of single ones, in a compact and suitable to algorithmic manipulation way. In particular, we describe an application of UPA to a concrete time granularity scenario taken from clinical medicine.

A short preliminary version of this paper appeared in [4].

D. Bresolin
Department of Computer Science, University of Verona
Ca' Vignal 2, strada le Grazie 15 - 37134 Verona - Italy
E-mail: bresolin@sci.univr.it

A. Montanari
Department of Mathematics and Computer Science, University of Udine
via delle Scienze 206 - 33100 Udine - Italy
E-mail: angelo.montanari@dimi.uniud.it

G. Puppis
Department of Mathematics and Computer Science, University of Udine
via delle Scienze 206 - 33100 Udine - Italy
E-mail: gabriele.puppis@dimi.uniud.it

1 Introduction

The relevance of the problem of managing periodic phenomena is widely recognized in different areas of computer science. Various formalizations of periodicity have been proposed in the literature, following algebraic, logical, string-based, and automaton-based approaches. We focus our attention on the problem of representing and reasoning about *sets* of ultimately periodic words. We first introduce the notion of ultimately periodic regular ω-language, namely, a Büchi-recognizable language that consists of ultimately periodic words only, and then we develop a theory of automata for this class of languages.

In analogy with the case of Büchi-recognizable languages, regular ω-languages of ultimately periodic words can be expressed as finite unions of languages of the form $U\{v\}^\omega$, where U is a regular language of finite words and v is a non-empty finite word. Each language in such a class may feature infinitely many distinct prefixes, but only a finite number of non-equivalent repeating patterns. Taking advantage of such a characterization, we show that these languages can be finitely represented by means of a suitable class of automata, called Ultimately Periodic Automata (UPA). UPA can be equivalently viewed as a proper subclass of Büchi automata or as a class of generalized non-deterministic finite state automata (NFA). The similarities among UPA, Büchi automata, and NFA make it possible to devise efficient solutions to a number of basic problems for regular ω-languages of ultimately periodic words, including the emptiness problem, that is, the problem of deciding whether a given language is empty, the membership problem, that is, the problem of deciding whether a certain word belongs to a given language, the equivalence problem, that is, the problem of deciding whether or not two given languages coincide, the inclusion problem, that is, the problem of deciding whether a given language is included in another one, and the size-optimization problem, that is, the problem of computing compact representations of a given language.

Ultimately periodic languages and automata are then used to model and to reason about possibly infinite sets of ultimately periodic time granularities, that is, temporal structures that, starting from a given point, periodically group instants of an underlying temporal domain. As long as one confines oneself to single time granularities taken in isolation, there is a plenty of formal systems for dealing with them in a systematic way (most notably, Calendar Algebra [2], Linear Temporal Logic over integer periodicity constraints [13], and Single-String Automata [9]). Things become much more complex when one must cope with sets of ultimately periodic granularities instead of single ones. We show that UPA can be successfully exploited to address a number of basic problems about sets of time granularities. In particular, we show how UPA allow one to solve the crucial problem of granularity comparison, that is, the problem of deciding, given two sets of granularities $\mathcal{G}$ and $\mathcal{H}$, whether there exist $G \in \mathcal{G}$ and $H \in \mathcal{H}$ such that $G \sim H$, where $\sim$ is one of the commonly-used relations between granularities, e.g, partition, group, refinement, aligned refinement [2]. A real-world application taken from clinical medicine, showing how the processes of specification and validation of

therapy plans benefit from the ability of managing sets of ultimately periodic time granularities via UPA, concludes the paper.

The paper is organized as follows. In Section 2, we briefly describe the different approaches to time granularity proposed in the literature, pointing out their relationships and limitations. In Section 3, we give a characterization of regular ω-languages consisting of ultimately periodic words only. Next, in Section 4 we define the class of UPA as the automaton-based counterpart to ultimately periodic languages, and we introduce different normal forms for them. In Section 5, we provide a simple, but efficient, solution to the emptiness, membership, equivalence, inclusion, and size-optimization problems for UPA. The application of UPA to the management of time granularities is given in Section 6, together with a real-world medical example that requires the ability of simultaneously dealing with sets of time granularities.

2 A framework for time granularity

The original motivation for developing a theory of ultimately periodic languages and automata was to represent and to reason about sets of time granularities in a systematic way. In this section, we provide some background knowledge about time granularity. First, we give a formal definition of it. Then, we briefly survey the various approaches to time granularity proposed in the literature. As a matter of fact, all of them confine themselves to the case of single time granularities.

According to a commonly accepted perspective, any time granularity can be viewed as the partitioning of a temporal domain in groups of elements, where each group is perceived as an indivisible unit (a granule) [2]. In particular, most granularities of interest are modeled as infinite sequences of granules, that present a repeating pattern and, possibly, temporal gaps within and between granules.

As it happens in most application domains, let us assume the underlying temporal domain to be (isomorphic to) the linear order $(\mathbb{N}, <)$ of the natural numbers. A time granularity is formally defined as follows.

Definition 1 A *time granularity* is a collection $\mathsf{G} \subseteq \mathscr{P}(\mathbb{N})$ of subsets of the temporal domain such that distinct sets in G (henceforth called *granules*) do not overlap, namely, for every pair of distinct granules $\mathsf{g}, \mathsf{g}' \in \mathsf{G}$, we have either $\mathsf{t} < \mathsf{t}'$ for all $\mathsf{t} \in \mathsf{g}$ and $\mathsf{t}' \in \mathsf{g}'$ or $\mathsf{t}' < \mathsf{t}$ for all $\mathsf{t} \in \mathsf{g}$ and $\mathsf{t}' \in \mathsf{g}'$.

Such a definition captures both time granularities that cover the whole temporal domain, such as `Day`, `Week`, and `Month`, and time granularities with gaps within (e.g., `BusinessMonth`) and between granules (e.g., `BusinessDay` and `BusinessWeek`). Figure 1 depicts some of these granularities. Moreover, the natural order on the elements of the temporal domain $\mathbb{N}$ induces a similar order on the granules of a granularity G. Thus, given two granules $\mathsf{g}, \mathsf{g}' \in \mathsf{G}$, we can write $\mathsf{g} < \mathsf{g}'$ whenever $\mathsf{t} < \mathsf{t}'$ holds for every $\mathsf{t} \in \mathsf{g}$ and $\mathsf{t}' \in \mathsf{g}'$. Such an order naturally yields a labeling of the granules of G: we say that $\mathsf{x} \in \mathbb{N}$ is the *index* of a granule $\mathsf{g} \in \mathsf{G}$, and we write $\mathsf{G}(\mathsf{x}) = \mathsf{g}$, if g is the $\mathsf{x}+1$-th element of G according to the induced order on the granules.

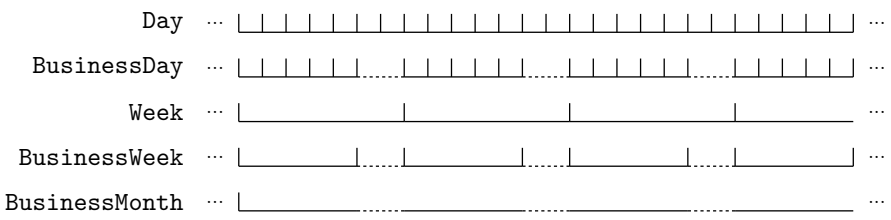

Fig. 1 Some examples of time granularities.

Various relations can be defined between pairs of time granularities, including, for instance, grouping, refinement, partition, and aligned refinement, which are defined as follows (further relations between time granularities are given in [2]). A granularity G *groups into* a granularity H if each granule of H is the union of some granules of G, while a granularity G *refines* a granularity H if every granule of G is contained in some granule of H. A granularity G *partitions* a granularity H if G both groups into H and refines H. Finally, a granularity G is an *aligned refinement* of H if, for every $n \in \mathbb{N}$, the granule $\mathsf{G}(n)$ is included in the granule $\mathsf{H}(n)$. As an example, in Figure 1, we have that `Day` groups into `BusinessMonth`, `BusinessDay` refines `Week`, `Day` partitions `Week`, and `BusinessWeek` is an aligned refinement of `Week`.

It is immediate to realize that the set of all structures that satisfy the given definition of time granularity becomes uncountable as soon as the underlying temporal domain is infinite. As a consequence, it is not possible to deal with all these structures by means of a finitary formalism. However, the problem of mastering time granularities can be tackled in an effective way by restricting to ultimately periodic granularities, namely, to those structures that, ultimately, periodically group elements (time points) of the underlying temporal domain.

A number of approaches to deal with time granularity have been proposed in the literature. The most common approach is the algebraic one, which represents time granularities as suitable symbolic expressions and defines the relationships between pairs of granularities by means of algebraic operators. In this setting, the most important contributions are the formalism of collection expressions [21], that of slice expressions [27], and the Calendar Algebra [28]. All of them make it possible to capture granularities of practical interest, including infinite periodical ones. The different sets of algebraic operators provided by the three formal systems and their expressiveness are investigated in [2], where it is proved that Calendar Algebra actually subsumes the other two systems.

A logical account of Calendar Algebra has been provided by Combi et al. in [8]. It defines time granularities as models of formulas in Propositional Linear Time Logic (PLTL) [14], where suitable propositional symbols are used to mark the starting and ending points of granules. The expressiveness of PLTL makes it possible to capture a large set of regular granularities, such as, for instance, repeating patterns that can start at an arbitrary time instant. Furthermore, problems like checking the consistency of a granularity specification and testing the equivalence of two granularity expressions can

be solved in a uniform way by reducing them to the validity problem for PLTL, which is known to be decidable in polynomial space.

An alternative approach to the representation and manipulation of time granularities has been proposed by Wijsen [36]. It models infinite granularities as infinite sequences over an alphabet consisting of three symbols, namely, ■ (filler), □ (gap), and ≀ (separator), which are respectively used to denote time points covered by some granule, to denote time points not covered by any granule, and to delimit granules. Left bounded granularities that, starting from a given point, periodically group instants of the underlying temporal domain can be represented as ultimately periodic words over such an alphabet. For instance, the granularity `BusinessWeek` can be encoded by the ultimately periodic word ■■■■■□□≀ ■■■■■□□ ≀.... Ultimately periodic words can be finitely represented in terms of a (possibly empty) prefix and a repeating pattern. As an example, the granularity `BusinessWeek` is represented by the empty prefix ε and the repeating pattern ■■■■■□□≀. Wijsen's string-based model can be used to solve some basic problems about granularities, such as the equivalence problem, which consists of establishing whether or not two given representations define the same granularity, and the minimization problem, which consists of computing the most compact representation of a granularity. As an example, the equivalence problem can be solved by introducing a suitable *aligned form*, in which separators are forced to occur immediately after an occurrence of ■. In such a way, one can encode each occurrence of ■≀ by means of a single symbol ◀. Taking advantage from such a rewriting, it is possible to establish a one-to-one correspondence between strings and granularities, thus providing a straightforward solution to the equivalence problem.

The idea of viewing time granularities as periodic strings naturally connects the notion of time granularity to the field of formal languages and automata. An original automaton-based approach to time granularity has been proposed by Dal Lago and Montanari in [9], and later revisited by Dal Lago, Montanari and Puppis in [10, 11, 12, 31]. Granularities are viewed as strings generated by a specific class of automata, called Single-String Automata (SSA), thus making it possible to (re)use well-known results from automata theory. SSA recognize languages consisting of a single ultimately periodic word; moreover, they can be endowed with counters ranging over finite domains in order to compactly encode the redundancies of the temporal structures[1]. Properties of automata are then exploited in order to efficiently solve problems about single time granularities, such as the problem to establish whether two different SSA encode the same granularity and the granule conversion problem, that is, the problem of properly relating granules belonging to different time granularities. In the last part of the paper we will show that ultimately periodic languages and automata make it possible to formally represent (possibly infinite) sets of ultimately periodic granularities and to efficiently manipulate them.

[1] A logical account of SSA with counters has been given by Demri in [13].

3 Languages of ultimately periodic words

In this section, we study the properties of regular ω-languages, that is, languages of infinite words recognized by Büchi automata, consisting of ultimately periodic words only. To start with, we introduce the notation and we briefly recall basic definitions and properties of finite state and Büchi automata.

Given a finite or infinite word w over a finite alphabet Σ, we denote by $|w|$ the size of w, that is, the number of symbols in w. Moreover, we denote by $w(i)$ the i-th symbol of w and, given two indices i and j, we denote by $w[i,j]$ the substring $w(i)w(i+1)...w(j)$ of w. An infinite word w is said to be *ultimately periodic* if it can be written as uv^ω, where $u \in \Sigma^*$ and $v \in \Sigma^+$. The finite words u and v are respectively called an *initial pattern* and a *repeating pattern* of w. Notice that an ultimately periodic word can be finitely presented by several distinct pairs consisting of an initial pattern and a repeating pattern. However, among all pairs that represent the same ultimately periodic word, there exists exactly one which has minimum size (the size of a pair (u,v) is $|u|+|v|$).

A *(sequential) automaton* is a tuple $\mathcal{A} = (\Sigma, S, \Delta, \mathcal{I}, \mathcal{F})$, where Σ is a finite alphabet, S is a finite set of states, $\Delta \subseteq S \times \Sigma \times S$ is a transition relation, $\mathcal{I} \subseteq S$ is a set of initial states, and $\mathcal{F} \subseteq S$ is a set of final states. We define the size of an automaton $\mathcal{A}$ as the number of its states and transitions, and we denote it by $|\mathcal{A}|$ as usual.

A *run* of $\mathcal{A}$ over a finite (resp., infinite) word $w \in \Sigma^*$ (resp., $w \in \Sigma^\omega$) is a finite (resp., infinite) sequence of states ρ such that

- $|\rho| = |w| + 1$ (resp., $|\rho| = |w| = \omega$),
- for every $1 \leqslant i \leqslant |w|$ (resp., for every $i \geqslant 1$), $\big(\rho(i), w(i), \rho(i+1)\big) \in \Delta$.

Acceptance conditions for finite and infinite words are obviously different. If w is a finite word and ρ is a run of $\mathcal{A}$ on w such that $\rho(1) \in \mathcal{I}$ and $\rho(|\rho|) \in \mathcal{F}$, then we say that ρ is a *successful run* on w and that $\mathcal{A}$ *accepts* w. If w is an infinite word, we say that ρ is a *successful run* on w and that $\mathcal{A}$ *accepts* w if ρ is such that $\rho(1) \in \mathcal{I}$ and $\rho(i) \in \mathcal{F}$ for infinitely many indices $i \geqslant 1$.

The language of finite words *recognized* by $\mathcal{A}$, denoted $\mathscr{L}(\mathcal{A})$, is the set of all and only the finite words that are accepted by $\mathcal{A}$, while the ω-language of infinite words *recognized* by $\mathcal{A}$, denoted by $\mathscr{L}^\omega(\mathcal{A})$, is the set of all and only the infinite words that are accepted by $\mathcal{A}$. For the sake of simplicity, we call *(non-deterministic) finite state automaton* (NFA for short) a sequential automaton that recognizes a language of finite words; similarly, we call *Büchi automaton* a sequential automaton that recognizes an ω-language of infinite words.

Proposition 1 (Büchi [1]) *The class of regular ω-languages is effectively closed under union, intersection, and complementation, namely, given two Büchi automata $\mathcal{A}$ and $\mathcal{B}$, one can compute a Büchi automaton $\mathcal{A} \cup \mathcal{B}$ (resp., $\mathcal{A} \cap \mathcal{B}$, $\bar{\mathcal{A}}$) recognizing the ω-language $\mathscr{L}^\omega(\mathcal{A}) \cup \mathscr{L}^\omega(\mathcal{B})$ (resp., $\mathscr{L}^\omega(\mathcal{A}) \cap \mathscr{L}^\omega(\mathcal{B})$, $\Sigma^\omega \setminus \mathscr{L}^\omega(\mathcal{A})$).*

Proposition 2 (Büchi [1]) *An ω-language L is regular iff it is a finite union of sets of the form UV^ω, where U and V are regular languages of finite words.*

Hereafter, we denote by U_Σ the universal ω-language that consists of all and only the ultimately periodic words over Σ. Moreover, given an ω-language $L \subseteq \Sigma^\omega$, we denote by $\mathcal{UP}(L)$ the ω-language $L \cap U_\Sigma$, which consists of all and only the ultimately periodic words that belong to L. Clearly, an ω-language L consists only of ultimately periodic words if and only if $L = \mathcal{UP}(L)$.

Proposition 3 (Büchi [1], Calbrix et al. [6]) *Every non-empty regular ω-language contains at least one ultimately periodic word. Moreover, if L_1 and L_2 are two regular ω-languages, then $L_1 = L_2$ iff $\mathcal{UP}(L_1) = \mathcal{UP}(L_2)$.*

Proof As for the first claim, by Proposition 2, any regular ω-language L can be written as $\bigcup_{1 \leqslant i \leqslant n} U_i V_i^\omega$, with $\varepsilon \notin V_i$ for every $1 \leqslant i \leqslant n$. Since L is not empty, there exists an index $1 \leqslant i \leqslant n$ such that both U_i and V_i are not empty. Therefore, L must contain an ultimately periodic word of the form $w = uv^\omega$, with $u \in U_i$ and $v \in V_i$.
As for the second claim, let L_1 and L_2 be two regular ω-languages containing the same ultimately periodic words. The left-to-right implication is trivial. For the converse implication, we know, from closure properties of regular ω-languages (see Proposition 1), that $(L_1 \setminus L_2) \cup (L_2 \setminus L_1)$ is a regular ω-language, which contains no ultimately periodic words. Thus $(L_1 \setminus L_2) \cup (L_2 \setminus L_1)$ is empty and $L_1 = L_2$ follows. □

In the following, we provide a characterization of regular ω-languages of ultimately periodic words only, in analogy with that of Proposition 2.

To start with, we point out that there exist non-regular ω-languages consisting of ultimately periodic words only: for instance, since Σ^ω is a regular ω-language, $\mathcal{UP}(U_\Sigma) = \mathcal{UP}(\Sigma^\omega)$, and $U_\Sigma \neq \Sigma^\omega$, then, by Proposition 3, U_Σ cannot be a regular ω-language.

Proposition 4 *The following closure properties hold:*

i) *if v is a non-empty finite word, $\{v\}^\omega$ is a regular ω-language consisting of a single ultimately periodic word;*

ii) *if U is a regular language and V is a regular ω-language of ultimately periodic words, then UV is a regular ω-language of ultimately periodic words;*

iii) *if L_1 and L_2 are regular ω-languages of ultimately periodic words, then $L_1 \cup L_2$ and $L_1 \cap L_2$ are regular ω-languages of ultimately periodic words.*

Proof The claim immediately follows from closure properties of regular ω-languages, since the above operations do not introduce words which are not ultimately periodic. □

As for the complementation of an ω-language of ultimately periodic words, it must be obviously defined with respect to the universe U_Σ, that is, the complement of $L \in U_\Sigma$ is $\bar{L} = U_\Sigma \setminus L$. Notice that there is no guarantee that $\bar{L}$ is regular whenever L is regular.

Proposition 5 *Regular ω-languages of ultimately periodic words are not closed under complementation.*

Proof A counterexample is given by the empty set: it trivially is a regular ω-language of ultimately periodic words, but its complement is the universal ω-language U_Σ, which is not regular. In fact, the complement of *any* regular ω-language L of ultimately periodic words is not regular, since $U_\Sigma = L \cup \bar{L}$. $\square$

Given an ultimately periodic word $w = uv^\omega$, the set of its repeating patterns is clearly infinite and it contains, among others, the finite words v, v^2, v^3, ... To group together the different repeating patterns of an ultimately periodic word, we define a suitable equivalence relation. Such an equivalence will play an essential role in the characterization of regular ω-languages of ultimately periodic words we are going to provide.

Definition 2 Let $\cong \subseteq \Sigma^* \times \Sigma^*$ be an equivalence relation such that $u \cong v$ iff the two infinite periodic words u^ω and v^ω share a common suffix, namely, there exist $x, y \in \Sigma^*$ and $z \in \Sigma^\omega$ such that $u^\omega = xz$ and $v^\omega = yz$.

Notice that in Definition 2 one can always assume either x or y to be ε.

It can be easily checked that all repeating patterns of a given ultimately periodic word w are equivalent. Moreover, they can be obtained by choosing different rotations and/or different repetitions of the *primitive* repeating pattern of w, namely, the shortest substring $w[i, j]$ of w such that (i) $w = w[1, i-1](w[i, j])^\omega$ and (ii) either $i = 1$ or $w(j) \neq w(i-1)$. Conversely, if v is a repeating pattern of an ultimately periodic word w and v' is equivalent to v, then v' is also a repeating pattern of w.

Given an ω-language L and a finite word v, we say that L *features* v as a repeating pattern if L contains an ultimately periodic word w having v as a repeating pattern; moreover, if v belongs to a language of the form V^+, with $V \subseteq \Sigma^*$, then we say that v is a V-*aligned* repeating pattern.

Below, we prove some fundamental properties of ω-languages of the form V^ω, where $V \subseteq \Sigma^*$, with respect to the repeating patterns they feature.

Lemma 1 *Given a language V, for every repeating pattern v featured by V^ω, there exists an equivalent V-aligned repeating pattern z featured by V^ω.*

Proof Let v be a repeating pattern featured by V^ω. By definition, V^ω contains an infinite word $w = u_1u_2u_3...$, with $u_i \in V$ for all $i \geqslant 0$, which is ultimately periodic with v as a repeating pattern. Thus, w can be written as uv^ω, where u is a suitable finite word. Let i_0 be a sufficiently large index such that u turns out to be a prefix of $u_1u_2...u_{i_0}$ (or, equivalently, $u_{i_0+1}u_{i_0+2}...$ turns out to be a suffix of v^ω). Moreover, let f be the function that maps any natural number $i \geqslant i_0$ to the value

$$f(i) = \big(|u_1u_2...u_i| - |u|\big) \bmod |v|.$$

Since the image of f is finite, by the Pigeonhole Principle there exist two indices i, i', with $i_0 \leqslant i < i'$, such that $f(i) = f(i')$. By definition of f, we

have that the length of the substring $z = u_{i+1}...u_{i'}$ of w is a multiple of $|v|$. Since $i \geqslant i_0$, z is also a substring of v^ω, which implies that (i) z is a repeating pattern equivalent to v and (ii) $z \in V^+$. □

Proposition 6 *Given a language V, if V^ω is non-empty and it features only equivalent repeating patterns, then $V^\omega = \{v\}^\omega$ for a suitable non-empty finite word v.*

Proof Suppose that V^ω is a non-empty ω-language featuring only equivalent repeating patterns. Let $v_1, v_2, v_3, ...$ be all and only the V-aligned repeating patterns featured by V^ω.
We first prove that, for every pair of indices $i, j > 0$, $v_i^\omega = v_j^\omega$. Let $i, j > 0$ be two generic indices and let q_i and q_j be two positive natural numbers such that $q_i|v_i| = q_j|v_j|$. We define $v_i' = v_i^{q_i}$ and $v_j' = v_j^{q_j}$. By hypothesis, we have $v_i \cong v_j$, from which $v_i' \cong v_j'$ follows. Below, we prove that v_i' and v_j' coincide. Since $v_i' \cong v_j'$ and $|v_i'| = |v_j'|$, v_i' must be a rotation of v_j', namely, there exist two finite words x and y such that $v_i' = xy$ and $v_j' = yx$. Since both v_i' and v_j' belong to V^+, we have that $v_{i,j}' = v_i'v_j'$ belongs to V^+ and thus it is a repeating pattern for V^ω. Hence, by hypothesis, $v_{i,j}' \cong v_i'$. This implies that $v_{i,j}'$ $(= xyyx)$ is a rotation of $v_i'v_i'$ $(= xyxy)$ and hence there is a suitable (possibly empty) word z such that $(z)(xyyx)$ is a prefix of $(xy)(xy)(xy)$. Now, let us denote by u the primitive repeating pattern of xy. Since $(z)(xy)$ is a prefix of $(xy)(xy)(xy)$, we have that either $z = \varepsilon$ or $z^\omega = (xy)^\omega$. From the minimality of u, it follows that $z = u^p$ for some $p \geqslant 0$. Therefore, since $(zxy)(yx)$ is also a prefix of $(xy)(xy)(xy)$, $|zxy|$ is a multiple of $|u|$, and $|yx| = |xy|$, we have that $yx = u^q$ $(= xy)$ for some $q > 0$. This allows us to conclude that, for every pair of indices $i, j > 0$, v_i' $(= xy)$ $= v_j'$ $(= yx)$ and hence $v_i^\omega = v_j^\omega$.
Now, let v be the shortest repeating pattern of the infinite periodic word v_i^ω, where $i > 0$ is an arbitrary index (v does not depend on i). We have that $V^\omega = \{v\}^\omega$. □

Proposition 7 *Given a language V, if V^ω features at least two repeating patterns which are not equivalent, then V^ω contains an infinite word which is not ultimately periodic.*

Proof Let V^ω be an ω-language featuring two non-equivalent repeating patterns u and v. By Lemma 1, there exist two V-aligned repeating patterns $u', v' \in V^+$ such that $u' \cong u$ and $v'v$. For every $i > 0$, we denote by z_i the finite word $(u')^i(v')^i$ and we define the infinite word $w = z_1z_2....$ Clearly, $z_i \in V^+$ holds for all $i > 0$ and hence w belongs to V^ω. It remains to show that w is not an ultimately periodic word. Suppose, by way of contradiction, that w is an ultimately periodic word having z' as repeating pattern. By construction, there exists an index i such that $(u')^{|z'|}$ is a substring of z_i and thus of w. Since z' is a repeating pattern of w and $\left|(u')^{|z'|}\right|$ is a multiple of $|z'|$, we have that $(u')^{|z'|}$ is a repetition of some rotation of z' and hence $u' \cong z'$. In a similar way, we can show that $v' \cong z'$. By transitivity, we have $u' \cong v'$ and thus $u \cong v$, which is against the hypothesis of u and v being two non-equivalent repeating patterns. □

Proposition 8 *Given a language* V*, exactly one of the following conditions holds:*

1. V^ω *features only equivalent repeating patterns;*
2. V^ω *features infinitely many non-equivalent repeating patterns.*

Proof Let us assume that V^ω features at least two non-equivalent repeating patterns u and v. By Lemma 1, V^ω features two V-aligned repeating patterns u' and v', with $u' \cong u$ and $v' \cong v$. Moreover, since $\cong$ is an equivalence relation and $u \ncong v$, we have $u' \ncong v'$. Now, let $n = |u'| + |v'|$ and, for every $i > 0$, $p_i = n^{i-1}$ and $z_i = (u')^{p_i}(v')^{p_i}$. Every word z_i is clearly a V-aligned repeating pattern featured by V^ω. We prove that the words z_i are pairwise non-equivalent, that is, $z_i \ncong z_j$ for every pair of distinct indices $i, j > 0$.
Suppose, by way of contradiction, that there exist two indices $0 < i < j$ such that $z_i \cong z_j$. By definition of $\cong$, there exists an infinite periodic word w that features both z_i and z_j $(= (u')^{p_j}(v')^{p_j})$ as repeating patterns. Moreover, since $i < j$, we have that p_j $(= n^{j-1})$ is a multiple of $|z_i|$ $(= n^i)$, which implies that $(u')^{p_j}$ is a rotation of some repetition of z_i. This shows that $u' \cong z_i$. A similar argument shows that $v' \cong z_i$. Thus, by transitivity, we obtain $u' \cong v'$, which contradicts the hypothesis of u' and v' being non-equivalent repeating patterns. □

The following theorem shows how the above characterization results can be easily generalized to the whole class of regular ω-languages consisting of ultimately periodic words only.

Theorem 1 *Given a regular* ω*-language* L*, the following conditions are equivalent:*

i) L *consists of ultimately periodic words only;*

ii) L *features only finitely many non-equivalent repeating patterns;*

iii) L *is a finite union of* ω*-languages of the form* $U\{v\}^\omega$*, where* U *is a regular language and* v *is a non-empty finite word.*

Proof We first prove the implication from i) to ii) by contraposition. Let L be a regular ω-language. We can write L as a finite union of the form $\bigcup_{1 \leqslant i \leqslant n} U_i V_i^\omega$. If L features infinitely many non-equivalent repeating patterns, then there exists an index $1 \leqslant i \leqslant n$ such that $U_i V_i^\omega$ (and hence V_i^ω) features at least two non-equivalent repeating patterns. Thus, by Proposition 7, it would follow that V_i^ω (and hence L) contains an infinite word which is not ultimately periodic.
As for the implication from ii) to iii), let L be a regular ω-language featuring only finitely many non-equivalent repeating patterns. By Proposition 2, we can write L as a finite union of the form $\bigcup_{1 \leqslant i \leqslant n} U_i V_i^\omega$. Moreover, from Proposition 8, it follows that each ω-language V_i^ω features only equivalent repeating patterns (otherwise, L would feature infinitely many non-equivalent repeating patterns). Then, by exploiting Proposition 6, we can write each ω-language V_i^ω as $\{v_i\}^\omega$, where v_i is a suitable non-empty finite word. As a consequence, L can be written as a finite union of the form $\bigcup_{1 \leqslant i \leqslant n} U_i \{v_i\}^\omega$.
The last implication from iii) to i) is trivial. □

4 Ultimately periodic automata

In this section, we provide an automata counterpart to regular ω-languages of ultimately periodic words. Theorem 1 basically states that these languages model sets of ultimately periodic words with possibly infinitely many initial patterns, but only a finite number of non-equivalent repeating patterns. Moreover, it yields a straightforward definition of a restricted class of Büchi automata that captures exactly the regular ω-languages of ultimately periodic words. As a matter of fact, an alternative view of such a class of automata is also possible: they can be seen as a natural extension of non-deterministic finite state automata (NFA for short), where final states actually recognize infinite words of the form ν^ω. This alternative view will clearly show up in the definition of prefix automaton given in Section 4.2.

As a preliminary step, we introduce the notion of strongly connected component of a state of an automaton $\mathcal{A} = (\Sigma, S, \Delta, \mathcal{I}, \mathcal{F})$. Let us view $\mathcal{A}$ as a finite labeled graph. The *strongly connected component* of a state $s \in S$ is the subgraph of $\mathcal{A}$ induced by the maximal set of states M_s that are reachable from s and from which s is reachable, that is, M_s consists of all and only the states s' in S such that both (s, s') and (s', s) belong to Δ^*. A state $s \in S$ is called *transient* if $(s, s) \notin \Delta^+$ (notice that it immediately follows that a transient state does not belong to any loop of $\mathcal{A}$). Let us consider the following subclass of Büchi automata (for the sake of simplicity, we assume every state of the automaton to be reachable from any initial state).

Definition 3 An *ultimately periodic automaton* (UPA for short) is a Büchi automaton $\mathcal{A} = (\Sigma, S, \Delta, \mathcal{I}, \mathcal{F})$ such that, for every final state $s \in \mathcal{F}$, either s is a transient state or the strongly connected component of s is a simple loop[2].

Notice that Definition 3 does not prevent non-transient final states from having in-degree or out-degree greater than 1 in (the finite labeled graph corresponding to) $\mathcal{A}$.

Examples of UPA are given in Figure 2, with reference to the alphabet for time granularity introduced in Section 2 which consists of the three symbols ■, □, and ◄. The UPA to the left recognizes the ω-language $\left(\{\square\}^*\{\blacksquare\blacktriangleleft\}^\omega\right) \cup \left(\{\square\}^*\{\blacksquare\blacktriangleleft\}^*\{\square\}^\omega\right)$ while that to the right recognizes the ω-language $\{\blacksquare\blacksquare\blacktriangleleft\}^\omega \cup \{\blacksquare\blacktriangleleft\}^\omega$. The former represents the (unanchored finite or infinite) granularities that group days two by two, while the latter represents the set consisting of two infinite granularities that group days respectively two by two and three by three.

By exploiting standard construction methods for Büchi automata, one can easily show that UPA-recognizable languages are effectively closed under unions, intersections with regular ω-languages, left-concatenations with regular languages, generalized products, and homomorphisms (i.e., substitutions of non-empty strings for symbols):

i) if L_1 and L_2 are two UPA-recognizable ω-languages, then $L_1 \cup L_2$ is an UPA-recognizable ω-language as well;

[2] A strongly connected component is said to be a simple loop if and only if all its vertices have both in-degree and out-degree equal to 1.

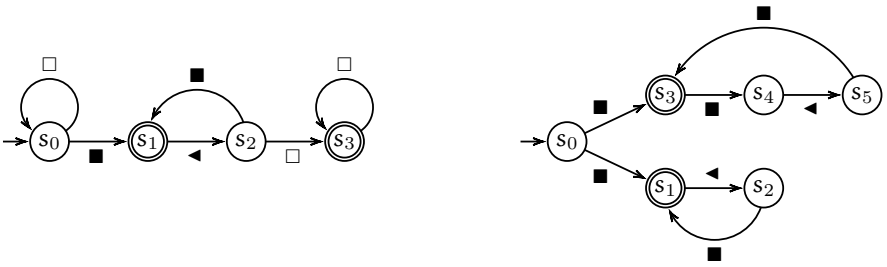

Fig. 2 Two examples of UPA.

ii) if L_1 is a regular ω-language and L_2 is an UPA-recognizable ω-language, then $L_1 \cap L_2$ is an UPA-recognizable ω-language as well;

iii) if L_1 and L_2 are two UPA-recognizable ω-languages, then the ω-language $L_1 \times L_2 = \{w \ : \ \exists\, u \in L_1.\ \exists\, v \in L_2.\ \forall\, i > 0.\ w(i) = (u(i), v(i))\}$ is UPA-recognizable as well;

iv) if L_1 is a regular language and L_2 is an UPA-recognizable ω-language, then $L_1 L_2$ is an UPA-recognizable ω-language as well;

v) if L is an UPA-recognizable ω-language and τ is a function from Σ to Σ^+, then the ω-language $\tau(L) = \{\tau(a_1)\tau(a_2)... \ : \ a_1 a_2... \in L\}$ is an UPA-recognizable ω-language as well.

In addition, it is easy to see that UPA satisfy a weak form of closure under ω-exponentiation, namely, for every non-empty finite word v, there exists an UPA recognizing the singleton ω-language $\{v\}^\omega$. On the other hand, UPA-recognizable languages are not closed under complementation: this is an immediate consequence of Proposition 5 and Theorem 2 below, which characterizes UPA-recognizable languages. Moreover, the *deterministic* versions of UPA are strictly less expressive than the non-deterministic ones: as it is well-known, the UPA-recognizable ω-language $\{a, b\}^*\{b\}^\omega$ is not recognizable by any deterministic Büchi automaton (and thus by any deterministic UPA).

Theorem 2 *UPA recognize exactly the regular ω-languages of ultimately periodic words.*

Proof Let $\mathcal{A} = (\Sigma, S, \Delta, \mathcal{I}, \mathcal{F})$ be an UPA, w be an infinite word accepted by $\mathcal{A}$, and ρ be a successful run of $\mathcal{A}$ on w. We denote by s a final state of $\mathcal{A}$ that occurs infinitely often in ρ. Clearly, s is not a transient state and hence, by definition of UPA, its strongly connected component is a simple loop. Thus, there is a *unique* infinite run ρ' of $\mathcal{A}$ that starts from s and visits s infinitely often. Such a run is a suffix of ρ of the form $\rho' = (\rho(i)\rho(i+1)...\rho(j-1))^\omega$, where i and j are the positions of two consecutive occurrences of s in ρ. This proves that ρ, and hence w, are ultimately periodic sequences.
As for the converse implication, we have to show that, given a regular ω-language L of ultimately periodic words, there is an UPA recognizing L. By exploiting Theorem 1, we know that $L = \bigcup_{1 \leqslant i \leqslant n} U_i \{v_i\}^\omega$ for a suitable n, suitable regular languages $U_1, ..., U_n$, and suitable non-empty finite words

$\nu_1, ..., \nu_n$. Such a characterization implicitly defines the three basic operations on UPA: the ω-exponentiation of a non-empty finite word, the concatenation with a regular language, and the finite union. Thus, from closure properties of UPA, it follows that there exists an UPA recognizing L. □

In the following subsections, we will introduce three normalized forms for UPA, which we will respectively call normal form, prefix-friendly form, and canonical form. We will prove that these normalized forms satisfy several desirable properties (e.g., the canonical form is proved to be unique, up to isomorphisms, among all equivalent UPA) and ease algorithmic manipulation. We will also prove that normal and prefix-friendly forms can be computed at low cost (precisely, the former can be computed in linear time and the latter can be computed in quadratic time with respect to the input UPA).

4.1 A normal form for UPA

Given a loop C of an UPA $\mathcal{A} = (\Sigma, S, \Delta, \mathcal{I}, \mathcal{F})$, we say that C is a *final* loop if it contains at least one final state. Moreover, we say that a final loop C *encodes* the repeating pattern $\nu \neq \varepsilon$ if and only if there is a run ρ of $\mathcal{A}$ on ν^ω that starts with a state $s \in C$. It is easy to see that a final loop C encodes only equivalent repeating patterns and, conversely, if ν and ν' are equivalent repeating patterns, then C encodes ν iff C encodes ν'. Thus, given two final loops C_1 and C_2, either C_1 and C_2 encode the same repeating patterns, or C_1 and C_2 encode repeating patterns which are pairwise non-equivalent.

Due to the peculiar structure of UPA, every successful run of an UPA consists of a finite prefix followed by an infinite repetition of a final loop. In particular, given a final loop C, the number and the positions of the final states of C are irrelevant (C encodes the same set of repeating patterns, independently from which states of C are chosen to be final). Similarly, marking a transient state as final state has no effect, since in any run of the automaton it occurs at most once. Finally, we can assume that no transitions exit from final loops (if this were the case, we could simply duplicate the final loop and let one copy of it to be final with no exiting transition and the other copy to be non-final with some exiting transitions). Putting together the above observations, we can obtain a normal form for UPA, which forces final states to coincide with the states of the final loops and forbids transitions exiting from final loops.

Definition 4 An UPA $\mathcal{A} = (\Sigma, S, \Delta, \mathcal{I}, \mathcal{F})$ is said to be in *normal form* if the following conditions hold:

- every final state is reachable from an initial state,
- every final state belongs to a (final) loop,
- every state in a final loop is final,
- there are no transitions exiting from final loops, namely, for every $(r, a, s) \in \Delta$, $r \in \mathcal{F}$ implies $s \in \mathcal{F}$.

By restricting to UPA in normal form, one can easily distinguish between components recognizing initial patterns and components recognizing repeating patterns of ultimately periodic words (note that the former components behave like NFA, while the latter ones behave like single-string automata [9]). The following proposition proves that there is no loss of expressiveness if we restrict ourselves to UPA in normal form.

Proposition 9 *Given an UPA $\mathcal{A}$, one can compute in time $\mathcal{O}(|\mathcal{A}|)$ an equivalent UPA $\mathcal{B}$ in normal form. Moreover, $|\mathcal{B}|$ is linear in $|\mathcal{A}|$.*

Proof Let $\mathcal{A} = (\Sigma, S, \Delta, \mathcal{I}, \mathcal{F})$ and $C_1, ..., C_k$ be all and only the final loops of $\mathcal{A}$. By definition of UPA, $C_1, ..., C_k$ are disjoint subgraphs of S. For every $1 \leqslant i \leqslant k$, we introduce a copy $\widetilde{C}_i$ of each final loop C_i and, for every state s of C_i, we denote by $\widetilde{s}$ the corresponding state of $\widetilde{C}_i$. We then define $\mathcal{B} = (\Sigma, S', \Delta', \mathcal{I}', \mathcal{F}')$ as follows:

- $S' = S \cup \bigcup_{1 \leqslant i \leqslant k} \{\widetilde{s} : s \in C_i\}$;
- Δ' contains all triples of the form
  1. (r, a, s), with $(r, a, s) \in \Delta$,
  2. $(r, a, \widetilde{s})$, with $(r, a, s) \in \Delta$, $r \notin C_i$, $s \in C_i$, and $1 \leqslant i \leqslant k$,
  3. $(\widetilde{r}, a, \widetilde{s})$, with $(r, a, s) \in \Delta$, $r, s \in C_i$, and $1 \leqslant i \leqslant k$;
- $\mathcal{I}' = \mathcal{I} \cup \bigcup_{1 \leqslant i \leqslant k} \{\widetilde{s} : s \in \mathcal{I}, s \in C_i\}$;
- $\mathcal{F}' = \bigcup_{1 \leqslant i \leqslant k} \{\widetilde{s} : s \in C_i\}$.

It can be easily checked that $\mathcal{B}$ is an UPA in normal form equivalent to $\mathcal{A}$. □

On the grounds of Proposition 9, one can devise a simple linear-time algorithm that receives a generic UPA as input and returns an equivalent UPA in normal form as its output.

4.2 A prefix-friendly form for UPA

In the following, we introduce an alternative way of representing regular ω-languages of ultimately periodic words as NFA over an extended alphabet. By definition, any UPA $\mathcal{A} = (\Sigma, S, \Delta, \mathcal{I}, \mathcal{F})$ in normal form contains only finitely many (pairwise disjoint) final loops, say $C_1, ..., C_k$. Hereafter, we denote by $\Sigma_{\mathcal{A}}$ an extended alphabet, which consists of symbols from Σ plus symbols of the form (s, C_i), with $1 \leqslant i \leqslant k$ and s being a state of C_i. Intuitively, an NFA representing $\mathcal{A}$ can be obtained by (i) adding a new global final state f, (ii) removing every transition departing from a final state, and (iii) adding a (s, C_i)-labeled transition from s to f for each state s belonging to the final loop C_i.

Definition 5 Given an UPA $\mathcal{A} = (\Sigma, S, \Delta, \mathcal{I}, \mathcal{F})$ in normal form, we define the *prefix automaton* of $\mathcal{A}$ as the NFA $\mathcal{A}_{pre} = (\Sigma_{\mathcal{A}}, S', \Delta', \mathcal{I}', \mathcal{F}')$, where

- $S' = S \cup \{f\}$, with f being a fresh state not belonging to S;
- Δ' contains all triples of the form

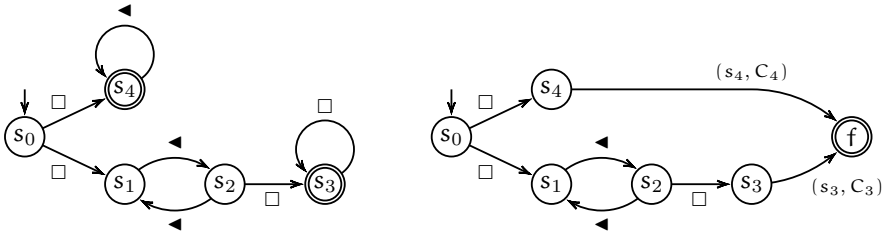

Fig. 3 An UPA in normal form and its prefix automaton.

1. (q, a, s), with $(q, a, s) \in \Delta$, $q \in S \setminus \mathcal{F}$, and $s \in S$,
2. (s, b, f), with $b = (s, C_i)$, $s \in C_i$, and $1 \leqslant i \leqslant k$;

- $\mathcal{I}' = \mathcal{I}$;
- $\mathcal{F}' = \{f\}$.

As an example, Figure 3 depicts an UPA $\mathcal{A}$ in normal form, which recognizes the language $\{\Box(\blacktriangleleft\blacktriangleleft)^n\blacktriangleleft\Box\Box^\omega : n \in \mathbb{N}\} \cup \{\Box\blacktriangleleft^\omega\}$, together with its prefix automaton $\mathcal{A}_{pre}$, which recognizes the language $\{\Box(\blacktriangleleft\blacktriangleleft)^n\Box\, b_3 : n \in \mathbb{N}\} \cup \{\Box\, b_4\}$, where $b_3 = (s_3, C_3)$, $b_4 = (s_4, C_4)$, C_3 is the final loop of s_3, and C_4 is the final loop of s_4.

Notice that the prefix automaton $\mathcal{A}_{pre}$ uniquely identifies the UPA $\mathcal{A}$, that is, one can obtain $\mathcal{A}$ from $\mathcal{A}_{pre}$ by (i) marking as final all states in $C_1, ..., C_k$, (ii) adding the transitions of $C_1, ..., C_k$, which can be recovered from the symbols belonging to extended alphabet $\Sigma_{\mathcal{A}}$, and (ii) removing the global final state f together with its entering transitions. This basically means that the NFA $\mathcal{A}_{pre}$ is nothing but an alternative representation of $\mathcal{A}$.

For the sake of brevity, given two states r, s of $\mathcal{A}$ and a finite word u, we write $r \xrightarrow{u} s$ whenever there exists a run of $\mathcal{A}$ on u that starts with r and ends with s. Similarly, we write $r \overset{u}{\circledcirc\!\!\rightarrow} s$ (resp., $r \overset{u}{\otimes\!\!\rightarrow} s$) whenever there exists a run of $\mathcal{A}$ on u that starts with r, ends with s, and traverses at least one final state of $\mathcal{A}$ (resp., no final states of $\mathcal{A}$). It is easy to see that the prefix automaton $\mathcal{A}_{pre}$ recognizes the language

$$\{ub : \exists\, 1 \leqslant i \leqslant k.\ \exists\, s_0 \in \mathcal{I}.\ \exists\, s \in C_i.\ b = (s, C_i),\ s_0 \overset{u}{\otimes\!\!\rightarrow} s\},$$

which is called the *prefix language* of $\mathcal{A}$.

The correspondence between UPA (in normal form) and prefix automata does not lift directly to the language level: the prefix languages of two UPA $\mathcal{A}$ and $\mathcal{A}'$ may be different even in the case in which $\mathscr{L}^\omega(\mathcal{A}) = \mathscr{L}^\omega(\mathcal{A}')$. As an example, Figure 4 depicts an UPA $\mathcal{A}'$, which is equivalent to the UPA of Figure 3, and its prefix automaton $\mathcal{A}'_{pre}$, which is not equivalent to the prefix automaton $\mathcal{A}_{pre}$ of Figure 3. To get rid of such an asymmetry, that is, to guarantee that $\mathscr{L}(\mathcal{A}_{pre}) = \mathscr{L}(\mathcal{A}'_{pre})$ if and only if $\mathscr{L}^\omega(\mathcal{A}) = \mathscr{L}^\omega(\mathcal{A}')$, we must impose suitable conditions on the structure of the transition relations of $\mathcal{A}$ and $\mathcal{A}'$.

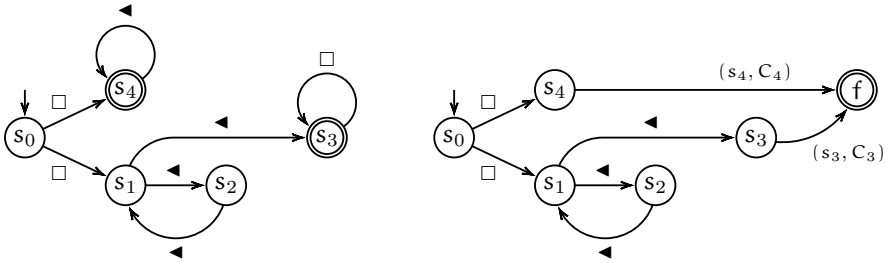

Fig. 4 Equivalence of UPA does not transfer to their prefix automata.

Definition 6 An UPA $\mathcal{A} = (\Sigma, S, \Delta, \mathcal{I}, \mathcal{F})$ is said to be *prefix-friendly* if it satisfies the following conditions:

(C1) $\mathcal{A}$ is in normal form,

(C2) every final loop C of $\mathcal{A}$ has the minimum number of states (among all loops that encode the same set of repeating patterns),

(C3) $\mathcal{A}$ has the minimum number of final loops (among all equivalent UPA),

(C4) there are no pairs of transitions of the form (q, a, s) and (r, a, s), with $q \in S \setminus \mathcal{F}$ and $r, s \in \mathcal{F}$.

Figure 3 and Figure 4 respectively show an *prefix-friendly* UPA and a *prefix-friendly* (equivalent) one.

Lemma 2 *Final loops of a prefix-friendly UPA are pairwise non-isomorphic.*

Proof This trivially follows from the minimality of the number of final loops. □

Lemma 3 *If $\mathcal{A}$ and $\mathcal{A}'$ are equivalent prefix-friendly UPA, then $\mathcal{A}$ and $\mathcal{A}'$ have isomorphic final loops.*

Proof Let C be a final loop of $\mathcal{A}$. Since all states of $\mathcal{A}$ are reachable from initial states and since $\mathcal{A}$ has the minimum number of final loops, there exists a word $w \in \mathscr{L}^\omega(\mathcal{A})$ that features all and only the repeating patterns encoded by C. Moreover, since $\mathcal{A}'$ is equivalent to $\mathcal{A}$, we have $w \in \mathscr{L}^\omega(\mathcal{A}')$ and hence there is a final loop C' in $\mathcal{A}'$ that encodes all and only the repeating patterns featured by w. From this, it easily follows that C and C' are bisimilar loops. Finally, since both C and C' have the minimum number of states, it immediately follows that C and C' are isomorphic. □

In virtue of Lemma 2 and Lemma 3, given two equivalent prefix-friendly UPA $\mathcal{A}$ and $\mathcal{A}'$, we can identify the symbols of the alphabet $\Sigma_{\mathcal{A}}$ and those of the alphabet $\Sigma_{\mathcal{A}'}$. Formally, we say that two symbols $b \in \Sigma_{\mathcal{A}}$ and $b' \in \Sigma_{\mathcal{A}'}$ *coincide* iff

1. either they both belong to Σ and $b = b'$,

2. or $\mathfrak{b} = (\mathsf{s}, \mathsf{C})$ and $\mathfrak{b}' = (\mathsf{s}', \mathsf{C}')$, where C and C' are isomorphic final loops of $\mathcal{A}$ and $\mathcal{A}'$, respectively, and s is the state of C that corresponds to s' in C' under the unique[3] isomorphism between C and C'.

The above correspondence can be naturally lifted to languages over $\Sigma_{\mathcal{A}}$ and $\Sigma_{\mathcal{A}'}$: we say that the two prefix automata $\mathcal{A}_{pre}$ and $\mathcal{A}'_{pre}$ are equivalent iff $\mathscr{L}(\mathcal{A}_{pre}) = \mathscr{L}(\mathcal{A}'_{pre})$, that is, for every finite word $\mathfrak{u} \in \mathscr{L}(\mathcal{A}_{pre})$ (resp., $\mathfrak{u}' \in \mathscr{L}(\mathcal{A}'_{pre})$), there is a finite word $\mathfrak{u}' \in \mathscr{L}(\mathcal{A}'_{pre})$ (resp., $\mathfrak{u} \in \mathscr{L}(\mathcal{A}_{pre})$) such that $|\mathfrak{u}| = |\mathfrak{u}'|$ and, for all $1 \leqslant \mathfrak{i} \leqslant |\mathfrak{u}|$, the symbols $\mathfrak{u}(\mathfrak{i})$ and $\mathfrak{u}'(\mathfrak{i})$ coincide.

Lemma 4 *Given a prefix-friendly UPA $\mathcal{A}$ over the alphabet Σ, its prefix automaton $\mathcal{A}_{pre}$ recognizes all and only the finite words of the form $\mathfrak{u}\mathfrak{b}$, with $\mathfrak{b} \in \Sigma_{\mathcal{A}} \backslash \Sigma$ and $\mathfrak{u}$ being the shortest initial pattern of some word $\mathfrak{w} \in \mathscr{L}^{\omega}(\mathcal{A})$.*

Proof Let $\mathcal{A} = (\Sigma, \mathsf{S}, \Delta, \mathcal{I}, \mathcal{F})$. Suppose that the prefix automaton $\mathcal{A}_{pre}$ accepts a finite word of the form $\mathfrak{u}\mathfrak{b}$, with $\mathfrak{u} \in \Sigma^*$ and $\mathfrak{b} = (\mathsf{s}, \mathsf{C})$. From the definition of prefix automaton, we know that there is a run ρ of $\mathcal{A}$ on $\mathfrak{u}$ that starts from an initial state, traverses only non-final states, and finally enters the final loop C at state s. Let ν be a repeating pattern encoded by C starting from s. Note that $\mathcal{A}$ accepts the ultimately periodic word $\mathfrak{w} = \mathfrak{u}\nu^{\omega}$. We have to show that $\mathfrak{u}$ is the shortest initial pattern of $\mathfrak{w}$, namely, that the last symbol $\mathfrak{u}(|\mathfrak{u}|)$ of $\mathfrak{u}$ differs from the last symbol $\nu(|\nu|)$ of ν. Let q be the (non-final) state that immediately precedes s inside the run ρ and let r be the (final) state that immediately precedes s inside the loop C. Clearly, we have that $\big(\mathsf{q}, \mathfrak{u}(|\mathfrak{u}|), \mathsf{s}\big) \in \Delta$. Moreover, since ν is encoded by C starting from s, we have that $\mathsf{t}\ \mathsf{s} \xrightarrow{\nu}\!\!\!\!\circ\!\!\!\!\rightarrow\ \mathsf{s}$. This shows that $\big(\mathsf{r}, \nu(|\nu|), \mathsf{s}\big) \in \Delta$. Finally, since $\mathcal{A}$ satisfies Condition C4 of Definition 6, we obtain $\mathfrak{u}(|\mathfrak{u}|) \neq \nu(|\nu|)$.
As for the converse direction, let $\mathfrak{w}$ be an ultimately periodic word accepted by $\mathcal{A}$ and let ρ be a successful run of $\mathcal{A}$ on $\mathfrak{w}$. We denote by $\mathfrak{i}$ the position of the first final state s that occurs in ρ and by $\mathfrak{j}$ the position of the second occurrence of s in ρ. We then denote by $\mathfrak{u}$ and ν, respectively, the substrings $\mathfrak{w}[1, \mathfrak{i}-1]$ and $\mathfrak{w}[\mathfrak{i}, \mathfrak{j}-1]$ of $\mathfrak{w}$. By definition of prefix automaton, the sequence $\rho(1)...\rho(\mathfrak{i})\mathsf{f}$, where f is the global final state of $\mathcal{A}_{pre}$, is a successful run of $\mathcal{A}_{pre}$ on the finite word $\mathfrak{u}\mathfrak{b}$, where $\mathfrak{b} = (\mathsf{s}, \mathsf{C})$ and C is the (unique) final loop of $\mathcal{A}$ that contains s. This shows that $\mathfrak{u}\mathfrak{b}$ is accepted by $\mathcal{A}_{pre}$. It remains to prove that $\mathfrak{u}$ is the shortest initial pattern of $\mathfrak{w}$. Let $\mathsf{q} = \rho(\mathfrak{i}-1)$ and $\mathsf{r} = \rho(\mathfrak{j}-1)$. Since, $\mathcal{A}$ satisfies Condition C4 of Definition 6 and, by construction, $\mathsf{q} \in \mathsf{S} \backslash \mathcal{F}$, $\mathsf{r}, \mathsf{s} \in \mathcal{F}$, and $(\mathsf{q}, \mathfrak{u}(|\mathfrak{u}|), \mathsf{s})$, $(\mathsf{r}, \nu(|\nu|), \mathsf{s}) \in \Delta$, we have that $\mathfrak{u}(|\mathfrak{u}|) \neq \nu(|\nu|)$. This shows that $\mathfrak{u}$ is the shortest initial pattern of $\mathfrak{w}$. □

The following theorem proves that equivalent prefix-friendly UPA have equivalent corresponding prefix automata.

[3] Given two loops C and C' with the minimum number of states, there exists at most one isomorphism between C and C', since otherwise, by transitivity, there would exist a non-trivial endomorphism in C, thus contradicting the minimality of C.

Theorem 3 *Let $\mathcal{A}$ and $\mathcal{A}'$ be two prefix-friendly UPA. We have that*

$$\mathscr{L}^{\omega}(\mathcal{A}) = \mathscr{L}^{\omega}(\mathcal{A}') \quad \Leftrightarrow \quad \mathscr{L}(\mathcal{A}_{pre}) = \mathscr{L}(\mathcal{A}'_{pre}).$$

Proof Every ultimately periodic word has a unique shortest initial pattern. Therefore, by Lemma 4, the ω-language recognized by $\mathcal{A}$ (resp., $\mathcal{A}'$) is uniquely determined by the language recognized by $\mathcal{A}_{pre}$ (resp., $\mathcal{A}'_{pre}$) and, vice versa, the language recognized by $\mathcal{A}_{pre}$ (resp., $\mathcal{A}'_{pre}$) is uniquely determined by the ω-language recognized by $\mathcal{A}$ (resp., $\mathcal{A}'$). □

Given an UPA $\mathcal{A}$ in normal form, one can efficiently build an equivalent prefix-friendly UPA $\mathcal{B}$ by applying the following sequence of normalization steps:

i) **Minimize the size of each final loop**. Such an operation collapses all equivalent states in each final loop, thus producing an UPA that satisfies Conditions C1, C2 of Definition 6.

ii) **Minimize the number of final loops**. Such an operation collapses all isomorphic (minimal) final loops, thus producing an UPA that satisfies Conditions C1–C3 of Definition 6.

iii) **Add shortcuts towards final loops**. Such an operation produces an UPA that satisfies all conditions of Definition 6.

The above normalizations steps can be implemented as follows. As a preliminary remark, we note that the minimization procedures for the size and the number of final loops can be viewed as particular cases of the solution to the *single function coarsest partition problem*. We thus refer the reader to [29] for further details and proofs.

Let $\mathcal{A} = (\Sigma, S, \Delta, \mathcal{I}, \mathcal{F})$ be an UPA in normal form and let us consider a final loop C of $\mathcal{A}$. Two states s, s' of C are said to be *equivalent* if we have $s \xrightarrow{\nu} s$ and $s' \xrightarrow{\nu} s'$, for some finite word ν. Minimizing the size of the final loop C amounts to collapse all equivalent states of C. Thus, suppose that C is a final loop of the form

$$s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \dots \xrightarrow{a_{n-1}} s_n \xrightarrow{a_n} s_1$$

and let ν be the repeating pattern $a_1 a_2 \dots a_n$. One can easily verify that two states s_i and s_j, with $1 \leqslant i \leqslant j \leqslant n$, are equivalent iff the value $j - i \bmod n$ is the offset of an occurrence of ν as a substring of $\nu\nu$. Thus, the equivalence class $[s_i]$ of a state s_i of C is given by the set $\{s_j : 1 \leqslant j \leqslant n,\ j \equiv i \pmod p\}$, where p denotes the offset of the *first non-trivial* occurrence of ν as a substring of $\nu\nu$ (note that the value p can be efficiently computed in linear time using Knuth-Morris-Pratt string matching algorithm [18]).

Finally, the operation of collapsing equivalence classes of C into single states can be implemented by first replacing the final loop C with a new final loop $[C]$ of the form

$$[s_1] \xrightarrow{a_1} [s_2] \xrightarrow{a_2} \dots \xrightarrow{a_{p-1}} [s_p] \xrightarrow{a_p} [s_1]$$

and then replacing every transition of $\mathcal{A}$ of the form (q, a, s_i), where $q \notin \mathcal{F}$, by the triple $\big(q, a, [s_i]\big)$. On the ground of the above arguments, it is easy to devise a linear time procedure that minimizes the size of each final loop of a given UPA $\mathcal{A}$ in normal form.

As for the minimization of the number of final loops, this amounts to collapse all isomorphic final loops of $\mathcal{A}$, under the assumption that $\mathcal{A}$ is an UPA that satisfies Conditions C1, C2 of Definition 6. Indeed, two final final loops C and C' of an UPA encode the same set of repeating patters iff they are bisimilar. Moreover, if C and C' have the minimum number of states, then they are bisimilar iff they are isomorphic.

Isomorphic final loops of $\mathcal{A}$ can be efficiently found by (i) defining a total ordering on the alphabet Σ, (ii) representing each final loop C of $\mathcal{A}$ with the *lexicographically least primitive repeating pattern* v_C encoded by C, and (iii) sorting the loops according to the lexicographic order of their representatives (in this way, isomorphic final loops have the same representatives and hence they appear contiguous in the ordered list).

We recall that, given a final loop C, the lexicographically least primitive repeating pattern v_C of C can be computed in linear time by using the algorithms described in [3, 34]. Moreover, sorting the loops according to the lexicographic order of their representatives can be done in linear time using the well-known radix-sort algorithm. This shows that the minimization of the number of final loops of a given UPA can be done in linear time.

The last normalization step consists in the removal of redundant transitions of $\mathcal{A}$ and in the addition of shortcuts towards their target states. Let $\mathcal{A} = (\Sigma, S, \Delta, \mathcal{I}, \mathcal{F})$ be an UPA that satisfies Conditions C1–C3 of Definition 6. We say that a transition (q, a, s) of $\mathcal{A}$ is *redundant* with respect to another transition (r, a', s'), if $q \in S \setminus \mathcal{F}$, $r \in \mathcal{F}$, $s = s' \in \mathcal{F}$, and $a = a'$ (notice that the UPA that satisfy Condition C4 of Definition 6 are exactly those UPA which contain no redundant transitions). The addition of shortcuts and the removal of redundant transitions are implemented in two phases as follows.

The first phase iteratively performs the following steps: (i) select two transition of $\mathcal{A}$ of the form (q, a, s) and (r, a, s), with (q, a, s) being redundant with respect to (r, a, s), (ii) mark the transition (q, a, s) so that it cannot be selected again, (iii) add a new transition (t, b, r) (if it does not exist already) for each existing transition (t, b, q), with $t \in S \setminus \mathcal{F}$, and (iv) mark r as a new initial state whenever q is an initial state.

The second phase starts when there are no more redundant (unmarked) transitions and it consists in the removal of previously-marked redundant transitions. Notice that it is necessary to postpone the removal of the redundant transitions to this second phase, since, otherwise, the algorithm may enter an infinite loop where a single transition is alternatively added to and removed from the automaton.

One can easily verify that the resulting automaton is a prefix-friendly UPA equivalent to the input automaton $\mathcal{A}$. In particular, the following invariant holds during the first phase: for every successful run on an infinite word w, there is a successful run on w that uses only unmarked transitions. Moreover, the whole process can be implemented by a procedure which takes quadratic time in the size of the input automaton $\mathcal{A}$.

Proposition 10 *Given an UPA $\mathcal{A}$ in normal form, one can compute in time $\mathcal{O}(|\mathcal{A}|^2)$ an equivalent prefix-friendly UPA $\mathcal{B}$. Moreover, $|\mathcal{B}|$ is at most quadratic in $|\mathcal{A}|$ and the number of states of $\mathcal{B}$ is less than or equal to the number of states of $\mathcal{A}$.*

Proof By applying the proposed sequence of normalization steps to a given UPA $\mathcal{A}$ in normal form, one obtains an equivalent prefix-friendly UPA $\mathcal{B}$. All normalization steps, but the last one, can be computed by suitable linear-time algorithms and the intermediate results are UPA with size (resp., number of states) less than or equal to the size (resp., number of states) of the original UPA $\mathcal{A}$. As for the last normalization step, the resulting prefix-friendly UPA $\mathcal{B}$ is obtained from an intermediate UPA $\mathcal{A}'$ by simply adding new transitions and removing redundant ones. This shows that the the size of $\mathcal{B}$ is at most quadratic in the size of the original UPA $\mathcal{A}$ and, similarly, that the number of states of $\mathcal{B}$ is less than or equal to the number of states of $\mathcal{A}$. □

4.3 The canonical form for UPA

We conclude the section by introducing a canonical form for UPA, , that is, a representation of regular ω-language of ultimately periodic words which turns out to be unique up to isomorphisms.

Definition 7 An UPA $\mathcal{A} = (\Sigma, \mathsf{S}, \Delta, \mathcal{I}, \mathcal{F})$ is said to be in *canonical form* if $\mathcal{A}$ is prefix-friendly and, in addition, the prefix automaton $\mathcal{A}_{pre}$ of $\mathcal{A}$ is a deterministic finite state automaton (DFA for short) having the minimum number of states (among all equivalent DFA).

As a matter of fact, an UPA in canonical form may be exponentially larger than an equivalent UPA in prefix-friendly form (this basically follows from the fact that DFA may be exponentially larger than equivalent NFA [15]).

The following theorem shows that the canonical form of an UPA is unique, up to isomorphisms, among all equivalent UPA.

Theorem 4 *Let $\mathcal{A}$ and $\mathcal{A}'$ be two UPA in canonical form. We have that $\mathcal{A}$ and $\mathcal{A}'$ are equivalent iff they are isomorphic.*

Proof By Theorem 3, the prefix-friendly UPA $\mathcal{A}$ and $\mathcal{A}'$ are equivalent iff the corresponding prefix automata $\mathcal{A}_{pre}$ and $\mathcal{A}'_{pre}$ are equivalent. Moroever, since $\mathcal{A}$ and $\mathcal{A}'$ are in canonical form, $\mathcal{A}_{pre}$ and $\mathcal{A}'_{pre}$ are DFA having the minimum number of states. Hence they are equivalent iff they are isomorphic. Finally, since the UPA $\mathcal{A}$ (resp., $\mathcal{A}'$) is uniquely determined by its prefix automaton $\mathcal{A}_{pre}$ (resp., $\mathcal{A}'_{pre}$), we can conclude that $\mathcal{A}$ and $\mathcal{A}'$ are equivalent iff they are isomorphic. □

Let us show now how to compute the canonical form of a given prefix-friendly UPA $\mathcal{A}$. As a preliminary remark, observe that any transformation of an UPA $\mathcal{A}$ that preserves the corresponding prefix language $\mathscr{L}(\mathcal{A}_{pre})$ results in an UPA $\mathcal{A}'$ which has the same structure of $\mathcal{A}$ with respect to the

transitions towards final states (intuitively, such a property follows from the fact that the final loops of $\mathcal{A}$ are "hidden" inside the alphabet of the prefix automaton). In particular, it follows that $\mathcal{A}'$ is prefix-friendly whenever $\mathcal{A}$ is prefix-friendly. Given such a property, it becomes clear that the canonical form of a prefix-friendly UPA $\mathcal{A}$ can be obtained by applying the following sequence of transformations:

i) **Compute the prefix automaton**. Such an operation produces an NFA that implicitly represents the original prefix-friendly UPA.

ii) **Compute the minimal equivalent DFA**. Such an operation produces a DFA that has the minimum number of states and that recognizes the prefix language of the original UPA.

iii) **Convert the minimal DFA back to an UPA**. Such an operation eventually produces an UPA in canonical form.

Note that all transformations, but the second one, can be computed at low cost. On the other hand, the second transformation is the most demanding one from the computational point of view, since, in the general case, deterministic prefix automata may be exponentially larger than equivalent nondeterministic prefix ones. From experimental comparisons [7], it turns out that Brzozowski's algorithm [5] is the most efficient solution to the problem of determinizing and minimizing prefix automata.

For the sake of completeness, we provide a short description of Brzozowski's algorithm. Given a generic NFA $\mathcal{A}'$, the algorithm first reverses the transitions of $\mathcal{A}'$ (we denote such an operation by *Rev*), then it performs a subset construction to build a DFA equivalent to the reversed copy of $\mathcal{A}'$ (we denote such an operation by *Det*), and finally it iterates such two operations once more. It can be proved that $Det(Rev(Det(Rev(\mathcal{A}'))))$ is a DFA equivalent to the NFA $\mathcal{A}'$ having the minimum number of states among all equivalent DFA. Moreover, such a construction requires, in the worst case, $\mathcal{O}(2^n)$ time and space, where n is the number of states of the input automaton $\mathcal{A}'$.

Proposition 11 *Given an UPA $\mathcal{A}$ in normal form, one can compute in time $\mathcal{O}(2^{|\mathcal{A}|})$ an equivalent UPA $\mathcal{B}$ in canonical form. Moreover, the size of $\mathcal{B}$ is, in the worst case, a simple exponential in the size of $\mathcal{A}$.*

Proof Let $\mathcal{A}$ be an UPA in normal form. By exploiting Proposition 10, one can compute an equivalent prefix-friendly UPA $\mathcal{A}'$ whose number of states is less than or equal to the number of states of $\mathcal{A}$. Then, by applying the above described sequence of normalization steps, one can transform $\mathcal{A}'$ into an equivalent UPA $\mathcal{B}$ in canonical form. As for the complexity of the whole procedure, recall that $\mathcal{A}'$ can be computed from $\mathcal{A}$ in quadratic time. Moreover, computing the prefix automaton of $\mathcal{A}'$ (and, vice-versa, computing the UPA which corresponds to any given prefix automaton), can be done in linear time. Finally, since Brzozowski's algorithm takes simple exponential time with respect to the number of states of the input automaton, computing the minimal DFA equivalent to the prefix automaton of $\mathcal{A}'$ requires time $\mathcal{O}(2^{|\mathcal{A}'|}) = \mathcal{O}(2^{|\mathcal{A}|})$. □

5 UPA-recognizable languages: algorithms and complexity

UPA can be successfully exploited to solve a number of classical problems about sets of ultimately periodic words. We will focus our attention on the following basic problems:

- **Emptiness problem**. It consists of deciding whether a given UPA $\mathcal{A}$ recognizes the empty language.
- **Membership problem**. It consists of deciding whether a given UPA $\mathcal{A}$ recognizes a given ultimately periodic word w, represented as a pair (u, v) consisting of a finite prefix and a finite non-empty repeating pattern.
- **Equivalence problem**. Given two UPA $\mathcal{A}$ and $\mathcal{B}$, it consists of deciding whether $\mathscr{L}^\omega(\mathcal{A}) = \mathscr{L}^\omega(\mathcal{B})$.
- **Inclusion problem**. Given two UPA $\mathcal{A}$ and $\mathcal{B}$, it consists of deciding whether $\mathscr{L}^\omega(\mathcal{A}) \subseteq \mathscr{L}^\omega(\mathcal{B})$.
- **Size-optimization problem**. Given an UPA $\mathcal{A}$ in a specific normalized form, it consists of building an equivalent UPA $\mathcal{B}$ having the smallest number of states among all equivalent UPA in that normalized form.

As UPA can be viewed both as a restricted class of Büchi automata and as an extension of NFA, we will compare the structure and the complexity of the proposed algorithms with those of both of them.

5.1 The emptiness and membership problems

In the case of a Büchi automaton, the emptiness problem is solved by (i) searching for a path departing from an initial state and reaching a final state and (ii) searching for a loop that contains such a final state. Since every final state of an UPA in normal form belongs to a final loop, the emptiness problem for UPA in normal form reduces to the problem of searching for a path from an initial state to a final state, as it happens with NFA. Thus, the emptiness problem can be solved in linear time $\mathcal{O}(|\mathcal{A}|)$.

As for the membership problem, there is a straightforward algorithm, which exploits basic closure properties, that decides whether a given UPA $\mathcal{A}$ accepts a given ultimately periodic word $w = uv^\omega$ in time $\mathcal{O}(|\mathcal{A}|(|u| + |v|))$. The problem of checking whether $w = uv^\omega$ belongs to the ω-language recognized by a given UPA $\mathcal{A}$ is indeed equivalent to the problem of testing the (non-)emptiness for the ω-language $\mathscr{L}^\omega(\mathcal{A}) \cap \mathscr{L}^\omega(\mathcal{B})$, where $\mathcal{B}$ is an UPA that recognizes the singleton $\{uv^\omega\}$. A (slightly) more efficient solution, which takes time $\mathcal{O}(|\mathcal{A}||u|+|v|)$, takes advantage of prefix automata. Given an UPA $\mathcal{A}$ and an ultimately periodic word $w = uv^\omega$, one can decide whether $w \in \mathscr{L}^\omega(\mathcal{A})$ by performing the following steps:

1. compute the prefix automaton $\mathcal{A}_{pre}$ of $\mathcal{A}$;
2. replace in $\mathcal{A}_{pre}$ every transition of the form (s, b, f), with $b = (s, C) \in \Sigma_{\mathcal{A}} \setminus \Sigma$, by a transition of the form (s, x, f), where x is either the symbol $\top$ or the symbol $\bot$, depending on whether C encodes the repeating pattern v starting from s or not;

3. decide whether the resulting NFA accepts the word $\mathfrak{u}\top$.

Note that the above steps, but the last one, can be performed in linear time with respect to the size of $\mathcal{A}$, $\mathfrak{u}$, and $\mathfrak{v}$, while the last step can be performed in time linear in $|\mathcal{A}||\mathfrak{u}|$.

5.2 The equivalence and inclusion problems

It is well known that the equivalence and inclusion problems, for any class of automata which is closed under intersection, are inter-reducible. As an example, given two Büchi automata $\mathcal{A}$ and $\mathcal{B}$, we have $\mathscr{L}^{\omega}(\mathcal{A}) = \mathscr{L}^{\omega}(\mathcal{B})$ iff $\mathscr{L}^{\omega}(\mathcal{A}) \subseteq \mathscr{L}^{\omega}(\mathcal{B}) \ \wedge \ \mathscr{L}^{\omega}(\mathcal{B}) \subseteq \mathscr{L}^{\omega}(\mathcal{A})$ and, similarly, $\mathscr{L}^{\omega}(\mathcal{A}) \subseteq \mathscr{L}^{\omega}(\mathcal{B})$ iff $\mathscr{L}^{\omega}(\mathcal{A}) = \mathscr{L}^{\omega}(\mathcal{A}) \cap \mathscr{L}^{\omega}(\mathcal{B})$. Moreover, if the class of automata is also closed under complementation, then both problems can be reduced to the emptiness problem. As an example, given two Büchi automata $\mathcal{A}$ and $\mathcal{B}$, we have $\mathscr{L}^{\omega}(\mathcal{A}) \subseteq \mathscr{L}^{\omega}(\mathcal{B})$ iff $\mathscr{L}^{\omega}(\mathcal{A}) \cap \mathscr{L}^{\omega}(\mathcal{C}) = \emptyset$, where $\mathcal{C}$ is the Büchi automaton recognizing the complement language of $\mathscr{L}^{\omega}(\mathcal{B})$.

In [35] an *implicit* construction of a complement Büchi automaton has been given, which allows one to solve the equivalence problem for Büchi automata in polynomial space. Such a construction is based on the ability to encode each state and checking each transition of the complement automaton by using only a polynomial amount of space. Since, in the worst case, the number of states of the complement automaton is $\Omega(2^{\mathfrak{n}\log\mathfrak{n}})$, where $\mathfrak{n}$ is the number of states of the input automaton (see [19, 24] for lower-bound results and [26, 32, 33] for constructions that match these bounds), it turns out that that any deterministic or non-deterministic algorithm based on an explicit or implicit construction of a complement automaton must use $\Omega(\mathfrak{n}\log\mathfrak{n})$ space.

As for NFA, both the equivalence and inclusion problems are proved to be PSPACE-complete [15]. Standard algorithms solving these problems are based on either explicit or implicit constructions of equivalent deterministic finite state automata and they use either simple exponential time $\Theta(2^{\mathfrak{n}})$ or linear space $\Theta(\mathfrak{n})$, where $\mathfrak{n}$ is the number of states of the input NFA. As an example, the inclusion problem for two NFA $\mathcal{A}$ and $\mathcal{B}$ can be solved by guessing a finite word $\mathfrak{u}$ which is a witness for the non-inclusion of $\mathscr{L}(\mathcal{A})$ in $\mathscr{L}(\mathcal{B})$, namely, such that $\mathfrak{u} \in \mathscr{L}(\mathcal{A})$ and $\mathfrak{u} \in \mathscr{L}(\mathcal{C})$, where $\mathcal{C}$ is the DFA recognizing the complement of $\mathscr{L}(\mathcal{B})$. Verifying that $\mathfrak{u} \in \mathscr{L}(\mathcal{C})$ can be done directly on the NFA $\mathcal{B}$ by first computing the set of states of $\mathcal{B}$ that are reachable from an initial state by reading $\mathfrak{u}$ and then verifying that such a set does not contain any final state. The described algorithm thus requires linear space with respect to the size of the input NFA $\mathcal{A}$ and $\mathcal{B}$.

Since UPA-recognizable languages have equivalent representations in terms of prefix automata, by exploiting existing algorithms for NFA we can devise suitable algorithms for the equivalence and inclusion problems of UPA of the same complexity. As a preliminary result, we provide a polynomial lower bound to the space complexity of the equivalence and inclusion problems for UPA.

Proposition 12 *The equivalence problem and inclusion problem for UPA are PSPACE-hard under LOGSPACE reductions.*

Proof We provide a LOGSPACE reduction from the equivalence problem for NFA, which is known to be PSPACE-hard under LOGSPACE reductions [15], to the equivalence problem for UPA. Let $\mathcal{A}$ and $\mathcal{B}$ be two NFA recognizing the languages $\mathscr{L}(\mathcal{A})$ and $\mathscr{L}(\mathcal{B})$, respectively. We extend the input alphabet with a new symbol # and we transform the NFA $\mathcal{A}$ (resp., $\mathcal{B}$) into an UPA $\mathcal{A}'$ (resp., $\mathcal{B}'$) that recognizes the ω-language $\mathscr{L}(\mathcal{A})\{\#\}^\omega$ (resp., $\mathscr{L}(\mathcal{A})\{\#\}^\omega$) as follows:

i) we add a new state $\mathtt{f}$ that becomes the unique final state of $\mathcal{A}'$ (resp., $\mathcal{B}'$),

ii) we add a new transition of the form $(\mathtt{f}, \#, \mathtt{f})$,

iii) for each final state $\mathtt{s}$ in $\mathcal{A}$ (resp., $\mathcal{B}$), we add a new transition $(\mathtt{s}, \#, \mathtt{f})$.

Clearly, we have $\mathscr{L}^\omega(\mathcal{A}') = \mathscr{L}^\omega(\mathcal{B}')$ iff $\mathscr{L}(\mathcal{A}) = \mathscr{L}(\mathcal{B})$. This allows us to conclude that the equivalence problem and the inclusion problem for UPA are PSPACE-hard under LOGSPACE reductions. □

Let us now provide optimal algorithms that solve the equivalence and inclusion problems for UPA.

The first solution to the equivalence and inclusion problems for UPA stems from the fact that the canonical form of an UPA is unique, up to isomorphisms, among all equivalent UPA. Thus, the problem of deciding whether two given UPA $\mathcal{A}$ and $\mathcal{B}$ recognize the same ω-language can be reduced to the problem of testing whether the canonical forms $\mathcal{A}'$ and $\mathcal{B}'$ of $\mathcal{A}$ and $\mathcal{B}$, respectively, are isomorphic. By Theorem 4, we know that the canonical form of an UPA is computable in exponential time. Moreover, since canonical forms of UPA are, basically, deterministic labeled graphs (with the only exception of the transitions entering the final loops), one can easily decide by a linear time procedure whether the canonical UPA $\mathcal{A}'$ and $\mathcal{B}'$ are isomorphic. This allows us to conclude that the equivalence problem for two generic UPA $\mathcal{A}$ and $\mathcal{B}$ can be decided by a deterministic procedure that requires *exponential time* in the size of the input UPA $\mathcal{A}$ and $\mathcal{B}$.

As for the inclusion problem, one can exploit the fact that, given two UPA $\mathcal{A}'$ and $\mathcal{B}'$ in canonical form, $\mathcal{O}(|\mathcal{A}'| + |\mathcal{B}'|)$ time suffices to compute an UPA $\mathcal{C}'$ in canonical form that recognizes the intersection language $\mathscr{L}^\omega(\mathcal{A}') \cap \mathscr{L}^\omega(\mathcal{B}')$. This yields a straightforward procedure that, given two UPA $\mathcal{A}$ and $\mathcal{B}$, decides in exponential time whether $\mathscr{L}^\omega(\mathcal{A}) \subseteq \mathscr{L}^\omega(\mathcal{B})$: such a procedure first computes the canonical forms $\mathcal{A}'$ and $\mathcal{B}'$ of $\mathcal{A}$ and $\mathcal{B}$, respectively, then it computes the UPA $\mathcal{C}'$ in canonical form recognizing $\mathscr{L}^\omega(\mathcal{A}') \cap \mathscr{L}^\omega(\mathcal{B}')$, and finally it decides whether $\mathscr{L}^\omega(\mathcal{A}') = \mathscr{L}^\omega(\mathcal{C}')$.

It is worth pointing out that the proposed deterministic solutions to the equivalence and inclusion problems for UPA outperform classical solutions for Büchi automata, which are based on the construction of complement languages, and their time complexity is comparable to the time complexity of analogous algorithms working on NFA.

We now describe alternative non-deterministic algorithms that solve the equivalence problem and the inclusion problem for UPA using at most a

linear amount of space. These are modified versions of standard algorithms working on NFA, which, basically, exploit an implicit subset construction to decide the (non-)inclusion problem for two given regular languages. Similar constructions have been also proposed in [20, 25].

Let $\mathcal{A} = (\Sigma, S, \Delta, \mathcal{I}, \mathcal{F})$ and $\mathcal{B} = (\Sigma, S', \Delta', \mathcal{I}', \mathcal{F}')$ be two generic UPA. By Theorem 3, one can apply the standard non-inclusion algorithm for NFA directly to the prefix automata corresponding to prefix-friendly UPA $\mathcal{A}'$ and $\mathcal{B}'$ obtained from $\mathcal{A}$ and $\mathcal{B}$, respectively. However, in such a case, the worst-case space complexity of the resulting algorithm is quadratic with respect to the size of the input UPA $\mathcal{A}$ and $\mathcal{B}$ (recall that prefix-friendly UPA $\mathcal{A}'$ and $\mathcal{B}'$ may have quadratic size with respect to the original UPA $\mathcal{A}$ and $\mathcal{B}$). Here, we describe a modified version of the non-inclusion algorithm that directly works on UPA and that requires linear space with respect to their size.

Without loss of generality, we can assume that the two input UPA $\mathcal{A}$ and $\mathcal{B}$ are in normal form (by Proposition 9, this does not imply any blowup of the size). The proposed algorithm exploits non-determinism to guess an ultimately periodic word w that belongs to $\mathscr{L}^\omega(\mathcal{A})$ and then verifies that w does not belong to $\mathscr{L}^\omega(\mathcal{B})$, thus certifying that $\mathscr{L}^\omega(\mathcal{A}) \not\subseteq \mathscr{L}^\omega(\mathcal{B})$. Since any run of $\mathcal{A}$ that accepts an ultimately periodic word w is eventually trapped inside a simple final loop, we can distinguish two phases of the algorithm, the first one dealing with a prefix of the run of $\mathcal{A}$ that reaches the final loop, the second one dealing with the (unique) suffix encoded by the final loop[4].

During the first phase, a finite prefix $s_0 s_1 ... s_n$ of a run of $\mathcal{A}$ and a word $a_1 ... a_n$ recognized by it are guessed. At the same time, the sets $S'_0, S'_1, ..., S'_n$ of states of $\mathcal{B}$ which are visited while reading the word $a_1 ... a_n$ are computed. Configurations are thus described by pairs of the form (s_i, S'_i), with $s_i \in S$ and $S'_i \subseteq S'$. The algorithm starts with an initial configuration of the form (s_0, S'_0), where s_0 is an initial state of $\mathcal{A}$ and S'_0 is the set of initial states of $\mathcal{B}$. At each step, the algorithm chooses a transition of $\mathcal{A}$ of the form (s_i, a_{i+1}, s_{i+1}), with a_{i+1} being a symbol from Σ and s_{i+1} being a state of $\mathcal{A}$, and it then computes the next configuration (s_{i+1}, S'_{i+1}), where S'_{i+1} is defined as the set of all states r' of $\mathcal{B}$ such that there exists $s' \in S'_i$, with (s', a_{i+1}, r') being a valid transition of $\mathcal{B}$. If, at some step n, s_n turns out to be a final state, then we know that $\mathcal{A}$ recognizes an ultimately periodic word with prefix $a_1 ... a_n$. At this point, the algorithm switches to the second phase. Note that, even though the first phase can be carried on for arbitrarily many steps (this is the case, for instance, when a non-final loop of $\mathcal{A}$ is reached), we can limit the number of iterations during this phase to $|S|2^{|S'|}$: indeed, if $n \geqslant |S|2^{|S'|}$ steps were performed during the first phase, then, by the Pigeonhole Principle, there would exist two indices n', n'', with $n' < n'' \leqslant |S|2^{|S'|}$, such that $(s_{n'}, S'_{n'}) = (s_{n''}, S'_{n''})$ and hence the word $a_1 ... a_{n'} a_{n''+1} ... a_n$ would be the prefix of an alternative witness for the non-inclusion.

During the second phase, the computation proceeds in a deterministic way on the basis of the (unique) infinite periodic word $b_1 b_2 ...$ which is recognized by $\mathcal{A}$ starting from the last visited (final) state s_n. Configurations are again

[4] It is worth remarking that such a technique cannot be extended to generic Büchi automata, since their runs may visit distinct final loops infinitely often.

described by pairs of the form (q_i, Q'_i), with $q_i \in S$ and $Q'_i \subseteq S'$, and they obey to the following constraints:

- the first configuration (q_0, Q'_0) coincides with the last configuration (s_n, S'_n) computed during the first phase;
- (b_{i+1}, q_{i+1}) is the unique pair such that (q_i, b_{i+1}, q_{i+1}) is a valid transition of $\mathcal{A}$;
- Q'_{i+1} is the set of all states r' of $\mathcal{B}$ for which there exists $q' \in Q'_i$ such that (q', b_{i+1}, r') is a valid transition of $\mathcal{B}$.

By a simple application of the Pigeonhole Principle, we have that there exist two indices m, m', with $m < m' \leqslant |S|2^{|S'|}$, such that $(q_m, Q'_m) = (q_{m'}, Q'_{m'})$. Hence, if Q'_m contains no final state, then we know that the ultimately periodic word

$$(a_1 ... a_n)(b_1 ... b_m)(b_{m+1} ... b_{m'})^\omega$$

is recognized by $\mathcal{A}$, but not by $\mathcal{B}$, thus certifying that $\mathscr{L}^\omega(\mathcal{A}) \not\subseteq \mathscr{L}^\omega(\mathcal{B})$. Otherwise, if Q'_m contains at least one final state, then the computation is discarded. It is worth pointing out that we do not need to exhibit the candidate ultimately periodic word $(a_1 ... a_n)(b_1 ... b_m)(b_{m+1} ... b_{m'})^\omega$ in $\mathscr{L}^\omega(\mathcal{A}) \setminus \mathscr{L}^\omega(\mathcal{B})$, and thus we do not need to keep track of its symbols during the computation.

The described non-deterministic algorithm solves the non-inclusion (and hence the non-equivalence) problem by using *linear space* in the size of the input UPA (precisely, it requires $\mathcal{O}(\log |S| + |S'|)$ space to store a configuration of the form (s_i, S'_i), with $s_i \in S$ and $S'_i \subseteq S'$, and the value of a counter i ranging over $\{0, ..., |S|2^{|S'|}\}$). The space complexity of the proposed algorithm is comparable with that of classical non-deterministic algorithms working on NFA and it is strictly lower than that of classical non-deterministic algorithms working on Büchi automata.

Finally, we recall that, by exploiting Savitch's Theorem [30], the above non-deterministic procedure can be turned into a deterministic one that solves the inclusion (and hence the equivalence) problem for UPA by using at most quadratic space.

5.3 The size-optimization problem

Let us consider now the minimization problem for UPA. We first prove that, similarly to the case of NFA (see [16, 17, 23]), the size-optimization problem for UPA is PSPACE-complete and it may yield different (non-isomorphic) solutions. Then, we show that standard minimization algorithms for NFA can be *directly* exploited to minimize the size of any UPA in normal form. These algorithms can be applied to generic UPA as well; however, in such a case the resulting UPA are, in general, approximations of size-optimal ones.

To start with, we provide a sufficient condition for UPA to be size-optimal among all equivalent UPA in normal form.

Proposition 13 *Let $\mathcal{A}$ be an UPA in prefix-friendly form and let $\mathcal{A}_{pre}$ be its prefix automaton. If $\mathcal{A}_{pre}$ has the minimum number of states among all equivalent NFA, then $\mathcal{A}$ has the minimum number of states among all equivalent UPA in normal form.*

Proof For any given automaton (UPA or NFA) $\mathcal{A}$, let us denote by $\mathfrak{n}(\mathcal{A})$ the number of its states. Assume $\mathcal{A}$ to be a prefix-friendly UPA, whose prefix automaton $\mathcal{A}_{pre}$ has the minimum number of states among all equivalent NFA. By definition of prefix automaton, we have that $\mathfrak{n}(\mathcal{A}_{pre}) = \mathfrak{n}(\mathcal{A}) + 1$. Let us now consider a generic UPA $\mathcal{B}$ in normal form equivalent to $\mathcal{A}$. By Proposition 10, there is a prefix-friendly UPA $\mathcal{B}'$ equivalent to $\mathcal{B}$ such that $\mathfrak{n}(\mathcal{B}') \leqslant \mathfrak{n}(\mathcal{B})$. Moreover, since $\mathcal{B}'$ is prefix-friendly, by Theorem 3 we have that the prefix automaton $\mathcal{B}'_{pre}$ is equivalent to $\mathcal{A}_{pre}$. From the minimality of $\mathcal{A}_{pre}$, we obtain $\mathfrak{n}(\mathcal{A}_{pre}) \leqslant \mathfrak{n}(\mathcal{B}'_{pre})$. Moreover, by construction, $\mathfrak{n}(\mathcal{B}'_{pre}) = \mathfrak{n}(\mathcal{B}') + 1$. Summing up, we have

$$\mathfrak{n}(\mathcal{A}) = \mathfrak{n}(\mathcal{A}_{pre}) - 1 \leqslant \mathfrak{n}(\mathcal{B}'_{pre}) - 1 = \mathfrak{n}(\mathcal{B}') \leqslant \mathfrak{n}(\mathcal{B})$$

which proves that $\mathcal{A}$ has the minimum number of states among all equivalent UPA in normal form. □

By exploiting Proposition 13, we can devise a PSPACE solution to the size-optimization problem for UPA in normal form. Such a solution exploits an auxiliary procedure that minimizes the number of states of prefix automata (see [17, 23] for implementation details). Since the size-optimization problem for NFA is known to be PSPACE-complete [16, 17, 23] and since (by an argument similar to that of Proposition 12) it is inter-reducible with the size-optimization problem for UPA in normal form, we can conclude that the latter problem is PSPACE-complete as well.

If we do not restrict to the class of UPA in normal form, then the number of states of UPA can be further reduced. Below, we show that some final loops of UPA may be safely removed, under the proviso that another 'subsuming' non-final loop takes their role.

Definition 8 Let $\mathcal{A} = (\Sigma, \mathsf{S}, \Delta, \mathcal{I}, \mathcal{F})$ be an UPA, C a final loop, and C' a simple non-final loop. We say that C is *subsumed* by C' if C has no exiting transitions and there is a surjective function $\mathsf{f} : \mathsf{C}' \to \mathsf{C}$ that satisfies the following two conditions:

- for every pair of states r, s in C', $(\mathsf{r}, \mathfrak{a}, \mathsf{s}) \in \Delta$ iff $(\mathsf{f}(\mathsf{r}), \mathfrak{a}, \mathsf{f}(\mathsf{s})) \in \Delta$ (intuitively, C and C' are bisimilar),
- for every state r neither in C nor in C' and every state s in C, $(\mathsf{r}, \mathfrak{a}, \mathsf{s}) \in \Delta$ iff there is a state s' in C' such that $\mathsf{s} = \mathsf{f}(\mathsf{s}')$ and $(\mathsf{r}, \mathfrak{a}, \mathsf{s}') \in \Delta$ (intuitively, the loop C' augmented with its entering transitions simulates the loop C augmented with its entering transitions).

If C is a final loop of $\mathcal{A}$ which is subsumed by a simple non-final loop C', then we can obtain an equivalent UPA $\mathcal{B}$ with fewer states by (i) removing the loop C and (ii) letting C' be a final loop in $\mathcal{B}$. As an example, the right-hand side UPA in Figure 5 is obtained from the left-hand side one by removing the final loop on state s_4 and by letting the subsuming loop on states s_1 and s_2 be final.

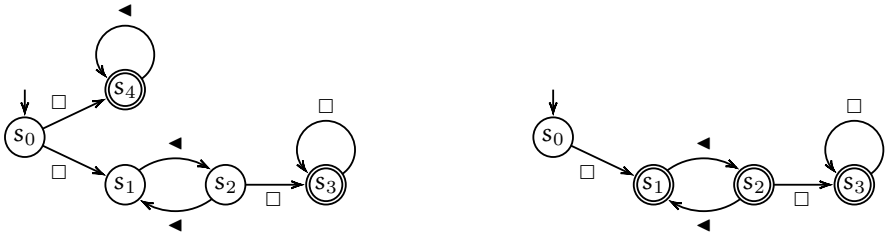

Fig. 5 An example of subsumed final loop.

Proposition 14 *Given an UPA* $\mathcal{A} = (\Sigma, S, \Delta, \mathcal{I}, \mathcal{F})$*, a final loop* C*, and a simple non-final loop* C' *that subsumes* C*, the automaton* $\mathcal{B} = (\Sigma, S \setminus C, \Delta, \mathcal{I}, (\mathcal{F} \cup C') \setminus C)$ *is an UPA equivalent to* $\mathcal{A}$*.*

Proof Since every final state s of $\mathcal{B}$ is contained either in C' (and thus the strongly connected component of s in $\mathcal{B}$ is a simple loop) or in $\mathcal{F} \setminus C$ (and thus the strongly connected component of s in $\mathcal{B}$ is a single transient state or a simple final loop), it immediately follows that $\mathcal{B}$ is an UPA. To complete the proof, it suffices to show that $\mathcal{B}$ is equivalent to $\mathcal{A}$. Let f be the surjective function from C' to C of Definition 8. We first prove that $\mathscr{L}^\omega(\mathcal{A}) \subseteq \mathscr{L}^\omega(\mathcal{B})$. Let w an ultimately periodic word in $\mathscr{L}^\omega(\mathcal{A})$ and let ρ be a successful run of $\mathcal{A}$ on w. By Definition 8 the subsumed loop C has no exiting transition and thus either ρ does not contain any state in C or, all, but finitely many, states in ρ belong to C. In the former case, it immediately follows that ρ is also a successful run of $\mathscr{L}^\omega(\mathcal{B})$ on w. In the latter case, we can obtain a successful run ρ' of $\mathcal{B}$ on w by simply replacing each state s of C with a suitable state s' such that $f(s') = s$ (given the properties of the function f, ρ' respects the transitions of $\mathcal{B}$ on w). This shows that w is accepted by $\mathcal{B}$. As for the converse inclusion, let w be an ultimately periodic word accepted by $\mathcal{B}$ and let ρ' be a successful run of $\mathcal{B}$ on w. By definition of UPA, the set of all states that occur infinitely often in ρ' is either disjoint from C' or it coincides with the set of states of C'. In the former case, ρ' is a successful run of $\mathcal{A}$ on w as well and thus w is accepted by $\mathcal{A}$; in the latter case, we can obtain a successful run ρ of $\mathcal{A}$ on w by simply replacing each state s that occur infinitely often in ρ' with the state $f(s)$. □

On the basis of Proposition 14, we can devise an algorithm that uses polynomial space to remove all subsumed final loops from a given UPA $\mathcal{A}$ in normal form. Clearly, the removal of all subsumed final loops of $\mathcal{A}$ (if any) results in an equivalent UPA $\mathcal{B}$ with fewer states. Moreover, since all final loops of $\mathcal{A}$ are disjoint, the size of the resulting UPA $\mathcal{B}$ does not depend on the removal order of the subsumed final loops.

It is worth pointing out that, in general, the UPA $\mathcal{B}$ resulting from the removal of all subsumed final loops of $\mathcal{A}$ is not guaranteed to be size-optimal among all equivalent UPA, even in the case in which $\mathcal{A}$ is a size-optimal UPA in normal form. As an example, consider the two equivalent UPA in normal form depicted in Figure 6. The left-hand side automaton has no subsumed final loops, while the right-hand side automaton has a final loop (the one

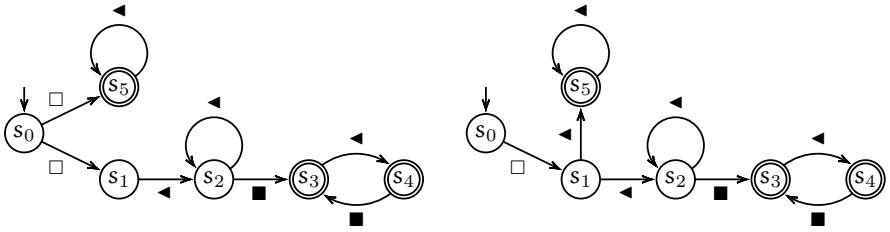

Fig. 6 Two equivalent UPA in normal form.

on state s_5) which is subsumed by a non-final loop (the one on state s_2). The optimization algorithm has no effect on the left-hand side UPA, while it reduces the size of the equivalent right-end side UPA. In general, to compute an UPA with the minimum number of states among all equivalent UPA one must resort to costly algorithms based on trace equivalence. These algorithms can be obtained as generalizations of standard minimization algorithms for NFA [17, 23].

6 Applications to time granularity

The algorithms for solving the emptiness, membership, equivalence, inclusion, and size-optimization problems, which have been described in the previous section, can be directly exploited to reason on possibly infinite sets of time granularities, e.g., the algorithm that solves the equivalence problem for UPA can be used to check whether or not two given automaton-based representations define the same set of granularities. In the following, we show that UPA turn out to be useful also in solving another basic problem for time granularity, namely, the comparison of pairs of sets of time granularities with respect to various standard relations, e.g., partition, group, sub-granularity, aligned refinement.

In its most common formulation, the granularity comparison problem is viewed as the problem of deciding whether a designated relation $\sim$ holds between two granularities $\mathsf{G} \in \mathcal{G}$ and $\mathsf{H} \in \mathcal{H}$, where $\mathcal{G}$ and $\mathcal{H}$ are two given sets of granularities. According to such a definition, the granularity comparison problem is actually a family of problems, whose different concrete instances are obtained by specifying the relation that must hold between the pairs of granularities that belong to the two sets $\mathcal{G}$ and $\mathcal{H}$.

Within the framework of automaton-based representations, a granularity comparison problem is reducible to the problem of deciding, given two UPA $\mathcal{A}$ and $\mathcal{B}$ representing two sets of ultimately periodic granularities, whether there exist some words $w_{\mathsf{G}} \in \mathscr{L}^{\omega}(\mathcal{A})$ and $w_{\mathsf{H}} \in \mathscr{L}^{\omega}(\mathcal{B})$ that satisfy a certain relation $\sim$, which basically capture the designated relation between time granularities.

To start with, we provide string-based characterizations of standard relations between time granularities, that is, group into, refine, partition, and aligned refinement. Hereafter, given a word w over the alphabet $\Sigma = \{\blacksquare, \square, \blacktriangleleft\}$

and a symbol $a \in \Sigma$, we denote by $|w|_a$ the number of occurrences (possibly ∞) of a in w. Note that, given an infinite word w that represents a time granularity G and given two natural numbers $t, x \in \mathbb{N}$, x is the index of the (unique) granule of G that contains t iff $w[t+1] \in \{\blacksquare, \blacktriangleleft\}$ and $|w[1,t]|_{\blacktriangleleft} = x$.

Proposition 15 *Let u and v be two infinite words over the alphabet $\{\blacksquare, \square, \blacktriangleleft\}$ that represent, respectively, two granularities G and H. We have that*

- G *groups into* H *iff for every* $t \in \mathbb{N}$,
  - *i)* $v(t+1) = \blacksquare$ *implies* $u(t+1) \in \{\blacksquare, \blacktriangleleft\}$,
  - *ii)* $v(t+1) = \blacktriangleleft$ *implies* $u(t+1) = \blacktriangleleft$,
  - *iii)* $v(t+1) = \square$ *implies either* $u(t+1) = \square$ *or* $v(t'+1) = \square$ *for all* $t' \in \mathbb{N}$ *such that* $|u[1,t]|_{\blacktriangleleft} = |u[1,t']|_{\blacktriangleleft}$;
- G *refines* H *iff for every* $t \in \mathbb{N}$,
  - *i)* $u(t+1) = \blacksquare$ *implies* $v(t+1) = \blacksquare$,
  - *ii)* $u(t+1) = \blacktriangleleft$ *implies* $v(t+1) \in \{\blacksquare, \blacktriangleleft\}$;
- G *partitions* H *iff for every* $t \in \mathbb{N}$,
  - *i)* $u(t+1) = \square$ *if and only if* $v(t+1) = \square$,
  - *ii)* $u(t+1) = \blacksquare$ *implies* $v(t+1) = \blacksquare$;
- G *is an aligned refinement of* H *iff for every* $t \in \mathbb{N}$,
  - *i)* $u(t+1) = \blacksquare$ *implies* $v(t+1) = \blacksquare$,
  - *ii)* $u(t+1) = \blacktriangleleft$ *implies* $v(t+1) \in \{\blacksquare, \blacktriangleleft\}$;
  - *iii)* $u(t+1) = \blacktriangleleft$ *implies* $|u[1,t]|_{\blacktriangleleft} = |v[1,t]|_{\blacktriangleleft}$.

Proof We only prove the characterization for the alignment refinement relation. Suppose that G is an aligned refinement of H. By definition, every time point $t \in \mathbb{N}$ which is covered by G is covered by H as well. Thus, for every $t \in \mathbb{N}$, $u(t+1) \in \{\blacksquare, \blacktriangleleft\}$ implies $v(t+1) \in \{\blacksquare, \blacktriangleleft\}$. Moreover, it cannot happen that $u(t+1) = \blacksquare$ and $v(t+1) = \blacktriangleleft$ because, otherwise, the granule of G that contains the time point t would not be included in a granule of H. Finally, for every $t \in \mathbb{N}$, if $u(t+1) = \blacktriangleleft$, then t is the last time point of the granule $G(x)$, where $x = |u[1,t]|_{\blacktriangleleft}$. Since $G(x) \subseteq H(x)$, we have that $|v[1,t]|_{\blacktriangleleft} = x$. The converse implication follows by a symmetric argument. □

Let $\sim$ be one of the standard relations between time granularities. By exploiting Proposition 15, it is immediate to see that the ω-language $L_\sim$ of all pairs of infinite words (including non-periodical ones) that represent pairs of time granularities G, H such that $G \sim H$ is Büchi-recognizable. In particular, one can build a Büchi automaton $\mathcal{C}_\sim$ over the alphabet $\{\blacksquare, \square, \blacktriangleleft\} \times \{\blacksquare, \square, \blacktriangleleft\}$ that recognizes the language $L_\sim$. As an example, Figure 7 depicts the Büchi automaton $\mathcal{C}_\sim$, where $\sim$ is the aligned refinement relation.

Moreover, given two UPA $\mathcal{A}$ and $\mathcal{B}$ representing some sets $\mathcal{G}$ and $\mathcal{H}$ of time granularities, one can build a suitable product automaton (an UPA) that recognizes the ω-language $L = \mathscr{L}^\omega(\mathcal{A}) \times \mathscr{L}^\omega(\mathcal{B})$, which basically represents all pairs (G, H) of time granularities, with $G \in \mathcal{G}$ and $H \in \mathcal{H}$. Thus, one can solve the comparison problem for the designated relation $\sim$ by simply testing the emptiness of the intersection language $L \cap L_\sim$, which is known

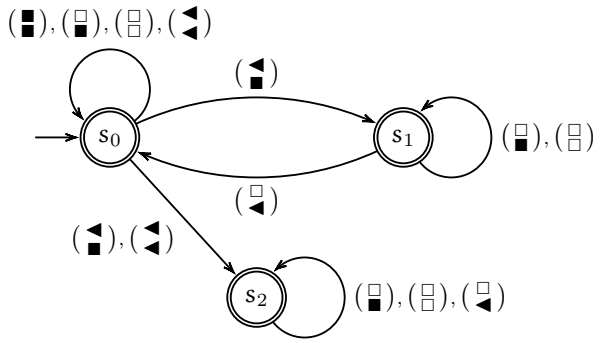

Fig. 7 The automaton for the aligned refinement relation.

to be UPA-recognizable. Note that, since the Büchi automaton $\mathcal{C}_{\sim}$ is fixed with the designated relation $\sim$, the complexity of the resulting algorithm is $\mathcal{O}(|\mathcal{A}||\mathcal{B}|)$.

We conclude this section by showing how a concrete problem, taken from the medical domain of heart transplant patients, can be addressed and efficiently solved by exploiting the notion of time granularity and the properties of UPA-recognizable ω-languages.

Posttransplantation guidelines require outpatients to take drugs and to submit to periodical visits for life. These requirements are usually collected in formal protocols with schedules specifying the therapies and the frequency of the check-ups. We report an excerpt of the guidelines for a heart transplant patient in [22]. Depending on the physical conditions of the patient, the guidelines can require, together with other treatments, an estimation of the glomerular filtration rate (GFR) with one of the following schedules:

- 3 months and 12 months posttransplantation and every year thereafter;
- 3 months and 12 months posttransplantation and every 2 years thereafter.

These protocols involve the so-called unanchored granularities, to manage the various admissible starting points for the scheduled therapies (and/or check-ups), as well as sets of granularities with different repeating patterns, to capture the set of distinct periodicities of the scheduled therapies. In particular, since different protocols can be specified for the same class of patients by different people/institutions, it is a crucial problem to decide whether two protocols define the same set of therapies/granularities (equivalence problem). Solving this problem makes it possible to choose the most compact, or the most suitable, representation for a given protocol.

Another meaningful reasoning task is that of checking whether a given therapy/granularity assigned to a patient satisfies the prescribed protocol, that is, whether it belongs to the set of therapies/granularities of the protocol (consistency-checking problem). As an example, consider the (sub)set of therapies/check-ups of the above protocol for heart transplant patients and the instance of the temporal relation `Visits(patientId, date, treatment)`, represented in Table 1. Given a representation of the single granularity G for the specific therapy (up to a certain date) of a certain patient and given a

patientId	date (MM/DD/YYYY)	treatment
1001	02/10/2003	transplant
1001	04/26/2003	GFR
1002	06/07/2003	GFR
1001	06/08/2003	biopsy
1001	02/10/2004	GFR
1001	01/11/2005	GFR
1001	01/29/2006	GFR

Table 1 A hypothetical schedule for therapies/check-ups.

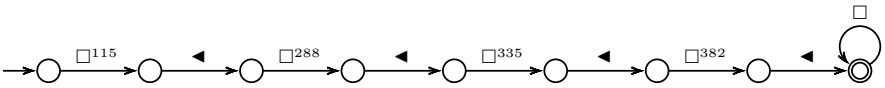

Fig. 8 An UPA representing GFR measurements for a patient.

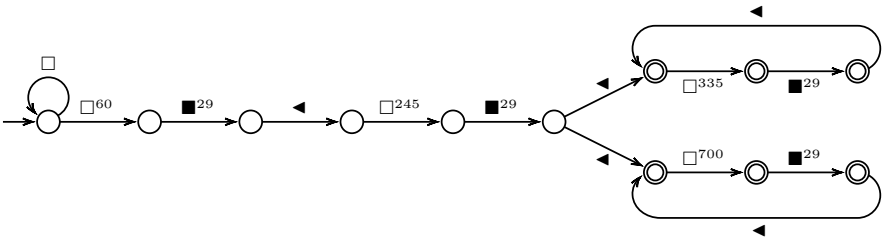

Fig. 9 An UPA-based specification of the protocol.

representation of the set of (periodic) time granularities for the prescribed therapies/check-ups, the consistency-checking problem can be decided by testing whether granularity G properly relates to some granularity in $\mathcal{H}$. The consistency-checking problem can thus be viewed as a particular case of granularity comparison problem. Below, we show how such a problem can be effectively solved by means of UPA.

For the sake of simplicity, we consider months of 30 days and years of 365 days (relaxing such a simplification is tedious, but trivial). By properly selecting records in Table 1, we can build the granularity G of GFR measurements for the patient identified by `patientId` 1001. We represent such a granularity as a single ultimately periodic word w (starting from 01/01/2003), in which the occurrences of ◀ denote the days of the visits. The UPA $\mathcal{A}$ that accepts w is depicted in Figure 8, where we use the shorthand $\circ \xrightarrow{\mathtt{a}^n} \circ$ to denote a sequence of $n+1$ states and n $\mathtt{a}$-labeled transitions.

The set $\mathcal{H}$ of periodic time granularities that encode the set of valid therapies of the protocol is represented by the UPA $\mathcal{B}$ depicted in Figure 9.

Checking the consistency of GFR measurements for patient 1001 with respect to the prescribed protocol amounts to check whether granularity G is an aligned refinement of some granularity in $\mathcal{H}$. We can solve the latter problem by first building the UPA $\mathcal{D} = (\mathcal{A} \times \mathcal{B}) \cap \mathcal{C}_{\sim}$, where $\mathcal{C}_{\sim}$ is the Büchi automaton depicted in Figure 7, and then checking whether the recognized ω-language $\mathscr{L}^{\omega}(\mathcal{D})$ is non-empty. In our case, we easily see that $\mathcal{D}$ recognizes

the (singleton) ω-language

$$\left\{ (\begin{smallmatrix}\square\\\square\end{smallmatrix})^{100}(\begin{smallmatrix}\square\\\blacksquare\end{smallmatrix})^{15}(\begin{smallmatrix}\blacktriangleleft\\\blacksquare\end{smallmatrix})(\begin{smallmatrix}\square\\\blacksquare\end{smallmatrix})^{13}(\begin{smallmatrix}\square\\\blacktriangleleft\end{smallmatrix})(\begin{smallmatrix}\square\\\square\end{smallmatrix})^{245}(\begin{smallmatrix}\square\\\blacksquare\end{smallmatrix})^{29}(\begin{smallmatrix}\blacktriangleleft\\\blacktriangleleft\end{smallmatrix})(\begin{smallmatrix}\square\\\square\end{smallmatrix})^{335}(\begin{smallmatrix}\blacktriangleleft\\\blacksquare\end{smallmatrix})(\begin{smallmatrix}\square\\\blacksquare\end{smallmatrix})^{28}(\begin{smallmatrix}\square\\\blacktriangleleft\end{smallmatrix})\cdot \cdot(\begin{smallmatrix}\square\\\square\end{smallmatrix})^{335}(\begin{smallmatrix}\square\\\blacksquare\end{smallmatrix})^{18}(\begin{smallmatrix}\blacktriangleleft\\\blacksquare\end{smallmatrix})(\begin{smallmatrix}\square\\\blacksquare\end{smallmatrix})^{10}(\begin{smallmatrix}\square\\\blacktriangleleft\end{smallmatrix})\Big((\begin{smallmatrix}\square\\\square\end{smallmatrix})^{335}(\begin{smallmatrix}\square\\\blacksquare\end{smallmatrix})^{29}(\begin{smallmatrix}\square\\\blacktriangleleft\end{smallmatrix})\Big)^{\omega} \right\}.$$

Since the resulting language is non-empty, we can conclude that G is an aligned refinement of some granularity in $\mathcal{H}$ and thus the considered therapy satisfies the prescribed protocol.

7 Conclusions

In this paper, we developed a theory of regular ω-languages that consist of ultimately periodic words only and we provided it with an automaton-based characterization. Furthermore, we showed how well-known results coming from automata theory can be exploited to solve a number of basic problems about regular ω-languages of ultimately periodic words. In particular, we provided effective solutions to the problems of emptiness, membership, equivalence, inclusion, and size-optimization. We also showed that the proposed framework allows one to represent and to reason about sets of granularities that have possibly infinitely many different prefixes and a finite number of non-equivalent repeating patterns.

As it has been shown in Section 4, UPA-recognizable languages, which are a proper subclass of Büchi-recognizable ones, are not closed under complementation. In fact, for every UPA-recognizable language L, the complement $\bar{\mathsf{L}}$ of L is not UPA-recognizable. The relationships among UPA-recognizable languages, their complements, and Büchi-recognizable languages can be summarized as follows. Let us call *co-UPA-recognizable* language the complement $\bar{\mathsf{L}}$, with respect to the set U_Σ of all ultimately periodic words, of any UPA-recognizable language L. Any co-UPA-recognizable language features all, but finitely many, non-equivalent repeating patterns and, as pointed out in the proof of Proposition 5, it is not Büchi-recognizable. Moreover, for every pair of UPA-recognizable languages L_1 and L_2, the language $\mathsf{L}_1 \cap \bar{\mathsf{L}}_2$ $(= \mathsf{L}_1 \cap (\mathsf{U}_\Sigma \setminus \mathsf{L}_2) = \mathsf{L}_1 \cap (\Sigma^\omega \setminus \mathsf{L}_2))$ is UPA-recognizable as well. Analogously, the union of an UPA-recognizable language with a co-UPA-recognizable language is a co-UPA-recognizable language. This basically means that the class of languages that contains all UPA-recognizable languages and all their complements is the smallest class that includes all regular ω-languages of ultimately periodic words and is closed under finite unions, finite intersections, and complementation with respect to U_Σ. Moreover, it is straightforward to generalize the proposed solutions to the emptiness, membership, equivalence, and inclusion problems for UPA to solutions of analogous problems for co-UPA.

It is worth pointing out that there exist interesting connections between UPA-recognizable languages and other subclasses of regular ω-languages.

In particular, the class of UPA can be viewed as a proper subclass of non-deterministic co-Büchi (or safety) automata. Moreover, non-deterministic co-Büchi automata are known to be equivalent to their deterministic versions [25]. It thus follows that the languages recognized by UPA can be also recognized by a proper subclass of deterministic co-Büchi automata.

A natural development of the present work is the definition of a high-level logical language, e.g., a variant of propositional linear temporal logic [14], that allows one to represent all (co-)UPA-recognizable languages by means of suitable formulas. Besides its theoretical relevance, pairing such a logical framework with the proposed automaton-based one would allow one to use the former as a high-level interface for the specification of granularities and the latter as an internal formalism for efficiently reasoning about them.

References

1. Büchi, J.: On a decision method in restricted second order arithmetic. In: Proceedings of the International Congress for Logic, Methodology and Philosophy of Science, pp. 1–11. Stanford University Press (1962)
2. Bettini, C., Jajodia, S., Wang, X.: Time Granularities in Databases, Data Mining, and Temporal Reasoning. Springer (2000)
3. Booth, K.: Lexicographically least circular substrings. Information Processing Letters **10**(4-5), 240–242 (1980)
4. Bresolin, D., Montanari, A., Puppis, G.: Time granularities and ultimately periodic automata. In: Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA), *Lecture Notes in Artificial Intelligence*, vol. 3229, pp. 513–525. Springer (2004)
5. Brzozowski, J.: Canonical regular expressions and minimal state graphs for definite events. Mathematical Theory of Automata **12**, 529–561 (1962)
6. Calbrix, H., Nivat, M., Podelski, A.: Ultimately periodic words of rational ω-languages. In: Proceedings of the 9th International Conference on Mathematical Foundations of Programming Semantics, *Lecture Notes in Computer Science*, vol. 802, pp. 554–566. Springer (1994)
7. Campeanu, C., II, K.C., Salomaa, K., Yu, S.: State complexity of basic operations on finite languages. In: Proceedings of the 4th International Workshop on Implementing Automata (WIA), *Lecture Notes in Computer Science*, vol. 2214, pp. 60–70. Springer (1999)
8. Combi, C., Franceschet, M., Peron, A.: Representing and reasoning about temporal granularities. Journal of Logic and Computation **14**, 51–77 (2004)
9. Dal Lago, U., Montanari, A.: Calendars, time granularities, and automata. In: Proceedings of the 7th International Symposium on Spatial and Temporal Databases (SSTD), *Lecture Notes in Computer Science*, vol. 2121, pp. 279–298. Springer (2001)
10. Dal Lago, U., Montanari, A., Puppis, G.: Towards compact and tractable automaton-based representations of time granularity. In: Proceedings of

the 8th Italian Conference on Theoretical Computer Science (ICTCS), *Lecture Notes in Computer Science*, vol. 2841, pp. 72–85. Springer (2003)
11. Dal Lago, U., Montanari, A., Puppis, G.: Compact and tractable automaton-based representations for time granularities. Theoretical Computer Science **373**(1-2), 115–141 (2007)
12. Dal Lago, U., Montanari, A., Puppis, G.: On the equivalence of automaton-based representations of time granularities. In: Proceedings of the 14th International Symposium on Temporal Representation and Reasoning (TIME), pp. 82–93. IEEE Computer Society (2007)
13. Demri, S.: LTL over integer periodicity constraints. Theor. Comput. Sci. **360**(1-3), 96–123 (2006)
14. Emerson, E.: Temporal and modal logic. In: Handbook of Theoretical Computer Science, vol. B: formal models and semantics, pp. 995–1072. Elsevier MIT Press (1990)
15. Hopcroft, J., Motwani, R., Ullman, J.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley Longman Publishing Co., Inc. (2001)
16. Jiang, T., Ravikumar, B.: Minimal NFA problems are hard. SIAM Journal on Computing **22**(6), 1117–1141 (1993)
17. Kameda, T., Weiner, P.: On the state minimization of nondeterministic finite automata. IEEE Transactions on Computers **19**(7), 617–627 (1970)
18. Knuth, D., Morris, J., Pratt, V.: Fast pattern matching in strings. SIAM Journal on Computing **6**, 323–350 (1977)
19. Löding, C.: Optimal bounds for the transformation of omega-automata. In: Proceedings of the 19th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), *Lecture Notes in Computer Science*, vol. 1738, pp. 97–109. Springer (1999)
20. Löding, C., Thomas, W.: Alternating automata and logics over infinite words. In: Proceedings of the IFIP International Conference on Theoretical Computer Science (IFIP TCS), Exploring new frontiers of theoretical informatics, *Lecture Notes in Computer Science*, vol. 1872, pp. 521–535. Springer (2000)
21. Leban, B., McDonald, D., Foster, D.: A representation for collections of temporal intervals. In: Proceedings of the AAAI National Conference on Artificial Intelligence, vol. 1, pp. 367–371. AAAI Press (1986)
22. Loma Linda International Heart Institute: Pediatric heart transplantation protocol. Tech. rep., International Heart Institute, Loma Linda University Medical Center, Loma Linda, CA (2002). Available at: http://www.llu.edu/ihi/pedproto.pdf
23. Matz, O., Potthoff, A.: Computing small nondeterministic automata. In: Proceedings of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), BRICS Notes Series, pp. 74–88 (1995)
24. Michel, M.: Complementation is more difficult with automata on infinite words (1988). CNET, Paris, Manuscript
25. Miyano, S., Hayashi, T.: Alternating finite automata on ω-words. Theoretical Computer Science **32**, 321–330 (1984)

26. Muller, D., Schupp, P.: Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. Theoretical Computer Science **141**(1-2), 69–107 (1995)
27. Niezette, M., Stevenne, J.: An efficient symbolic representation of periodic time. In: Proceedings of the International Conference on Information and Knowledge Management (CIKM), pp. 161–168. Association for Computing Machinery (1992)
28. Ning, P., Jajodia, S., Wang, X.: An algebraic representation of calendars. Annals of Mathematics and Artificial Intelligence **36**, 5–38 (2002)
29. Paige, R., Tarjan, R., Bonic, R.: A linear time solution to the single function coarsest partition problem. Theoretical Computer Science **40**, 67–84 (1985)
30. Papadimitriou, C.: Computational Complexity. Addison-Wesley Longman Publishing Co., Inc. (1994)
31. Puppis, G.: Automata for branching and layered temporal structures. Ph.D. thesis, Department of Mathematics and Computer Science, Udine University, Udine, Italy (CS 2006/08). Available at: http://users.dimi.uniud.it/~gabriele.puppis/files/PhDThesis.pdf
32. Safra, S.: On the complexity of ω-automata. In: Proceedings of the 29th Annual Symposium on Foundations of Computer Science, pp. 319–327. IEEE Computer Society (1988)
33. Safra, S.: Complexity of automata on infinite objects. Ph.D. thesis, Weizmann Institute of Science, Rehovot, Israel (1989)
34. Shiloach, Y.: Fast canonization of circular strings. Journal of Algorithms **2**(2), 107–121 (1981)
35. Sistla, A., Vardi, M., Wolper, P.: The complementation problem for Büchi automata with applications to temporal logic. Theoretical Computer Science **49**, 217–237 (1987)
36. Wijsen, J.: A string-based model for infinite granularities. In: Proceedings of the AAAI Workshop on Spatial and Temporal Granularities, pp. 9–16. AAAI Press (2000)