Time Granularities and Ultimately Periodic Automata

Davide Bresolin, Angelo Montanari and Gabriele Puppis

Dipartimento di Matematica e Informatica,
Università di Udine
via delle Scienze 206, 33100 Udine, Italy

{bresolin,montana,puppis}@dimi.uniud.it

Abstract

The relevance of the problem of managing periodic phenomena is widely recognized in the area of knowledge representation and reasoning. One of the most effective attempts at dealing with this problem has been the addition of a notion of time granularity to knowledge representation systems. Different formalizations of such a notion have been proposed in the literature, following algebraic, logical, string-based, and automaton-based approaches. In this paper, we focus our attention on the automaton-based one, which allows one to represent a large class of granularities in a compact and suitable to algorithmic manipulation form. We further develop such an approach to make it possible to deal with (possibly infinite) sets of granularities instead of single ones. We define a new class of automata, called Ultimately Periodic Automata, we give a characterization of their expressiveness, and we show how they can be used to encode and to solve a number of fundamental problems, such as the membership problem, the equivalence problem, and the problem of granularity comparison. Moreover, we give an example of their application to a concrete problem taken from clinical medicine.

Contents

1 Introduction

The importance of managing periodic phenomena is widely recognized in a variety of applications in the areas of artificial intelligence and databases, including planning, natural language processing, temporal database inter-operability, data mining, and time management in workflow systems. One of the most effective attempts at dealing with this problem has been the addition of a notion of time granularity to knowledge and database systems. Different time granularities can be used to specify the occurrence times of different classes of events. For instance, the temporal characterizations of a flight departure, a business appointment, and a birthdate are usually given in terms of minutes, hours, and days, respectively. Furthermore, the ability of properly relating different time granularities is needed to process temporal information. As an example, when a computation involves pieces of information expressed at different time granularities, the system must integrate them in a principled way. Such an integration presupposes the formalization of the notion of granularity and the analysis of the relationships between different time granularities.

According to a commonly accepted perspective [BJW00], 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). 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. Even though conceptually clean, this point of view does not address the problem of finitely (and compactly) representing granularities in order to make it possible to deal with them in an effective (and efficient) way. To be computationally acceptable, any formal system should indeed satisfy the following requirements:

- **Expressiveness.** The set of all possible granularities is not countable. Thus, every representation formalism is bound to be incomplete. The class of granularities captured by the formalism should be large enough to be of practical interest.

- **Effectiveness.** The formalism must provide infinite granularities with a finite representation. Data structures, which are used to store the information, should ease access to and manipulation of time granularities.

- **Compactness.** The formalism should exploit regularities exhibited by the considered structures in order to make their representation as compact as possible.

In this paper, we outline a general framework for time granularity, that refines and extends the automaton-based approach originally proposed by Dal Lago and Montanari in [DM01]. In particular, we show how to generalize it from single periodical granularities to (possibly infinite) sets of granularities. To this end, we identify a proper subclass of Büchi automata, called Ultimately Periodic Automata (UPA), that captures regular sets consisting of only ultimately periodic words. UPA allow one to encode single granularities, (possibly infinite) sets of granularities which have the same repeating pattern and different prefixes, and (possibly infinite) sets of granularities characterized by a finite set of non-equivalent patterns (the notion of equivalent patterns is formally defined in Section 4), as well as any possible combination of them.

This paper is organized as follows. Section 2 recalls some basic concepts from strings and automata theory, that will be useful in the rest of the paper. In Section 3, we briefly survey the most relevant formalisms for time granularity proposed in the literature. Next, in Section 4, we exploit a characterization of ω-regular languages of ultimately periodic words, we define UPA, and we use them to represent sets of periodical granularities. Successively, in Section 5, we show how to explore noticeable properties of (languages recognized by) UPA to address a number of crucial problems about sets of time granularities:

- **Emptiness**. It consists in deciding whether a given set of granularities is empty;
- **Membership**. It consists in deciding whether a granularity belongs to a given set of granularities;
- **Equivalence**. It consists in deciding whether two representations define the same set of granularities;
- **Inclusion**. It consists in deciding whether a given set of granularities is included in another one.

Section 6 is dedicated to our solution to the equivalence problem for UPA, while in Section 7 we show how to produce compact representation for time granularities. In Section 8 we explore a meaningful class of problems, the *granularity comparison problems*. Given two sets $\mathcal{G}, \mathcal{H}$ of granularities, a granularity comparison problem consists in deciding whether there exist $G \in \mathcal{G}$ and $H \in \mathcal{H}$ such that $G \sim H$, where $\sim$ is one of the usual relationships between granularities, such as "finer than", "group", and "partition" [BJW00]. Successively, in Section 9, we analyze variants and extensions of UPA, in order to increase expressiveness of granularity representations. Finally, we show how to apply the proposed framework to a real-world application taken from clinical medicine, where the ability of managing sets of time granularities is exploited to specify and to validate therapy plans. We conclude the paper with a short discussion about achieved results and future research directions.

2 Background Knowledge

In this section, we introduce some basic notations and results about formal languages and automata.

A *finite word* (or *finite string*) over a finite *alphabet* A is a finite (possibly empty) sequence of symbols, belonging to A. We denote by A^* the set of all the finite words over A; $\varepsilon \in A^*$ is the empty string and A^+ stands for the set $A^* \setminus \{\varepsilon\}$ of all non-empty finite strings over A.

By analogy, an *infinite word* (or *infinite string*, or ω-*word*) over A is an infinite sequence of symbols taken from A. A^ω denotes the set of all the infinite strings over A and $A^\diamond$ is simply the set $A^* \cup A^\omega$. The function $| \ | : A^\diamond \to \mathbb{N} \cup \{\omega\}$ returns the length of a string (by convention, $|u| = \omega$ for every ω-word u). The parametric operator $\#_a : A \to \mathbb{N} \cup \{\omega\}$ returns the number of occurrences of the symbol a in a word $u \in A^\diamond$. For instance, $\#_b(abcabbc) = 3$. The following notations will be broadly used in the following:

$$u(i) = u_i \qquad u[i,j] = \begin{cases} u_i \dots u_j & \text{if } i \leq j, \\ \varepsilon & \text{otherwise;} \end{cases}$$

$$u[i,\omega[= u_i u_{i+1} \dots \qquad u[i,j[= \begin{cases} u_i \dots u_{j-1} & \text{if } i < j, \\ \varepsilon & \text{otherwise,} \end{cases}$$

where $u = u_1 u_2 u_3 \dots$ is a word in $A^\diamond$ and i and j are two positive integers.

The concatenation operator $\cdot$ can be easily extended from A^* to $A^\diamond$ by assuming $u \cdot v = u$ for every $u \in A^\omega$ and $v \in A^\diamond$. We will also use the following notations:

- $v^0 = \varepsilon$ and $v^k = v^{k-1} \cdot v$ for every $k \geq 1$;
- $v^\omega = v \cdot v \cdot v \cdot \dots$;
- $V^0 = \{\varepsilon\}$;
- $V^n = \{v_1 \cdot v_2 \cdot \dots \cdot v_n | \forall i \in [1,n].v_i \in V\}$;
- $V^* = \bigcup_{n \in \mathbb{N}} V^n$;
- $V^\omega = \{v_1 \cdot v_2 \dots | \forall i \in [1,\omega[.v_i \in V\}$,

where $v \in A^*$ and $V \subseteq A^*$.

If w is a word in $A^\diamond$ and there exists a finite word v such that $w = v^k$, with $k(\in \mathbb{N}) \geq 2$ or $k = \omega$, then w is said to be *periodic*, v is called a *pattern* of w, and $|v|$ is called the *period* of w. A pattern p is said to be *primitive* iff for every non-empty word $v \in A^*$, $p = v^k \Rightarrow k = 1$. The following proposition relates the different periods of a word.

Proposition 2.1 (Fine-Wilf [FW65]). *A word v with periods q and q' and length $|v| \leq q + q' - GCD(q,q')$ has period $GCD(q,q')$.*

An infinite word w is said to be *ultimately periodic* iff there exist two finite words u and v such that $w = u \cdot v^\omega$; u is said to be a *prefix* of w and v is said to be a *pattern* of w. Furthermore, a *period* of w is the length of a pattern of w.

2.1 Büchi automata and ω-regular languages

We now present some basic results coming from automata theory. For a complete discussion on Büchi automata and ω-regular languages, refer to [Tho90].

Definition 2.2. A *finite automaton* over an alphabet A is a tuple $\mathcal{A} = (Q, \Delta, q_0, F)$, where:

- Q is a finite set of states;
- $\Delta \subseteq Q \times A \times Q$ is a transition relation;
- $q_0 \in Q$ is an initial state;
- $F \subseteq Q$ is a set of final states.

A *run* of $\mathcal{A}$ over a word (finite or infinite) $w = w(1)w(2)\dots$ is a sequence of states $\rho = \rho(1)\rho(2)\dots$ such that $(\rho(i), w(i), \rho(i+1)) \in \Delta$ for each $1 \le i \le |w|$ if w is finite, and for each $i \ge 1$ if w is infinite.

If w is a non-empty finite word and $p, q \in Q$, we'll write $p \xrightarrow[\mathcal{A}]{w} q$ iff there exists a run ρ of $\mathcal{A}$ over w that starts in p and ends in q. If ρ contains a final state we'll write $p \overset{w}{\underset{\mathcal{A}}{-\!\!\circ\!\!\rightarrow}} q$, otherwise we'll write $p \overset{w}{\underset{\mathcal{A}}{-\!\!\sim\!\!\rightarrow}} q$. Given two states of $\mathcal{A}$, p and q, $p \rightarrow^* q$ means that the automaton can reach the state q starting from p, namely, there exists a (possibly empty) word v such that $p \xrightarrow{v} q$. With $p \rightarrow^+ q$ we indicate that there exists a *non-empty* word v such that $p \xrightarrow{v} q$.

A run $\rho = \rho(1)\dots\rho(n)$ of $\mathcal{A}$ over a finite word w is said to be *accepting* if $\rho(1) = q_0$ and if $\rho(n) \in F$. The automaton $\mathcal{A}$ *accepts* a finite word w if there is an accepting run of $\mathcal{A}$ over w. The *language* $\mathscr{L}(\mathcal{A})$ *recognized by* $\mathcal{A}$ is the language of all the finite words accepted by $\mathcal{A}$. A language $L \subseteq A^*$ is said to be *regular* iff there exists some automaton $\mathcal{A}$ such that $L = \mathscr{L}(\mathcal{A})$.

When considering languages of infinite words, $\mathcal{A}$ is called a *Büchi automaton*, and a run of $\mathcal{A}$ over an ω-word w is said to be *accepting* if $\rho(1) = q_0$ and if there exists a final state $f \in F$ that occurs infinitely often in ρ. The automaton $\mathcal{A}$ *accepts* an ω-word w if there is an accepting run of $\mathcal{A}$ over w. The *language* $\mathscr{L}^\omega(\mathcal{A})$ *recognized by* $\mathcal{A}$ is the language of all the ω-words accepted by $\mathcal{A}$. Any language of infinite words recognized by a Büchi automaton is said to be *ω-regular*.

We now recall some relevant properties of Büchi automata and ω-regular languages.

Lemma 2.3. (a) *If $V \subseteq A^+$ is a regular language, then $V^\omega \subseteq A^\omega$ is a ω-regular language.*

(b) *If $U \subseteq A^*$ is regular and $V \subseteq A^\omega$ is ω-regular, then $U \cdot V$ is ω-regular.*

(c) *If $L_1, L_2 \subseteq A^\omega$ are ω-regular, then $L_1 \cap L_2$ and $L_1 \cup L_2$ are ω-regular.*

(d) *If $L \subseteq A^\omega$ is ω-regular, then $\overline{L} = A^\omega \setminus L$ is a ω-regular language.*

Proof. (a) Since $V^\omega = (V \setminus \{\varepsilon\})^\omega$, we can assume that V does not contain the empty word. We further assume that there are no transitions reaching the initial state q_0 of a finite automaton $\mathcal{A}$ recognizing V. A Büchi automaton $\mathcal{A}'$ that recognizes V^ω can be obtained from $\mathcal{A}$ by adding a transition (q, a, q_0) for each transition (q, a, q') with $q' \in F$, and by defining q_0 to be the only final state of $\mathcal{A}'$. Note that every accepting run of $\mathcal{A}'$ contains infinite occurrences of q_0 and, between two consecutive occurrences of q_0, the automaton reads exactly a word belonging to V.

For the proofs of claims (b), (c) and (d) see [Tho90]. □

Theorem 2.4. *An ω-language $L \subseteq A^\omega$ is recognized by a Büchi automaton iff it is a finite union of sets of the form $U \cdot V^\omega$, with $U \subseteq A^*$ and $V \subseteq A^+$ being regular languages of finite words.*

The above theorem is the starting point for a characterization of ω-regular languages of ultimately periodic words. The following results allows us to identify an ω-regular language with the set $\mathrm{UP}(L)$ of its ultimately periodic words.

Proposition 2.5. (a) *Every non-empty ω-regular language contains at least one ultimately periodic word.*

(b) *If L_1, L_2 are ω-regular languages, then $\mathrm{UP}(L_1) = \mathrm{UP}(L_2)$ iff $L_1 = L_2$.*

Proof. (a) By Theorem 2.4, any ω-regular language L can be written as

$$\bigcup_{i=1}^{n} U_i \cdot V_i^{\omega}.$$

Since L is not empty, it contains a word $w = u \cdot v_1 \cdot v_2 \cdot \ldots$, with $u \in U_i$ and $v_1, v_2, \ldots \in V_i$ for a suitable index i. Thus the ultimately periodic word $w' = u \cdot v_1 \cdot v_1 \cdot \ldots$ is also a word of L.

(b) If L_1, L_2 are ω-regular languages containing the same ultimately periodic words, then by Lemma 2.3 $(L_1 \setminus L_2) \cup (L_2 \setminus L_1)$ is an ω-regular language that does not contains any ultimately periodic word. Thus $(L_1 \setminus L_2) \cup (L_2 \setminus L_1)$ is empty and $L_1 = L_2$ follows. The converse implication follows trivially. $\square$

3 Formal Systems for Time Granularity

In the database literature, there exists a widespread recognition of the relevance of the problem of properly managing temporal information associated with different time granularities, e.g., [CSS94]. In particular, such a problem has been extensively dealt with in temporal databases [WJS95, WJL91]. As an example, a significant component of TSQL2, the major proposal for a temporal extension of SQL [SAA$^+$94], is devoted to the management of time granularity. TSQL2 allows one to define quite expressive granularity structures, including granularities with temporal gaps and incomparable granularities, and it features some primitive operations, such as SCALE and CAST, that support conversions from a given granularity to a coarser/finer one. A significant limitation of most these systems, however, is the lack of a general framework for time granularity.

Some attempts at overcoming such a weakness have been pursued during the last years. A detailed presentation of the set-theoretic/algebraic approach to time granularity, that subsumes well-known formalisms for time granularity developed in the areas of artificial intelligence and temporal databases, such as the temporal interval collection formalism [LMF86] and the slice formalism [NS92], is given in [BJW00]. Assuming the temporal domain to be isomorphic to $\mathbb{N}$ (this is the assumption of most applications), a *time granularity* can be defined as a partition $G \subseteq 2^T$ of a set $T \subseteq \mathbb{N}$ such that distinct sets in G (hereafter called *granules*) do not overlap, namely, for every pair of distinct granules $g, g' \in G$, we have that either $\forall\, t \in g.\ \forall\, t' \in g'.\ t < t'$, or $\forall\, t \in g.\ \forall\, t' \in g'.\ t' < t$. This definition captures both time granularities that cover the whole temporal domain, such as `Day`, `Week`, and `Month`, and time granularities with gaps within and between granules, like, for instance, `BusinessDay`, `BusinessWeek`, and `BusinessMonth`. Figure 1 depicts some of these granularities.

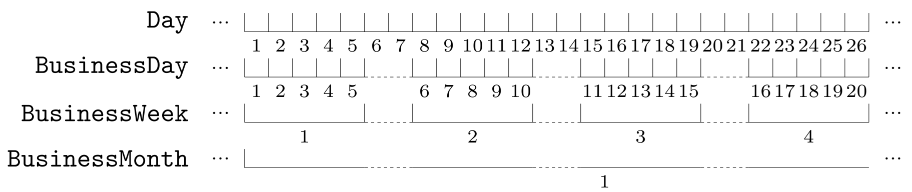

Figure 1: Some examples of time granularities.

Most formalisms proposed in the literature follow the set-theoretic approach to time granularity, originally outlined by Clifford and Rao [CR88], as a suitable way of structuring information with a temporal dimension, independently of any particular calendric system. The proposed framework consists of a finite, ordered set of temporal structures built upon some discrete, infinite, linearly ordered, temporal domain, whose elements are the smallest observable/interesting time units. Such an approach has been later extended and refined by several authors (a comprehensive survey can be found in [BJW00]).

A symbolic representation of a significant class of time granularities has been obtained by means of a *Calendar Algebra* [NJW02]. Such a formalism represents time granularities as expressions built up from a finite set of basic granularities through the application of suitable algebraic operators. For instance, the granularity `Week` can be generated by applying the operation $Group_7$ to the granularity `Day`. Since the operations of Calendar Algebra reflect the ways in which people define new granularities from existing ones, such a formalism turns out to be fairly natural. In despite of that, Calendar Algebra suffers from some non-trivial drawbacks [FM02]. Some basic problems of obvious theoretical and practical importance, such as the equivalence problem, have not been addressed satisfactorily. Other problems, such as the problem of supporting granularity conversion, have been only partially worked out in a rather complex way.

A string-based model for time granularities has been proposed by Wijsen [Wij00]. Infinite granularities are modeled as infinite words over an alphabet consisting of three symbols, namely, $\blacksquare$ (filler), $\square$ (gap), and $\wr$ (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. He focuses his attention on the class of periodical granularities, that is, to granularities that, ultimately, periodically groups instants of the underlying temporal domain. Periodical granularities can be identified with ultimately periodic words, and they can be finitely represented by specifying a (possibly empty) prefix and a repeating pattern. As an example, the granularity `BusinessWeek` $\blacksquare\blacksquare\blacksquare\blacksquare\blacksquare\square\square \wr \blacksquare\blacksquare\blacksquare\blacksquare\blacksquare\square\square \wr \ldots$ can be encoded by the empty prefix ε and the repeating pattern $\blacksquare\blacksquare\blacksquare\blacksquare\blacksquare\square\square\wr$. In order to solve the equivalence problem for string-based representations of time granularities, Wijsen proposes a suitable *aligned form*, which forces separators to occur immediately after an occurrence of $\blacksquare$ (in such a case, one can encode each occurrence of the substring $\blacksquare\wr$ by means of a single symbol $\blacktriangleleft$). The aligned form guarantees a one-to-one correspondence between strings and granularities, thus providing a straightforward solution to the equivalence problem. Moreover, the string-based formalism allow us to easily extend the definition of time granularity, associating to each granule a unique *index* $i \in \mathbb{N}$ in the following way. Given a granularity G represented by the word $g \in \{\square, \blacksquare, \blacktriangleleft\}^\omega$, we say that a time instant t belongs to the i-th granule of G, denoted $G(i)$, iff $g(t) \in \{\blacksquare, \blacktriangleleft\}$ and $\#_{\blacktriangleleft}(g[1, t-1]) = i - 1$.

The idea of viewing time granularities as periodic strings establishes a natural connection with the field of formal languages and automata. The basic idea underlying the automaton-based approach to time granularity is simple: we take an automaton $\mathcal{A}$ recognizing a *single* word $u \in \{\blacksquare, \square, \blacktriangleleft\}^\omega$ and we say that $\mathcal{A}$ represents the granularity G if and only if u represents G. Such an automaton-based approach to time granularity has been proposed by Dal Lago and Montanari in [DM01], and later revisited by Dal Lago, Montanari, and Puppis in [DMP03a, DMP03b]. The resulting framework views granularities 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. In order to compactly encode the redundancies of the temporal structures, SSA are endowed with counters ranging over discrete finite domains. Properties of SSA have been exploited to efficiently solve the equivalence and the granule conversion problems for single time granularities. The relationships between the automaton-based and the algebraic approaches to time granularity have been investigated in [DMP03a], where a number of algorithms that map Calendar Algebra expressions into automaton-based representations of time granularities are given. Such an encoding allows one to reduce problems about Calendar Algebra expressions to equivalent problems for SSA. More generally, the operational flavor of SSA suggests an alternative point of view on the role of automaton-based representations: besides a formalism for the direct specification of time granularities, SSA can be viewed as a low-level formalism into which high-level time granularity specifications, such as those of Calendar Algebra, can be mapped.

A logical framework for time granularity, based on a many-level view of temporal structures, with matching logics and decidability results, has been proposed in [Mon96] and later extended in [Fra01]. Temporal logics for time granularity pair the usual temporal logic operators with contextualization and projection operators to select and to move across granularities. They have been interpreted over a variety of (possibly infinite) layered temporal structures, and their expressive power has been systematically studied. The choice of Propositional Linear Temporal Logic (PLTL) as a logical tool for granularity management has been recently advocated by Combi et al. in [CFP04]. Time granularities are defined as models of PLTL formulas, where suitable propositional symbols are used to mark the endpoints of granules. In this way, a large set of regular granularities, such as, for instance, repeating patterns that can start at an arbitrary time point, can be captured. Moreover, problems like checking the consistency of a granularity specification or 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 extension of LTL that replaces propositional variables by first-order formulas defining integer constraints, e.g., $x \equiv_k y$, has been proposed by Demri [Dem04]. The resulting logic, denoted by PLTL^{mod}(Past LTL with integer periodicity constraints), generalizes both the logical framework proposed by Combi et al. and Dal Lago and Montanari's automaton-based one, and it allows one to compactly define granularities as periodicity constraints. In particular, the author shows how to reduce the equivalence problem for ESSA to the model checking problem for PLTL^{mod}(-automata), which turns out to be in PSPACE, as for LTL.

4 Ultimately Periodic Automata

In the previous section we introduced an automaton-based approach for representing and reasoning about time granularities. The described framework allows one to represent a single granularity at a time. However, in many cases we are required to deal with (possibly infinite) sets of granularities.

In this section we propose an automaton-based approach for representing *sets of granularities*, starting from the investigation of properties of ω-regular languages consisting of only ultimately periodic words. Afterwards, we define an appropriate subclass of Büchi automata that captures all the ω-regular languages of ultimately periodic words.

4.1 Languages of ultimately periodic words

In this section we provide a characterization of ω-regular languages consisting of only ultimately periodic words, namely, of ω-regular languages L such that $L = \mathrm{UP}(L)$.

4.1.1 Closure properties

Here we investigate several closure properties for the class $\mathcal{F}$ of ω-regular languages of ultimately periodic words. It is worth mentioning that there exists some languages of ultimately periodic words that are not ω-regular: for instance the language

$$A^{UP} = \{u \cdot v^\omega | u, v \in A^*\},$$

of all the ultimately periodic words over a given alphabet turns out to be not ω-regular (Proposition 4.1).

Proposition 4.1. *The language A^{UP} of all ultimately periodic words is not ω-regular.*

Proof. It's easy to see that $\mathrm{UP}(A^{UP}) = \mathrm{UP}(A^\omega)$. By Proposition 2.5 follows that A^{UP} couldn't be ω-regular (because $A^{UP} \neq A^\omega$). $\square$

Proposition 4.2. (a) *If $U \subseteq A^*$ is regular and $V \in \mathcal{F}$, then $U \cdot V$ is an ω-regular language of ultimately periodic words.*

(b) *If $U, V \in \mathcal{F}$, then $U \cup V$ and $U \cap V$ are ω-regular languages of ultimately periodic words.*

We can redefine complementation and ω-power as follows:

$$\begin{cases} \overline{U} = A^{UP} \setminus U, & \text{for any } U \subseteq A^{UP} \\ V^{inf} = \{v^\omega | v \in V\}, & \text{for any } V \in A^*. \end{cases}$$

Such a restriction implies that the complement and the ω-power of a given language consist of only ultimately periodic words. Nevertheless, this does not suffice to guarantee that $\overline{U}$ and V^{inf} are ω-regular languages, even with the proviso that U is ω-regular and V is regular. This is the claim of the following proposition.

Proposition 4.3. *ω-regular sets of ultimately periodic words are not closed under (redefined operations of) complementation and ω-power.*

Proof. By definition, $A^{UP} = U \cup \overline{U}$ and $A^{UP} = A^* \cdot (A^+)^{inf}$ hold. Since $\mathcal{F}$ is closed under union and under concatenation with regular sets and from the fact that A^{UP} is not ω-regular, it follows that $\mathcal{F}$ cannot be closed under complementation and ω-power. $\square$

4.1.2 Equivalences of repeating patterns

The set of all the possible patterns of an ultimately periodic word is infinite, so it's useful to define an appropriate equivalence relation that groups together all the patterns of the same word. In this way it's possible to reason about the index of the equivalence relation in order to characterize regular languages of ultimately periodic words.

Definition 4.4. We define the equivalence relation $\cong \subseteq A^* \times A^*$ such that $u \cong v$ iff the ultimately periodic words u^ω and v^ω have at least one common pattern, namely, iff there exist finite words $x, y, z \in A^*$ such that $u^\omega = x \cdot z^\omega$ and $v^\omega = y \cdot z^\omega$.

Two patterns p_1 and p_2 are said to be *non-equivalent* iff $p_1 \ncong p_2$.

Proposition 4.5. *The patterns of an ultimately periodic word are $\cong$-equivalent.*

Proof. Let w be an ultimately periodic word, and p_1, p_2 be two patterns of w. By definition, there exist two prefixes u and v of w such that $w = u \cdot p_1^\omega = v \cdot p_2^\omega$ and $|u| \leq |v|$.

Let $v = v_1 \cdot v_2$, with $|v_1| = |u|$. Thus, w is of the form $w = u \cdot p_1^\omega = v_1 \cdot v_2 \cdot p_2^\omega$ and $p_1^\omega = v_2 \cdot p_2^\omega$. Therefore, p_2 is a pattern of both p_1^ω and p_2^ω, thus proving that $p_1 \cong p_2$. ☐

Definition 4.6. Given an ω-regular language L, the set P of *all non-equivalent patterns of* L is the $\cong$-quotient set of all the possible patterns of the ultimately periodic words belonging to L.

Now we investigate several properties of $\cong$-equivalent patterns. In particular, we study the ways how two equivalent patterns could differ. Intuitively, if w is a ultimately periodic word, we can build different patterns of w in two ways:

1. by choosing different prefixes of w;
2. by choosing different repetitions of the primitive pattern (note that if p is a pattern of w, then $p^2, p^3, \ldots, p^n, \ldots$ are patterns, as well).

Definition 4.7. Two distinct patterns p_1 and p_2 are said to be *reducible by alignment* if p_1^ω is a proper suffix of p_2^ω or vice versa (i.e. if $p_1^\omega \neq p_2^\omega$ and there exists v such that $p_2^\omega = v \cdot p_1^\omega$).

Definition 4.8. Two distinct patterns p_1 and p_2 are said to be *reducible by multiplicity* if there exist two positive integers n and m such that $p_1^n = p_2^m$.

The following theorem connects the previous definitions with the notion of $\cong$-equivalence.

Theorem 4.9. *Two distinct patterns p_1 and p_2 are $\cong$-equivalent iff they are reducible either by alignment or by multiplicity.*

Proof. It is easy to see that if two patterns p_1 and p_2 are reducible by alignment or by multiplicity then they are $\cong$-equivalent:

- If they are reducible by alignment, assume that p_1^ω is a proper suffix of p_2^ω. Then p_1 is both a pattern of p_1^ω and of p_2^ω, proving that $p_1 \cong p_2$.

- If they are reducible by multiplicity, then $p_1^\omega = p_2^\omega$ and $p_1 \cong p_2$.

Let p_1 and p_2 be two distinct equivalent patterns. The words p_1^ω and p_2^ω have at least one common pattern x: let u and v satisfy $p_1^\omega = u \cdot x^\omega$ and $p_2^\omega = v \cdot x^\omega$. Let k be the least integer such that $k \cdot |p_1| \geq |u|$. Then, there exists a suitable word y such that

$$p_1^k = u \cdot y,\ x^\omega = y \cdot p_1^\omega,$$

and p_1 is a pattern of x^ω. But p_1 is also a pattern of p_2^ω, so

$$p_2^\omega = v \cdot x^\omega = v \cdot y \cdot p_1^\omega.$$

Now two cases arise:

- $p_1^\omega \neq p_2^\omega$, and hence p_1^ω is a proper suffix of p_2^ω and the patterns are reducible by alignment;
- $p_1^\omega = p_2^\omega$, and hence there exist two positive integers n, m such that $n \cdot |p_1| = m \cdot |p_2|$, $p_1^n = p_2^m$ and the patterns are reducible by multiplicity.

□

4.1.3 Patterns of V^ω-languages

Due to the structure of ω-regular languages (which are finite unions of sets of the form $U \cdot V^\omega$), each pattern featured by an ω-regular language belongs to some set V^ω, with V being a suitable regular language. In this section we characterize the languages of the form V^ω that contain only ultimately periodic words.

We introduce some properties about patterns of V^ω-languages.

Lemma 4.10. *If $L = V^\omega$ is an ω-regular language that contains the ultimately periodic word $u \cdot v^\omega$, then L contains also the ultimately periodic word w^ω, with v and w being reducible by alignment.*

Proof. We start from a Büchi automaton $\mathcal{A}$ that recognizes L (such an automaton can be generated from the automaton recognizing V, as shown in Lemma 2.3). We can further assume that the initial state is the only final state.

Let ρ be an accepting run of $\mathcal{A}$ over $u \cdot v^\omega$. Such ρ is of the form:

$$q_0 \xrightarrow{u(1)} q_1 \xrightarrow{u(2)} \dots \xrightarrow{u(|u|)} q_{|u|} \xrightarrow{v(1)} q_{|u|+1} \xrightarrow{v(2)} \dots$$

Consider now the suffix $\sigma = \rho[|u|, \infty[$ of ρ. Because ρ is an accepting run, there exist infinitely many occurrences of q_0. Let j be the position of the first occurrence of q_0 in σ. The infinite suffix $\sigma[j, \infty[$ is thus an accepting run of $\mathcal{A}$ and there exists k such that $1 \leq k \leq |v|$ and:

$$\sigma[j, \infty[= p_j \xrightarrow{v(k)} \dots \xrightarrow{v(|v|)} p_{j+|v|+1} \xrightarrow{v(1)} \dots \xrightarrow{v(k)} p_{j+|v|+k} \xrightarrow{v(k)} \dots$$

Therefore, $\sigma[j, \infty[$ recognizes the ultimately periodic word

$$w^\omega = (v[k, |v|] \cdot v[1, k-1])^\omega,$$

and the patterns u and w are reducible by alignment. □

Definition 4.11. Let $L = V^\omega$ and let P be the set of non-equivalent patterns of L: a word $w \in P$ is said to be an *aligned witness* iff $w^\omega \in L$.

Corollary 4.12. *Let $L = V^\omega$ and let P be the set of non-equivalent patterns of L: for each equivalence class of P there exists at least one aligned witness.*

Lemma 4.13. *Let $L = V^\omega$ and let $P_A = \{p_1, p_2, \ldots\}$ be a set of aligned witnesses of L. Then, for each $p \in P_A$, there exist $v_1, v_2, \ldots, v_k \in V$ such that $p \cong v_1 \cdot v_2 \cdot \ldots \cdot v_k$.*

Proof. We start from a Büchi automaton $\mathcal{A}$ recognizing L. Such $\mathcal{A}$ has a unique final state q_0, which is also the initial state. Furthermore, it reads a word from V between every pair of consecutive occurrences of q_0.

Let $p \in P_A$ be an aligned witness and let $\rho = q_0 q_1 q_2 \ldots$ be an accepting run of $\mathcal{A}$ over p^ω. Consider the states of ρ from which the automaton reads the initial symbol of p: they define the infinite sequence $\rho_{|p|} = q_0 q_{|p|} q_{2|p|} \cdots q_{n|p|} \cdots$.

Let q be a state occurring infinitely many times in $\rho_{|p|}$ and let i, j be distinct integers such that:

1. $q_{i|p|}$ is the first occurrence of q in $\rho_{|p|}$;
2. $q_{j|p|}$ is the first occurrence of q following $q_{i|p|}$ such that there exists at least one occurrence of the final state q_0 in $\rho[i|p|, j|p|]$.

Consider now $\rho[i|p|, j|p|]$: it recognizes the pattern p^{j-i}, which is reducible to p by multiplicity. Let l be the first occurrence of q_0 in $\rho[i|p|, j|p|]$: then

$$\sigma = \rho[l, j|p|] \cdot \rho[i|p| + 1, l]$$

starts and ends with the state q_0 and it recognizes a pattern p_σ of p^ω which is reducible to p^{j-i} by alignment. Moreover there exist k words in V (one for each occurrence of q_0) such that $p_\sigma = v_1 \cdot v_2 \cdot \ldots \cdot v_k$. □

Definition 4.14. Let be $L = V^\omega$ and let be P the set of distinct patterns of L: a witness $p \in P$ *respects* V iff there are $v_1, v_2, \ldots, v_k \in V$ such that $p = v_1 \cdot v_2 \cdot \ldots \cdot v_k$.

Corollary 4.15. *Let be $L = V^\omega$ and let be P the set of non-equivalent patterns of L. For each equivalence class of P there exists at least one witness that respects V.*

The last two results allow us to establish sufficient conditions for the existence of non ultimately periodic words in languages of the form V^ω.

Theorem 4.16. *Every ω-regular language of the form V^ω that features at least two non-equivalent patterns contains at least one non ultimately periodic word.*

Proof. Let be $L = V^\omega$ and let be $P_V = \{p_1, p_2, \ldots\}$ a set that contains, for each equivalence class of $\cong$, a witness that respects V.

P_V contains at least two non-equivalent patterns, say p_1 and p_2. It's easy to see that the ω-word

$$w = p_1 \cdot p_2 \cdot (p_1)^2 \cdot (p_2)^2 \cdot \ldots \cdot (p_1)^k \cdot (p_2)^k \cdot (p_1)^{k+1} \cdot (p_2)^{k+1} \cdot \ldots$$

is a word of L that is not ultimately periodic. □

The proof of the previous theorem suggests that a language that features at least two non-equivalent patterns actually contains infinitely many non-equivalent patterns built up by composition.

Lemma 4.17. *Every ω-regular language of the form V^ω that features at least two non-equivalent patterns features infinitely many non-equivalent patterns.*

Proof. Let be $L = V^\omega$ and let be $P_V = \{p_1, p_2, \ldots\}$ a set that contains, for each equivalence class of $\cong$, a pattern that respects V.

Then L contains also the infinite series of non-equivalent patterns:

$$\begin{array}{c} p_1 \cdot p_1 \\ (p_1)^2 \cdot (p_2)^2 \\ (p_1)^3 \cdot (p_2)^3 \\ \ldots \\ (p_1)^n \cdot (p_2)^n \\ \ldots \end{array}$$

□

Theorem 4.18. *Every ω-regular language of the form V^ω featuring only equivalent patterns is a singleton of the form $\{v^\omega\}$.*

Proof. Since V^ω features only equivalent patterns, for every pair of words $v_i, v_j \in V$, we have $v_i \cong v_j$. Hence, two words in V must be reducible one to each other either by alignment or by multiplicity.

We show by contraposition that such words are indeed reducible by multiplicity. So, suppose that $v_1, v_2 \in V$ are reducible by alignment. Then, there exists a minimal word u such that

$$v_2^\omega = u \cdot v_1^\omega.$$

Let n, m be such that $n \cdot |v_1| = m \cdot |v_2| > |u|$, and let $w_1 = v_1^n$ and $w_2 = v_2^n$. Clearly, w_1 and w_2 are reducible by alignment and we have

$$w_2^\omega = u \cdot w_1^\omega.$$

Then w_1 could be obtained from w_2 by rotation, namely there exists a word x such that

$$w_2 = u \cdot x \text{ and } w_1 = x \cdot u.$$

Consider now the ultimately periodic word $(w_1 \cdot w_2)^\omega = (x \cdot u \cdot u \cdot x)^\omega$: it belongs to L, but its pattern $x \cdot u \cdot u \cdot x$ is not equivalent neither to w_1 nor to w_2, in contradiction with the hypothesis of L featuring only equivalent patterns.

Thus, the words of V are reducible by multiplicity: by Fine-Wilf's Lemma (Lemma 2.1) there exists a pattern v of length $|v| = GCD(|v_1|, |v_2|, \ldots)$ such that, for every $v_i \in V$, $v_i = v^k$ for a suitable k. So, every possible infinite sequence of words of V is an infinite repetition of the primitive pattern v, and hence L consists of only a word of the form v^ω. □

In the next section, we generalize the previous results to ω-regular languages.

4.1.4 General characterization

ω-regular languages are characterized by finite unions of languages of the form $U \cdot V^\omega$, so the patterns of the ultimately periodic words are generated by the corresponding V^ω-components. The results of the previous section allow us to characterize ω-regular languages of ultimately periodic words in terms of ω-regular expressions.

Theorem 4.19. *Every ω-regular language for which there are infinitely many non-equivalent patterns contains at least one non ultimately periodic word.*

Proof. Let be $L = \bigcup_{i=1}^n U_i \cdot V_i^\omega$ an ω-regular language such that the equivalence relation over patterns $\cong$ has infinite index. Since patterns are generated by the V^ω-components, there is a component V_i^ω that contains infinitely many non-equivalent patterns and, by Theorem 4.16, it contains at least one non ultimately periodic word w. Given a word $u \in U_i$, the word $u \cdot w \in L$ is not ultimately periodic. □

Theorem 4.20 (Charachterization of ω-regular languages of ultimately periodic words). *An ω-regular language L contains only ultimately periodic words iff it contains a finite number of non-equivalent patterns, i.e. iff it can be defined by an ω-regular expression of the form $\bigcup_{i=1}^n U_i \cdot \{v_i\}^\omega$.*

Proof. The left-to-right implication is easily proved by contraposition with Theorem 4.19.

We prove the converse implication, namely that ω-regular languages featuring finitely many non-equivalent patterns consist of only ultimately periodic words. To this end, we exploit the characterization of an ω-regular language L:

$$L = \bigcup_{i=1}^{n} U_i \cdot V_i^\omega.$$

Lemma 4.17 allows only two cases for the number of non-equivalent patterns featuref by V_i^ω: such a number can be either one or infinite. Because L contains only a finite number of non-equivalent patterns, then every V_i^ω is of the form $V_i^\omega = \{v_i^\omega\}$, for an appropriate v_i. Hence, each component $U_i \cdot V_i^\omega$ of L could be written as

$$U_i \cdot V_i^\omega = \{u \cdot v_i^\omega | u \in U_i\}.$$

Thus, the language L is a finite union of languages consisting of only ultimately periodic words. □

Remark 4.21. Theorem 4.20 shows that Büchi automata can represent only a small class of languages of ultimately periodic words, namely the class of languages of ultimately periodic words featuring a *finite number* of patterns and a (possibly) *infinite number* of prefixes. Nevertheless, this class of languages represents a meaningful class of granularities, such as the *dynamic granularities* (or *unanchored granularities*), which are granularities that can be arbitrarily shifted among the time line.

4.2 Ultimately periodic automata

In this section we show how to effectively build, starting from smaller components, suitable Büchi automata recognizing languages of ultimately periodic words. In addition, we define an appropriate subclass of Büchi automata that captures exactly the ω-regular languages of ultimately periodic words and that can be exploited to represent and manipulate sets of granularities.

4.2.1 Basic operations

Theorem 4.20 states that ω-regular languages of ultimately periodic words are generated by ω-regular expressions of the form

$$L = \bigcup_{i=1}^{n} U_i \cdot \{v_i^\omega\}.$$

This implicitly defines the basic operations for effectively building automata that recognize such languages. These operations are:

1. the iteration v_i^ω of a given word v_i;
2. the concatenation of a regular language U_i with a language of ultimately periodic words;
3. the finite union of languages of ultimately periodic words.

The following propositions shows how to implement these operations in the automaton-based framework.

Proposition 4.22. *Given a word $v \in A^*$ it is possible to build a Büchi automaton $\mathcal{A}_v$ that recognizes $\{v^\omega\}$.*

Proof. Let be $v = a_0a_1a_2 \dots a_n$ a finite word of length $n+1$. The Büchi automaton $\mathcal{A}_v = (Q, \Delta, q_0, F)$ recognizing the language $\{v^\omega\}$ can be defined as follows:

- $Q = \{q_0, q_1, \dots, q_n\}$;
- $\Delta = \{(q_i, a_i, q_{i+1}) | 0 \leq i \leq n-1\} \cup \{(q_n, a_n, q_0)\}$;
- $F = \{q_0\}$.

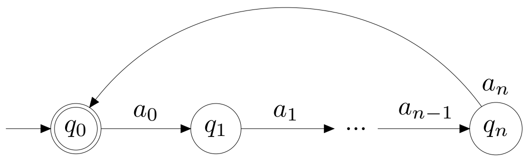

Figure 2: Automaton $\mathcal{A}_v$ that recognize $\{v^\omega\}$

An example of such an automaton is depicted in Figure 2: it has one state for each symbol of v and it reads a single symbol a_i from each state q_i. In order to encode repeating occurrences of v, a transition departs from q_n and reaches

q_0. The only possible run of $\mathcal{A}_v$ is $\rho = q_0 q_1 \dots q_{n-1} q_n q_0 q_1 \dots$, which recognizes the infinite word $v^\omega = a_0 a_1 \dots a_{n-1} a_n a_0 a_1 \dots$. Hence the language recognized by $\mathcal{A}_v$ is $\{v^\omega\}$. $\square$

Proposition 4.23. *Let L be an ω-regular language of ultimately periodic words recognized by an automaton $\mathcal{A}_L$, and let U be a regular language recognized by a finite state automaton $\mathcal{A}_U$. Then it is possible to build a Büchi automaton $\mathcal{A}_{U\cdot L}$ that recognizes $U \cdot L$.*

Proof. Given $\mathcal{A}_L = (Q_L, \Delta_L, q_0^L, F_L)$ and $\mathcal{A}_U = (Q_U, \Delta_U, q_0^U, F_U)$, the automaton $\mathcal{A}_{U\cdot L} = (Q, \Delta, q_0, F)$ is defined in the usual way:

- $Q = Q_U \cup Q_L$, with $Q_U \cap Q_L = \emptyset$;
- $\Delta = \Delta_U \cup \Delta_L \cap \{(q, a, q') | q \in F_U \text{ and } (q_0^L, a, q') \in \Delta_L\}$;
- $q_0 = q_0^U$;
- $F = F_L$.

$\square$

Proposition 4.24. *Let L_1 and L_2 be two ω-regular languages of ultimately periodic words, recognized respectively by $\mathcal{A}_1$ and $\mathcal{A}_2$. It is possible to build a Büchi automaton $\mathcal{A}_3$ that recognizes $L_1 \cup L_2$.*

Proof. Given $\mathcal{A}_1 = (Q_1, \Delta_1, q_0^1, F_1)$ and $\mathcal{A}_2 = (Q_2, \Delta_2, q_0^2, F_2)$, the automaton $\mathcal{A}_3 = (Q, \Delta, q_0, F)$ is defined in the usual way:

- $Q = Q_1 \cup Q_2 \cup \{q_0\}$, with $Q_U \cap Q_L = \emptyset$ and $q_0 \notin Q_1 \cup Q_2$;
- $\Delta = \Delta_1 \cup \Delta_2 \cap \{(q_0, a, q) | (q_0^1, a, q) \in \Delta_1 \text{ or } (q_0^2, a, q) \in \Delta_2\}$;
- $F = F_1 \cup F_2$.

$\square$

These basic operations allow us to build an automaton that recognizes a language of ultimately periodic words starting from the corresponding ω-regular expression. The effectiveness of such a construction yields to a characterization of a class of automata that recognizes exactly the ω-regular languages of ultimately periodic words.

Indeed, the automata resulting from Propositions 4.22, 4.23 and 4.24 satisfy the following properties:

1. the components that recognize the infinite suffixes v^ω behave like single-string automata, namely there is a unique accepting run that recognizes exactly v^ω;

2. final states appears only in those components that recognize a word v^ω.

Now we define an appropriate class of Büchi automata that meets such properties. We recall that a *strongly connected component* (SCC) S of an automaton $\mathcal{A}$ is a maximal set of states such that for every couple of states $u, v \in S$, both $u \to^* v$ and $v \to^* u$ holds. Moreover, a state $q \in Q$ is called *transient* if you cannot return to it (namely, $q \not\to^+ q$).

Definition 4.25. A Büchi automaton $\mathcal{A} = (Q, \Delta, q_0, F)$ is an *ultimately periodic automata* (UPA) iff, for every final state f, the SCC of f is either a single transient state or a simple loop without exiting transitions, namely, for every $f \in F$, $f \to^+ f$ implies that for every state q such that $f \to^* q$ there is a unique transition $(q, a, q') \in \Delta$ departing from q.

Given a loop C_f in $\mathcal{A}$ we say that C_f is a *final loop* if there exists at least one final state $f \in C_f$. We also say that C_f *encodes the pattern* $v \neq \varepsilon$ if there exists a state $q \in C_f$ such that $q \xrightarrow{v} q$ and v is primitive (namely, such that for every $u \in A^+$, $q \xrightarrow{u} q$ implies $|v| \leq |u|$). It's easy to see that whenever a final loop encodes a pattern v, then it encodes also every rotation of v: thus by saying that two final loops C_1 and C_2 *encode non-equivalent patterns* we mean that they encode two patterns that are not reducible by rotation.

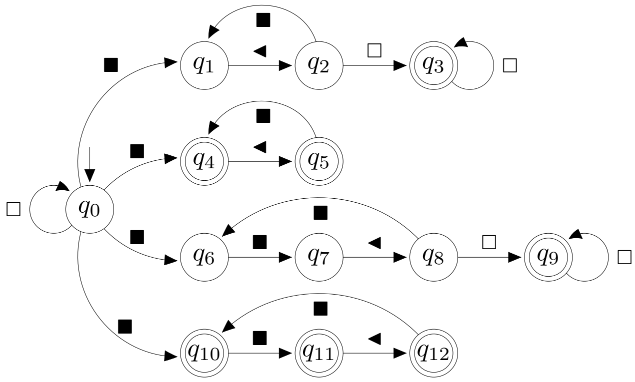

Figure 3: Example of UPA

Example 4.26. Figure 3 depicts an UPA representing the set of finite and infinite granularities that ultimately groups days either two by two or three by three (the embedding of finite granularities into infinite ones is obtained in the usual way).

From Definition 4.25, it follows that the automata built in Propositions 4.22, 4.23 and 4.24 are ultimately periodic. Thus, every ω-regular language of ultimately periodic words is recognizable by an UPA. The following theorem shows that also the converse holds.

Theorem 4.27. *Ultimately periodic automata recognize languages of ultimately periodic words.*

Proof. Let $\mathcal{A} = (Q, \Delta, q_0, F)$ be an UPA and let $w \in \mathscr{L}^\omega(\mathcal{A})$. Consider an accepting run ρ of $\mathcal{A}$ over w, and let f be a final state that occurs infinitely often in ρ. It is clear that $f \to^+ f$ and thus (by definition of UPA), there exists only a unique run σ of $\mathcal{A}$ starting from f:

$$\sigma = \sigma(1) \to \sigma(2) \to \ldots \to \sigma(n) \to \ldots$$

Since $f \to^+ f$, there exists $k > 1$ such that $\sigma(k) = f$ and σ is of the form:

$$\sigma = (\sigma(1) \cdot \ldots \cdot \sigma(k-1))^\omega.$$

Consequently, σ recognizes the infinite word v_f^ω, with v_f being the pattern encoded by the loop $\sigma(1) \cdot \ldots \cdot \sigma(k-1)$.

Therefore, ρ has an an infinite suffix recognizing v_f^ω, and w is necessarily an ultimately periodic word. This proves that $\mathcal{A}$ recognizes only languages of ultimately periodic words. $\square$

4.2.2 Normal form of UPA

Due to the particular structure of UPA, every accepting run of an UPA consists of a finite prefix followed by the infinite repetition of a *final loop*. Consequently, in a final loop the number and the positions of the final states is irrelevant: a single final state in the loop is sufficient to recognize the infinite suffix v^ω, where v is the pattern encoded by the loop. Conversely, if you set a non-final state of a final loop as final, the recognized language remains the same. Moreover, Definition 4.25 allows the final states of an automaton to be transient states. It is clear that transient final states cannot recognize repeating patterns, because they cannot occur infinitely often in a run, and thus, if you set a transient final state of the automaton as non-final, the recognized language remains the same.

These observations leads to the following definition of a *normal form* of UPA, which forces all states of a final loop to be final and does not allow the existence of transient final states, thus simplifying algorithmic manipulation of UPA.

Definition 4.28. An UPA $\mathcal{A} = (Q, \Delta, q_0, F)$ is said to be in *normal form* if:

- $\forall f \in F. f \to^+ f$ (namely, there are no transient final states);
- $\forall f \in F, q \in Q. f \to^* q \Rightarrow q \in F$ (namely, every state of a final loop is final);
- the initial state q_0 is not a final state ($q_0 \notin F$).

Such conditions allow us to easily distinguish between components recognizing finite prefixes and components recognizing infinite suffixes. The third condition implies that the automaton must recognize at least one symbol before reaching a final loop. Such a condition will be useful in order to simplify the algorithmic manipulation of ultimately periodic automata.

The following theorem shows that normal forms of UPA do not reduce the expressiveness of the class of automata.

Theorem 4.29. *For every UPA $\mathcal{A} = (Q, \Delta, q_0, F)$ there exists an equivalent UPA $\mathcal{A}'$ in normal form.*

Proof. We define $\mathcal{A}' = (Q', \Delta', q_0', F')$ as follows:

1. $Q' = Q$, $\Delta' = \Delta$;
2. $q \in F'$ iff $q \in Q$ belongs to a final loop of $\mathcal{A}$;
3. if $q_0 \in F'$, then we define a new initial state q_0' in $\mathcal{A}'$ and a new transition (q_0', a, q) for each state q such that $(q_0, a, q) \in \Delta$, otherwise we let $q_0' = q_0$ to be the initial state of $\mathcal{A}'$.

Let $w \in \mathscr{L}^\omega(\mathcal{A})$ and let ρ be an accepting run of $\mathcal{A}$ over w. If $q_0' = q_0$, then ρ is also an accepting run of $\mathcal{A}'$ over w (because every final state of $\mathcal{A}$ occurring infinitely often in ρ is a final state of $\mathcal{A}'$), otherwise we can build an accepting run of $\mathcal{A}'$ simply by substituting the initial transition (q_0, a, q) with (q'_0, a, q). Therefore, $\mathscr{L}^\omega(\mathcal{A}) \subseteq \mathscr{L}^\omega(\mathcal{A}')$.

Let be $w \in \mathscr{L}^\omega(\mathcal{A}')$ and let be ρ an accepting run of $\mathcal{A}'$ over w. If $q'_0 = q_0$ then ρ is also a run of $\mathcal{A}$, otherwise we can build a run of $\mathcal{A}$ over w simply by substituting the initial transition (q'_0, a, q) with (q_0, a, q). Because ρ is an accepting run of $\mathcal{A}'$, there exists a final state $f \in F'$ that occurs infinitely often in ρ. Since a final state of $\mathcal{A}'$ is a state that belongs to a final loop of $\mathcal{A}$, there exists a final state $f' \in F$ such that $f \to^* f'$ and it occurs infinitely often in ρ. Thus, ρ is also an accepting run of $\mathcal{A}$ and $\mathscr{L}^\omega(\mathcal{A}') \subseteq \mathscr{L}^\omega(\mathcal{A})$. $\square$

4.2.3 Closure properties

UPA are (effectively) closed under union and concatenation with regular languages, as shown by Propositions 4.23 and 4.24. From the closure properties of ω-regular languages, they also turn out to be closed under intersection. The effectiveness of such a closure property is claimed in the following theorem.

Theorem 4.30. *Given two UPA $\mathcal{A}_1$ and $\mathcal{A}_2$ that recognize, respectively, the languages L_1 and L_2, it is possible to build an automaton $\mathcal{A}_1 \cap \mathcal{A}_2$ that recognizes the language $L_1 \cap L_2$.*

Proof. Let $\mathcal{A}_1 = (Q_1, \Delta_1, q_0^1, F_1)$ and $\mathcal{A}_2 = (Q_2, \Delta_2, q_0^2, F_2)$ be two UPA in normal form. The automaton $\mathcal{A}_1 \cap \mathcal{A}_2 = (Q, \Delta, q_0, F)$ is defined as follows:

- $Q = Q_1 \times Q_2$;
- Δ is such that $((q_1, q_2), a(q'_1, q'_2)) \in \Delta$ iff $(q_1, a, q'_1) \in \Delta_1$ and $(q_2, a, q'_2) \in \Delta_2$;
- $q_0 = (q_0^1, q_0^2)$;
- $F = \{(q_1, q_2) | q_1 \in F_1 \text{ and } q_2 \in F_2\}$.

For every $(q_1, q_2), (q'_1, q'_2) \in Q$ and $v \in A^*$, we have $(q_1, q_2) \xrightarrow[\mathcal{A}_1 \cap \mathcal{A}_2]{v} (q'_1, q'_2)$ iff $q_1 \xrightarrow[\mathcal{A}_1]{v} q'_1$ and $q_2 \xrightarrow[\mathcal{A}_2]{v} q'_2$. We show that $\mathcal{A}_1 \cap \mathcal{A}_2$ recognizes the language $L_1 \cap L_2$.

If $w \in L_1 \cap L_2$, there exist an accepting run ρ_1 of $\mathcal{A}_1$ over w and an accepting run ρ_2 of $\mathcal{A}_2$ over w. Consequently,

$$\rho_1 \times \rho_2 = (\rho_1(1), \rho_2(1)) \xrightarrow{w(1)} \dots \xrightarrow{w(n-1)} (\rho_1(n), \rho_2(n)) \xrightarrow{w(n)} \dots$$

is also a run of $\mathcal{A}_1 \cap \mathcal{A}_2$. Because $\mathcal{A}_1$ and $\mathcal{A}_2$ are in normal form, there exists an integer n such that the infinite suffixes $\rho_1[n, +\infty[$ and $\rho_2[n, +\infty[$ encompass only final states. Therefore, the infinite suffix $(\rho_1 \times \rho_2)[n, +\infty[$ encompass only final states and there exists a final state of $\mathcal{A}_1 \cap \mathcal{A}_2$ that occurs infinitely often in it. Hence $\rho_1 \times \rho_2$ is an accepting run of $\mathcal{A}_1 \cap \mathcal{A}_2$ over w, so $w \in \mathscr{L}^\omega(\mathcal{A}_1 \cap \mathcal{A}_2)$.

Conversely, let $w \in \mathscr{L}^\omega(\mathcal{A}_1 \cap \mathcal{A}_2)$ and let ρ be an accepting run of $\mathcal{A}_1 \cap \mathcal{A}_2$ over w. Then there exists a final state (q_1, q_2) that occurs infinitely often in ρ. The projections ρ_1, ρ_2 with respect to the first and second component of ρ are accepting runs of $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively, because q_1 occurs infinitely often in ρ_1 and q_2 occurs infinitely often in ρ_2. Thus $w \in L_1 \cap L_2$.

We proved that the automaton $\mathcal{A}_1 \cap \mathcal{A}_2$ recognizes exactly the language $L_1 \cap L_2$; now we have to to show that it is an UPA. If $(q_1, q_2) \in F$, then $q_1 \in F_1$, $q_2 \in F_2$ and there exist two transitions $(q_1, a_1, q_1') \in \Delta_1$ and $(q_2, a_2, q_2') \in \Delta_2$ such that $q_1' \in F_1$ and $q_2' \in F_2$. Therefore, $(q_1, q_2) \in F$ satisfies the following properties:

- $((q_1, q_2), a_1, (q_1', q_2'))$ is the only transition departing from (q_1, q_2) and $(q_1', q_2') \in F$, provided that $a_1 = a_2$;
- there are no transition departing from (q_1, q_2), provided that $a_1 \neq a_2$.

Therefore, in order to obtain an UPA in normal form it is sufficient to discard all those final states f that do not belongs to a final loop of $\mathcal{A}_1 \cap \mathcal{A}_2$, namely, all final states f such that $f \not\rightarrow^+ f$. □

5 Some Paradigmatic Problems

We study now some interesting problems of automata theory that could be easily solved in UPA.

Emptiness problem

Definition 5.1. Given an UPA $\mathcal{A}$, the *emptiness problem* is the problem of deciding whether or not $\mathscr{L}^\omega(\mathcal{A}) = \emptyset$.

In the automaton-based framework, the emptiness problem is answered in linear time by searching *i)* for a path departing from the initial state and reaching a final state, and *ii)* for a loop that contains such a final state. Since every final state of an UPA in normal form belongs to a loop, in order to solve emptiness problem it's sufficient to search for a path from the initial state to a final state.

Membership problem

Definition 5.2. Given an ultimately periodic word w and a UPA $\mathcal{A}$, the *membership problem* is the problem of deciding whether or not $w \in \mathscr{L}^\omega(\mathcal{A})$.

Such a problem checks if a given granularity (represented by a word w) belongs to the set of granularities captured by $\mathcal{A}$. As it is claimed in the following proposition, the membership problem can be reduced to the emptiness problem.

Proposition 5.3. *The membership problem can be reduced to the emptiness problem.*

Proof. The ultimately periodic word w belongs to $\mathscr{L}^\omega(\mathcal{A})$ if and only if $\{w\} \cap \mathscr{L}^\omega(\mathcal{A})$ is not empty. To solve the membership problem it's sufficient to build the automaton that recognizes $\{w\}$ and build the automaton that recognizes the language $\{w\} \cap \mathscr{L}^\omega(\mathcal{A})$, and finally to test the emptiness of $\{w\} \cap \mathscr{L}^\omega(\mathcal{A})$. □

Remark 5.4. The membership problem is non-well founded for Büchi automata, because is not possible to provide a finite representation for any word that may belong to the language (i.e. non ultimately periodic words). On the other hand, the problem becomes decidable when we restrict ourselves to ultimately periodic words.

Equivalence problem

Definition 5.5. Given two UPA $\mathcal{A}_1$ and $\mathcal{A}_2$, the *equivalence problem* is the problem of deciding whether or not $\mathscr{L}^\omega(\mathcal{A}_1) = \mathscr{L}^\omega(\mathcal{A}_2)$.

One can obviously decide whether two given UPA $\mathcal{A}$ and $\mathcal{B}$ recognize the same language by considering $\mathcal{A}$ and $\mathcal{B}$ as generic Büchi automata, by computing their complements $\overline{\mathcal{A}}$ and $\overline{\mathcal{B}}$, and by testing the emptiness of $\mathscr{L}^\omega(\mathcal{A}) \cap \mathscr{L}^\omega(\overline{\mathcal{B}})$ and $\mathscr{L}^\omega(\mathcal{B}) \cap \mathscr{L}^\omega(\overline{\mathcal{A}})$. However, there exists an alternative direct method for deciding the equivalence problem that exploits a suitable canonical form of UPA, which turns out to be unique up to isomorphisms. This will be the topic of discussion in the next section, where we show in details the construction of the canonical form of UPA.

Inclusion problem

Definition 5.6. Given two UPA $\mathcal{A}_1$ and $\mathcal{A}_2$, the *inclusion problem* is the problem of deciding whether or not $\mathscr{L}^\omega(\mathcal{A}_1) \subseteq \mathscr{L}^\omega(\mathcal{A}_2)$.

The inclusion problem allows one to solve all inclusion problems between sets of granularities.

Proposition 5.7. *The inclusion problem can be reduced to the equivalence problem.*

Proof. Given two UPA $\mathcal{A}_1$ and $\mathcal{A}_2$, $\mathscr{L}^\omega(\mathcal{A}_1) \subseteq \mathscr{L}^\omega(\mathcal{A}_2)$ holds if and only if $\mathscr{L}^\omega(\mathcal{A}_1) \cap \mathscr{L}^\omega(\mathcal{A}_2) = \mathscr{L}^\omega(\mathcal{A}_1)$. Since ultimately periodic automata are closed under intersection, the inclusion problem can be reduced to equivalence problem. $\square$

6 The Equivalence Problem

In this section, we solve the equivalence problem for UPA by introducing a suitable *canonical form* for UPA. It will turn out that equivalent canonical UPA are isomorphic, so the equivalence problem will be reduced to an isomorphism test between canonical UPA. Such an approach is similar to the one adopted for the equivalence problem of *deterministic finite automata*: in that case the existence of a *unique minimum automaton* was exploited to solve the equivalence problem (for further information, see [HMU01]). However, our approach does not generate a minimum automaton, but it rather uses minimization algorithms for finite automata to obtain a canonical form that is unique up to isomorphism.

6.1 Prefix and pattern languages

The structure of UPA follows the *prefix-pattern* subdivision of ultimately periodic words, and implicitly splits every recognized word into a prefix (recognized before reaching a final loop) and a pattern (recognized in a final loop).

This observation allows us to define the *prefix language* and the *pattern language* of an UPA.

Definition 6.1. Given an UPA $\mathcal{A} = (Q, \Delta, q_0, F)$, the *prefix language* of $\mathcal{A}$ is defined as $\mathscr{L}^{pre}(\mathcal{A}) = \{v \in A^* | q_0 \overset{v}{\leadsto} q_f, q_f \in F\}$ and the *pattern language* of $\mathcal{A}$ is defined as $\mathscr{L}^{pat}(\mathcal{A}) = \{v \in A^+ | \exists f \in F. f \xrightarrow{v} f \text{ and } \forall u \in A^+. f \xrightarrow{u} f \Rightarrow |v| \leq |u|\}$.

It's easy to see that both $\mathscr{L}^{pre}(\mathcal{A})$ and $\mathscr{L}^{pat}(\mathcal{A})$ are regular languages and it is worth noticing that automata recognizing the same language could have different prefix and pattern languages. On the other hand, automata having the same prefix and pattern languages may recognize different languages, since they can combine prefixes and patterns in different ways.

To overcome this problem, we first define a unique minimal prefix-pattern subdivision of an UPA. The computation of such a subdivision will be the preliminary step of our algorithm. In particular, we plan to solve the equivalence problem in three computational steps:

1. **minimization of the pattern language:** after such an operation, the automaton will recognize only primitive patterns and the graph of final states will be unique and minimal;
2. **minimization of the prefix language:** after such an operation, the prefix-pattern subdivision of every recognized word is minimal, and the prefix and pattern languages are unique;
3. **canonization of non-final states:** using minimization algorithms for finite automata, we build a canonical form for non-final states and hence for the entire automaton.

6.1.1 Minimization of the pattern language

The procedure for minimizing the pattern language is based on the solution of the *single function coarsest partition problem*, which was described by Paige, Tarjan and Bonic in [PTB85].

The set of final states of an UPA can be viewed as a directed graph of out-degree one satisfying the following properties:

1. there exists at least one loop (provided that the set of final states is non-empty);
2. there are no paths outside loops;
3. no path of a loop can leave the loop;

Thus, the graph of final states satisfies the same properties of the graph representation of the single function coarsest partition problem; then the Paige, Tarjan and Bonic algorithm can be used to minimize the final states.

Before describing in details the algorithm, we recall some properties of patterns and string rotations that turn out to be useful to compute canonical form.

Definition 6.2. The *smallest repeating prefix* of a string $s \in A^*$ is the smallest prefix p of s such that $p^k = s$.

Definition 6.3. If $v = v_1 \cdot \ldots v_k$ is a finite word, a *rotation of* v is a word $v' = v_i \cdot \ldots \cdot v_k \cdot v_1 \cdot \ldots \cdot v_{i-1}$.

Definition 6.4. Let $<$ be a total ordering on a finite alphabet A. The *lexicographical ordering* $<_{lex}$ is the total ordering over strings in A^* such that $x \leq_{lex} y$ iff one of the following conditions holds:

1. $\exists j . x(j) < y(j)$ and $x(i) = y(i), \forall i = 1, \ldots, j-1$;
2. $|x| \leq |y|$ and $x(i) = y(i), \forall i = 1, \ldots, |x|$.

Definition 6.5. Given a word v, a rotation v' of v is said to be the *lexicographically least rotation* if, for every rotation v'' of v, $v' \leq_{lex} v''$ holds.

The algorithm minimizes the graph of final states for a given automaton in the following way:

1. substitute each final loop recognizing a pattern p by a loop corresponding to the smallest repeating prefix of p;

2. compute the lexicographically least rotation of each pattern;

3. lexicographically sort the loops with respect to their patterns;

4. merge ordered loops that recognize the same pattern.

The above steps are effective. The smallest repeating prefix of a word v can be computed by finding the first non trivial occurrence of v in $v \cdot v$ through string-matching algorithms. The problem of computing the lexicographically least rotation of a word is solved in [Boo80] and [Shi81] through a linear-time algorithm.

Algorithm 1 shows in details how the procedure works. The input is an UPA in normal form, and the algorithm returns an equivalent UPA which final loops recognize only non-equivalent primitive patterns. The array *Cycles* is used to store and sort final loops.

Theorem 6.6. *After the execution of Algorithm 1 with an UPA in normal form $\mathcal{A}$ as input, the output automaton $\mathcal{A}'$ satisfies:*

1. $\mathscr{L}^{\omega}(\mathcal{A}) = \mathscr{L}^{\omega}(\mathcal{A}')$*;*
2. *the structure of the final states of $\mathcal{A}'$ is unique and minimal, with respect to the recognized language.*

Proof. (1) To prove that $\mathscr{L}^{\omega}(\mathcal{A}) = \mathscr{L}^{\omega}(\mathcal{A}')$, we study the structure of the final states of the automaton before the execution of line 21 and after the execution of the entire algorithm.

If C_f is a final loop of the initial automaton $\mathcal{A}$ and v_f is the recognized pattern, after the execution of lines 1-20 two possible cases arise:

1. v_f is a primitive pattern and C_f has not been modified;

2. if v_f is not primitive, then C_f has been modified in order to recognize a primitive pattern (the transitions reaching the deleted part of the loop have been moved in the initial part).

Thus, after the first part of the algorithm the automaton recognizes the same language of the original automaton.

In the second stage, the algorithm merges together those loops that recognize the same pattern. If C_f is a final loop of the automaton, after the execution of lines 21-35 two possible cases arises:

1. C_f is still a loop of the output automaton $\mathcal{A}'$;

Algorithm 1 Minimization of final states

1: **for all** unmarked $q_f \in F$ **do**
2: $\quad C_f \leftarrow$ final loop that starts in q_f
3: $\quad v_f \leftarrow$ pattern recognized by C_f
4: $\quad i \leftarrow$ position of first non trivial occurrence of v_f in $v_f \cdot v_f$
5: $\quad$ **if** $i < |v|$ **then**
6: $\quad\quad$ *delete transition* $(C_f[i], v[i], C_f[i+1])$
7: $\quad\quad$ *insert transition* $(C_f[i], v[i], C_f[1])$
8: $\quad\quad$ **for** $j \leftarrow i+1, \ldots, |v|$ **do**
9: $\quad\quad\quad$ **for all** $(q', a', C_f[j])$ such that $q' \notin F$ **do**
10: $\quad\quad\quad\quad$ *delete transition* $(q', a', C_f[j])$
11: $\quad\quad\quad\quad$ *insert transition* $(q', a', C_f[((j-1) \ mod \ i)+1])$
12: $\quad\quad\quad$ **end for**
13: $\quad\quad\quad$ *delete state* $(C_f[j])$
14: $\quad\quad$ **end for**
15: $\quad$ **end if**
16: $\quad v'_f \leftarrow$ lexicographically least rotation of v_f
17: $\quad$ *insert* (C_f, v'_f) *in Cycles*
18: $\quad$ *mark all states* (C_f)
19: **end for**
20: *sort* $(Cycles)$
21: $(C'_f, v'_f) \leftarrow Cycles[1]$
22: **for** $i \leftarrow 2, \ldots, |Cycles|$ **do**
23: $\quad (C_f, v_f) \leftarrow Cycles[i]$
24: $\quad$ **if** $v_f = v'_f$ **then** $\quad$ // *Merge* C_f *and* C'_f
25: $\quad\quad$ **for** $j \leftarrow 1, \ldots, |C'_f|$ **do**
26: $\quad\quad\quad$ **for all** $(q, a, C'_f[j]) \in \Delta$ such that $q \notin F$ **do**
27: $\quad\quad\quad\quad$ *delete transition* $(q, a, C'_f[j])$
28: $\quad\quad\quad\quad$ *insert transition* $(q, a, C_f[j])$
29: $\quad\quad\quad$ **end for**
30: $\quad\quad\quad$ *delete state* $(C'_f[j])$
31: $\quad\quad$ **end for**
32: $\quad$ **end if**
33: $\quad (C'_f, v'_f) \leftarrow (C_f, v_f)$
34: **end for**

2. C_f has been removed and there exists a final loop C'_f of $\mathcal{A}'$ that recognizes the same pattern (transitions reaching C_f have been moved to C'_f).

Thus, at the end of the algorithm the automaton recognizes the same language of $\mathcal{A}$.

(2) Let be $\mathcal{A}_1$ and $\mathcal{A}_2$ two UPA such that $\mathscr{L}^\omega(\mathcal{A}_1) = \mathscr{L}^\omega(\mathcal{A}_2)$: we show that after the execution of Algorithm 1 the corresponding output automata $\mathcal{A}'_1$ and $\mathcal{A}'_2$ have the same graph of the final states (up to isomorphism), and that this graph is minimal.

To prove this, is sufficient to notice that, after the first part of the algorithm (lines 1-20), every final loop recognizes a primitive pattern: then the two automata recognize the same pattern language, which contains only primitive patterns.

After the second part of the algorithm, final loops that recognize the same pattern are merged, so every final state recognizes a pattern different from the others. Because every final state recognizes exactly one pattern, at the end of the algorithm $\mathcal{A}_1$ and $\mathcal{A}_2$ have the same number of final states (one for each minimal pattern) and furthermore the structures of the transitions connecting final states are isomorphic.

Thus, the structure of final states is unique and minimal after the execution of Algorithm 1. $\square$

6.1.2 Minimization of the prefix language

Given a ultimately periodic word $w = u \cdot v^\omega$, the prefix u is said to be minimal if and only if $u(|u|) \neq v(|v|)$ (the last symbol of u differs from the last symbol of v). Conversely, if $w = u \cdot v^\omega$ is such that $u(|u|) = v(|v|)$ then $w = u[1, |u| - 1] \cdot (v(|v|) \cdot v[1, |v| - 1])^\omega$ and $u[1, |u| - 1]$ is a shorter prefix of w. By iterating such an operation, we can find the minimal prefix of w, which is proved to be unique.

Such a construction can be mimicked in the run of an UPA. An accepting run ρ of an UPA over w splits the word into *i)* a prefix u such that $q_0 \overset{u}{\rightsquigarrow} q_f$ (with q_f being the first final state occurring in ρ) and *ii)* a suffix v^ω such that v is recognized by the final loop of q_f. If u were not minimal, then it would follow that $u(|u|) = v(|v|)$ and

$$\rho = q_0 \overset{u[1,|u|-1]}{\rightsquigarrow} q' \xrightarrow{u(|u|)} q_f \xrightarrow{v[1,|v|-1]} q'_f \xrightarrow{v(|v|)} q_f \rightarrow \dots$$

Now, it's clear that ρ can be modified to by replacing the transition $q' \xrightarrow{u(|u|)} q_f$ with $q'_f \xrightarrow{v(|v|)} q_f$. In this case the first final state occurring in ρ would be q'_f. This would lead to a different prefix-pattern subdivision of the word w, precisely by defining the prefix to be $u[1, |u| - 1]$ and the pattern to be $v(|v|) \cdot v[1, |v| - 1]$. The structure of $\mathcal{A}$ has to be modified accordingly: the transition $(q', u(|u|), q_f)$ should be deleted and, for each transition $(q'', a, q') \in \Delta$ reaching q', a transition (q'', a, q'_f) should be introduced.

Algorithm 2 shows in details how to minimize the prefix language of an UPA $\mathcal{A}$ provided in input. In order to assure termination, we require that $\mathcal{A}$ is in normal form and that the initial state q_0 has no entering transitions. The procedure *Clear* deletes marked transitions and those states which are unreachable from q_0 or from which you cannot reach a final state.

It is worth noticing that marked transitions are deleted at the end of the procedure; this avoids situations in which a transition is deleted and added infinitely many times.

To prove the correctness of the algorithm, we show that *i)* after each step of the computation, the automaton recognizes the same language, and *ii)* when the algorithm terminates, the prefix language of the automaton is minimal.

Lemma 6.7. *Given an UPA in normal form $\mathcal{A} = (Q, \Delta, q_0, F)$ and a state $q_f \in F$ such that $(q'_f, a, q_f) \in \Delta$, with $q'_f \in F$, and $(q, a, q_f) \in \Delta$ (with $q \in Q \setminus F \setminus \{q_0\}$), the automaton $\mathcal{A}' = (Q, \Delta', q_0, F)$, where*

$$\Delta' = (\Delta \setminus \{(q, a, q_f)\}) \cup \{(q', a', q'_f) | (q', a', q) \in \Delta\},$$

recognizes the same language of $\mathcal{A}$.

Algorithm 2 Minimization of the prefix language of $\mathcal{A}$

1: $c \leftarrow$ true
2: **while** $c =$ true **do**
3:     $c \leftarrow$ false
4:     **for all** $q_f \in F$ **do**
5:         **let** q'_f, a_f such that $(q'_f, a_f, q_f) \in \Delta$ and $q'_f \in F$
6:         **for all** unmarked (q, a, q_f) such that $a = a_f$ and $q \notin F$ **do**
7:             $c \leftarrow$ true
8:             $mark(q, a, q_f)$
9:             **for all** $(q', a', q) \in \Delta$ **do**
10:                 $insert\ transition(q', a', q'_f)$
11:             **end for**
12:         **end for**
13:     **end for**
14: **end while**
15: $clear(\mathcal{A})$

Proof. We have to prove that $\mathscr{L}^\omega(\mathcal{A}) = \mathscr{L}^\omega(\mathcal{A}')$.

Let $w \in \mathscr{L}^\omega(\mathcal{A})$ and let ρ be an accepting run of $\mathcal{A}$ over w. If ρ contains no transitions $q \xrightarrow{a} q_f$, then it is also an accepting run of $\mathcal{A}'$ and $w \in \mathscr{L}^\omega(\mathcal{A}')$. Otherwise, it contains exactly a transition $q \xrightarrow{a} q_f$ (because, after a final state is reached, $\mathcal{A}$ can visit only final states) and hence it is possible to build an accepting run ρ' of $\mathcal{A}'$ over w by replacing $q \xrightarrow{a} q_f$ with $q'_f \xrightarrow{a} q_f$ and the preceding transition $q' \xrightarrow{a'} q$ with $q' \xrightarrow{a'} q'_f$. Thus $w \in \mathscr{L}^\omega(\mathcal{A}')$.

Conversely, let $w \in \mathscr{L}^\omega(\mathcal{A}')$ and let ρ' be an accepting run of $\mathcal{A}'$ over w. If ρ contains no transitions $q' \xrightarrow{a'} q'_f$ (with $(q', a', q'_f) \in \Delta' \setminus \Delta$), then it is also an accepting run of $\mathcal{A}$ and $w \in \mathscr{L}^\omega(\mathcal{A})$. Otherwise, it contains exactly one transition $q' \xrightarrow{a'} q'_f$ that is immediately followed by the transition $q'_f \xrightarrow{a} q_f$. An accepting run ρ of $\mathcal{A}$ over w can be obtained by replacing $q' \xrightarrow{a'} q'_f$ with $q' \xrightarrow{a'} q$ and $q'_f \xrightarrow{a} q_f$ with $q \xrightarrow{a} q_f$. Then $w \in \mathscr{L}^\omega(\mathcal{A})$ and so we proved that $\mathscr{L}^\omega(\mathcal{A}) = \mathscr{L}^\omega(\mathcal{A}')$. $\square$

To show that Algorithm 2 preserves the language recognized by the original automaton, we consider a sequence of automata $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n, \ldots$ such that $\mathcal{A}_1$ is the input automaton and for every $n \geq 1$, $\mathcal{A}_{n+1}$ is the automaton resulting from the n-th iteration of lines 7-11 (without considering marked transitions). The following lemma states that the automata $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n, \ldots$ are all equivalent.

Lemma 6.8. *For every $n \geq 1$, the automaton $\mathcal{A}_{n+1}$ satisfies $\mathscr{L}^\omega(\mathcal{A}_n) = \mathscr{L}^\omega(\mathcal{A}_{n+1})$.*

Proof. Let $\mathcal{A}_n = (Q, \Delta, q_0, F)$. After the execution of lines 7-11, there exists a final state q_f, a transition $q'_f \xrightarrow{a} q_f$, with $q'_f \in F$, and an unmarked transition $q \xrightarrow{a} q_f$, with $q \in Q \setminus F \setminus \{q_0\}$, satisfying the hypothesis of Lemma 6.7.

Then, the algorithm proceed as follows:

1. it marks the transition $q \xrightarrow{a} q_f$ (hence $q \xrightarrow{a} q_f$ will not appear in $\mathcal{A}_{n+1}$);

2. for each transition $q' \xrightarrow{a'} q$, it inserts a transition $q' \xrightarrow{a'} q'_f$, provided that $q' \xrightarrow{a'} q'_f$ was not already marked.

$\mathcal{A}_{n+1}$ is slightly different from the automaton $\mathcal{A}'$ of Lemma 6.7, since some transitions added to $\mathcal{A}'$ may be missing in $\mathcal{A}_{n+1}$. However, we can show that $\mathscr{L}^\omega(\mathcal{A}') = \mathscr{L}^\omega(\mathcal{A}_{n+1})$ still holds.

It's easy to see that $\mathscr{L}^\omega(\mathcal{A}_{n+1}) \subseteq \mathscr{L}^\omega(\mathcal{A}')$, because every transition of $\mathcal{A}_{n+1}$ belongs to $\mathcal{A}'$, as well.

Let $q' \xrightarrow{a'} q'_f$ be a transition of $\mathcal{A}'$ that is not in $\mathcal{A}_{n+1}$. Then, such a transition has been marked in the previous steps of the algorithm, and there exists a final state q''_f such that $q''_f \xrightarrow{a'} q'_f \in \Delta$.

Given a prefix of the run ρ of $\mathcal{A}'$ from q_0 to q'_f, with $q' \xrightarrow{a'} q'_f$ being the last transition of ρ, let $q'' \xrightarrow{a''} q'$ be the transition preceding $q' \xrightarrow{a'} q'_f$ in ρ. When the algorithm marks $q' \xrightarrow{a'} q'_f$ it also inserts a transition $q'' \xrightarrow{a''} q''_f$. Hence, two cases arise:

- $q'' \xrightarrow{a''} q''_f$ is a transition of $\mathcal{A}_{n+1}$, and hence the sequence ρ' obtained from ρ by substituting $q' \xrightarrow{a'} q'_f$ by $q''_f \xrightarrow{a'} q'_f$ and $q'' \xrightarrow{a''} q'$ by $q'' \xrightarrow{a''} q''_f$ is a run of $\mathcal{A}_{n+1}$ recognizing the same word of ρ;

- $q'' \xrightarrow{a''} q''_f$ is not a transition of $\mathcal{A}_{n+1}$, and hence it has been marked in a previous step. In such a case, we consider the preceding transition $q''' \xrightarrow{a'''} q''$ in ρ and we repeat the previous argument: there exists a transition $q''' \xrightarrow{a'''} q'''_f$ that was added to the automaton and that, if it were not marked, it would allow to build a run ρ' of $\mathcal{A}_{n+1}$. The fact that q_0 has no entering transition guarantees that, after at most $|\rho|$ steps, we will find an unmarked transition and we will obtain a run ρ' of $\mathcal{A}_{n+1}$ that recognizes the same word of ρ.

Summing up, for every run of $\mathcal{A}'$ we obtain a run of $\mathcal{A}_{n+1}$ that accepts the same word and, by Lemma 6.7, $\mathscr{L}^\omega(\mathcal{A}_{n+1}) = \mathscr{L}^\omega(\mathcal{A}') = \mathscr{L}^\omega(\mathcal{A}_n)$. □

The following theorem states that the prefix language of the output automaton of Algorithm 2 is minimal.

Theorem 6.9. *Let $\mathcal{A}$ be the automaton computed by Algorithm 2 and $v \in \mathscr{L}^{pre}(\mathcal{A})$. Then v has length 1 or is the minimal prefix of a word $w \in \mathscr{L}^\omega(\mathcal{A})$.*

Proof. Algorithm 2 terminates whenever the flag variable c is set to false. This happens only in two cases:

1. if every transition reaching a final states is marked; hence the transitions entering in a final loop are only the ones departing from q_0.

2. If for every unmarked transition connecting a non-final state with a final state q_f the corresponding label is different from the label of the unique transition reaching q_f from a final state, or it comes from q_0.

Let $w \in \mathscr{L}^\omega(\mathcal{A})$ and let ρ be an accepting run over w. The run ρ of the automaton $\mathcal{A}$ satisfies:

$$\rho = q_0 \overset{u}{\rightsquigarrow} q_f \overset{v}{\multimap} q_f \overset{v}{\multimap} q_f \to \dots$$

with $q_f \in F$ and $u \in \mathscr{L}^{pre}(\mathcal{A})$. Then we have:

1. $|u| \geq 1$, because $q_0 \notin F$;

2. $|u| > 1$ and $u(|u|) \neq v(|v|)$ (otherwise, the algorithm would have marked the transition $q' \xrightarrow{u(|u|)} q_f$).

We have shown that every $u \in \mathscr{L}^{pre}(\mathcal{A})$ has length 1 or it is minimal. □

Corollary 6.10. *Given two equivalent UPA $\mathcal{A}_1$ and $\mathcal{A}_2$, Algorithm 2 modifies them in such a way that $\mathscr{L}^{pre}(\mathcal{A}_1) = \mathscr{L}^{pre}(\mathcal{A}_2)$.*

Proof. By Theorem 6.9, $\mathscr{L}^{pre}(\mathcal{A}_1)$ and $\mathscr{L}^{pre}(\mathcal{A}_2)$ consist only of non-empty minimal prefixes. From the fact that the minimal prefix of a ultimately periodic word is unique, it follows that $\mathscr{L}^{pre}(\mathcal{A}_1) = \mathscr{L}^{pre}(\mathcal{A}_2)$. □

We have shown that, when Algorithm 2 terminates, the prefix language of the automaton is unique and minimal. To assure the termination of the algorithm it is sufficient to note that:

1. the algorithm does not add new sates to the automaton;

2. the algorithm only marks transitions and it adds new transitions;

3. the number of all possible transitions between the states of the automaton is finite.

Summing up, after the execution of Algorithm 1 and Algorithm 2 in succession, the resulting automaton satisfy the following properties:

1. the pattern language is minimal, and the structure of final states is unique and minimal;

2. the prefix language consists only of non-empty minimal prefixes.

In the following, we shall complete our discussion of the solution of the equivalence problem by introducing the canonical form of UPA.

6.2 Canonical form of UPA

In this section we provide the details for the last stage of the canonization algorithm. The main steps of the algorithm are the following:

1. starting from an UPA $\mathcal{A}$, build the *prefix automaton*, which is a finite automaton;

2. compute the equivalent minimum deterministic prefix automaton;

3. make the prefix automaton into an UPA in canonical form.

6.2.1 Prefix automaton

The prefix automaton in 1 satisfies the following properties: it resembles the structure of non-final states of $\mathcal{A}$ (recognizing the prefix language $\mathscr{L}^{pre}(\mathcal{A})$) and, along with the information for the correspondence between prefixes and patterns, it allows the reconstruction of an UPA that recognizes $\mathscr{L}^{\omega}(\mathcal{A})$.

In practice, the prefix automaton is built as follows:

1. all transitions connecting pairs of final states are removed;
2. the alphabet of the automaton is extended to $A \cup (A \times F)$;
3. every transition (q, a, f) reaching a final state f from a non-final state is substituted with $(q, (a, f), f)$.

Here we provide the formal definition of the prefix automaton. Note that this is an automaton that recognizes *finite words*, hence we denote its language with $\mathscr{L}(pre(\mathcal{A}))$.

Definition 6.11. If $\mathcal{A} = (Q, \Delta, q_0, F)$ is an UPA in normal form, the *prefix automaton* $pre(\mathcal{A}) = (Q^P, \Delta^P, q_0^P, F^P)$ is defined as follows:

- the alphabet of $pre(\mathcal{A})$ is $A \cup (A \times F)$;
- $Q^P = Q$;
- $\Delta^P = (\Delta \setminus \{(q, a, f) | f \in F\}) \cup \{(q, (a, f), f) | f \in F, q \notin F$ and $(q, a, f) \in \Delta\}$;
- $q_0^P = q_0$;
- $F^P = F$.

Proposition 6.12. *Given an UPA $\mathcal{A} = (Q, \Delta, q_0, F)$ and its prefix automaton $pre(\mathcal{A})$, the language recognized by $pre(\mathcal{A})$ satisfies:*

1. *for every $v \in \mathscr{L}(pre(\mathcal{A}))$, $v = u \cdot (a, q_f)$, with $u \in A^*$, $a \in A$ and $q_f \in F$;*
2. *$u \cdot (a, q_f) \in \mathscr{L}(pre(\mathcal{A}))$ iff $q_0 \underset{\mathcal{A}}{\overset{u \cdot a}{\rightsquigarrow}} q_f$, with $q_f \in F$.*

Proof. Let $\mathcal{A} = (Q, \Delta, q_0, F)$ be an UPA in normal form and let $pre(\mathcal{A}) = (Q^P, \Delta^P, q_0, F)$ be the corresponding prefix automaton.

To prove property 1. it is sufficient to note that any transition of $pre(\mathcal{A})$ labelled with a pair (a, q_f) has q_f as target state, all other transitions are labelled with a symbol from A. Then, any accepting run of $pre(\mathcal{A})$ has an initial sequence of transitions labelled with symbols from A, and it ends with a single transition labelled with a pair (a, q_f). Thus, the words accepted by $pre(\mathcal{A})$ are necessarily of the form $u \cdot (a, q_f)$, with $u \in A^*$, $a \in A$ and $q_f \in F$.

We now prove property 2. Given a word $u \cdot (a, q_f) \in \mathscr{L}(pre(\mathcal{A}))$ and an accepting run σ over it, we have

$$\sigma = q_0 \overset{u}{\rightsquigarrow} q \xrightarrow{(a, q_f)} q_f.$$

By definition of prefix automaton, the transition (q, a, q_f) is a transition of $\mathcal{A}$ and q is not a final state. Thus, $q_0 \xrightarrow{u} q \xrightarrow{a} q_f$ is a run of $\mathcal{A}$ over $u \cdot a$ with no final states (except from q_f), and $u \cdot (a, q_f) \in \mathscr{L}(pre(\mathcal{A})) \Rightarrow q_0 \underset{\mathcal{A}}{\overset{u \cdot a}{\rightsquigarrow}} q_f$.

Conversely, because $pre(\mathcal{A})$ has the same structure of $\mathcal{A}$, given a word $u \cdot a$ such that $q_0 \underset{\mathcal{A}}{\overset{u \cdot a}{\rightsquigarrow}} q_f$, there exists a non-final state q such that $q_0 \underset{\mathcal{A}}{\overset{u}{\rightsquigarrow}} q$ and (q, a, q_f) is a transition of $\mathcal{A}$. Thus, it is possible to build the accepting run of $pre(\mathcal{A})$

$$q_0 \overset{u}{\rightsquigarrow} q \xrightarrow{(a,q_f)} q_f;$$

in such a way that $u \cdot (a, q_f) \in \mathscr{L}(pre(\mathcal{A}))$. $\square$

6.2.2 Building the UPA

In this section we show how to combine a suitable DFA equivalent to $pre(\mathcal{A})$ with the final loops of an UPA $\mathcal{A}$ in normal form, in order to obtain an UPA $\mathcal{A}'$ equivalent to $\mathcal{A}$ and in canonical form.

Algorithm 3 Build the UPA

1: **let** $\mathcal{A} = (Q, \Delta, q_0, F)$
2: **let** $\mathcal{P} = (Q^P, \Delta^P, q_0^P, F^P)$
3: **let** $\mathcal{A}' = (Q', \Delta', q_0', F')$
4: $Q' \leftarrow Q^P \cup F$ *// with* $Q^P \cap F = \emptyset$
5: $\Delta' \leftarrow \Delta^P \cup \{(f, a, f') \in \Delta | f, f' \in F\}$
6: **for all** $f \in F^P$ **do**
7:     **for all** $(q, (a, q_f), f) \in \Delta'$ such that $q_f \in F$ **do**
8:        *insert transition* (q, a, q_f) *in* Δ'
9:        *delete transition* $(q, (a, q_f), f)$
10:     **end for**
11: **end for**
12: $F' \leftarrow F$
13: $q_0' \leftarrow q_0^P$
14: $clear(\mathcal{A}')$

Theorem 6.13. *Given an UPA in normal form $\mathcal{A} = (Q, \Delta, q_0, F)$ and a finite automaton $\mathcal{P} = (Q^P, \Delta^P, q_0^P, F^P)$ recognizing the same language of $pre(\mathcal{A})$, Algorithm 3 builds an UPA $\mathcal{A}' = (Q', \Delta', q_0', F')$ such that $\mathscr{L}^\omega(\mathcal{A}') = \mathscr{L}^\omega(\mathcal{A})$.*

Proof. Algorithm 3 works as follows:

1. it defines the set of states of $\mathcal{A}'$ as the union of the states of $\mathcal{P}$ and the final states of $\mathcal{A}$ (line 4);

2. it defines the transitions of $\mathcal{A}'$ according to those in $\mathcal{P}$ and in $\mathcal{A}$ (line 5);

3. for every final state f of $\mathcal{P}$, *i)* it removes all those transitions that reach f and that are labelled with a pair $(a, q_f) \in A \times F$, and *ii)* it introduces appropriate transitions connecting the final loop (lines 6-13);

4. it declares F to be the set of final states of $\mathcal{A}'$ (line 14);

5. it declares q_0 to be the initial state of $\mathcal{A}'$ (line 15).

At the end of the algorithm, $\mathcal{A}'$ is an UPA in normal form, since every final state belongs to a single final loop (consisting of final states) without exiting transitions, and since $q'_0 \notin F'$.

We now show that $\mathscr{L}^\omega(\mathcal{A}') = \mathscr{L}^\omega(\mathcal{A})$. Given a word $w \in \mathscr{L}^\omega(\mathcal{A})$ and an accepting run ρ of $\mathcal{A}$ over w, if q_f is the first final state that occurs in ρ then ρ has the following form:

$$\rho = q_0 \overset{u}{\leadsto} q_f \overset{v}{\multimap} q_f \overset{v}{\multimap} \dots$$

By Proposition 6.12, whenever $u = u' \cdot a$ holds, $u' \cdot (a, q_f) \in \mathscr{L}(pre(\mathcal{A})) = \mathscr{L}(\mathcal{P})$ follows, and there exists a run σ of $\mathcal{P}$ such that

$$\sigma = q_0^P \overset{u'}{\leadsto} q \overset{a}{\to} f, \text{ with } f \in F^P.$$

All the transitions in σ are transitions of $\mathcal{A}'$, with exception for the transition of the form $(q, (a, q_f), f)$, which have been substituted by (q, a, q_f). Thus we can define the following run of $\mathcal{A}'$ over w:

$$\rho' = q'_0 \overset{u'}{\leadsto} q \overset{a}{\to} q_f \overset{v}{\multimap} q_f \overset{v}{\multimap} \dots$$

So, $w \in \mathscr{L}^\omega(\mathcal{A}')$ and $\mathscr{L}^\omega(\mathcal{A}) \subseteq \mathscr{L}^\omega(\mathcal{A}')$.

Conversely, given a word $w \in \mathscr{L}^\omega(\mathcal{A}')$ and an accepting run ρ' of $\mathcal{A}'$ over w, whenever q_f is the first final state occurring in ρ', then ρ' has the following form:

$$\rho' = q'_0 \overset{u}{\leadsto} q \overset{a}{\to} q_f \overset{v}{\multimap} q_f \overset{v}{\multimap} \dots,$$

with $u, v \in A^*$ and $a \in A$. In one iteration of the main loop (lines 6-13), the transition (q, a, q_f) is introduced; hence $\mathcal{P}$ contains the transition $(q, (a, q_f), f)$, for an appropriate $f \in F^P$. Consequently, $u \cdot (a, q_f) \in \mathscr{L}(\mathcal{P}) = \mathscr{L}(pre(\mathcal{A}))$ holds and, by Proposition 6.12, $\mathcal{A}$ satisfies $q_0 \underset{\mathcal{A}}{\overset{u \cdot a}{\leadsto}} q_f$. Thus, it is possible to build the following accepting run of $\mathcal{A}$ over w:

$$\rho = q_0 \overset{u \cdot a}{\leadsto} q_f \overset{v}{\multimap} q_f \overset{v}{\multimap} \dots$$

Finally, $w \in \mathscr{L}^\omega(\mathcal{A})$ and $\mathscr{L}^\omega(\mathcal{A}) = \mathscr{L}^\omega(\mathcal{A}')$ holds. ☐

6.2.3 Canonization algorithm

Algorithms 1, 2 and 3 allow us to build, starting from an UPA in normal form $\mathcal{A}$, an UPA in canonical form as shown in Algorithm 4.

Algorithm 4 Canonization algorithm

1: Execute Algorithm 1 over $\mathcal{A}$ to minimize final states structure;
2: minimize $\mathscr{L}^{pre}(\mathcal{A})$ by Algorithm 2;
3: build the prefix automaton $pre(\mathcal{A})$;
4: minimize $pre(\mathcal{A})$;
5: build the UPA $\mathcal{A}'$ in canonical form by Algorithm 3.

Each step preserves the recognized language (as stated by Theorems 6.6, 6.13 and by Lemma 6.8). Theorem 6.6 and Corollary 6.10 state that, after line 2, the structure of the final states for the automaton $\mathcal{A}$ is unique for every equivalent automaton provided in input. The following lemma states that the language recognized by the prefix automaton is unique (up to renaming of final states).

Lemma 6.14. *Given two equivalent UPA in normal form $\mathcal{A} = (Q, \Delta, q_0, F)$ and $\mathcal{A}' = (Q', \Delta', q'_0, F')$, after the execution of lines 1-3 of Algorithm 4, $pre(\mathcal{A})$ and $pre(\mathcal{A}')$ are equivalent (up to renaming of final states).*

Proof. After the execution of Algorithms 1 and 2, we know that

1. $\mathcal{A}$ and $\mathcal{A}'$ recognize the same minimal prefix language;
2. $\mathcal{A}$ and $\mathcal{A}'$ recognize the same minimal pattern language;
3. $\mathcal{A}$ and $\mathcal{A}'$ have the same number of final states with every final state corresponding to a different pattern.

Then, there exists a bijection that relates final states of $\mathcal{A}$ to final states of $\mathcal{A}'$ recognizing the same patterns. Thus, final states of $\mathcal{A}'$ can be renamed in such a way that final states of $\mathcal{A}$ and $\mathcal{A}'$ recognize the same patterns.

Let be $u \cdot (a, q_f) \in \mathscr{L}(pre(\mathcal{A}))$: by Proposition 6.12 there exists a ultimately periodic word $w \in \mathscr{L}^\omega(\mathcal{A})$ such that $w = u \cdot v^\omega$ and

$$q_0 \xrightarrow{u \cdot a} q_f \xrightarrow{v} q_f \xrightarrow{v} \dots$$

By Theorems 6.6 and 6.9 $u \cdot a$ and v are minimal, and there exists a run ρ' of $\mathcal{A}'$ such that

$$\rho' = q'_0 \xrightarrow{u \cdot a} q_f \xrightarrow{v} q_f \dots$$

Then we have $u \cdot (a, q_f) \in \mathscr{L}(pre(\mathcal{A}'))$ and $\mathscr{L}(pre(\mathcal{A})) \subseteq \mathscr{L}(pre(\mathcal{A}'))$. In a similar way it is possible to show that $\mathscr{L}(pre(\mathcal{A}')) \subseteq \mathscr{L}(pre(\mathcal{A}))$ holds, thus proving that $\mathscr{L}(pre(\mathcal{A})) = \mathscr{L}(pre(\mathcal{A}'))$. $\square$

Theorem 6.15. *Given two equivalent UPA in normal form $\mathcal{A} = (Q, \Delta, q_0, F)$ and $\mathcal{A}' = (Q', \Delta', q'_0, F')$, after the execution of Algorithm 4, $\mathcal{A}$ and $\mathcal{A}'$ are isomorphic.*

Proof. We have already proved, by Lemma 6.14, that after the execution of first two steps of Algorithm 4 we have that:

1. $\mathcal{A}$ and $\mathcal{A}'$ recognize the same minimal prefix and pattern languages;
2. the graph of final states of $\mathcal{A}$ and $\mathcal{A}'$ are isomorphic;
3. $pre(\mathcal{A})$ and $pre(\mathcal{A}')$ recognize the same language (up to renaming of final states).

Thus, after minimization of $pre(\mathcal{A})$ and $pre(\mathcal{A}')$ (step 4), we obtain the same DFA, (recall that the minimum DFA is unique). Consequently, line 5, which rebuilds the UPA from the minimum prefix automaton and from the structure of final states, generates the same UPA for $\mathcal{A}$ and for $\mathcal{A}'$ (up to isomorphism).

In such a way, we have shown that Algorithm 4 generates a canonical form for UPA. $\square$

Remark 6.16. Note that Algorithm 4 generates a canonical form for ultimately periodic automaton that is not guaranteed to be minimal because the minimization of prefix automaton generates a deterministic automaton from a nondeterministic one, with possible blow-up of states.

Corollary 6.17. *The equivalence problem for UPA is decidable.*

Proof. Given two UPA $\mathcal{A}$ and $\mathcal{A}'$, by Theorem 6.15, we have $\mathscr{L}^\omega(\mathcal{A}) = \mathscr{L}^\omega(\mathcal{A}')$ iff after the execution of Algorithm 4, their canonical forms are isomorphic. $\square$

7 Achieving Compactness

In this section we show how to obtain compact automaton-based representations of sets of time granularities. First we introduce a minimization algorithm for UPA and then we show how to further decrease the size of the automaton by defining a more general subclass of Büchi automata, called Relaxed UPA (RUPA), which are expressively equivalent to UPA but generally provide more compact representations of time granularities.

7.1 Minimization of UPA

In the previous section, we have discussed the canonization algorithm for UPA. This algorithm can be easily modified in order to obtain, instead of a canonical UPA, a *minimum (normal form) UPA*, as we show in this section. However, as in the case of nondeterministic finite automata, there may exist different (non isomorphic) minimum UPA. The main idea underlying the algorithm is very simple: consider the canonization algorithm (Algorithm 4) and instead of building a minimum *deterministic* prefix automaton, build a minimum *nondeterministic* one. Algorithm 5 shows the procedure in details.

Algorithm 5 Minimization algorithm

1: Execute Algorithm 1 over $\mathcal{A}$ to minimize final states structure;
2: minimize $\mathscr{L}^{pre}(\mathcal{A})$ by Algorithm 2;
3: build the prefix automaton $pre(\mathcal{A})$;
4: build a minimum NFA recognizing $\mathscr{L}(pre(\mathcal{A}))$;
5: build a minimum UPA $\mathcal{A}'$ by Algorithm 3.

Minimization of nondeterministic finite automata (NFA) is more difficult than minimization of deterministic finite automata. The problem is PSPACE-complete (except for the case of one-letter-alphabet) [GJ78], whereas deterministic finite automata can be minimized in time $O(n \log n)$ [Hop71]. NFA minimization is investigated in several papers, such as Kameda and Weiner [KW70] and Matz and Potthoff [MP95]. We recall some basic properties of minimum NFA.

Definition 7.1. Given a finite automaton $\mathcal{A} = (Q, \Delta, q_0, F)$ and a state $q \in Q$, the *behaviour of* q is the set $bh(q) = \{v \in A^* | \exists f \in F. q \xrightarrow{v} f\}$.

That is, the behaviour of a state q is the set of all words that reaches a final state. Such a notion leads to the following definition of *state equivalence*.

Definition 7.2. Two states $q_1, q_2 \in Q$ are *equivalent* if and only if $bh(q_1) = bh(q_2)$.

Clearly, in a minimum automaton there are no pairs of equivalent states: otherwise, one could build a smaller automaton by simply merging equivalent states. Consider now the prefix automaton $pre(\mathcal{A})$: because of its particular language (where the recognized word are of the form $u \cdot (a, q_f)$, with $u \in A^*$, $a \in A$, and $q_f \in F$), any minimum NFA recognizing $\mathscr{L}(pre(\mathcal{A}))$ has the particular structure described in the following lemma.

Lemma 7.3. *Given an UPA in normal form $\mathcal{A} = (Q, \Delta, q_0, F)$, any minimum NFA $\mathcal{M}$ equivalent to $pre(\mathcal{A})$ satisfies the following properties:*

1. *$\mathcal{M}$ only has a single final state f with no exiting transitions;*
2. *the transitions labelled with symbols in $A \times F$ are exactly the transitions reaching f.*

Proof. Let $\mathcal{A} = (Q, \Delta, q_0, F)$ be an UPA in normal form, let $pre(\mathcal{A}) = (Q^P, \Delta^P, q_0, F)$ be the corresponding prefix automaton, and let $\mathcal{M}$ be a minimum NFA recognizing the language $\mathscr{L}(pre(\mathcal{A}))$.

By Propostion 6.12, $\mathcal{M}$ recognizes only words of the form $u \cdot (a, q_f)$: thus the transitions reaching a final state of $\mathcal{M}$ are labelled with symbols from $A \times F$, while all other transitions are labelled with symbols of A. Furthermore, the final states of $\mathcal{M}$ have no exiting transitions (because symbols of $A \times F$ can occur only as the last letter of a recognized word). Consequently, they have the same behaviour (i.e. $\{\varepsilon\}$).

Because a minimum NFA has no equivalent states, $\mathcal{M}$ could have only one final state f, and the transitions that reach f are exactly the ones labelled with symbols from $A \times F$. $\square$

The following result follows from Lemma 7.3 and it shows that our minimization algorithm builds an automaton that is at least of the same size of the input one.

Lemma 7.4. *Given an UPA in normal form $\mathcal{A} = (Q, \Delta, q_0, F)$, Algorithm 5 builds an UPA $\mathcal{A}' = (Q', \Delta', q_0', F')$ such that $|Q'| \leq |Q|$.*

Proof. In the previous sections we have already shown that lines 1 and 2 of the algorithm cannot increase the number of states of the automaton. Now we prove that neither line 4 nor line 5 can do it.

Consider the final states of $pre(\mathcal{A})$. They have no exiting transitions and, because they have the same behaviour $\{\varepsilon\}$, they are equivalent. We can build a smaller NFA by simply merging all final states of $pre(\mathcal{A})$. Thus, the minimum NFA $\mathcal{M}$ equivalent to $pre(\mathcal{A})$ (built at line 4) has at most $|Q \setminus F| + 1$ states.

When Algorithm 3 builds the output automaton $\mathcal{A}'$, it adds the final loops of $\mathcal{A}$ to $\mathcal{M}$, so $\mathcal{A}'$ has at most $|Q| + 1$ states. However, this upper bound is not strict: by Lemma 7.3, $\mathcal{M}$ has only a single final state f with no exiting transitions, and all transitions reaching f are labelled with pairs from $A \times F$. Algorithm 3 eliminates all the transitions reaching f, disconnecting it from the other states of the automaton. f is not a state of the output automaton $\mathcal{A}'$; consequently, $\mathcal{A}'$ has at most $|Q \setminus F| + |F| = |Q|$ states. $\square$

The preceding lemma shows that out algorithm builds an automaton that is (in general) smaller that the input one. To prove that it is minimum, we must prove some relevant properties of minimum normal form UPA.

Algorithm 5 builds an automaton having a minimal pattern language, and a prefix language which consists of only non-empty minimal prefixes. In general, it is not guaranteed that a minimum normal form UPA respects such conditions. In the following, we point out some relevant properties of prefix and pattern languages of minimum normal form UPA.

Lemma 7.5. *Every minimum UPA recognizes a minimal pattern language.*

Proof. Suppose by absurd that there exists a minimum UPA $\mathcal{M}$ such that $\mathscr{L}^{pat}(\mathcal{M})$ is not minimal. So, there exists a non primitive pattern v encoded by some final loop C_f of $\mathcal{M}$. You can build a smaller automaton by simply substituting C_f by a smaller loop encoding a primitive pattern, against the hypothesis that $\mathcal{M}$ is minimum. □

Lemma 7.6. *For every ω-regular language of ultimately periodic words L, there exists a minimum normal form UPA $\mathcal{M}$ such that $\mathscr{L}^{pre}(\mathcal{M})$ contains only non-empty minimal prefixes.*

Proof. Let $\mathcal{M}$ be a minimum normal form UPA recognizing L. Then, by Algorithm 2 one could build an UPA $\mathcal{N}$ such that the prefix language contains only non-empty minimal prefixes and it has the same size of $\mathcal{M}$. □

The previous results allow us to prove the following theorem, that shows that the minimization algorithm is correct.

Theorem 7.7. *Given a normal form UPA $\mathcal{A} = (Q, \Delta, q_0, F)$, Algorithm 5 generates a minimum normal form UPA $\mathcal{A}' = (Q', \Delta', q_0', F')$.*

Proof. Let $\mathcal{A} = (Q, \Delta, q_0, F)$ be a normal form UPA recognizing L, and let $\mathcal{A}' = (Q', \Delta', q_0', F')$ be the output automaton of Algorithm 5.

By Lemmas 7.5 and 7.6, there exists a minimum normal form UPA $\mathcal{M}$ recognizing L that has the same prefix and pattern language of $\mathcal{A}'$. Thus, $\mathcal{M}$ and $\mathcal{A}'$ have the same graph of final states, because the graph of final states of $\mathcal{A}'$ is unique and minimal and $\mathcal{M}$ is minimum. As a consequence, there exists a renaming of the final states of $\mathcal{M}$ such that the prefix automata $pre(\mathcal{A}')$ and $pre(\mathcal{M})$ recognize the same language.

By Lemma 7.3, whenever you minimize $pre(\mathcal{M})$ and rebuild the UPA by adding the final loops you obtain an automaton that has the same number of states of $\mathcal{M}$. Thus, because Algorithm 5 operates in the same way on automaton $\mathcal{A}$ (by minimizing the prefix automaton and by adding the same structure of final loops), the automaton $\mathcal{A}'$ has the same number of states of $\mathcal{M}$, thus proving that our algorithm builds a minimal normal form UPA. □

It is worth mentioning that our algorithm generates an UPA that is minimum with respect to *normal form UPA*. In general, there could exists a smaller UPA that is not in normal form, but only when the automaton recognizes a single word of the form v^ω. It's easy to see that in this case the minimum (non normal form) UPA has exactly $|v|$ sates (if v is primitive) and it consists of a single simple loop encoding v. The definition of normal form forces the automaton to have an extra initial state that cannot belongs to the final loop, thus the minimum normal form UPA has exactly $|v|+1$ states. However, because we introduce UPA to deal with (possibly infinite) sets of granularities, this case rarely occurs and in most practical cases our algorithm builds an automaton that is minimum with respect to all possible equivalent UPA.

7.2 Relaxed Ultimately Periodic Automata

In the previous sections, we have defined the class of ultimately periodic automata by introducing strict limitations on the structure of Büchi automata. These restrictions simplified algorithmic manipulation of automata and allowed us to define a unique canonical form to solve the equivalence problem. However, they may generate redundancies in the automaton-based representations, due to the presence of duplicated loops encoding the same pattern. For example, consider the automata of Figure 4.

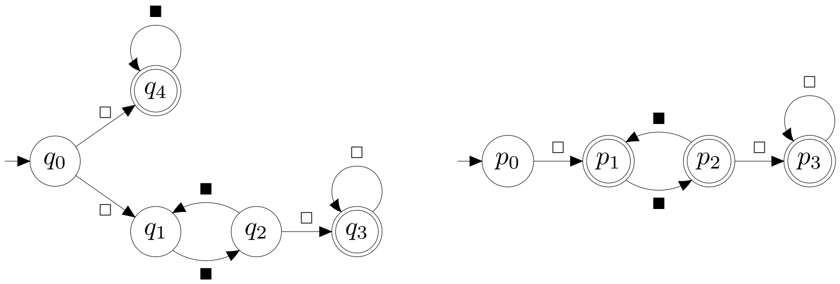

Figure 4: Examples of redundancy

It's easy to see that they recognize the same language of ultimately periodic words (that is, $\square\blacksquare^{\omega}+\square\blacksquare(\blacksquare\blacksquare)^{*}\square^{\omega}$) and that the one to the right-side has one state less than the other, but it is not a proper UPA (recall that we do not allow transitions departing from final loops). Thus it seems that the definition of UPA is too restrictive: indeed, in order to assure that it recognizes only ultimately periodic words, it would suffice to require that when the automaton leaves a final loop it cannot return to it.

In order to guarantee that the automata recognize only ultimately periodic words even with transitions exiting from final loops, we consider the strongly connected components. It's easy to see that the states of a final loop belong to the same SCC. Thus, to assure that whenever the automaton leaves a final loop it cannot return to it, it's sufficient to require that SCCs and final loops coincide. Indeed, if the automaton can leave a final loop C_f and then return to it, there exists a state q that is reachable from C_f and from which the automaton can reach C_f: thus q belongs to the same SCC of C_f but is not in C_f. These observations yield a definition of automata that "relaxes" the constraints of UPA and preserves the same expressiveness.

Definition 7.8. A Büchi automaton $\mathcal{A}=(Q,\Delta,q_0,F)$ is a *relaxed UPA* (RUPA) if for each $f\in F$, the SCC of f is a simple loop made only of final states.

Theorem 7.9. *The class of RUPA captures exactly the ω-regular languages of ultimately periodic words.*

Proof. It's easy to see that every UPA in normal form is a RUPA, thus any language recognized by an UPA is recognized by a RUPA.

We now prove that RUPA recognize only ω-regular languages of ultimately periodic words. Consider an accepting run of a RUPA $\mathcal{A} = (Q, \Delta, q_0, F)$. From the acceptance condition of Büchi automata, we know that there exists a final state f that occurs infinitely often in it. By definition of RUPA, there exists a unique simple loop C_f that contains f. Because C_f is a simple loop, it encodes a single pattern v_f, and the word recognized by $\mathcal{A}$ is also ultimately periodic. $\square$

7.2.1 From UPA to RUPA

In this section we show how to exploit the definition of RUPA in order to obtain more compact descriptions of time granularities.

To understand the underlying idea, consider the SCCs of an UPA. We have already noticed that final loops are itself single SCC. To reduce redundancies we search for a SCC S that meets the following conditions:

1. S is a single loop of non-final states;
2. S encodes a single pattern v (up to rotations);
3. there exists a final loop C_f that encodes a pattern v_f that is equivalent to v (with respect to Definition 4.4);
4. C_f and S have the same entering transitions.

If all such conditions are satisfied, you can merge S and C_f and preserve the recognized language.

As an example, consider the left-side automaton of Figure 4. In this case the SCC $\{q_1, q_2\}$ meets the above conditions:

1. it is a single loop of non final states;
2. it encodes a single pattern ■■;
3. ■■ is equivalent to the pattern encoded by the final loop $\{q_3\}$;
4. $\{q_1, q_2\}$ and q_3 have the same entering transitions.

Thus q_3 can be removed from the automaton by letting q_1 and q_2 to be final states. Note that the resulting automaton is the one on the right-side. The following definition generalizes such a construction.

Definition 7.10. A loop C_1 of the automaton $\mathcal{A} = (Q, \Delta, q_0, F)$, having no exiting transitions and encoding the pattern v is said to be *redundant* with respect to another loop C_2 iff C_2 encodes a pattern that is a multiple of v and has the same entering transitions of C_1, namely iff

1. C_1 is a SCC for $\mathcal{A}$, it is a simple loop encoding v and it has no transitions departing from it;
2. C_2 is a SCC for $\mathcal{A}$ and it is a simple loop that encodes a single word v^k, with $k \geq 1$;
3. there exists a surjective function $M : C_2 \to C_1$ that *respects the transitions*, namely such that:

- $\forall q, q' \in C_2.\ (q, a, q') \in \Delta \Leftrightarrow (M(q), a, M(q')) \in \Delta$;
- $\forall q \in (Q \setminus C_1 \setminus C_2), q' \in C_1.\ (q, a, q') \in \Delta \Leftrightarrow \exists q'' \in C_2.(q' = M(q'') \wedge (q, a, q'') \in \Delta$.

Instead of the more general condition that requires C_2 to encode a pattern $v_2 \cong v$, we required that C_2 encodes a pattern v^k. This because we are assuming to work on UPA that are minimal or in canonical form: in this case, final loops encode only minimal patterns and thus $v_2 \cong v$ holds iff there exists a rotation of v_2 that is equal to v^k, with $k \geq 1$.

Algorithm 6 shows how to compute compact RUPA from UPA by elimination of redundant final loops.

Algorithm 6 Sketch of an algorithm that eliminates redundant loops

```
1: let be 𝒜 = (Q, Δ, q_0, F)
2: while there are redundant final loops do
3:     find a final loop C_f with no exiting transitions redundant with respect to
       C_n
4:     delete C_f
5:     F ← F ∪ C_n
6: end while
```

Lemma 7.11. *Given a RUPA $\mathcal{A} = (Q, \Delta, q_0, F)$ and a final loop C_f that is redundant with respect to a non-final loop C_n, the automaton $\mathcal{A}' = (Q \setminus C_f, \Delta, q_0, (F \setminus C_f) \cup C_n)$ is a RUPA that recognizes the same language of $\mathcal{A}$.*

Proof. To prove this lemma we first show that $\mathcal{A}'$ is a RUPA. Consider a final state f of $\mathcal{A}'$. If $f \in F$ then it is also a final state of $\mathcal{A}$ and it satisfies the definition of RUPA. Otherwise, $f \in C_n$ and

1. C_n is a SCC of $\mathcal{A}$;
2. C_n is a simple loop of final states and it encodes a pattern v^k (where v is encoded by C_f).

Thus, f satisfies the conditions of Definition 7.8 and $\mathcal{A}'$ is a RUPA.

To show that $\mathscr{L}^\omega(\mathcal{A}) = \mathscr{L}^\omega(\mathcal{A}')$, consider a word $w \in \mathscr{L}^\omega(\mathcal{A})$ and an accepting run ρ of $\mathcal{A}$ over w. Further let $M : C_n \to C_f$ be a surjective function satisfying the condition 3. of Definition 7.10. Because C_f has no transitions departing from it, the states of C_f either do not occur in ρ or they occur infinitely often. If the states of C_f does not occur in ρ, then ρ is also an accepting run for $\mathcal{A}'$. Otherwise, we can build an accepting run ρ' for $\mathcal{A}'$ by replacing each occurrence of a state $f \in C_f$ with a state $q \in C_n$ such that $M(q) = f$ and the labelling of the transitions are respected: since M is surjective such a state q exists. Thus ρ' is a run of $\mathcal{A}'$ accepting w because C_n is a final loop of $\mathcal{A}'$.

For the converse inclusion, consider a word $w \in \mathscr{L}^\omega(\mathcal{A}')$ and an accepting run ρ' of $\mathcal{A}'$ over it. Because $\mathcal{A}'$ is a RUPA there exists a final loop whose states occur infinitely often in ρ'. If such a loop differs from C_n, then ρ' is also an accepting run of $\mathcal{A}$ over w. Otherwise, we can build an accepting run ρ of $\mathcal{A}$ by replacing each occurrence of a state $q \in C_n$ with $M(q)$: since M respects the transitions, ρ is a valid run of $\mathcal{A}$ and it accepts w. □

The preceding lemma implies that, after executing lines 3-5 of Algorithm 6, the resulting automaton recognizes the same language.

Theorem 7.12. *Given an UPA $\mathcal{A} = (Q, \Delta, q_0, F)$ that is either minimal or in canonical form, Algorithm 6 terminates and returns an equivalent RUPA $\mathcal{A}' = (Q', \Delta', q_0', F')$ that has no redundant loops.*

Proof. Because $\mathcal{A}$ is minimal or in canonical form, every final loop C_f has no exiting transitions and it cannot be redundant with respect to any other final loop (this because every final loop encodes a non-equivalent pattern). Thus, by Lemma 7.15, C_f can be redundant only with respect to non-final loops. When algorithm merges C_f and a non-final loop C_n (lines 4-5), it introduces a non-redundant final loop, because C_n must have exiting transitions (otherwise, C_n cannot reach a final state and it cannot be part of the automaton).

To prove that the algorithm terminates, it is sufficient to note that there are only finitely many redundant final loops and that they are merged at most once during the execution of the algorithm.

Consequently, the resulting automaton has no redundant final loops and it is equivalent to the original automata provided in input. □

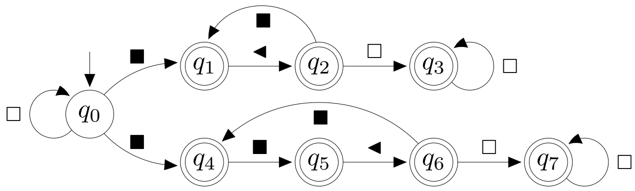

Figure 5: The RUPA obtained from UPA of Figure 3

Example 7.13. Figure 5 shows the RUPA that Algorithm 6 computes starting from the UPA depicted in Figure 3. It is easy to see that it recognizes the same language of the input one (namely, the set of all finite and infinite granularities that groups days either two by two or three by three) but is has 5 states less.

It is worth mentioning that line 3 of the algorithm randomly merges redundant loops, and that (in general) a final loop C_f could be redundant with respect to different non-final loops. Thus, the resulting automaton is not guaranteed to be unique. However, because the algorithm deletes all redundant loops in any case, all possible resulting automata have the same size, as stated in the following theorem.

Theorem 7.14. *Given an UPA $\mathcal{A} = (Q, \Delta, q_0, F)$, Algorithm 6 returns a RUPA that has a unique size, for every merging strategy the algorithm choose.*

Proof. We first show that the algorithm processes *every* redundant final loop of $\mathcal{A}$ before terminating. Suppose by absurd that Algorithm 6 terminates without merging a final loop C_f with the corresponding non-final loop C_n and consider the structure of the resulting automaton $\mathcal{A}'$: C_f and C_n are still present in the automaton and they are still redundant, because the algorithm doesn't add transitions nor states and it only merges final loops. Thus, $\mathcal{A}'$ contains a redundant final loop (i.e. C_f) and, accordingly to Theorem 7.12, Algorithm 6 must merge C_f before terminating.

Consequently, at the end of Algorithm 6, each redundant final loop of $\mathcal{A}$ is merged with the corresponding non-final loop. Consider two different merging strategies and suppose that the algorithm terminates by generating two automaton $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. We know that:

- they have the same states: algorithm doesn't add new states, and it merges the same final loops in both cases by deleting the same states from the automata;
- they have the same transitions: algorithm doesn't add new transitions and merges the same final loops in both cases by deleting the same transitions from the automata;

Thus, $\mathcal{A}_1$ and $\mathcal{A}_2$ have the same size, for any merging strategy chosen by the algorithm. □

Theorem 7.14 states that, for every merging strategy, the output automaton of Algorithm 6 has the same size. However, this does not mean that, whenever you start from a minimum UPA, it is a minimal RUPA .

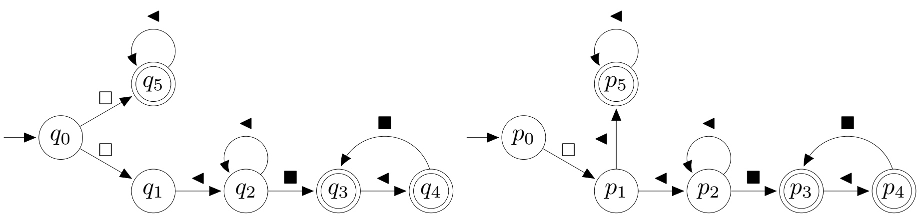

Figure 6: Two equivalent minimal UPA

Consider, for instance, the two UPA of Figure 6. They are both minimum UPA, but in the case of the leftmost one our algorithm cannot eliminates any redundancies, because loops $\{q_2\}$ and $\{q_5\}$ does not have the same entering transitions. On the other side, in the case of the rightmost automaton, loops $\{p_5\}$ and $\{p_2\}$ can be merged by the algorithm, producing a smaller RUPA.

7.2.2 Extending the canonical form

In the preceding section we showed how to create (starting from an UPA that is minimal or in canonical form) a smaller equivalent RUPA. We also showed that such a RUPA is not guaranteed to be unique or to be minimal. In this section we consider the case where the input automaton of Algorithm 6 is in canonical form, and we show that (only in this particular case) the resulting automaton is unique (up to isomorphisms). Because the prefix automaton of an UPA in canonical form is a deterministic automaton, there will be at most one redundant non final loop for every final loop of the UPA.

Lemma 7.15. *Given an UPA $\mathcal{A} = (Q, \Delta, q_0, F)$ in canonical form and a final loop C_f, there exists at most one non-final loop C_n such that C_f is redundant with respect to C_n.*

Proof. Suppose by contradiction that there exist two loops C_1 and C_2 with respect to which C_f is redundant. By Definition 7.10, C_1 and C_2 are disjoint SCCs encoding equivalent patterns and they have the same entering transitions. Since $\mathcal{A}$ has no unreachable states, there exists a state q from which you can reach both C_1 and C_2 by reading the same symbol. But $\mathcal{A}$ is in canonical form and its non-final states form a minimum deterministic finite automaton; hence, for each non final state q and for every symbol a, there is at most one transition departing from q and reading a (with exception for those transitions that reach final states). Then such a state cannot exist and C_f cannot be redundant with respect to both C_1 and C_2. □

As a consequence of the preceding lemma, the following theorem shows that the resulting automaton is unique, whatever strategy has been chosen for merging redundant final loops (provided that the input automaton is in canonical form).

Theorem 7.16. *Given an UPA $\mathcal{A} = (Q, \Delta, q_0, F)$ in canonical form, Algorithm 6 terminates and returns a RUPA that is unique for every merging strategy of final loops.*

Proof. We have already shown that the algorithm processes *every* redundant final loop of $\mathcal{A}$.

Consequently, at the end of Algorithm 6, every redundant final loop of $\mathcal{A}$ is merged with the corresponding non-final loop. Consider two different merging strategies and suppose that the algorithm terminates by generating two automaton $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. We know that:

- they have the same states: algorithm doesn't add new states, and it merges the same final loops in both cases by deleting the same states from the automata;

- they have the same transitions: algorithm doesn't add new transitions and merges the same final loops in both cases by deleting the same transitions from the automata;

- they have the same structure of final states: Each final loop is redundant with respect to at most one non-final loop; when the algorithm merges a final loop C_f it has only one possibility: it must add the states from C_n to F in both automata.

Thus, $\mathcal{A}_1 = \mathcal{A}_2$ for any merging strategy chosen by the algorithm. □

Theorem 7.16 proves that, starting from a canonical UPA, the resulting RUPA is unique with respect to the automaton provided in input: thus Algorithm 6 can be used to extend our canonization procedure for UPA, as exploited in the following.

Algorithm 7 Extended canonization algorithm

```
Let be 𝒜 = (Q, Δ, q_0, F) an UPA;
Execute Algorithm 4 over 𝒜 to canonize it;
Execute Algorithm 6 over 𝒜 to eliminate redundancies.
```

Algorithm 7 can be used to produce a RUPA with no redundancies starting from a generic UPA. The resulting automaton is, in general, smaller than the corresponding canonical UPA. Moreover, it could be used to solve the equivalence problem. Indeed, because an UPA in canonical form is unique with respect to the recognized language and because Algorithm 6 return an unique RUPA, Algorithm 7 produces a RUPA that has no redundancies and it is unique with respect to the recognized language.

8 Comparison of Granularities

In this section we describe a comprehensive set of problems involving time granularities that can be easily solved in the automaton-based framework. In [BJW00], Bettini et al. describe a wide set of relationships between granularities. These relationships yield a class of problems, called *granularity comparison problems*.

Definition 8.1. Given two sets of granularities $\mathcal{G}$ and $\mathcal{H}$, a *granularity comparison problem* is the problem of deciding whether or not there exist a granularity $G \in \mathcal{G}$ and a granularity $H \in \mathcal{H}$ such that $G \sim H$, with $\sim$ being one of the usual relations between granularities.

8.1 Relationships between granularities

Definition 8.2. Given two granularities G and H, we say that G *groups into* H and we write $G \trianglelefteq H$ iff, for every integer i, there is a (possibly infinite) set $S_i \subseteq \mathbb{N}^+$ such that $H(i) = \bigcup_{j \in S_i} G(j)$.

Definition 8.3. Given two granularities G and H, we say that G *is finer than* H and we write $G \preceq H$ iff, for every positive integer i, there is $j \in \mathbb{N}^+$ such that $G(i) \subseteq H(j)$.

Definition 8.4. Given two granularities G and H, we say that G is an *aligned refinement* of H and we write $G \ll H$ iff, for every index $i \in \mathbb{N}^+$, $G(i) \subseteq H(i)$.

Definition 8.4 says that a granularity G is an aligned refinement of another granularity H iff every non-empty granule of G is a subset of the granule of H with the same index. Thus, the refinement relation is a restriction of the relation finer-than.

8.2 Granularity relationship in terms of strings

In the case of time granularities represented by ω-words, all the relations defined in the previous section can be charachterized in terms of relations between ω-strings. In this section we focus on the aligned refinement relation, and we show how to solve the corresponding comparison problem in the automaton-based framework.

Lemma 8.5. *Given two granularities G and H represented by the ω-words g and h, respectively, $G \ll H$ holds iff $g(i) \in \{\blacksquare, \blacktriangleleft\}$ implies $h(i) \in \{\blacksquare, \blacktriangleleft\}$ and $\#_{\blacktriangleleft}(g[1, i-1]) = \#_{\blacktriangleleft}(h[1, i-1])$, for every $i \in \mathbb{N}^+$.*

Proof. Let G and H be two granularities such that $G \ll H$, and consider their string-based representations g and h, respectively. Given a non-empty granule $G(i)$, let $G(i) \subseteq H(i)$. Further, let j be a time point covered by $G(i)$. We know that j is covered by $H(i)$, as well. Consider now the symbols $g(j)$ and $h(j)$; we have:

- $g(j) \neq \square$ and $h(j) \neq \square$;
- if $g(j) \in \{\blacksquare, \blacktriangleleft\}$, then $h(j)$ is covered by $H(i)$, thus the number of occurrences of $\blacktriangleleft$ in $g[1, j-1]$ and $h[1, j-1]$ is equal to $i-1$ in both cases.

Therefore, g and h satisfy the two conditions of the lemma.

Conversely, given two strings g and h satisfying conditions 1. and 2., let G and H be the granularities represented by g and h, respectively. Consider the symbols $g(i)$ and $h(i)$; we have:

- if $h(i) = \square$, then $g(i) = \square$, and i is not covered by G nor by H;
- if $g(i) \in \{\blacksquare, \blacktriangleleft\}$, then $h(i) \in \{\blacksquare, \blacktriangleleft\}$. Let j be the number of occurrences of $\blacktriangleleft$ in $g[1, i-1]$ and $h[1, i-1]$: i is covered both by $G(j+1)$ and by $H(j+1)$.

So, every time point covered by a granule $G(j)$ is covered by $H(j)$, as well; thus $G \ll H$. $\square$

The granularity comparison problem for the aligned refinement relationship can be solved using RUPA.

Definition 8.6. Given two RUPA $\mathcal{A}_1 = (Q_1, \Delta_1, q_0^1, F_1)$ and $\mathcal{A}_2 = (Q_2, \Delta_2, q_0^2, F_2)$, the *product automaton* $\mathcal{A} = (Q, \Delta, q_0, F)$ is defined as follows:

- $Q = Q_1 \times Q_2 \times \{0, 1, 2\}$;
- $((q_1, q_2, n), (a_1, a_2), (q_1', q_2', m)) \in \Delta$ iff $(q_1, a_1, q_1') \in \Delta_1$, $(q_2, a_2, q_2') \in \Delta_2$ and one of the following holds:
  1. $n = m = 0$ and $a_1 = a_2 \in \{\blacksquare, \blacktriangleleft\}$;
  2. $n = 0$, $m = 1$, $a_1 = \blacktriangleleft$ and $a_2 = \blacksquare$;
  3. $n = 0$, $m = 2$, $a_1 = \blacksquare$ and $a_2 = \blacktriangleleft$;
  4. $n = m$, $a_1 = \square$ and $a_2 \in \{\blacksquare, \square\}$;
  5. $n = 1$, $m = 0$, $a_1 = \square$ and $a_2 = \blacktriangleleft$;
  6. $n = m = 2$ and $a_1 = \square$;
- $q_0 = (q_0^1, q_0^2, 0)$;
- $F = F_1 \times F_2 \times \{0, 1, 2\}$;

Definition 8.6 is a variation of the general definition of product automaton. It extends the set of states with a flag in $\{0, 1, 2\}$, and it imposes suitable restrictions on the structure of the transitions to meet the conditions of Lemma 8.5. By exploiting the solution of the emptiness problem, we can check whether there exist pairs of granularities represented by $\mathcal{A}_1$ and $\mathcal{A}_2$ that satisfy the refined alignement relationship.

Lemma 8.7. *Given two RUPA $\mathcal{A}_1 = (Q_1, \Delta_1, q_0^1, F_1)$ and $\mathcal{A}_2 = (Q_2, \Delta_2, q_0^2, F_2)$, their product automaton $\mathcal{A} = (Q, \Delta, q_0, F)$ is such that for every finite word $u \times v \in (A^2)^\omega$ the following properties hold:*

1. if $q_0 \xrightarrow{u\times v} (q_1, q_2, 0)$ *then* $q_0^1 \xrightarrow{u} q_1$, $q_0^2 \xrightarrow{v} q_2$ *and* $\#_{◀}(u) = \#_{◀}(v)$*;*

2. $q_0 \xrightarrow{u\times v} (q_1, q_2, 1)$ *iff* $q_0^1 \xrightarrow{u} q_1$, $q_0^2 \xrightarrow{v} q_2$ *and* $\#_{◀}(u) = \#_{◀}(v) + 1$*;*

3. $q_0 \xrightarrow{u\times v} (q_1, q_2, 2)$ *iff* $q_0^1 \xrightarrow{u} q_1$, $q_0^2 \xrightarrow{v} q_2$ *and* $\#_{◀}(u) < \#_{◀}(v)$*.*

Proof. We prove the Lemma by induction on the length of $u \times v$. If $u \times v = \varepsilon$ the properties trivially holds.

If $|u \times v| = n + 1$, let $u = u' \cdot a$ and $v = v' \cdot b$. Three cases arise:

1. if $\#_{◀}(u') = \#_{◀}(v')$, then $q_0 \xrightarrow{u'\times v'} (q_1, q_2, 0)$ holds. Consider the definition of product automaton; only conditions 1., 2., 3. and 4. on the transition relation can hold, and in any case lemma is satisfied.

2. If $\#_{◀}(u') = \#_{◀}(v')+1$, then $q_0 \xrightarrow{u'\times v'} (q_1, q_2, 1)$ holds. Consider the definition of product automaton; only conditions 4. and 5. on the transition relation can hold, and in any case lemma is satisfied.

3. If $\#_{◀}(u') < \#_{◀}(v')$, then $q_0 \xrightarrow{u'\times v'} (q_1, q_2, 2)$ holds. Consider the definition of product automaton; only conditions 4. and 6. on the transition relation can hold, and in any case lemma is satisfied.

□

Theorem 8.8. *Let $\mathcal{A}_1$ be a RUPA representing the granularity set $\mathcal{G}$ and $\mathcal{A}_2$ be a RUPA representing the granularity set $\mathcal{H}$. Then there exist two granularities $G \in \mathcal{G}$ and $H \in \mathcal{H}$ such that $G \ll H$ iff the language of the product automaton of $\mathcal{A}_1$ and $\mathcal{A}_2$ is nonempty.*

Proof. Given two RUPA $\mathcal{A}_1$ and $\mathcal{A}_2$ representing the sets of granularities $\mathcal{G}$ and $\mathcal{H}$, respectively, let $G \in \mathcal{G}$ and $H \in \mathcal{H}$ such that $G \ll H$. Consider the ω-words g and h representing G and H, respectively. It is clear that $g \in \mathscr{L}^\omega(\mathcal{A}_1)$, $h \in \mathscr{L}^\omega(\mathcal{A}_2)$, and g and h satisfy Lemma 8.5.

We show that it is possible to build an accepting run of the product automaton over $g \times h$, by proving that for every $i \in \mathbb{N}$, $g[1, i] \times h[1, i]$ satisfies Lemma 8.7 by induction on i. Let ρ_1 be a run of $\mathcal{A}_1$ over g and let ρ_2 be a run of $\mathcal{A}_2$ over h. It is clear that for every $i \in \mathbb{N}$, both $q_0^1 \xrightarrow{g[1,i]} \rho_1(i + 1)$ and $q_0^2 \xrightarrow{h[1,i]} \rho_2(i + 1)$ holds.

For $i = 0$, Lemma 8.7 trivially holds. For $i \geq 1$, let $u = g[1, i - 1]$ and $v = h[1, i - 1]$, and consider the symbols $g(i)$ and $h(i)$; we have:

- if $g(i) \in \{■, ◀\}$, then $h(i) \in \{◀, ■\}$ and $\#_{◀}(u) = \#_{◀}(v)$ hold. Thus, $q_0 \xrightarrow{u\times v} (\rho_1(i), \rho_2(i), 0)$ and you can build a run of the product automaton over $u \cdot g(i) \times v \cdot h(i)$ that satisfies Lemma 8.7;

- if $g(i) = □$, then (by induction hypothesis) $u \times v$ satisfies Lemma 8.7 and $q_0 \xrightarrow{u\times v} (\rho_1(i), \rho_2(i), n)$ holds. Thus, for any value of n and for any symbol $h(i)$, it is possible to build a run of the product automaton over $u \cdot g(i) \times v \cdot h(i)$ that satisfies Lemma 8.7.

Consequently, there exists a valid run ρ of the product automaton that reads $g \times h$. Furthermore, this run is also accepting: because $\mathcal{A}_1$ and $\mathcal{A}_2$ are RUPA, ρ_1

and ρ_2 are ultimately periodic and there exist a state (f_1, f_2, n) of the product automaton that occurs infinitely often in ρ.

Conversely, consider a word $g \times h$ recognized by the product automaton. It is clear that $g \in \mathscr{L}^\omega(\mathcal{A}_1)$ and $h \in \mathscr{L}^\omega(\mathcal{A}_2)$. Thus g represents a granularity $G \in \mathcal{G}$ and h represents a granularity $H \in \mathcal{H}$. For every $i \in \mathbb{N}^+$, consider the symbols $g(i)$ and $h(i)$: if $g(i) \in \{\blacksquare, \blacktriangleleft\}$ then $h(i) \in \{\blacksquare, \blacktriangleleft\}$ and (by conditions on the transition relation) $q_0 \xrightarrow{g[1,i-1] \times h[1,i-1]} (q_1, q_2, 0)$. Hence, $\#_{\blacktriangleleft}(g[1, i-1]) = \#_{\blacktriangleleft}(h[1, i-1])$ and g and h satisfy Lemma 8.5. $\square$

9 Beyond (R)UPA

In this section we briefly investigate possible extensions of (R)UPA. We have shown that the class of (R)UPA is the subclass of Büchi automata that recognize ω-regular languages of ultimately periodic words. As we shall illustrate in the next section, (R)UPA allow one to deal with meaningful real-world applications. Nevertheless, there are many sets of of periodical granularities which are not captured by (R)UPA.

Recall, from Section 4, that we identify with A^{UP} the set of all ultimately periodic words, and that we define the complementation of a language of ultimately periodic words L as $\overline{L} = A^{UP} \setminus L$.

9.1 Three-phase automata (3PA)

In this section, we characterize the language L for which there exists an ω-regular language L' such that $L = \mathrm{UP}(L')$ as the languages recognized by a new class of automata, called *Three-phase Automata* (3PA). This set of languages includes both ω-regular languages (the UPA-recognizable languages L, for which $L = \mathrm{UP}(L)$), and non-ω-regular languages (the languages L such that $L = \mathrm{UP}(L') \subset L$). In particular, unlike UPA, 3PA are able to capture sets of granularities featuring an infinite number of non-equivalent repeating patterns. Computations of 3PA consist in three steps: (i) the automaton guesses the prefix of an ultimately periodic word, then (ii) it guesses its repeating pattern and stores it in a queue, and finally (iii) it recognizes the stored pattern infinitely many times.

Definition 9.1. A *Three-phase Automata* (3PA) over an alphabet A is a tuple $\mathcal{A} = (Q, \Delta, q_0, F)$, where:

- Q is a finite set of states;
- $\Delta \subseteq Q \times A \times Q$ is the transition relation;
- $q_0 \in Q$ is the initial state;
- $F \subseteq Q$ is the set of final states.

The definition of 3PA is identical to the one of Büchi automata. What changes are the definition of computation and the acceptance condition.

Definition 9.2. A *configuration* of a Three-phase automaton $\mathcal{A}$ is a triple (q, P, s), where $q \in Q$, $P \in A^*$ and $s \in F \cup \overline{F} \cup \{\bot\}$.

A configuration (q, P, s) of an automaton represents its computational state at a given instant: q is the current state, P is the content of the queue and s represents one of the three possible phase of the computation:

- if $s = \perp$, the automaton reads the prefix of the word, and the queue is empty;
- if $s \in F$, the automaton reads the pattern for the first time, and stores it in the queue;
- if $s \in \overline{F} = \{\bar{f} | f \in F\}$ the automaton reads infinitely often the pattern contained in the queue.

To formalize this concept, we give the following definition of run of a Three-phase automaton.

Definition 9.3. A *run* (or computation) of a Three-phase automaton $\mathcal{A} = (Q, \Delta, q_0, F)$ over a (finite or infinite) word w is a sequence $\rho = \rho(1)\rho(2)\ldots$ of configurations where $\rho(i) = (q_i, P_i, s_i)$ and:

- $(q_i, w_i, q_{i+1}) \in \delta$ for each $1 \leq i \leq |w|$;
- one of the following conditions holds:
    1. $s_i = \perp$, $P_{i+1} = P_i$ and $s_{i+1} = \perp$;
    2. $s_i = \perp$, $q_{i+1} \in F$, $P_{i+1} = P_i$ and $s_{i+1} = q_{i+1}$;
    3. $s_i \in F$, $P_{i+1} = P_i \cdot w_i$ and $s_{i+1} = s_i$;
    4. $s_i \in F$, $q_{i+1} = s_i$, $P_{i+1} = P_i \cdot w_i$ and $s_{i+1} = \bar{s}_i$;
    5. $s_i \in \overline{F}$, $P_i = w_i \cdot P$, $P_{i+1} = P \cdot w_i$ and $s_{i+1} = s_i$.

A run of the 3PA $\mathcal{A}$ over an infinite word w is said to be *accepting* iff $\rho(1) = (q_0, \varepsilon, \perp)$ and there exists $f \in F$ and $P \in A^+$ such that $(f, P, \bar{f})$ occurs infinitely often in ρ. The language recognized by a three-phase automaton $\mathcal{A}$ is denoted by $\mathscr{L}^{up}(\mathcal{A})$. The following theorem shows that 3PA recognizes exactly the class of languages of ultimately periodic words L for which there is an ω-regular language L' such that $L = \mathrm{UP}(L')$.

Lemma 9.4. *If $\mathcal{A}$ is a Büchi automaton, for every ultimately periodic word $w \in \mathrm{UP}(\mathscr{L}^\omega(\mathcal{A}))$ there is an ultimately periodic run accepting it.*

Proof. Let $w = u \cdot v^\omega \in \mathscr{L}^\omega(\mathcal{A})$, and let ρ be an accepting run of $\mathcal{A}$ over w. If ρ is ultimately periodic, we have finished.

If ρ is not ultimately periodic, we show how to build an ultimately periodic accepting run ρ' of $\mathcal{A}$ starting from ρ. Consider now the suffix $\sigma = \rho[|u| + 1, \omega[$ of ρ that reads v^ω. Clearly, there exists a final state $f \in F$ that occurs infinitely often in it. We extend σ by associating at every state $\sigma(i)$ a tuple $(\sigma(i), a, k)$, where:

- a is the symbol $w(|u| + i)$ read by the automaton when it is in $\sigma(i)$;
- $k = (i \ mod \ |v|) + 1 \in [1, |v|]$ is the position in v of a.

Since σ is infinite, there exists at least one tuple (q, a, k) that occurs infinitely often in it: let i be the first occurrence of (q, a, k) in σ, and let j be the first occurrence of (q, a, k) after i such that there exists at least one occurrence of f in $\sigma[i, j - 1]$.

It is easy to see that the run $\sigma[1, i-1] \cdot (\sigma[i, j-1])^\omega$ recognizes v^ω. Therefore, an ultimately periodic run recognizing w can be built as:

$$\rho' = \rho[1, |u|] \cdot \sigma[1, i - 1] \cdot (\sigma[i, j - 1])^\omega.$$

□

Theorem 9.5. *For every automaton $\mathcal{A}$, $\mathscr{L}^{up}(\mathcal{A}) = \mathrm{UP}(\mathscr{L}^{\omega}(\mathcal{A}))$, namely, the set of ultimately periodic words contained in the language of $\mathcal{A}$, considered as a Büchi automaton, is exactly the language of ultimately periodic words recognized by $\mathcal{A}$, considered as a 3PA.*

Proof. Let $\mathcal{A} = (Q, \Delta, q_0, F)$ be an automaton. We prove that $\mathscr{L}^{up}(\mathcal{A}) = \mathrm{UP}(\mathscr{L}^{\omega}(\mathcal{A}))$. First, let $w \in \mathscr{L}^{up}(\mathcal{A})$. It is clear that every accepting run of a 3PA is also an accepting run of the corresponding Büchi automaton, thus $w \in \mathrm{UP}(\mathscr{L}^{\omega}(\mathcal{A}))$.

Conversely, let $w \in \mathrm{UP}(\mathscr{L}^{\omega}(\mathcal{A}))$. From Lemma 9.4, it follows that there exists an ultimately periodic run ρ of $\mathcal{A}$ over w. If $f \in F$ is a final state that occurs infinitely often in the run, then ρ is of the form:

$$\rho = q_0 \xrightarrow{u} f \xrightarrow{v} f \xrightarrow{v} f \xrightarrow{v} f \rightarrow \ldots, \text{ with } w = u \cdot v^{\omega}.$$

Hence, the following run ρ' is an accepting run of the 3PA $\mathcal{A}$ over w:

$$\rho' = \begin{pmatrix} q_0 \\ \varepsilon \\ \bot \end{pmatrix} \xrightarrow{u} \begin{pmatrix} f \\ \varepsilon \\ f \end{pmatrix} \xrightarrow{v} \begin{pmatrix} f \\ v \\ f \end{pmatrix} \xrightarrow{v} \begin{pmatrix} f \\ v \\ f \end{pmatrix} \xrightarrow{v} \begin{pmatrix} f \\ v \\ f \end{pmatrix} \rightarrow \ldots$$

and we have shown that $w \in \mathscr{L}^{up}(\mathcal{A})$. ☐

Corollary 9.6. *3PA recognize all and only the languages of ultimately periodic words L for which there exists an ω-regular language L' such that $L = \mathrm{UP}(L')$.*

Theorem 9.5 establishes a direct connection between Büchi automata and 3PA, that can be exploited in order to demonstrate the closure properties of 3PA under the main operations on languages.

Theorem 9.7. *3PA-recognizable languages are closed under union, intersection, concatenation and complementation.*

Proof. Given two 3PA-recognizable languages $L_1, L_2 \subseteq A^{UP}$, let $M_1, M_2 \subseteq A^{\omega}$ be the ω-regular languages such that $L_1 = \mathrm{UP}(M_1)$ and $L_2 = \mathrm{UP}(M_2)$. From Theorem 9.5, it is easy to see that the following properties holds:

- $\mathrm{UP}(M_1) \cup \mathrm{UP}(M_2) = \mathrm{UP}(M_1 \cup M_2)$;
- $\mathrm{UP}(M_1) \cap \mathrm{UP}(M_2) = \mathrm{UP}(M_1 \cap M_2)$;
- $V \cdot \mathrm{UP}(M_1) = \mathrm{UP}(V \cdot M_2)$, for every regular language $V \subseteq A^*$;
- $\overline{\mathrm{UP}(M_1)} = \mathrm{UP}(\overline{M_1})$.

Therefore, 3PA recognizing the languages $L_1 \cup L_2$, $L_1 \cap L_2$, $V \cdot L_1$ and $\overline{L_1}$ can be constructed using the same techniques of Büchi automata. ☐

Many interesting languages of ultimately periodic words are 3PA-recognizable: for instance, all UPA-recognizable languages and their complements. Moreover, there are other languages that are 3PA-recognizable, such as the language of the granularities that covers the whole temporal domain, that is $L = \{u \cdot v^{\omega} | u, v \in \{\blacksquare, \blacktriangleleft\}^+\} = \mathrm{UP}(\{\blacksquare, \blacktriangleleft\}^{\omega})$, and its complement. Note that neither L nor $\overline{L}$ are UPA-recognizable. In the following we give a characterization of 3PA-recognizable languages in term of the operators of union, concatenation and infinite repetition (V^{inf}).

Theorem 9.8. *Every 3PA-recognizable language L is of the form*

$$L = \bigcup_{i=1}^{n} U_i \cdot V_i^{inf},$$

with U_i, V_i regular languages such that $U_i \cdot V_i \subseteq U_i$ and $V_i \cdot V_i \subseteq V_i$, for every $1 \leq i \leq n$.

Proof. Given a 3PA-recognizable language L, let M be the corresponding ω-regular language. M is of the form

$$M = \bigcup_{i=1}^{n} U_i \cdot V_i^{\omega},$$

with U_i, V_i regular languages of finite words. Thus, L can be characterized by the following expressions:

$$\begin{aligned} L &= \mathrm{UP}\left(\bigcup_{i=1}^{n} U_i \cdot V_i^{\omega}\right) \\ &= \bigcup_{i=1}^{n} \mathrm{UP}(U_i \cdot V_i^{\omega}) \\ &= \bigcup_{i=1}^{n} U_i \cdot \mathrm{UP}(V_i^{\omega}) \\ &= \bigcup_{i=1}^{n} U_i \cdot V_i^{*} \cdot (V_i^{+})^{inf} \\ &= \bigcup_{i=1}^{n} U_i' \cdot V_i'^{inf}, \end{aligned}$$

with $U_i' = U_i \cdot V_i^{*}$ and $V_i' = V_i^{+}$. □

The theorem shows that 3PA-recognizable languages can contain infinitely many non-equivalent patterns, but with some limitations. In particular, the following restrictions on the structure of 3PA-recognizable languages holds:

1. a 3PA cannot recognize languages with an empty set of prefixes: for example, the language of all *periodic words*, that is $\{v^{\omega} | v \in A^{+}\}$, is not 3PA-recognizable;
2. every set of the form $U_i \cdot V_i^{inf}$ that contains infinitely many non-equivalent patterns, must contains every pattern that is combination of them. This means, for instance, that the language of the unanchored granularities that groups days n by n, that is $\{\square^{*} \cdot (\blacksquare^{n} \blacktriangleleft)^{\omega} | n \geq 0\}$, is not 3PA-recognizable: every 3PA that recognize these repeating patterns must indeed also recognize all, but finitely many, combinations of them.

Figure 7 summarize the relations between the class of languages that we have investigated till now. It also depicts the following example languages, placing them in the appropriate class:

- $L_1 = \{\square\}^{*} \cdot \{\blacksquare \blacktriangleleft\}^{\omega}$, that is an UPA-recognizable language, and its complement $\overline{L}_1$, that is 3PA-recognizable;

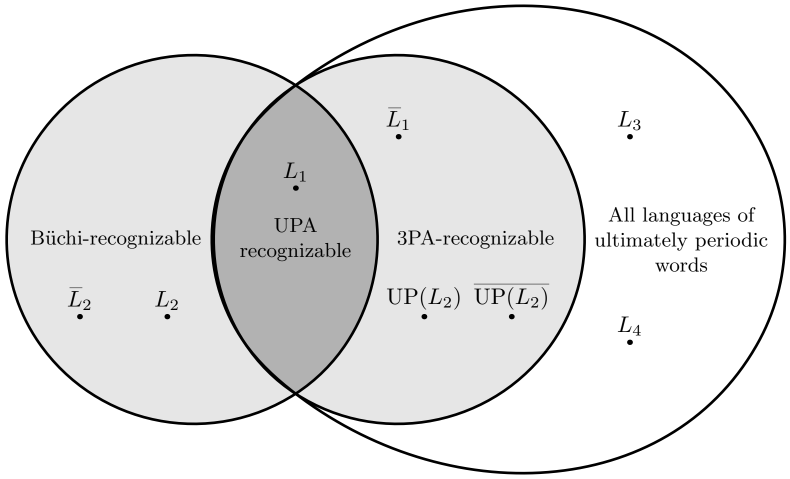

Figure 7: Relations between Büchi, UPA, and 3PA-recognizable languages

- $L_2 = \{\blacksquare, \blacktriangleleft\}^\omega$, and its complement $\overline{L}_2$, that are both Büchi-recognizable;
- the corresponding 3PA-recognizable languages $\mathrm{UP}(L_2)$ and $\overline{\mathrm{UP}(L_2)}$;
- the two example of non 3PA-recognizable languages given above, namely, $L_3 = \{v^\omega | v \in A^+\}$ and $L_4 = \{\square^*(\blacksquare^n \blacktriangleleft)^\omega | n \geq 0\}$.

The class of UPA-recognizable language is the smallest one, and it is contained in both Büchi-recognizable and 3PA-recognizable languages. Furthermore, UPA-recognizable languages are exactly the intersection between Büchi-recognizable and 3PA-recognizable languages. We have also shown that there are languages of ultimately periodic words that are not 3PA-recognizable.

9.2 Algorithms on 3PA

In the previous section, we have exploited the results of Theorem 9.5 in order to prove closure properties of 3PA. Furthermore, the strict correlation between 3PA and Büchi automata implies that every problem that can be solved on Büchi automata can be solved in the same way for 3PA (and viceversa). This observation gives us a straightforward way to effectively manage the problems that we have introduced in 5:

- **Emptiness**. The emptiness problem is solved in polynomial time by testing the existence of a loop involving some final state reachable from the initial state.
- **Membership**. The membership problem consists in deciding whether a 3PA $\mathcal{A}$ recognizes a given ultimately periodic word w. Given a 3PA $\mathcal{B}$ recognizing the singleton $\{w\}$, one can decide in polynomial time whether $w \in \mathscr{L}^{up}(\mathcal{A})$ by testing the emptiness of the language $\mathscr{L}^{up}(\mathcal{A}) \cap \mathscr{L}^{up}(\mathcal{B})$.
- **Equivalence**. One can decide whether two given 3PA $\mathcal{A}$ and $\mathcal{B}$ are equivalent in the same way as generic Büchi automata, by computing their

complements $\overline{\mathcal{A}}$ and $\overline{\mathcal{B}}$, and by testing the emptiness of both $\mathscr{L}^{up}(\mathcal{A}) \cap \mathscr{L}^{up}(\overline{\mathcal{B}})$ and $\mathscr{L}^{up}(\mathcal{B}) \cap \mathscr{L}^{up}(\overline{\mathcal{A}})$.

- **Comparison of granularities**. As for UPA, granularity comparison problems can be easily reduced to the emptiness problem for suitable product automata. In fact, the technique used in Section 8 to solve the aligned refinement problem for UPA can be generalized to 3PA.

10 A Real-World Application

The need of dealing with sets of time granularities arises in several application domains. We focus our attention on the medical domain of heart transplant patients. 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 an heart transplant patient reported in [Cen02]. 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. The ability of dealing with sets of granularities, and not only with single granularities, is thus needed to reason about protocols and patient schedules. As an example, since different protocols can be specified for the same class of patients by different people/institutions, it is a critical problem to decide whether two protocols define the same set of therapies/granularities (equivalence problem). The decidability of this problem gives the possibility of choosing the most compact, or 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 (granularity comparison problem).

Let us consider this latter problem. Consider the above given (sub)set of therapies/check-ups of a protocol for hearth transplant patients. Given an UPA $\mathcal{A}$ encoding (the granularities of) such a set of therapies/check-ups and an UPA $\mathcal{B}$ representing the single granularity of the specific therapy (up to a certain date), the granularity comparison problem can be decided by checking the existence of a word in $\mathscr{L}^{\omega}(\mathcal{A})$ that properly relates to the one contained in $\mathscr{L}^{\omega}(\mathcal{B})$. For the sake of simplicity, we consider months of 30 days and years of 365 days (relaxing such a simplification is tedious, but trivial). The UPA $\mathcal{A}$ is depicted in Figure 8, where we use the shorthand $\circ\xrightarrow{a^n}\circ$ to denote a sequence of $n+1$ states and n a-labeled transitions.

We model the granularity of the therapy assigned to the patient with a single ultimately periodic word v (equivalently, an SSA $\mathcal{B}$), where the occurrences of

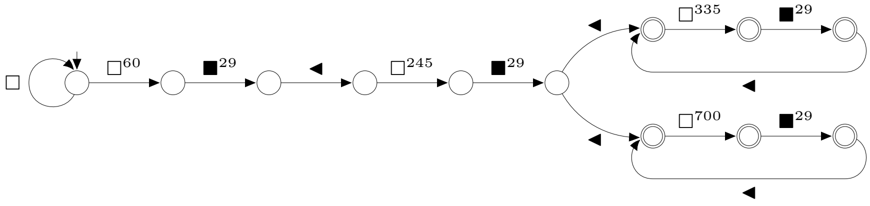

Figure 8: The UPA-based specification of the protocol.

$\blacktriangleleft$ denote the days of the visits. We can check the consistency of the therapy with respect to the prescribed protocol by testing whether the granularity v is an *aligned refinement* of some granularity $u \in \mathscr{L}^\omega(\mathcal{A})$. Thus, the given consistency-checking problem can be seen as a particular case of granularity comparison problem. Such a condition can be easily verified in polynomial time as shown in Section 8.

As an example, consider the following instance of the temporal relation $\texttt{VISITS}(\texttt{PatientId}, \texttt{Date}, \texttt{Treatment})$.

`PatientId`	`Date`	`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	01/11/2004	GFR
1001	01/29/2005	GFR

By properly selecting records, we can build the granularity of GFR measurements for the patient identified by 1001. We represent this granularity as a single ultimately periodic word v, in which the occurrences of $\blacktriangleleft$ denote the days of the visits. The UPA $\mathcal{B}$ recognizing v is depicted in Figure 9.

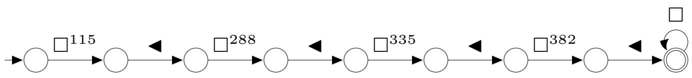

Figure 9: The UPA representing GFR measurements for patient 1001.

In order to check whether the granularity of GFR measurements for patient 1001 is an aligned refinement of some granularity in $\mathscr{L}^\omega(\mathcal{A})$, we must construct the product automaton for the relation of aligned refinement. Such an automaton recognizes the language

$$\Big\{ \tbinom{\square}{\square}^{100}\tbinom{\blacksquare}{\square}^{15}\tbinom{\blacksquare}{\blacktriangleleft}\tbinom{\blacksquare}{\square}^{13}\tbinom{\blacktriangleleft}{\square}\tbinom{\square}{\square}^{245}\tbinom{\blacksquare}{\square}^{29}\tbinom{\blacktriangleleft}{\blacktriangleleft}\tbinom{\square}{\square}^{335}\tbinom{\blacksquare}{\blacktriangleleft}\tbinom{\blacksquare}{\square}^{28}\tbinom{\blacktriangleleft}{\square}\tbinom{\square}{\square}^{335}\cdot$$
$$\cdot\tbinom{\blacksquare}{\square}^{18}\tbinom{\blacksquare}{\blacktriangleleft}\tbinom{\blacksquare}{\square}^{10}\tbinom{\blacktriangleleft}{\square}\Big(\tbinom{\square}{\square}^{335}\tbinom{\blacksquare}{\square}^{29}\tbinom{\blacktriangleleft}{\square}\Big)^{\omega} \Big\}$$

over the alphabet $\{\tbinom{\square}{\square}, \tbinom{\blacksquare}{\square}, \tbinom{\blacktriangleleft}{\square}, \tbinom{\blacksquare}{\blacksquare}, \tbinom{\blacksquare}{\blacktriangleleft}, \tbinom{\blacktriangleleft}{\blacksquare}, \tbinom{\blacktriangleleft}{\blacktriangleleft}\}$. Since the resulting language is not empty, we can conclude that the therapy satisfies the prescribed protocol.

11 Discussion

In this paper, we developed an original automaton-based approach to the management of sets of granularities. Such an approach makes it possible to represent sets of granularities possibly having infinitely many different prefixes and a finite number of non-equivalent repeating patterns. Furthermore, we showed how well-known results coming from automata theory can be exploited to solve several crucial problems. In particular, we provided effective solutions to the problems of emptiness, membership, equivalence, inclusion, minimization, and comparison of granularities. Successively, we discussed variants and extensions of UPA (RUPA, 3PA) that increase compactness and expressiveness of granularity representations. Finally, we discussed an example of application of these problems in the context of clinical medicine.

As for open problems, on the one hand we are looking for larger classes of languages of ultimately periodic words that extends the class of 3PA, possibly preserving closure and (some) decidability properties. On the other hand, we are currently looking for the (proper) fragment of PLTL^{mod} defining the temporal logic counterpart of UPA as well as for the logical counterpart of 3PA. Pairing the logical formalism with the automaton-based one would allow us 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

[BJW00] C. Bettini, S. Jajodia, and X.S. Wang. *Time Granularities in Databases, Data Mining, and Temporal Reasoning*. Springer, July 2000.

[Boo80] K. S. Booth. Lexicographically least circular substrings. *Information Processing Letters*, 10(4,5):240–242, 1980.

[Cen02] Loma Linda University Medical Center. Pediatric heart transplantation protocol, 2002.

[CFP04] C. Combi, M. Franceschet, and A. Peron. Representing and reasoning about temporal granularities. *Journal of Logic and Computation*, 14(1):51–77, 2004.

[CR88] J. Clifford and A. Rao. A simple general structure for temporal domains. In C. Rolland and M. Leonard, editors, *Temporal Aspects of Information Systems*, pages 17–28. Elsevier Science Publishers, North-Holland, 1988.

[CSS94] R. Chandra, A. Segev, and M. Stonebraker. Implementing calendars temporal rules in next generation databases. In *International Conference on Data Engineering*, pages 264–273. IEEE Computer Society Press, 1994.

[Dem04] S. Demri. LTL over integer periodicity constraints (extended abstract). In *Proceedings of the 7th Int. Conf. on Foundations of Software Science and Computation Structures (FOSSACS)*, volume 2987 of *Lecture Notes in Computer Science*, pages 121–135. Springer, April 2004.

[DM01] U. Dal Lago and A. Montanari. Calendars, time granularities, and automata. In *7th International Symposium on Spatial and Temporal Databases (SSTD)*, volume 2121 of *LNCS*, pages 279–298. Springer, 2001.

[DMP03a] U. Dal Lago, A. Montanari, and G. Puppis. Time granularities, calendar algebra, and automata. Technical Report 4, Dipartimento di Matematica e Informatica, Università di Udine, Italy, February 2003.

[DMP03b] U. Dal Lago, A. Montanari, and G. Puppis. Towards compact and tractable automaton-based representations of time granularity. In *8th Italian Conference on Theoretical Computer Science (ICTCS)*, volume 2841 of *LNCS*, pages 72–85. Springer, 2003.

[FM02] M. Franceschet and A. Montanari. Time granularities in databases, data mining, and temporal reasoning, by Claudio Bettini, Sushil Jajodia, and Sean X. Wang (book review). *The Computer Journal*, 45(6):683–685, 2002.

[Fra01] M. Franceschet. *Dividing and Conquering the Layered Land*. PhD thesis, Dipartimento di Matematica e Informatica, Università di Udine, Italy, 2001.

[FW65] N. J. Fine and H. S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16:109–114, 1965.

[GJ78] Michael R. Garey and David S. Johnson. *Computers and Intractability / A Guide to the Theory of NP-Completeness*. W.H. Freeman & Company, San Francisco, 1978.

[HMU01] J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 2001.

[Hop71] J.E. Hopcroft. An $n \, log \, n$ algorithm for minimizing the states in a finite automaton. In Z. Kohavi, editor, *Theory of Machines and Computations*, pages 189–196. Academic Press, New York, 1971.

[KW70] T. Kameda and P. Weiner. On the state minimization of nondeterministic finite automata. *IEEE Transactions on Computers*, C-19(7), July 1970.

[LMF86] B. Leban, D. McDonald, and D. Foster. A representation for collections of temporal intervals. In *AAAI National Conference on Artificial Intelligence*, volume 1, pages 367–371. AAAI Press, 1986.

[Mon96] A. Montanari. *Metric and Layered Temporal Logic for Time Granularity*. ILLC Dissertation Series 1996-02. Institute for Logic, Language and Computation, University of Amsterdam, 1996.

[MP95] O. Matz and A. Potthoff. Computing small nondeterministic automata. In *Proceedings of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, BRICS Notes Series, pages 74–88, 1995.

[NJW02] P. Ning, S. Jajodia, and X.S. Wang. An algebraic representation of calendars. *Annals of Mathematics and Artificial Intelligence*, 36:5–38, 2002.

[NS92] M. Niezette and J. Stevenne. An efficient symbolic representation of periodic time. In *International Conference on Information and Knowledge Management (CIKM)*, pages 161–168, Baltimore, MD, 1992. ACM Press.

[PTB85] R. Paige, R.E. Tarjan, and R. Bonic. A linear time solution to the single function coarsest partition problem. *Theoretical Computer Science*, 40:67–84, 1985.

[SAA+94] R.T. Snodgrass, I. Ahn, G. Ariav, D.S. Batory, J. Clifford, C.E. Dyreson, R. Elmasri, F. Grandi, C.S. Jensen, W. Käfer, N. Kline, K. Kulkarni, T.Y.C. Leung, N. Lorentzos, J.F. Roddick, A. Segev, M.D. Soo, and S.M. Sripada. TSQL2 language specification. *SIGMOD Record*, 23(1):65–86, 1994.

[Shi81] Y. Shiloach. Fast canonization of circular strings. *Journal of algorithms*, 2:107–121, 1981.

[Tho90] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 133–192. Elsevier MIT Press, 1990.

[Wij00] J. Wijsen. A string-based model for infinite granularities. In C. Bettini and A. Montanari, editors, *AAAI Workshop on Spatial and Temporal Granularities*, pages 9–16. AAAI Press, 2000.

[WJL91] G. Wiederhold, S. Jajodia, and W. Litwin. Dealing with granularity of time in temporal databases. In *Conference on Advanced Information Systems Engineering (CAiSE)*, pages 124–140. LNCS 498, Springer, 1991.

[WJS95] X.S. Wang, S. Jajodia, and V.S. Subrahmanian. Temporal modules: an approach toward federated temporal databases. *Information sciences*, 82:103–128, 1995.