Compact and Tractable Automaton-based Representations of Time Granularities

Ugo Dal Lago [a], Angelo Montanari [b], Gabriele Puppis [b]

[a] *Dipartimento di Scienze dell'Informazione, Università di Bologna, Mura Anteo Zamboni 7, 40127 Bologna, Italy*

[b] *Dipartimento di Matematica e Informatica, Università di Udine, via delle Scienze 206, 33100 Udine, Italy*

Abstract

Most approaches to time granularity proposed in the literature are based on algebraic and logical formalisms [11]. Here we follow an alternative automaton-based approach, originally outlined in [7], which makes it possible to deal with infinite time granularities in an effective and efficient way. Such an approach provides a neat solution to fundamental algorithmic problems, such as the granularity equivalence and granule conversion problems, which have been often neglected in the literature. In this paper, we focus our attention on two basic optimization problems for the automaton-based representation of time granularities, namely, the problem of computing the smallest representation of a time granularity and that of computing the most tractable representation of it, that is, the one on which crucial algorithms, such as granule conversion algorithms, run fastest.

1 Introduction

The notion of time granularity comes into play in a variety of computer science problems, including time representation and management in database applications, specification and verification of temporal properties of reactive systems, and temporal representation and reasoning in artificial intelligence. To give a few examples, time granularity is involved in temporal database design, temporal database inter-operability, temporal data conversion, data

⋆ A short preliminary version of this paper appeared in [9].

Email addresses: `dallago@cs.unibo.it` (Ugo Dal Lago), `montana@dimi.uniud.it` (Angelo Montanari), `puppis@dimi.uniud.it` (Gabriele Puppis).

mining, reactive system satisfiability and model checking, synthesis, execution, and monitoring of timed workflow systems, temporal constraint representation and reasoning, and temporal abstraction. Different approaches to time granularity have been proposed in the literature [11], based on algebraic [2, 3, 19], logical [5, 13, 14, 17], and string-based [22] formalisms. We focus our attention on the latter.

The string-based formalism eases access to and manipulation of data associated with different granularities, allowing one to solve some basic problems about time granularities, such as the equivalence and conversion problems, which have been neglected by many existing formalisms [16, 18, 19]. String-based algorithms, however, may potentially process every element (symbol) of representations, independently from their redundancy, thus requiring a large amount of computational time. This efficiency problem can be dealt with by the automaton-based approach to time granularity [7], which revises and extends the string-based one. According to such an approach, granularities are viewed as strings generated by a specific class of automata, called Single-String Automata (SSA for short), thus making it possible to (re)use well-known results from automata theory. SSA were originally proposed by Dal Lago and Montanari to model infinite periodical granularities [7]. Furthermore, they showed that regularities of modeled granularities can be naturally expressed by extending SSA with counters (let us call Extended SSA the resulting class of automata). This extension makes the structure of the automata more compact, and it allows one to efficiently deal with those granularities which have a quasi-periodic structure. In [8], we showed that Extended SSA can be exploited to solve the equivalence and the granule conversion problems. The equivalence problem consists in establishing whether two different representations define the same granularity, while the granule conversion problem is the problem of determining a set of granules of a granularity H which are in some specific relation with a set of granules of a coarser/finer granularity G [11]. As a matter a fact, there are as many granule conversion problems as the specific (meaningful) granularity relations are. To solve these problems, we introduced a suitable variant of Extended SSA, called Restricted Labeled Single-String Automata (RLA for short), and we demonstrated that these automata are at least as expressive as the string-based formalism, better suited for direct algorithmic manipulation. As an example, we showed that, in many relevant cases (i.e., those in which there are no gaps within and between granules), the granule conversion problem can be solved in polynomial time with respect to the size of the involved RLA.

The algorithmic nature of automaton-based representations of time granularity suggests an alternative point of view on their role: RLA, as well as SSA and Extended SSA, can be used not only as a formalism for the direct specification of time granularities, but also (and mainly) as a low-level formalism into which high-level time granularity specifications can be mapped. We fully explored such a possibility in the case of Calendar Algebra [2, 19]. The Calendar Algebra is a high-level formalism for modeling time granularities, which subsumes a number of previous proposals including the formalism of Collection Expressions proposed by Leban et al. [16] and the formalism of Slice Expressions developed by Niezette and Stevenne [18]. In [8], we defined a suitable set of algorithms mapping expressions of Calendar Algebra into equivalent RLA-based representations. In view of this operational flavor of RLA (resp. SSA, Extended SSA), the problem of reducing as much as possible the complexity of basic algorithms operating on automaton-based representations of time granularities becomes even more crucial.

In this paper, we focus our attention on optimization problems for RLA. There exist at least two possible notions of RLA-optimization. According to the first one, optimizing means computing the smallest representation of a given time granularity; according to the second one, optimizing means computing the most tractable representation of a given granularity, that is, the one on which crucial algorithms run fastest. We call an automaton-based representation of the first (resp. second) type a *size-optimal* (resp. *complexity-optimal*) representation. The two optimization criteria are not equivalent, since the smallest representation is not necessarily the most tractable one, and vice versa. Furthermore, both problems yield non-unique solutions. In the following, we tackle them by taking advantage of dynamic programming techniques: we first state some closure properties of RLA with respect to concatenation and repetition of words, and then we show how to compute size- and complexity-optimal automata from smaller (optimal) ones in a bottom-up fashion. The resulting algorithms run in polynomial time with respect to the size of the string-based description of the involved granularity.

The rest of the paper is organized as follows. In Section 2, we formalize the notion of time granularity and we briefly describe the main features of Wijsen's string-based formalism, which represents regular granularities by means of (encodings of) ultimately periodic words. In Section 3 we give some preliminary definitions and results about repeating patterns of strings. In particular, we provide an efficient algorithm to compute the minimum periods of all the substrings of a given word. In Section 4 we introduce the automaton-based approach to time granularity. We formally define RLA and we state some basic properties of them. In Section 5 we describe some basic algorithms that can be used to efficiently solve granule conversion problems for RLA-based representations of time granularities. In Section 6 we introduce the size-optimization and complexity-optimization problems and we point out important aspects about their solutions. In Sections 7 and 8 we provide polynomial-time solutions to the complexity-optimization and size-optimization problems, respectively. As a matter of fact, the size-optimization problem turns out to be more difficult than the complexity-optimization one. For this reason, we first deal with the latter problem and we then adapt the achieved results to the case of size-optimal automata, devising an algorithm that computes size-optimal automata for a restricted class of automata. In Section 9 we briefly summarize the outcomes of the work, and we outline future research directions, with a special emphasis on possible improvements on the proposed algorithms.

2 The string-based model of time granularity

Temporal information is often associated with different temporal domains at different granularities. As an example, in many information systems different time granularities can be used to specify the validity intervals of different facts [2] and thus their database component need the ability of properly relating temporal elements belonging to different time granularities. Such an ability rests on a suitable formalization of a notion of granularity. In this section, we give a formal definition of time granularity, which captures a large class of temporal structures; then, we specialize it in order to allow a finite representation and an efficient manipulation of a meaningful subclass of temporal structures. We assume the temporal domain to be isomorphic to the set $\mathbb{N}^+$ of positive natural numbers (as a matter of fact, most applications view time as a discrete left-bounded linear structure).

Definition 1 *A* time granularity *is a partition G of a subset T of $\mathbb{N}^+$ such that, for every pair of sets g, g' (called* granules*) in G, either $\forall\, t \in g\ \forall\, t' \in g'$ $(t < t')$ or $\forall\, t \in g\ \forall\, t' \in g'$ $(t' < t)$.*

Definition 1 captures both time granularities that cover the entire temporal domain, such as `Day`, `Week`, and `Month`, and time granularities with gaps within and between granules (gaps exactly consist of those elements that belong to $\mathbb{N}^+ \setminus T$), like, for instance, `BusinessDay`, `BusinessWeek`, and `BusinessMonth`. Figure 1 depicts some of these granularities.

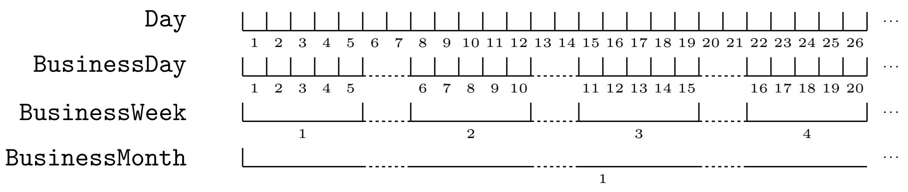

Figure 1. Some examples of time granularities.

The ordering on $\mathbb{N}^+$ induces an ordering on G. Given $g, g' \in G$, if for every $t \in g$, $t' \in g'$, $t < t'$, then we can write $g < g'$. Such an ordering naturally yields a labeling of granules: we say that $x \in \mathbb{N}^+$ is the *label* of a granule $g \in G$, and we write $G(x) = g$, if g is the x-th element of G, according to the induced order $<$. The proposed definition of granularity is equivalent to that provided by Wijsen in [22] and (up to a shift of labels) to the notion of *full labeled* *granularity* (also called *simple granularity*) given by Wang et al. in [1, 2, 19]. In [2] a more general notion of granularity is introduced, which allows labels to be non-contiguous. On the basis of such a notion, granularities are symbolically represented as suitable terms of a Calendar Algebra [2]. However, it is not difficult to show that the automaton-based approach to time granularity [7, 8], including the results reported in the present paper, can be extended to capture this general notion of granularity.

Consider now the set of time granularities of Definition 1. Since the underlying temporal domain is isomorphic to $\mathbb{N}^+$, the set of all partitions that satisfy Definition 1 is uncountable and thus it is not possible to deal with all of them by means of a finitary formalism. However, the problem of dealing with temporal structures for time granularity in an effective way can be tackled by restricting to (infinite) periodic granularities. In [22], Wijsen shows that such granularities can be naturally expressed in terms of ultimately periodic strings over an alphabet 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. In order to guarantee a one-to-one correspondence between infinite strings and granularities, as well as to ease the treatment of the problems of granularity equivalence and granule conversion, Wijsen introduces an aligned form for string-based specifications of granularities. Such a form forces any separator $\wr$ to occur immediately after an occurrence of $\blacksquare$. As pointed out by Dal Lago and Montanari [7], if we encode each occurrence of the substring $\blacksquare\wr$ by a single symbol $\blacktriangleleft$, we align the symbols of the string-based representation and the elements of the temporal domain, that is, we establish a one-to-one correspondence between strings and granularities. In the following, we shall adopt this simplified setting to represent granularities.

We assume the reader to be familiar with basic terminology and notation on finite and infinite strings (if this is not the case, we refer the reader to [21]). In particular, we shall write a generic string u as $u[1]u[2]u[3]\ldots$, where $u[i]$ denotes the i-th symbol of the string, and we shall use the notation $u[i,j]$ to denote the substring $u[i]u[i+1]\ldots u[j]$ of u. Furthermore, given a finite set S, we shall denote by S^∞ the set $S^\omega \cup S^*$, where S^ω (respectively, S^*) stands for the set of all infinite (respectively, finite) strings over S. The operation of concatenation $\cdot$ in S^* can be extended to S^∞ by letting $u \cdot v = u$ whenever $u \in S^\omega$. Similarly, we denote by $|w|$ the length of the string $w \in S^*$ and, for all $w \in S^\omega$, we assume $|w|$ to be equal to ω.

Definition 2 *Given a string $w \in \{\blacksquare, \square, \blacktriangleleft\}^\omega$, we say that w* represents a granularity *G if, for every $t, x \in \mathbb{N}^+$, we have $t \in G(x)$ iff $w[t] \neq \square$ and $w[1, t-1]$ contains exactly $x-1$ occurrences of $\blacktriangleleft$.*

In order to *finitely* model (infinite) periodic time granularities, Wijsen introduces the notion of *granspec*. A granspec is an ordered pair (u, v) of finite strings over the alphabet $\{\blacksquare, \square, \blacktriangleleft\}$ such that v is not the empty string ε. It can be viewed as a finite representation of the infinite string $u \cdot v^\omega$. As an example, the granularity `BusinessWeek` can be expressed, in terms of days, by the ultimately periodic string $\blacksquare\blacksquare\blacksquare\blacksquare\blacktriangleleft\square\square\blacksquare\blacksquare\blacksquare\blacksquare\blacktriangleleft\square\square\ldots$, itself represented by the granspec $(\varepsilon, \blacksquare\blacksquare\blacksquare\blacksquare\blacktriangleleft\square\square)$. In general, a granspec (u, v) represents the same granularity uv^ω represents (see Definition 2).

A major limitation of Wijsen's granspec formalism is that, whenever the granularity to be represented has a long prefix and/or a long period, it produces lengthy representations. In such a case, computations on time granularities represented by granspecs turn out to be rather expensive because their worst-case time complexity is linear in $|u|$ and $|v|$. As an example [7], if (u, v) is a granspec representing months of the Gregorian Calendar in terms of days, we have that $|u| + |v| \geq 10^5$. Indeed, the Gregorial Calendar has a very long periodicity (400 years) and a year includes at least 365 days.

In the following, we shall introduce the automaton-based approach, which yields more succinct representations of time granularities by using counters to encode repetitions in strings. As a preliminary step, we provide a characterization of repeating patterns of strings.

3 On repeating patterns of strings

In this section we establish some fundamental properties of repeating patterns of strings. In particular, we show how to compute the minimum periods of all substrings of a given string in quadratic time. These results will be extensively used in the following sections to cope with the size- and complexity-optimization problems. To start with, we introduce the notions of period, partial period, and border.

Definition 3 *A finite (resp. infinite) string u has* a period *p if, for some $k > 0$ (resp. for $k = \omega$), we have $u = u[1, p]^k$.* The period *of u is its minimum period. An ultimately periodic string is any infinite string of the form $w = uv^\omega$, where u is a finite string and v is a non-empty finite string. The strings u and v are respectively called* a prefix *and* a repeating pattern *of w. If u and v are the shortest strings such that $w = uv^\omega$, then $|u|$ and $|v|$ are said to be* the prefix length *and* the period *of w, respectively. By analogy, we say that p is* a partial period *of a finite string u if u is a prefix of $u[1, p]^\omega$. Finally, a* border *of a finite string u is a string u', different from u, such that u' is both a prefix and a suffix of u.*

The following lemma relates distinct (partial) periods of strings. It is a straightforward generalization of the well-known Fine-Wilf's Lemma [12].

Lemma 1 *For any finite non-empty string u, if p and q are partial periods of u and $|u| \geq p + q$, then $gcd(p, q)$ is a partial period of u.*

Proof. We prove the claim by induction on $p + q$. Assume that $p < q$ and denote by r the value $q - p$. Since p and q are both partial periods of u, for every $i \in [1, |u| - q]$, we have $u[i] = u[i + q] = u[i + q - p] = u[i + r]$. Similarly, for every $i \in [|u| - q + 1, |u| - r]$, $u[i] = u[i - p]$ (since $|u| \geq p + q$) and $u[i - p] = u[i + q - p] = u[i + r]$ hold. Thus r is partial period of u as well. Since $p + r < p + q$ and $gcd(p, q) = gcd(p, r)$ hold, we conclude by induction that $gcd(p, q)$ is a partial period of u. □

We now show that, for every finite string u (or any finite prefix u of an ultimately periodic string), the periods of all substrings of u can be efficiently computed in time $\Theta(|u|^2)$. The algorithm rests on noticeable properties of *partial* periods and borders, and it is closely related to the way Knuth, Morris, and Pratt define the prefix function of a string in the context of string-matching problems [15]. From now on, we shall use the abbreviations $v \dashv u$ and $v \dashv\!\mid u$ to respectively say that "v is a border of u" and "v is the maximum border of u". We begin by proving some distinctive properties of (partial) periods and borders. The following proposition establishes a correspondence between (maximum) borders and (minimum) partial periods.

Proposition 1 *Given a finite string u, $u[1, q]$ is a (maximum) border of u if and only if $|u| - q$ is a (minimum) partial period of u.*

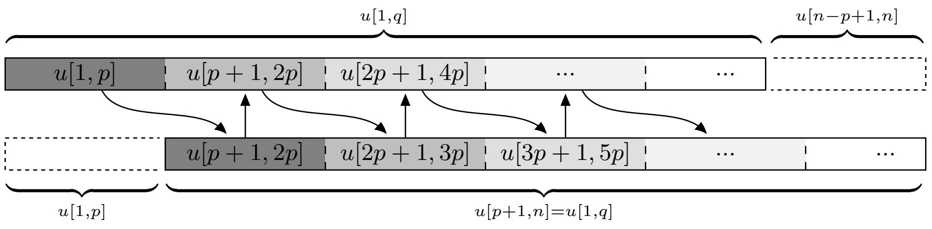

Figure 2. Relationship between partial periods and borders.

Proof. Let u be a finite string of length n and q be a natural number such that $u[1, q] = u[n-q+1, n]$. We define $p = n-q$ and we show, by induction on k, that for every $i \in [1, p]$, $kp+i \leq n$ implies $u[i] = u[kp+i]$ (see Figure 2). For $k = 0$, the property trivially holds. For $k > 0$ and $i \leq n - kp$, by the inductive hypothesis, we have that $u[i] = u[(k - 1)p + i]$ and, since (i) $u[1, q]$ is a border of u, (ii) $kp + i \geq n - q + 1$, and (iii) $(k - 1)p + i \leq (k - 1)p + n - kp = q$, $u[i] = u[(k - 1)p + i] = u[(k - 1)p + i + (n - q)] = u[kp + i]$ holds. Hence, u has a partial period $p = n - q$. For the converse, let u be a finite string of length n and p be a partial period of u. We have that, for every $i \in [1, p]$, $kp + i \leq n$ implies $u[i] = u[kp + i]$. Now, let $q = n - p$. We have that

$u[1,q] = u[p+1,p+q] = u[n-q+1,n]$. Therefore, $u[1,q]$ is a border of u. Finally, the maximum border is the border of maximum length q, hence $p = n - q$ is the minimum partial period of u, and vice versa. □

Let us now focus our attention on the computation of the maximum border of each prefix of a given string u. We preliminarily establish some interesting properties of the relations "border of" and "maximum border of". They will allow us to devise an algorithm that computes the partial periods of all the prefixes of a given finite string u in linear time with respect to $|u|$. Taking advantage of such an algorithm, we shall be able to compute the partial periods of all substrings of u in $\Theta(|u|^2)$ time by simply iterating the computation on each suffix of u. Since u contains exactly $\frac{|u|(|u|+1)}{2}$ substrings, the resulting algorithm turns out to be asymptotically optimal. The following lemma determines the relation between the borders of a given string u and the borders of the extended string ua, by showing that va is a border of ua only if v is a border of u.

Lemma 2 *The relation $\dashv$ respects the extension of strings to the right, that is, $(v \cdot a) \dashv (u \cdot a)$ holds if and only if both $v \dashv u$ and $u[|v|+1] = a$ hold (see Figure 3).*

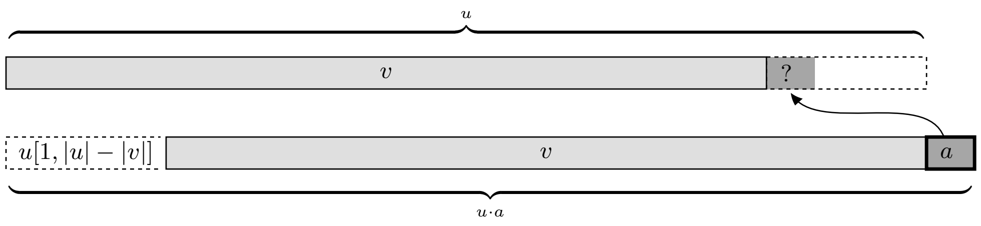

Figure 3. Right extensions of borders.

From Lemma 2, we can easily devise a dynamic-programming-oriented algorithm that computes *all* borders of all prefixes of a given string u: for each border $u[1,q]$ of some prefix $u[1,j]$, check whether $u[q+1] = u[j+1]$ (this suffices to establish whether $u[1,q+1]$ is a border of $u[1,j+1]$). From the lemma it follows that, given the borders of $u[1,j]$, one can easily compute the maximum border of $u[1,j+1]$. In fact, it is not necessary to store *all* borders of all prefixes in order to compute the *maximum* borders of all prefixes, as the following lemma shows (cf. Figure 4, where we depict the transitive reduction of a simple instance of the relation "border of").

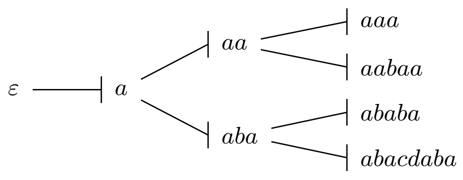

Figure 4. Left linearity of the relation "border of".

Lemma 3 *The relation $\dashv$ is linear to the left, that is, whenever both $v \dashv u$ and $w \dashv u$ hold, then we have $v = w$ or $v \dashv w$ or $w \dashv v$.*

From Lemma 3 it follows that, whenever v is a border of $u[1, j]$, then v is either the maximum border of $u[1, j]$ or a border of the maximum border of $u[1, j]$. This property can be exploited to prove the following corollary (it basically coincides with the Prefix-Function Iteration Lemma in [6]).

Corollary 1 *Let u be a finite string and let $u_1, \ldots, u_n$ be the (unique) sequence of finite strings such that $\varepsilon = u_1 \dashv\vdash \ldots \dashv\vdash u_n \dashv\vdash u$. If $v \dashv u$, then there is $1 \leq k \leq n$ such that $v = u_k$.*

The upshot of Lemma 2 and Corollary 1 is that, in order to compute the maximum border of $u[1, j+1]$, it is sufficient to recursively determine the maximum border of each proper prefix of $u[1, j+1]$ and then descend the chain of relationships $\dashv\vdash$, searching for the longest (i.e., the first) border whose extension matches with $u[j+1]$. As an example, consider the sequence $\varepsilon \dashv \dashv a \dashv\vdash aba \dashv\vdash abacdaba$. The maximum border of the string $abacdaba \cdot b$ is $a \cdot b$, which is precisely the string obtained by extending with b the rightmost string v in the sequence ε, a, aba such that $u[|v| + 1] = b$ (if any). The above argument is formally stated by the next theorem. Subsequently, we shall show that, even if some steps of the computation of a maximum border may take linear time, the total time needed to compute the maximum borders of all prefixes of u is still linear in $|u|$.

Theorem 1 *Let u be a finite string and let $u_1, \ldots, u_n$ be the (unique) sequence of finite strings such that $\varepsilon = u_1 \dashv\vdash \ldots \dashv\vdash u_n \dashv\vdash u$. For any given v, the following two conditions are equivalent:*
1. *$(v \cdot a) \dashv\vdash (u \cdot a)$,*
2. *there is a $1 \leq k \leq n$ such that $u_k = v$, $u[|u_k| + 1] = a$, and $u[|u_h| + 1] \neq a$ for all $h > k$.*

Let us provide now an algorithm that computes the minimum partial period of each prefix $u[1, j]$ of a given finite string u. By denoting with $p(j)$ (resp. $q(j)$) the minimum partial period (resp. the length of the maximum border) of $u[1, j]$, the recurrence equations

$$q(1) = 0, \tag{1}$$
$$q(j+1) = max\Big\{0, r+1 \,:\, u[r+1] = u[j+1] \;\wedge\; \exists\, i > 0 \; (r = q^i(j))\Big\}, \tag{2}$$

follow directly from Theorem 1.

The algorithm *PartialPeriodsOfAllPrefixes* uses the above equations to compute $q(j)$ and $p(j)$, for every $1 \leq j \leq |u|$ (as a matter of fact, up to line 13 it is the algorithm *ComputePrefixFunction* in [6]).

PartialPeriodsOfAllPrefixes(u)

```
1: $n \leftarrow |u|$
2: $q(1) \leftarrow 0$
3: **for all** $j \in [2, n]$ **do**
4:    $r \leftarrow q(j-1)$
5:    **while** $u[r+1] \neq u[j]$ **and** $r > 0$ **do**
6:       $r \leftarrow q(r)$
7:    **end while**
8:    **if** $u[r+1] = u[j]$ **then**
9:       $q(j) \leftarrow r+1$
10:   **else**
11:      $q(j) \leftarrow 0$
12:   **end if**
13: **end for**
14: **for all** $j \in [1, n]$ **do**
15:    $p(j) \leftarrow j - q(j)$
16: **end for**
17: **return** $(p(j))_{j \in [1,n]}$
```

It remains to show that the execution of *PartialPeriodsOfAllPrefixes*(u) takes time linear in $n = |u|$. This can be done by using amortized analysis as in [6]. First, note that $r = q(j-1)$ just before entering the "while" loop at lines 5–7. Furthermore, at lines 8–12, either $r+1$ or 0 is assigned to $q(j)$. Since the variable r decreases at least by 1 at each iteration of the inner loop, for each j the number of iterations is bounded by $q(j-1) - (q(j) - 1)$. Hence, the computation of the length $q(j)$ of the maximum border of $u[1, j]$, with $j \in [2, n]$, takes time proportional to $q(j-1) - (q(j) - 1)$. Therefore, the total time required to execute *PartialPeriodsOfAllPrefixes*(u) is proportional to $\sum_{j \in [2,n]} (q(j-1) - (q(j) - 1)) = \Theta(n)$. Since there is a linear lower bound to the complexity of *PartialPeriodsOfAllPrefixes*(u), the proposed algorithm is asymptotically optimal.

Finally, we provide an asymptotically optimal algorithm for computing the periods of all the substrings of u. The algorithm rests on the following proposition, which connects periods to partial periods.

Proposition 2 *For every finite non-empty string u and for every positive natural number $p < |u|$, p is the minimum period of u if and only if p divides $|u|$ and it is the minimum partial period of u.*

Proof. The right-to-left implication is trivial. Conversely, if p is a period of u, then p is a partial period of u as well. Suppose that p $(< |u|)$ is the minimum period of u. We have that p is a partial period of u. Now, if u had a partial period $q < p$, then $p + q < 2p \leq |u|$ and then, by Lemma 1, $gcd(p, q)$ would be another partial period of u. Since p divides $|u|$ and $gcd(p,q)$ divides p, $gcd(p,q)$ would be a period of $|u|$, and hence p would not be the minimum period of u. This is a contradiction and thus p must be the minimum partial period of u. $\Box$

The following algorithm computes the periods of all substrings of a string u (we use $p(i,j)$ and $P(i,j)$ to respectively denote the minimum partial period and the minimum period of a substring $u[i,j]$).

PeriodsOfAllSubstrings(u)

```
1: $n \leftarrow |u|$
2: for all $i \in [1,n]$ do
3:     $(p(i,j))_{j\in[i,n]} \leftarrow \mathit{PartialPeriodsOfAllPrefixes}(u[i,n])$
4: end for
5: for all $i \in [1,n]$ do
6:     for all $j \in [i,n]$ do
7:         if $p(i,j)$ divides $(j-i+1)$ then
8:             $P(i,j) \leftarrow p(i,j)$
9:         else
10:            $P(i,j) \leftarrow j-i+1$
11:        end if
12:    end for
13: end for
14: return $(P(i,j))_{i\in[1,n],j\in[i,n]}$
```

4 From strings to automata

The idea of viewing granularities as ultimately periodic strings (words) naturally connects time granularity to the fields of formal languages and automata, because any ω-regular language is uniquely determined by its ultimately periodic words [4]. An automaton-based approach to time granularity, that generalizes the string-based one in several respects, was originally proposed in [7], and systematically explored in [8]. It allows one to take advantage of some well-known results coming from automata theory, such as, for instance, closure properties of automata with respect to Boolean operations and concatenation. The basic idea underlying the automaton-based approach to time granularity is simple: we take an automaton M recognizing a *single* word $u \in \{\blacksquare, \square, \blacktriangleleft\}^\omega$ and we say that M represents granularity G if and only if u represents G. In the following, we introduce Restricted Labeled Single-String Automata (RLA for short), which, unlike finite automata and Büchi automata, only accept single words. As a matter of fact, RLA can also be viewed as a variant of

SSA [7], where counters over discrete domains are exploited to obtain succinct representations of time granularities.

Before formalizing the notion of RLA, we give an intuitive description of their structure and behavior. In order to simplify the notation and the formalization of useful properties, RLA label states instead of transitions. The set of control states is partitioned into two groups, respectively denoted by S_Σ and S_ε. S_Σ is the set of states where the labeling function is defined, while S_ε is the set of states where it is not defined. Furthermore, to succinctly represent repetitions, we introduce two kinds of transitions, respectively called *primary* and *secondary* transitions. We have that at most one primary transition can be defined in any given state and at most one secondary transition can be defined in any given non-labeled state (no secondary transitions can be associated with labeled states). At any point of the computation, at most one (primary or secondary) transition is taken according to an appropriate rule depending on the state at which the automaton lies and the value of a *counter*. Moreover, a primary transition can be taken in a non-labeled state $s \in S_\varepsilon$ only once the secondary transition associated with s has been consecutively taken $C_0(s)$ times, where $C_0(s)$ is the initial valuation for the counter associated with s.

Figure 5 depicts an RLA recognizing the word $(\blacktriangleleft\square^6)^\omega$, which represents Mondays in terms of days. States in S_Σ are represented by Σ-labeled circles, while states in S_ε are represented by triangles. Primary and secondary transitions are represented by continuous and dashed arrows, respectively. The initial state is identified by a little triangular tip. The (initial values of) counters are associated with states in S_ε (for the sake of readability, we depict them as labels of the secondary transitions exiting states in S_ε). This simple example gives an intuitive account of how RLA allow one to compactly encode repeating patterns in granularities by means of counters and transitions.

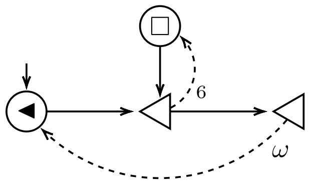

Figure 5. An RLA that represents Mondays in terms of days.

In the sequel, we deal with counters ranging over the discrete domain $\mathbb{N} \cup \{\omega\}$. Counters can be either set to their initial value or decremented (remember that $\omega - 1 = \omega$). We now proceed with the formal definitions.

Definition 4 *A* Restricted Labeled *(*Single-String*)* Automaton *(*RLA *for short) is an 8-tuple $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, C_0)$, where*

- *S_Σ and S_ε are disjoint finite sets of* control states *(in the following, we shall denote $S_\Sigma \cup S_\varepsilon$ by S);*

- Σ *is a finite alphabet;*
- $\Omega : S_\Sigma \rightarrow \Sigma$ *is the (total)* labeling function*;*
- $\delta : S \rightharpoonup S$ *is the (partial)* primary transition function *whose transitive closure δ^+ is irreflexive (namely, it never happens that $(s, s) \in \delta^+$);*
- $\gamma : S_\varepsilon \rightarrow S$ *is the (total)* secondary transition function *such that for every $s \in S_\varepsilon$, $(\gamma(s), s) \in \delta^+$;*
- $s_0 \in S$ *is the* initial state*;*
- $C_0 : S_\varepsilon \rightarrow \mathbb{N}^+ \cup \{\omega\}$ *is the* initial valuation.

The restrictions on the transition functions can be motivated as follows. Constraining the transitive closure of the primary transition function to be irreflexive guarantees that any cycle involves at least one secondary transition. Moreover, the source state of every secondary transition can always be reached from the target state via a sequence of primary transitions.

The definition of RLA run is based on the notion of configuration. Let us denote by $\mathcal{V}$ the set of all the valuations of the form $C : S_\varepsilon \rightarrow (\mathbb{N} \cup \{\omega\})$ for the counters of an RLA $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, C_0)$. A *configuration* is a state-valuation pair (s, C), with $s \in S$ and $C \in \mathcal{V}$. The transitions of M are taken according to a (partial) function $\Delta_M : S \times \mathcal{V} \rightharpoonup S \times \mathcal{V}$ satisfying:

i) whenever $s \in S_\Sigma$ and $\delta(s)$ is defined, $\Delta_M((s, C)) = (\delta(s), C)$ (namely, whenever the automaton lies in a labeled state and there exists an exiting primary transition, then it takes the primary transition, which does not change the valuation);

ii) whenever $s \in S_\varepsilon$, $C(s) = 0$, and $\delta(s)$ is defined, $\Delta_M((s, C)) = (\delta(s), D)$, where $D(s) = C_0(s)$ and, for every non-labeled state $r \neq s$, $D(r) = C(r)$ (namely, whenever the automaton lies in a non-labeled state, whose counter has value 0, and there exists an exiting primary transition, then it takes the primary transition and it re-initializes the counter);

iii) whenever $s \in S_\varepsilon$ and $C(s) > 0$, $\Delta_M((s, C)) = (\gamma(s), D)$, where $D(s) = C(s) - 1$ and, for every non-labeled state $r \neq s$, $D(r) = C(r)$ (namely, whenever the automaton lies in a non-labeled state whose counter has a positive value, then it takes the secondary transition and it decrements the counter by 1).

The *run* of an RLA M is defined as a pair $(\mathbf{s}, \mathbf{C}) \in S^\infty \times \mathcal{V}^\infty$ of maximum (possibly infinite) sequences of states and valuations such that

- $\mathbf{s}[1] = s_0$,
- $\mathbf{C}[1] = C_0$,
- for all $0 < i < |\mathbf{s}| (= |\mathbf{C}|)$, $\Delta_M((\mathbf{s}[i], \mathbf{C}[i])) = (\mathbf{s}[i+1], \mathbf{C}[i+1])$.

As can be easily shown, for any RLA M, there is exactly one run of M. Given the RLA M and its run $(\mathbf{s}, \mathbf{C})$, one can extract a (finite or infinite) sequence of labeled states $\mathbf{s}_\Sigma \in S_\Sigma^\infty$ by discarding the valuations and the non-labeled states. We shall say that M recognizes the word u if and only if $u = \Omega(\mathbf{s}_\Sigma)$ (where $\Omega(\mathbf{s}_\Sigma)$ is the obvious shorthand). Notice that Definition 4 allows situations where states and transitions of an RLA form an unconnected (directed) graph.

We can overcome these clumsy situations by discarding useless states and transitions of RLA. It is easy to prove that RLA recognize either finite or ultimately periodic words.

We conclude the section by providing a formal characterization of the words recognized by RLA. It is based on the notions of δ-*degree* and γ-*degree* of states. The δ-*degree* of a state $s \in S$ is the unique natural number n such that $\delta^n(s)$ is defined, but $\delta^{n+1}(s)$ is not (such an n exists since δ^+ is irreflexive). For each non-labeled state $s \in S_\varepsilon$, the γ-*degree* of s is the least $n \in \mathbb{N}$ such that $(\gamma(s), s) \in \delta^n$ (this is well-defined as well, given the constraints on the secondary transition function). We then define a binary relation Γ_M over the set S_ε as follows: $(s, r) \in \Gamma_M$ if and only if $s = \delta^i(\gamma(r))$, with i less than the γ-degree of r. Notice that the reflexive and transitive closure Γ^*_M is antisymmetric, from which it follows that Γ^*_M is a well-founded partial order over the set of non-labeled states. Thus, we can take advantage of induction on such a partial order (we call it γ-*induction*) in both formal definitions and proofs. As an example, if we denote by s_0 the initial state of the RLA of Figure 5, by s_1 its successor, by s_2 the top-most state, and by s_3 the right-most state, we have that

- the δ-degree of s_0 (respectively, s_1, s_2, s_3) is 2 (respectively, 1, 2, 0),
- the γ-degree of s_1 is 1 and the γ-degree of s_3 is 2,
- $\Gamma_M = \{(s_1, s_3)\}$ and Γ^*_M consists the pair in Γ_M plus the pairs (s_1, s_1) and (s_3, s_3).

As already pointed out, the distinctive feature of RLA is the way they encode repeating patterns of words. In order to provide a characterization of the words recognized by RLA in terms of repetitions of smaller substrings, we need some preliminary definitions.

Suppose that $(\mathbf{s}, \mathbf{C}) \in S^n \times \mathcal{V}^n$ is a finite sequence of states and valuations satisfying $\Delta_M((\mathbf{s}[i], \mathbf{C}[i])) = (\mathbf{s}[i+1], \mathbf{C}[i+1])$ for every $i \in \{1, \ldots, n-1\}$. Then we shall write $(\mathbf{s}[1], \mathbf{C}[1]) \to^w (\mathbf{s}[n], \mathbf{C}[n])$, where $w = \Omega(\mathbf{s}_\Sigma)$. Analogously, if $(\mathbf{s}, \mathbf{C}) \in S^\omega \times \mathcal{V}^\omega$ is an infinite sequence of states and valuations satisfying $\Delta_M((\mathbf{s}[i], \mathbf{C}[i])) = (\mathbf{s}[i+1], \mathbf{C}[i+1])$ for every $i \geq 1$, then we shall write $(\mathbf{s}[1], \mathbf{C}[1]) \to^w$, where $w = \Omega(\mathbf{s}_\Sigma)$. In the following, we denote by σ^M_s the sequence of symbols inductively defined as follows [1]:

- $\Omega(s)$, if $s \in S_\Sigma$,
- $(\sigma^M_{\gamma(s)} \cdot \sigma^M_{\delta(\gamma(s))} \cdot \ldots \cdot \sigma^M_{\delta^{m-1}(\gamma(s))})^{C_0(s)}$, if $s \in S_\varepsilon$ and m is the γ-degree of s.

Hereafter, for any $s \in S_\varepsilon$, we denote by ρ^M_s the word $\sigma^M_{\gamma(s)} \cdot \sigma^M_{\delta(\gamma(s))} \cdot \ldots \cdot \sigma^M_{\delta^{m-1}(\gamma(s))}$.

Lemma 4 *Let $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, C_0)$ be an RLA, s a non-labeled state, and C a valuation such that $C(r) = C_0(r)$ whenever $(r, s) \in \Gamma^+_M$. Then*

[1] Note that the well-definedness of σ^M_s directly follows from the principle of γ-induction.

exactly one of the following conditions holds:
1. $C(s) = 0$,
2. $(s, C) \rightarrow^{w} (s, D)$*, where* $w = \rho^M_s \in \Sigma^*$*,* $D(s) = C(s) - 1$*, and* $D(r) = C(r)$ *for every non-labeled state* $r \neq s$*; moreover, if* $w = u \cdot v$*, with* $u, v \neq \varepsilon$*, and* $(s, C) \rightarrow^{u} (t, E) \rightarrow^{v} (s, D)$*, with* t *being a non-labeled state, then we have* $(t, s) \in \Gamma^+_M$*,*
3. $(s, C) \rightarrow^{w}$*, where* $w = \rho^M_s \in \Sigma^\omega$*; moreover, if* $(s, C) \rightarrow^{u} (t, D)$ *holds, with* t *being a non-labeled state, then we have* $(t, s) \in \Gamma^+_M$*.*

Proof. We prove the lemma by γ-induction on s. Let m be the γ-degree of s, let $r_i = \delta^i(\gamma(s))$, where i ranges over $\{0, \ldots, m\}$, and let $C(s) > 0$ (otherwise Condition *1.* trivially holds). Observe that, by definition of γ-degree, r_m must be s. If s is a minimal non-labeled state with respect to the ordering given by Γ^*_M, then we know that every state r_i, with $0 \leq i < m$, is a labeled state and hence Condition *2.* follows almost trivially. Otherwise, if s is not a minimal element with respect to Γ^*_M, by the inductive hypothesis, we can distinguish between two cases: either
i) every non-labeled state r_i, with $0 \leq i < m$, satisfies Condition *1.* or Condition *2.*,
ii) or there is a non-labeled state r_i, with $0 \leq i < m$, satisfying Condition *3.*.
Let first consider case i). We further distinguish between two sub-cases.

- If, for every $i \in \{0, \ldots, m-1\}$, $C(r_i) \neq \omega$, then we let D be the valuation such that $D(s) = C(s) - 1$ and $D(r) = C(r)$ for all $r \neq s$. We verify that

$$(s, C) \rightarrow^{\varepsilon} (r_0, D) \rightarrow^{w_0} (r_1, D) \rightarrow^{w_1} \ldots \rightarrow^{w_{m-2}} (r_{m-1}, D) \rightarrow^{w_{m-1}} (s, D)$$

  holds, where $w_i = \sigma^M_{r_i}$. Indeed, $r_i \in S_\Sigma$ implies $(r_i, D) \rightarrow^{\Omega(r_i)} (r_{i+1}, D)$, by definition. Otherwise, if $r_i \in S_\varepsilon$, by the inductive hypothesis, we have

$$(r_i, D) \rightarrow^{v_i} (r_i, E_1) \rightarrow^{v_i} (r_i, E_2) \rightarrow^{v_i} \ldots \rightarrow^{v_i} (r_i, E_{n_i}) \rightarrow^{\varepsilon} (r_{i+1}, D)$$

  where $v_i = \rho^M_{r_i}$, $n_i = D(r_i)$, and, for all $j \in \{1, \ldots, n_i\}$, $E_j(r_i) = D(r_i) - j$ and $E_j(r) = D(r)$, for all $r \neq r_j$. Now, suppose that $(s, C) \rightarrow^{u} (t, E) \rightarrow^{v} (s, D)$ holds, with $w = u \cdot v$, $u, v \neq \varepsilon$, and t being a non-labeled state. Clearly, there is $j \in \{0, \ldots, m-1\}$ satisfying either $t = r_j$ or, by the inductive hypothesis, $(r_j, t) \in \Gamma^+_M$. In both cases, $(t, s) \in \Gamma^+_M$ follows and s satisfies Condition *2.*.
- If there is an index $k \in \{0, \ldots, m-1\}$ such that $C(r_k) = \omega$, then we can assume, without loss of generality, that k is the least of such indices. Now, since every non-labeled state r_i, with $0 \leq i < k$, satisfies $C(r_i) \neq \omega$ and either Condition *1.* or Condition *2.*, we can exploit an argument similar to that for the preceding sub-case to show that

$$(s, C) \rightarrow^{\varepsilon} (r_0, D) \rightarrow^{w_0} (r_1, D) \rightarrow^{w_1} \ldots \rightarrow^{w_{k-1}} (r_k, D)$$

  where $w_i = \sigma^M_{r_i} \in \Sigma^+$, $D(s) = C(s) - 1$, and $D(r) = C(r)$ for all $r \neq s$. Again, by the inductive hypothesis, since $D(r_k) = C(r_k) = \omega$, we know that $(r_k, D) \rightarrow^{v_k} (r_k, D)$, where $v_k = \rho^M_{r_k}$. Thus, by letting $w_k = \sigma^M_{r_k} \in \Sigma^\omega$, $(r_k, D) \rightarrow^{w_k}$ follows and hence $(s, C) \rightarrow^w$, for $w = w_0 \cdot w_1 \cdot \ldots \cdot w_k = \sigma^M_s$. Now, let $(s, C) \rightarrow^u (t, D)$. Clearly, there is $j \in \{0, \ldots, k\}$ satisfying either $t = r_j$ or $(t, r_j) \in \Gamma^+_M$ and in both cases $(t, s) \in \Gamma^+_M$ follows and s satisfies Condition *3.*.

Let now consider case ii), namely, let i be the least index from $\{0, \ldots, m-1\}$ such that r_i is a non-labeled state satisfying Condition *3.*. Clearly, for every $j \in \{0, \ldots, i-1\}$, r_j satisfies either Condition *1.* or Condition *2.*. Thus, we can proceed, exactly as in the previous case, by distinguishing between two sub-cases depending on whether there is $k \in \{0, \ldots, i-1\}$ such that $C(r_k) = \omega$ or not. It is easy to verify that in both sub-cases s satisfies Condition *3.*. □

Now, we can state and prove the following proposition.

Proposition 3 *The word recognized by an RLA* $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, C_0)$ *is of the form* $\sigma^M_{s_0} \cdot \sigma^M_{\delta(s_0)} \cdot \ldots \cdot \sigma^M_{\delta^n(s_0)}$. *where n is the δ-degree of s_0.*

Proof. This is a direct consequence of Lemma 4. □

As an example, consider the case of the RLA of Figure 5. According to Proposition 3, the recognized word is $u_0^1 \cdot u_1^6 \cdot u_2^\omega$, where $u_0 = ◀$, $u_1 = \square$, and $u_2 = ◀ \cdot \square^6$. The words u_0, u_1, and u_2 are recognized by the RLA M_0, M_1, and M_2 of Figure 6, respectively.

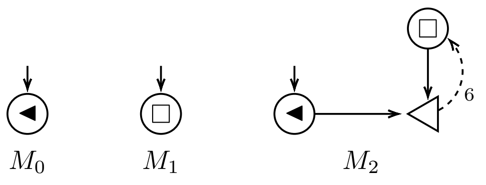

Figure 6. The resulting decomposition of the RLA of Figure 5.

5 Granule conversion problems

RLA can be used to effectively solve the fundamental problems of granularity equivalence and granule conversion. The problem of granularity equivalence is the problem of establishing whether two different representations define the same granularity. Solving this problem gives the possibility of effectively testing the semantic equivalence of two descriptions, making it possible to use smaller, or more tractable, representations in place of bigger, or less tractable, ones. The problem of granule conversion is the problem of relating granules of a given granularity to those of another one. The importance of this problem, that comes into play in a large set of granularity comparison problems [8], has been highlighted by several authors, e.g., Bettini et al. in [2]. Nevertheless, in many approaches it has been only partially worked out in a rather intricate way.

In this paper, we restrict our attention to the latter problem (we extensively deal with the former one in [8], where we show that it is in co-NP). In this section, we present some algorithms to solve the granule conversion problem for RLA-based representations of time granularities; in the next sections, we shall describe in detail some optimization techniques that make it possible to considerably improve the proposed algorithms for granule conversion.

For the sake of simplicity, given an RLA $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, C_0)$, a state $s \in S_\varepsilon$, and a symbol $a \in \Sigma$, we shall denote by $|\rho_s^M|$ and by $|\rho_s^M|_a$ respectively the length of ρ_s^M and the number of occurrences of a in ρ_s^M. Furthermore, we denote by $|M|$ the number of states of M (denoting the number of states of M by $|M|$, instead of by $|S|$, we can avoid to make the components of M explicit whenever it is not really necessary). The whole set of values $|\rho_s^M|$ and $|\rho_s^M|_a$ can be pre-computed in quadratic time with respect to $|M|$. Hereafter, we assume that these values are stored into appropriate data structures for M.

5.1 Searching for symbol occurrences

To explain our solution to the granule conversion problem, we first address a simpler problem, which arises very often when dealing with time granularities as well as with infinite words in general, namely, the problem of finding the n-th occurrence of a given symbol in a word. Such a problem can be easily solved in linear time *with respect to the number of transitions* needed to reach the n-th occurrence of the symbol: it suffices to follow the transitions of the automaton until the n-th occurrence of the symbol is recognized. Nevertheless, we can improve this straightforward solution by taking advantage of the structure of RLA. For instance, if we are searching for an occurrence of a symbol $a \in \Sigma$ in a word u recognized by an RLA $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, C_0)$ and we have that $s_0 \in S_\varepsilon$ and $\rho_{s_0}^M$ contains no occurrences of the symbol a, then we can avoid processing the first $C_0(s_0) \cdot |\rho_{s_0}^M|$ symbols in u. Similarly, if $s_0 \in S_\varepsilon$, $\gamma(s_0) \in S_\varepsilon$, and $\rho_{s_0}^M$ contains at least one occurrence of a, but $\rho_{\gamma(s_0)}^M$ does not, then we can start searching for an occurrence of a in u from the position $C_0(\gamma(s_0)) \cdot |\rho_{\gamma(s_0)}^M|$. By applying the same argument to any state of M, we can define an algorithm, called *SeekAtOccurrence*, which returns the configuration reached by simulating transitions of M from a given configuration (s, C) until the n-th occurrence of a symbol belonging to a distinguished set $A \subseteq \Sigma$ has been read. As a side effect, *SeekAtOccurrence*$(M, s, C, A, n, counter)$ returns in *counter*$[a]$ the number of processed occurrences of every symbol $a \in \Sigma$.

```
SeekAtOccurrence(M, s, C, A, n, counter)
1: let $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, C_0)$
2: for all $a \in \Sigma$ do
```

```
3:      $counter[a] \leftarrow 0$
4:  end for
5:  $i \leftarrow 0$
6:  while $i < n$ do
7:    if $s = \bot$ then
8:      fail
9:    end if
10:   if $s \in S_\Sigma$ then
11:     if $\Omega(s) \in A$ then
12:       $i \leftarrow i + 1$
13:     end if
14:     $counter[\Omega(s)] \leftarrow counter[\Omega(s)] + 1$
15:     $s \leftarrow \delta(s)$
16:   else
17:     $q \leftarrow \sum_{a \in A} |\rho_s^M|_a$
18:     $r \leftarrow s$
19:     if $i + q * C(s) \leq n$ then
20:       if $|\rho_s^M| * C(s) = \omega$ then
21:         fail
22:       else
23:         $l \leftarrow C(s)$
24:         $C(s) \leftarrow C_0(s)$
25:         $s \leftarrow \delta(s)$
26:       end if
27:     else
28:       $l \leftarrow (n - i)\ div\ q$
29:       $C(s) \leftarrow C(s) - l$
30:       $s \leftarrow \gamma(s)$
31:     end if
32:     $i \leftarrow i + l * q$
33:     for all $a \in \Sigma$ do
34:       $counter[a] \leftarrow counter[a] + |\rho_r^M|_a * l$
35:     end for
36:   end if
37: end while
38: return $(s, C)$
```

In spite of the simplicity of the idea, the analysis of the complexity of the above algorithm is rather involved. To make it precise, we introduce a complexity measure $\|M\|$, which depends on the nesting structure of the transition functions of M, defined as follows. For every state s of M and any integer n, let $\mathbf{C}^M_{s,n}$ be defined as follows[2]:

[2] Here we use double induction on s and n, where the ordering for the first, domi-

- 0, if $n < 0$;
- 1, if $n \geq 0$, $s \in S_\Sigma$, and $\delta(s)$ is undefined;
- $1 + \mathbf{C}^M_{\delta(s),n-1}$, if $n \geq 0$, $s \in S_\Sigma$, and $\delta(s)$ is defined;
- $1 + \mathbf{C}^M_{\gamma(s),m-1}$, where m is the γ-degree of s, if $n \geq 0$, $s \in S_\varepsilon$, and $\delta(s)$ is undefined;
- $1 + max\{\mathbf{C}^M_{\delta(s),n-1}, \mathbf{C}^M_{\gamma(s),m-1}\}$, where m is the γ-degree of s, if $n \geq 0$, $s \in S_\varepsilon$, and $\delta(s)$ is defined.

The complexity $\|M\|$ is defined as

$$\|M\| = \mathbf{C}^M_{s_0,n},$$

where s_0 is the initial state of M and n is the δ-degree of s_0.

As an example, the complexity $\|M\|$ of the RLA M of Figure 5 is 6. It is possible to show that the worst-case time complexity of $\mathit{SeekAtOccurrence}(M, s, C, A, n, \mathit{counter})$ is $\Theta(\|M\|)$.

As for the relationships between the complexities of the algorithms operating on automaton-based representations and string-based ones (which are linear in the size of granspecs), it is immediate to see that, for every granspec, there exists an RLA, that represents the same granularity, whose complexity does not exceed the size of the granspec. Moreover, there exist several meaningful cases in which such a complexity turns out to be much lower, thus accounting for the tractability of RLA with respect to granspecs. As an example, it is not difficult to provide an RLA representing the granularity `Month` in terms of days and having complexity 520, which is significantly less than the size of any equivalent granspec (see Section 2).

It turns out that the running time of many other algorithms working on RLA can be expressed in terms of the complexities of the involved automata. In particular, we can write simple algorithms that look for occurrences of symbols in the word recognized by a given RLA, which use *SeekAtOccurrence* as a subroutine. As an example, let u be the word recognized by a given RLA M. The following algorithm computes the position of the last occurrence of the symbol a in u which precedes the first occurrence of the symbol b:

```
OccurrenceLastBeforeFirst($M, a, b$)
1: let $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, C_0)$
2: $(s, C) \leftarrow (s_0, C_0)$
3: $\mathit{SeekAtOccurrence}(M, s, C, \{b\}, 1, \mathit{counter}_1)$
4: $(s, C) \leftarrow (s_0, C_0)$
5: $\mathit{SeekAtOccurrence}(M, s, C, \{a\}, \mathit{counter}_1[a], \mathit{counter}_2)$
6: return $\sum_{c \in \Sigma} \mathit{counter}_2[c]$
```

nant, argument is given by the relation Γ^*_M.

Table 1 reports the heading and a short description of the behavior of other basic algorithms, that will be exploited in the next sections to manipulate RLA representing time granularities (their structure is quite similar to that of the *OccurrenceLastBeforeFirst* algorithm, and thus omitted). It is worth pointing out that, in general, the complexity of such algorithms is *sub-linear* with respect to the number of transitions needed to reach the addressed symbol occurrence.

Table 1
Some basic algorithms running in time $\mathcal{O}(\|M\|)$.

Algorithm	**Behavior**
$\mathit{Occurrence}(M, i, A)$	Returns the position of the i-th occurrence of a symbol in $A \subseteq \Sigma$ in the word u recognized by M.
$\mathit{OccurrenceFirstAfter}(M, i, A, B)$	Returns the position of the first occurrence of a symbol in $A \subseteq \Sigma$ after the i-th occurrence of a symbol in $B \subseteq \Sigma$ in the word u recognized by M.
$\mathit{OccurrenceLastBefore}(M, i, A, B)$	Returns the position of the last occurrence of a symbol in $A \subseteq \Sigma$ before the i-th occurrence of a symbol in $B \subseteq \Sigma$ in the word u recognized by M.
$\mathit{OccurrencesBetween}(M, i, j, A)$	Returns the number of occurrences of symbols in $A \subseteq \Sigma$ in the subword $u[i, j-1]$ of the word u recognized by M.

5.2 Solving the conversion problem

In its most common formulation, the problem of granule conversion is viewed as the problem of determining a set of granules of a granularity H which are in some specific relation with a set of granules of a coarser/finer granularity G (see Section 1). According to such a definition, the granule conversion problem is actually a family of problems, whose different concrete instances are obtained by specifying the relation that must hold between the granules of the source granularity G and the destination granularity H. Here we consider the cases of the relations cover, covered-by, and intersect (the other relations can be dealt with in a similar way). The relation *cover* holds between a set R of granules of G and a set S of granules of H if S is the largest set such that $\bigcup_{g \in R} g \supseteq \bigcup_{g \in S} g$. The relation *covered-by* is the converse of the relation cover and it holds between a set R of granules of G and a set S of granules of H if S is the smallest set such that $\bigcup_{g \in R} g \subseteq \bigcup_{g \in S} g$. Note that the relation cover defines a total function mapping a set of granules of G into a possibly empty set of granules of H, while the relation covered-by only defines a partial function, since it may happen that some sets of granules of G are not covered by any set of granules of H. Finally, the relation *intersect* holds between a set R of granules of G and a set S of granules of H if $S = \{h \in H \,:\, \exists\, g \in R \;\; (g \cap h \neq \emptyset)\}$ Notice that, if $R \subseteq G$, $S, T \subseteq H$, (R, S) is an instance of the relation covered-by and (R, T) is an instance of the relation intersect, then $S = T$. In some cases, however, given $R \subseteq G$, there may not exist any set S such that (R, S) is an instance of the relation covered-by, while there always exists a set T such that (R, T) is an instance of the relation intersect.

For any possible instance of the granule conversion problem, we distinguish two distinct variants of increasing complexity. In the simplest case (case 1), the granularities involved have no gaps within or between granules; in the second case (case 2), gaps may occur both within and between the granules of the granularities involved. Efficient automaton-based solutions for most common relations can be obtained in the first case, provided that we work with intervals. In such a case, the set of granules of the destination granularity H that correspond to an interval of granules of G, is an interval that can be dealt with as a whole. On the contrary, in case 2 we cannot guarantee that the resulting set of granules is an interval and thus we must consider one granule at a time.

The solutions to the conversion problems take advantage of some auxiliary functions, called downward conversions and upward conversions, which are quite similar to the conversion operators introduced by Snodgrass et al. in [10, 20]. Downward conversion receives a granularity G and a set (an interval, if we restrict to case 1) R of granules of G as input and it returns as output the set (respectively the interval) $T = \bigcup_{g \in R} g$ of time points. Upward conversion is the dual operation and it comes in three different variants:

- the *cover upward conversion* of a granularity G and a set/interval T of time points is the smallest set/interval S of granules of G such that $T \subseteq \bigcup_{g \in S} g$,
- the *covered-by upward conversion* of a granularity G and a set/interval T of time points is the largest set/interval S of granules of G such that $T \supseteq \bigcup_{g \in S} g$,
- the *intersect upward conversion* of a granularity G and a set/interval T of time points is the set/interval S of all granules g of G such that $g \cap T \neq \emptyset$.

We now provide the algorithms that compute downward and upward conversions for case 1 (no gaps allowed) and case 2 (gaps allowed). In case 1, since the input set R is assumed to be an interval of granules, we use $min(R)$ and $max(R)$ to denote the least and the greatest element of R ($max(R) = \omega$ if R is not bounded). As previously mentioned, since intervals can be dealt with as a whole, downward and upward conversions in case 1 can be implemented using only a finite number of calls to *SeekAtOccurrence* and thus their worst-case running time is linear with respect to $\|M\|$, where M is the RLA representing the involved granularity.

Here are the algorithms for downward conversion in case 1 and 2; those for upward conversion will be given later.

DownwardConversion1(M, R)

```
1: $i \leftarrow OccurrenceFirstAfter(M, min(R) - 1, \{\blacksquare, \blacktriangleleft\}, \{\blacktriangleleft\})$
2: $j \leftarrow Occurrence(M, max(R), \{\blacktriangleleft\})$
3: return $\{k : i \leq k \leq j\}$
```

The downward conversion for case 2 is performed by *DownwardConversion2*, which processes one granule of the input set R at a time. In particular, for each such granule, *DownwardConversion2* first computes the smallest interval $[i, j]$ of time points covering that granule and then collects the time points $k \in [i, j]$ that belongs to the granule. The union T of all such time points gives the output of the algorithm.

It is worth noticing that termination is not guaranteed if the input set R is finite, but its last granule is infinite. In such a case, the smallest interval covering the set R of granules is infinite (procedure $Occurrence(M, x\{\blacktriangleleft\})$ at line 4 assigns ω to j) and the algorithm cycles at lines 5–9. In order to guarantee termination, one can exploit the fact that M recognizes an ultimately periodic word $w = u \cdot v^\omega$ and then reason on the prefix and the repeating pattern of w. This allows one to detect a non-terminating loop and, accordingly, to return the (possibly infinite) set T of converted time points, which can be represented as an arithmetic progression of the form $A \cup \{i + jq : i \in B, j \in \mathbb{N}\}$, where A and B are finite disjoint sets of indices and q is a positive natural number. A similar argument can be applied in the case R is an infinite set of granules, represented as an arithmetic progression. From now on, we shall not consider cases where infinite sets of granules or infinite sets of time points are involved (however, by reasoning on prefixes and repeating patterns of RLA-recognizable words, it is always possible to manage such cases in an effective way).

DownwardConversion2(M, R)

```
 1: $T \leftarrow \emptyset$
 2: for all $x \in R$ do
 3:    $i \leftarrow OccurrenceFirstAfter(M, x - 1, \{\blacksquare, \blacktriangleleft\}, \{\blacktriangleleft\})$
 4:    $j \leftarrow Occurrence(M, x, \{\blacktriangleleft\})$
 5:    for all $k \in [i, j]$ do
 6:       if $OccurrencesBetween(M, k, k + 1, \{\blacksquare, \blacktriangleleft\}) = 1$ then
 7:          $T \leftarrow T \cup \{k\}$
 8:       end if
 9:    end for
10: end for
```

```
11: return $T$
```

In succession, we provide the algorithms for cover, covered-by, and intersect upward conversions, for both case 1 and case 2. Note that all algorithms for case 1 work in worst-case linear time with respect to $\|M\|$, where M is the RLA representing the involved granularity. The algorithms for case 2 are more general but less efficient, since we need to process one element of the input set T at a time.

The correctness of the first algorithm, *CoverUpwardConversion1*, stems from the following observation: if a granularity G is represented by the RLA M and t is a time point, then $\mathit{OccurrencesBetween}(M, 1, t, \{◀\}) + 1$ is the label of the granule of G including t.

CoverUpwardConversion1(M, T)

```
1: $x \leftarrow \mathit{OccurrencesBetween}(M, 1, \mathit{min}(T), \{◀\}) + 1$
2: $y \leftarrow \mathit{OccurrencesBetween}(M, 1, \mathit{max}(T), \{◀\}) + 1$
3: return $\{z : x \leq z \leq y\}$
```

CoverUpwardConversion2(M, T)

```
1: $S \leftarrow \emptyset$
2: for all $i \in T$ do
3:   if $\mathit{OccurrencesBetween}(M, i, i + 1, \{■, ◀\}) = 1$ then
4:     $S \leftarrow S \cup \{\mathit{OccurrencesBetween}(M, 1, i, \{◀\}) + 1\}$
5:   else
6:     fail
7:   end if
8: end for
9: return $S$
```

Covered-by upward conversions are computed by the following algorithms. Note that, in case 2, the covered-by upward conversion is computed by first collecting all granules of G (i.e. the granularity represented by the RLA M) that intersect the time points in T (lines 2-6), and then discarding those granules which are not entirely covered by T (lines 7-15).

CoveredByUpwardConversion1(M, T)

```
1: if $\mathit{min}(T) = 1$ then
2:   $x \leftarrow 1$
3: else
4:   $x \leftarrow \mathit{OccurrencesBetween}(M, 1, \mathit{min}(T) - 1, \{◀\}) + 2$
5: end if
```

```
6: $y \leftarrow \mathit{OccurrencesBetween}(M, 1, \mathit{max}(T) + 1, \{◀\})$
7: return $\{z : x \leq z \leq y\}$
```

CoveredByUpwardConversion2(M, T)

```
 1: $S \leftarrow \emptyset$
 2: for all $i \in T$ do
 3:   if $\mathit{OccurrencesBetween}(M, i, i + 1, \{■, ◀\}) = 1$ then
 4:     $S \leftarrow S \cup \{\mathit{OccurrencesBetween}(M, 1, i, \{◀\}) + 1\}$
 5:   end if
 6: end for
 7: for all $x \in S$ do
 8:   $i \leftarrow \mathit{OccurrenceFirstAfter}(M, x - 1, \{■, ◀\}, \{◀\})$
 9:   $j \leftarrow \mathit{Occurrence}(M, x, \{◀\})$
10:   for all $k \in [i, j]$ do
11:     if $\mathit{OccurrencesBetween}(M, k, k + 1, \{■, ◀\}) = 1$ and $k \notin T$ then
12:       $S \leftarrow S \setminus \{x\}$
13:     end if
14:   end for
15: end for
16: return $S$
```

Finally, intersect upward conversions are simplified versions of cover upward conversions (here we do not need to check for failure).

IntersectUpwardConversion1(M, T)

```
1: $x \leftarrow \mathit{OccurrencesBetween}(M, 1, \mathit{min}(T), \{◀\}) + 1$
2: $y \leftarrow \mathit{OccurrencesBetween}(M, 1, \mathit{max}(T), \{◀\}) + 1$
3: return $\{z : x \leq x \leq y\}$
```

IntersectUpwardConversion2(M, T)

```
1: $S \leftarrow \emptyset$
2: for all $i \in T$ do
3:   if $\mathit{OccurrencesBetween}(M, i, i + 1, \{■, ◀\}) = 1$ then
4:     $S \leftarrow S \cup \{\mathit{OccurrencesBetween}(M, 1, i, \{◀\}) + 1\}$
5:   end if
6: end for
7: return $S$
```

The above-defined conversion operations are strictly connected to the relations introduced at the beginning of the section: each of them can be computed by performing a downward conversion followed by the corresponding upward conversion. As an example, the relation cover can be computed as follows.

```
Cover(M, N, R)
1: return CoverUpwardConversion(N, DownwardConversion(M, R))
```

Depending on the type of granularities involved, we use different implementations of the conversion algorithms. In particular, if we restrict to granularities without gaps and to intervals (case 1), we can use more efficient implementations, that run in worst-case linear time with respect to $\|M\|$ and $\|N\|$, where M and N are the two RLA representing the involved granularities.

6 Optimality of automaton-based representations

In Section 5 we have outlined some basic algorithms which compute granule conversions in worst-case linear time with respect to the complexities of the involved RLA. It immediately follows that, given an RLA M, it is worth minimizing its complexity $\|M\|$. Furthermore, there exists a widespread recognition of the fact that state minimization is an important problem in classical automata theory as well as in the theory of reactive systems, and thus another goal of practical interest is the minimization of $|M|$. The former problem is called *complexity-optimization problem*, while the latter is called *size-optimization problem*. Even though the size and complexity measures associated with an RLA are clearly related one to the other, they are not equivalent, and the same holds for the corresponding minimization problems. In particular, the size-optimization problem seems to be harder than the complexity-optimization, and only a partial solution to it will be given here. It is also worth remarking that optimal automata are not guaranteed to be unique (up to isomorphism) as it happens, for instance, for deterministic finite automata. As an example, the three automata M, N, and O of Figure 7 recognize the same finite word ■■□■■□■■□. M and N are size-optimal automata ($|M| = |N| = 4$, while $|O| = 6$), and M and O are complexity-optimal automata ($\|M\| = \|O\| = 5$, while $\|N\| = 7$).

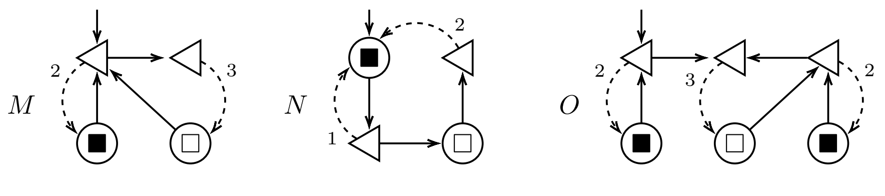

Figure 7. Size-optimal and complexity-optimal automata.

Automata optimization problems can be addressed in many different ways, e.g., by partitioning the state space or by exploiting noticeable relations between automata and expressions encoding recognized words. In the following, we tackle both the complexity-optimization problem and the size-optimization problem by using dynamic programming, namely, by computing an optimal automaton starting from smaller (optimal) ones in a bottom-up fashion. The key point of such a solution is the proof that the optimization problem enjoys an *optimal-substructure property*. In the following, we describe three operations on RLA and we prove closure properties for them; then, we compare the complexity and the size of compound automata with that of their components. In Section 7 and Section 8 we take advantage of these results to provide optimal substructure properties for the two optimization problems for RLA.

6.1 Closure properties

The class of RLA is closed with respect to the operations of concatenation and repetition. Given two RLA M and N, that recognize a finite or infinite word u and a finite word v, respectively, let $\mathit{AppendChar}(a, M)$, where a is a symbol, and $\mathit{AppendRepeat}(N, k, M)$ be, respectively, the *concatenation of a to M*, which recognizes the word $a \cdot u$, and the *concatenation of a k-repetition of N to M*, which recognizes the word $v^k \cdot u$ (with $k \in \mathbb{N}^+ \cup \{\omega\}$). The resulting automata can be computed as follows:

- the automaton $\mathit{AppendChar}(a, M)$ can be obtained from M by (i) adding a new a-labeled state s_0, (ii) linking it to the initial state of M, and (iii) giving it the status of initial state of the resulting automaton (see Figure 8);
- the automaton $\mathit{AppendRepeat}(N, k, M)$ can be obtained from N and M by (i) adding a new non-labeled state s_{loop} (the triangular state of Figure 9), (ii) introducing a secondary transition from s_{loop} to the initial state of N, a primary transition from the final state of N to s_{loop}, and a primary transition from s_{loop} to the initial state of M, and (iii) giving s_{loop} the status of initial state of the resulting automaton (see Figure 9).

We can actually give *AppendChar* and *AppendRepeat* the status of algorithms running in linear time. If the argument M in the above definitions is missing,

s_1
t_1
M
a
s_0
s_1
t_1
$\mathit{AppendChar}(a, M)$

Figure 8. The concatenation of a to M.

we have

- the automaton $\mathit{AppendChar}(a)$, which recognizes a single character a;
- the automaton $\mathit{AppendRepeat}(N, k)$, which recognizes v^k.

Moreover, the size (resp. the complexity) of the resulting automata can be specified in terms of the size (resp. the complexity) of the component automata

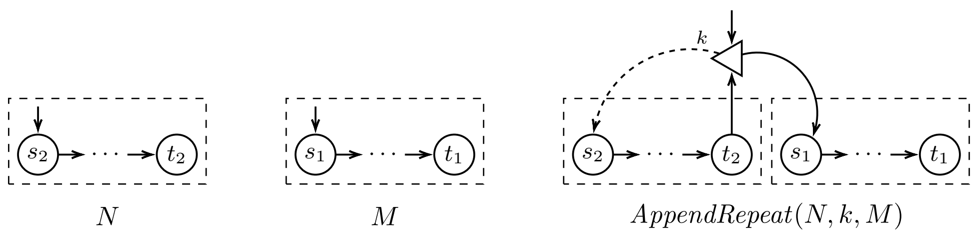

Figure 9. The concatenation of a k-repetition of N to M.

as follows:

- $\mathit{AppendChar}(a, M)$ has size $1 + |M|$ and complexity $1 + \|M\|$;
- $\mathit{AppendRepeat}(N, k, M)$ has size $1+|N|+|M|$ and complexity $1+\mathit{max}\{\|N\|, \|M\|\}$.

Now, let Σ be a finite alphabet and let $\mathcal{C}_\Sigma$ be the class of all RLA obtained from symbols in Σ by applying the operators *AppendChar* and *AppendRepeat*. Clearly, $\mathcal{C}_\Sigma$ is *properly included* in the class of all RLA, that is, there exist some RLA, including size-optimal and complexity-optimal ones (e.g., the automaton M in Figure 7), that cannot be generated via *AppendChar* and *AppendRepeat*. Nevertheless, it turns out that, for every RLA M, $\mathcal{C}_\Sigma$ always contains at least one RLA which is equivalent to M and has the same *complexity*. The above mentioned property can be exploited to prove that a complexity-optimal automaton for a given string can be generated by composing smaller (complexity-optimal) automata using the above mentioned operators. Unfortunately, similar properties do not hold for size-optimal RLA.

Lemma 5 *For every RLA $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, C_0)$, every state $s \in S$, and every $n \in \mathbb{N}$ not exceeding the δ-degree of s, there is an RLA $N_{s,n} \in \mathcal{C}_\Sigma$ such that $N_{s,n}$ recognizes $u_{s,n} = \sigma^M_s \cdot \sigma^M_{\delta(s)} \cdot \ldots \cdot \sigma^M_{\delta^n(s)}$ and $\|N_{s,n}\| \leq \mathbf{C}^M_{s,n}$.*

Proof. The proof is by induction on $\mathbf{C}^M_{s,n} \in \mathbb{N}$. We distinguish some cases depending on s being an element of S_Σ or S_ε and on n being 0 or a positive integer.

- If $s \in S_\Sigma$ and $n = 0$, then we have $\mathbf{C}^M_{s,n} = 1$ and $u_{s,n} = \Omega(s)$. Hence, the thesis easily follows, since $\|\mathit{AppendChar}(\Omega(s))\| = 1$.
- If $s \in S_\Sigma$ and $n > 0$, then we have $\mathbf{C}^M_{s,n} = 1 + \mathbf{C}^M_{\delta(s),n-1}$ and $u_{s,n} = \Omega(s) \cdot u_{\delta(s),n-1}$. Hence, we can apply the inductive hypothesis to $\delta(s)$ and $n-1$, so that the thesis follows, since $\|\mathit{AppendChar}(\Omega(s), N_{\delta(s),n-1})\| = 1 + \|N_{\delta(s),n-1}\|$.
- If $s \in S_\varepsilon$ and $n = 0$, then we have $\mathbf{C}^M_{s,n} = 1+\mathbf{C}^M_{\gamma(s),m-1}$ and $u_{s,n} = u^{C_0(s)}_{\gamma(s),m-1}$, with m being the γ-degree of s, under the assumption that $m > 0$ (the case $m = 0$ can be easily handled). By applying the inductive hypothesis to $\gamma(s)$ and $m-1$, we can obtain an automaton $N_{\gamma(s),m-1}$ so that $N_{s,n} = \mathit{AppendRepeat}(N_{\gamma(s),m-1}, C_0(s))$ is the desired automaton.
- If $s \in S_\varepsilon$ and $n > 0$, then we have $\mathbf{C}^M_{s,n} = 1 + \mathit{max}\{\mathbf{C}^M_{\gamma(s),m-1}, \mathbf{C}^M_{\delta(s),n-1}\}$ and $u_{s,n} = u^{C_0(s)}_{\gamma(s),m-1} \cdot u_{\delta(s),n-1}$, with m being the γ-degree of s, under the

assumption that $m > 0$ (the case $m = 0$ can be easily handled). From the inductive hypothesis, we can obtain two RLA $N_{\gamma(s),m-1}$ and $N_{\delta(s),n-1}$ such that $N_{s,n} = \mathit{AppendRepeat}(N_{\gamma(s),m-1}, C_0(s), N_{\delta(s),n-1})$ is the desired automaton. □

Proposition 4 *For every RLA M, there is an equivalent RLA $N \in \mathcal{C}_\Sigma$ such that $\|N\| \leq \|M\|$.*

Proof. This is trivial in view of Lemma 5. Let s_0 be the initial state of M and n its δ-degree. Consider the RLA $N_{s_0,n}$ obtained from Lemma 5. Clearly, $\|N_{s_0,n}\| \leq \|M\|$ and N recognizes the same string as M. □

As an example, consider the automata M and O of Figure 7. They have the same complexity ($\|M\| = \|O\| = 5$), but $O \in \mathcal{C}_\Sigma$, while $M \notin \mathcal{C}_\Sigma$. It is easy to show that O can be obtained from M by applying the transformations given in Lemma 5.

7 Computing complexity-optimal automata

In this section, we exploit the closure properties of RLA to devise a polynomial-time solution for the complexity-optimization problem. In virtue of Proposition 4, we have that for any (finite or ultimately periodic) word $u \in \Sigma^\infty$, there exists a complexity-optimal automaton M that recognizes u and belongs to $\mathcal{C}_\Sigma$. As a matter of fact, we can prove that, for any word u, there exists one such M that is decomposable into complexity-optimal automata. As a preliminary result, we establish the following technical lemma.

Lemma 6 (Prefix Property) *Given an RLA M recognizing a (finite or infinite) word u and a natural number $1 \leq n \leq |u|$, there is an RLA in $\mathcal{C}_\Sigma$, denoted $\mathit{Prefix}(M, n)$, recognizing the prefix $u[1, n]$ and satisfying $\|\mathit{Prefix}(M, n)\| \leq \|M\|$.*

Proof. By Proposition 4, we have that there exists $N \in \mathcal{C}_\Sigma$ (equivalent to M) such that $\|N\| \leq \|M\|$. We prove the thesis by induction on the structure of $N \in \mathcal{C}_\Sigma$. We only consider the non-trivial cases.

- Suppose $N = \mathit{AppendChar}(a, L)$. We further distinguish the following subcases:
  - i) if $n = 0$, then the required automaton is $\mathit{AppendRepeat}(L, 0)$;
  - ii) if $n = 1$, then the required automaton is $\mathit{AppendChar}(a)$;
  - iii) if $n > 1$, then the required automaton is $\mathit{AppendChar}(a, O)$, where O is the automaton obtained by applying the inductive hypothesis to L and $n - 1$.
- Suppose $N = \mathit{AppendRepeat}(L, k, O)$, where k is a natural number. Let v

(respectively, t) be the word recognized by L (respectively, O). We distinguish the following subcases:

i) if $n \leq |v|$, then the required automaton is simply obtained by applying the inductive hypothesis on L and n;

ii) if $|v| < n \leq k|v|$, let m and l be natural numbers such that $n = m|v| + l$, with $l < |v|$, and let P be the automaton obtained by applying the inductive hypothesis on L and l. Then, the required automaton is $AppendRepeat(L, m, P)$;

iii) if $n > k|v|$, then, by applying the inductive hypothesis to O and $n - k|v|$, we obtain P. The required automaton is $AppendRepeat(L, k, P)$. □

The following two theorems are the basic ingredients of the solution to the complexity-minimization problem. They state optimal substructure properties for finite and ultimately periodic words, respectively.

Theorem 2 *Given a finite word u, at least one of the following conditions holds:*

1. *$|u| = 1$ and $AppendChar(a)$ is complexity optimal for u;*
2. *$|u| > 1$ and $AppendChar(a, M)$ is complexity-optimal for u whenever M is complexity-optimal for $u[2, |u|]$;*
3. *$|u| > 1$ and $AppendRepeat(M, |u|/p)$ is complexity-optimal for u whenever M is complexity-optimal for $u[1, p]$, with p being the period of u;*
4. *$|u| > 1$ and there exists $r < |u|$ such that $AppendRepeat(M, r/p, O)$ is complexity-optimal for u whenever M is complexity-optimal for $u[1, p]$, where p is the period of $u[1, r]$, and O is complexity-optimal for $u[r+1, |u|]$.*

Proof. By Proposition 4, we have that there exists a complexity-optimal automaton $N \in \mathcal{C}_\Sigma$ recognizing u. We prove the thesis by induction on the structure of N. We only consider the non-trivial cases.

- Suppose $N = AppendChar(a, L)$ and let M be a complexity-optimal automaton recognizing $u[2, |u|]$. Then we have:

$$\begin{aligned} \|AppendChar(a, M)\| &= 1 + \|M\| \leq \\ &\leq 1 + \|L\| = \|AppendChar(a, L)\| = \|N\| \end{aligned}$$

  This implies that $AppendChar(a, M)$ is complexity-optimal.

- Suppose $N = AppendRepeat(L, k, P)$, where k is a natural number. Clearly, $k > 0$, because otherwise N would not be complexity-optimal. Let v be the finite word recognized by L, let $r = k|v|$, and let p be the minimum period of $u[1, r]$ (and thus we have that $p \leq |v|$). Moreover, let M be a complexity-optimal automaton for $u[1, p]$ and O be a complexity-optimal automaton for $u[r + 1, |u|]$. By Lemma 6, we have that $\|M\| \leq \|L\|$ and thus

$$\begin{aligned} \|AppendRepeat(M, r/p, O)\| &= 1 + max\{\|M\|, \|O\|\} \leq \\ &\leq 1 + max\{\|L\|, \|P\|\} = \|N\|, \end{aligned}$$

which implies that $\mathit{AppendRepeat}(M, r/p, O)$ is complexity-optimal. $\square$

The case of ultimately periodic (infinite) words is more problematic, because it may happen that a complexity-optimal automaton operates on a non-minimum prefix. Consider, for instance, the word $(abc)^2ab(ce)^\omega$. Its minimum prefix length is 8 and its minimum period is 2. However, the complexity-optimal automata for it recognize the prefix $(abc)^3$ (of length 9) and the repeating pattern ec of length 2.

Theorem 3 *Given an ultimately periodic word u with minimum prefix length l and minimum period q, at least one of the following conditions holds:*

1. *$\mathit{AppendChar}(a, M)$ is complexity-optimal for u whenever M is complexity-optimal for $u[2, \omega]$;*
2. *$\mathit{AppendRepeat}(M, \omega)$ is complexity-optimal for u whenever M is complexity-optimal for $u[1, q]$;*
3. *There exists $r \leq 2l + 2q$ such that $\mathit{AppendRepeat}(M, r/p, O)$ is complexity-optimal for u whenever M is complexity-optimal for $u[1, p]$, where p is the period of $u[1, r]$, and O is complexity-optimal for $u[r + 1, \omega]$.*

Proof. We proceed in the usual way. By Proposition 4, we have that there is a complexity-optimal $N \in \mathcal{C}_\Sigma$ recognizing u and we prove the thesis by induction on the structure of N. We only consider the most complex case, that is, the case in which $N = \mathit{AppendRepeat}(L, k, P)$, where k is a *positive* natural number. We distinguish two cases: $k = 1$ and $k > 1$. Let $k = 1$ and let v be the finite word recognized by L. We define $r = \mathit{min}(|v|, ((|v| - l) \bmod q) + l)$. Note that r is always less than or equal to $|v|$ and it is strictly less than $l + q$. For every complexity-optimal automaton M that recognizes $u[1, r]$ and every complexity-optimal automaton O that recognizes $u[r + 1, \omega](= u[|v| + 1, \omega])$, we have that $\|M\| \leq \|L\|$ (by Lemma 6) and $\|O\| \leq \|P\|$. Hence, $\mathit{AppendRepeat}(M, 1, O)$ is a complexity-optimal automaton that recognizes u. Let $k > 1$. By proceeding as in the proof of Theorem 2, we can replace the automata L and P by equivalent complexity-optimal automata M and O, respectively. Let v be the finite word recognized by M, let $r = k|v|$, and let p be the minimum period of $u[1, r]$ (and thus we have that $p \leq |v|$). To complete the proof, we need to show that $r \leq 2l + 2q$. Suppose, by contradiction, that $r > 2l + 2q$ and consider the substring $t = u[l + 1, r]$ of u, which has partial periods q and $|v|$. We have that $r \geq \mathit{max}(2l + 2q, 2|v|)$ and thus $r \geq l + q + |v|$ which is equivalent to $|t| \geq q + |v|$. By Lemma 1, this means that t has partial period $m = \mathit{gcd}(q, |v|)$. Consider now the substring $u[2l + 1, 2l + 2q]$ of t. It has period q (since $2l + 1$ is greater than l) and partial period m. Since m divides q, m is in fact a period. From the fact that q is the minimum period, we have that $q = m$. It immediately follows that $|v|$ is a multiple of q $(= m)$, and thus v has period q. Moreover, from the minimality of l, it follows that $l = 0$ (if $l > 0$, then $u[l] \neq u[l + q]$, but, from $r > l + q$, we have that $u[l] = u[l + q]$). Hence, the repeating pattern of v of length q is equal to $u[r + 1, r + q]$ and we have

$$\begin{aligned} u &= u[1,r]u[r+1,\omega] = v^k u[r+1,r+q]^\omega = \\ &= u[1,q]^h u[r+1,r+q]^\omega = u[r+1,r+q]^\omega = u[r+1,\omega], \end{aligned}$$

which contradicts the hypothesis that N is complexity-optimal (O recognizes $u[r+1,\omega] = u$). □

According to Theorem 2 and Theorem 3 there exists only a *finite* number of ways of building a complexity-optimal automaton for a word u, given some (optimal) automata for the substrings of u. However, if u is an infinite word, we must show that there is an upper bound on the number of possible applications of case *3.* of Theorem 3.

For every $n \in \mathbb{N}$, we denote by $\mathcal{F}_\Sigma(n)$ the restriction of $\mathcal{C}_\Sigma$ to automata recognizing finite words of length at most n. Moreover, for every n, r, m, we define, by induction on n, the subclass $\mathcal{T}_\Sigma(n,r,m)$ of $\mathcal{C}_\Sigma$, which contains automata recognizing infinite words with period less than or equal to m.

$$\begin{aligned} \mathcal{T}_\Sigma(0,r,m) &= \{\mathit{AppendRepeat}(M,\omega) \ : \ M \in \mathcal{F}_\Sigma(m)\} \\ \mathcal{T}_\Sigma(n+1,r,m) &= \mathcal{T}_\Sigma(n,r,m) \ \cup \ \{\mathit{AppendChar}(a,M) \ : \ M \in \mathcal{T}_\Sigma(n,r,m)\} \\ &\quad \cup \{\mathit{AppendRepeat}(M,k,O) \ : \ kp \leq r, \\ &\qquad M \in \mathcal{F}_\Sigma(p), O \in \mathcal{T}_\Sigma(n,r,m)\}. \end{aligned}$$

Proposition 5 *For every ultimately periodic word u, with minimum prefix length l and minimum period q, there is $N \in \mathcal{T}_\Sigma(l+q, 2l+2q, q)$ that is complexity-optimal for u.*

Proof. We can recursively apply Theorem 3 and end up with a complexity-optimal automaton N for u such that $N = N_n$, where

$$\begin{aligned} & N_0 = \mathit{AppendRepeat}(M_0,\omega), \text{ and} \\ \forall\, 1 \leq i \leq n \quad & (N_i = \mathit{AppendRepeat}(M_i,k_i,N_{i-1}) \text{ or} \\ & N_i = \mathit{AppendChar}(a,N_{i-1})), \end{aligned}$$

for suitable natural numbers k_i and complexity-optimal automata M_i, with $0 \leq i \leq n$.

It is easy to see that $n \leq l+q$. By contradiction, if $n > l+q$, then $\|N\| \geq l+q+2$, which implies that N is not complexity-optimal. Moreover, Theorem 3 implies that, for every $1 \leq i \leq n$, $k_i|v_i| \leq 2l+2q$, where v_i is the string recognized by M_i. From the same theorem we also have that $M_0 \in \mathcal{F}_\Sigma(q)$. Therefore, $N \in \mathcal{T}_\Sigma(l+q, q, 2l+2q)$. □

On the basis of the above results, we can devise a simple polynomial-time algorithm that solves the complexity-optimization problem for ultimately periodic words (the algorithm for finite words is just a special case). Such an algorithm receives a pair (v, w) of finite strings, where v and w are assumed to be of minimum length, and it returns as output a complexity-optimal RLA that recognizes $u = vw^\omega$. The algorithm uses the following data structures and procedures:

- a matrix $M_{\mathit{fin}}(i, j)$, where entry (i, j) stores the generated complexity-optimal automata that recognize the substrings $u[i, j]$ of u, for $1 \leq i \leq j \leq 3l + 3q$;
- an array $M_{\mathit{inf}}(i)$, where entry i stores the generated complexity-optimal automata that recognize the suffixes $u[i, \omega]$ of u, for $1 \leq i \leq l + q$ (we assume the array to be initialized with sentinel automata of complexity infinity);
- a matrix $P(i, j)$, whose values are the periods of the substrings $u[i, j]$, for $1 \leq i \leq j \leq 3l + 3q$ (such periods are computed by the procedure *PeriodsOfAllSubstrings*);
- an auxiliary procedure *BestComplexity*, which receives a finite set of RLA as input and returns an RLA of minimum complexity as output;
- an auxiliary procedure *PrefixOfUltimatelyPeriodic*(v, w, n), which returns the string $(vw^\omega)[1, n]$;
- a routine *Normalize*(i, l, q), which returns i if $i \leq l$, and $l + ((i - l - 1)\ \mathit{mod}\ q) + 1$ otherwise (notice that, if u is an ultimately periodic word with prefix length l and period q, then, for every i, $u[\mathit{Normalize}(i, l, q), \omega] = u[i, \omega]$).

ComplexityOptimalAutomaton(v, w)

```
 1: $l \leftarrow |v|$
 2: $q \leftarrow |w|$
 3: $u \leftarrow \mathit{PrefixOfUltimatelyPeriodic}(v, w, 3l + 3q)$
 4: $(P(i, j))_{i \in [1, 3l+3q], j \in [i, 3l+3q]} \leftarrow \mathit{PeriodsOfAllSubstrings}(u)$
 5: for all $i \in [1, 3l + 3q]$ do
 6:    $M_{\mathit{fin}}(i, i) \leftarrow \mathit{AppendChar}(u[i])$
 7: end for
 8: for all $n \in [2, 3l + 3q]$ in increasing order do
 9:    for all $i \in [1, 3l + 3q - n - 1]$ in increasing order do
10:       $N \leftarrow \mathit{AppendChar}(u[i], M_{\mathit{fin}}(i + 1, i + n - 1))$
11:       $p \leftarrow P(i, i + n - 1)$
12:       if $p \neq n$ then
13:          $N \leftarrow \mathit{BestComplexity}(N, \mathit{AppendRepeat}(M_{\mathit{fin}}(i, i + p - 1), n/p))$
14:       end if
15:       for all $r \in [1, n - 1]$ in increasing order do
16:          $p \leftarrow P(i, i + r - 1)$
17:          $N \leftarrow \mathit{BestComplexity}(N, \mathit{AppendRepeat}(M_{\mathit{fin}}(i, i + p - 1), r/p,$
18:                                                   $M_{\mathit{fin}}(i + r, i + n - 1)))$
```

```
19:       end for
20:       $M_{\mathit{fin}}(i, i+n-1) \leftarrow N$
21:     end for
22: end for
23: for all $i \in [l+1, l+q]$ in increasing order do
24:     $M_{\mathit{inf}}(i) \leftarrow \mathit{AppendRepeat}(M_{\mathit{fin}}(i, i+q-1), \omega)$
25: end for
26: for all $n \in [1, l+q]$ in increasing order do
27:     for all $i \in [1, l+q]$ in increasing order do
28:       $N \leftarrow \mathit{BestComplexity}(M_{\mathit{inf}}(i),$
29:                   $\mathit{AppendChar}(u[i], M_{\mathit{inf}}(\mathit{Normalize}(i+1, l, q))))$
30:       for all $r \in [1, 2l+2q]$ in increasing order do
31:         $p \leftarrow P(i, r)$
32:         $N \leftarrow \mathit{BestComplexity}(N,$
33:                   $\mathit{AppendRepeat}(M_{\mathit{fin}}(i, i+p-1), r/p,$
34:                               $M_{\mathit{inf}}(\mathit{Normalize}(i+r, l, q))))$
35:       end for
36:       $M_{\mathit{inf}}(i) \leftarrow N$
37:     end for
38: end for
39: return $M_{\mathit{inf}}(1)$
```

Lines 5–22 (resp. 23–38) initialize the matrix M_{fin} (resp. M_{inf}). In particular, the cycle at lines 23–25 sets M_{inf} with complexity-optimal automata from $\mathcal{T}_{\Sigma}(0, q, 2l+2q)$, while the n-th iteration of the loop at lines 26–38 sets M_{inf} with complexity-optimal automata from $\mathcal{T}_{\Sigma}(n, q, 2l+2q)$.

A straightforward implementation of the above algorithm would require time linear in the length of v and w to compute each automaton in M_{fin} and in M_{inf}. Pairing such a complexity with the threefold nesting of the loops, we have that $\mathcal{O}((|v|+|w|)^4)$ is an upper bound to the complexity of the problem. As a matter of fact, it is possible to compare the complexities of the generated automata without really building them. By exploiting definitions from Section 6 and by using suitable data structures, the complexity of each automaton can indeed be calculated in constant time, and thus we only need to explicitly generate the final complexity-optimal automaton for each substring $u[i, j]$. This allows us to conclude that a wiser implementation of the above algorithm would require time $\Theta((|v| + |w|)^3)$.

8 Computing size-optimal automata

In this section, we adapt the results we obtained for complexity-optimal automata to size-optimal ones. Unfortunately, we do not have an analogous of Proposition 4 for size-optimal automata. However, we can still find size-optimal automata with respect to the subclass $\mathcal{C}_\Sigma$. To do that we restrict the search space to $\mathcal{C}_\Sigma$ and we proceed exactly as in Section 7. Hereafter, we define a size-optimal automaton for u as an automaton in $\mathcal{C}_\Sigma$, with the minimum number of states (with respect to automata in $\mathcal{C}_\Sigma$), that recognizes u.

Theorem 4 *Given a finite word u, at least one of the following conditions holds:*
1. $|u| = 1$ *and* $\mathit{AppendChar}(a)$ *is size-optimal for* u*;*
2. $|u| > 1$ *and* $\mathit{AppendChar}(a, M)$ *is size-optimal for* u *whenever* M *is size-optimal for* $u[2, |u|]$*;*
3. $|u| > 1$ *and* $\mathit{AppendRepeat}(M, |u|/p)$ *is size-optimal for* u *whenever* M *is size-optimal for* $u[1, p]$*, with* p *being a period of* u*;*
4. $|u| > 1$ *and there exists* $r < |u|$ *such that* $\mathit{AppendRepeat}(M, r/p, O)$ *is size-optimal for* u *whenever* M *is size-optimal for* $u[1, p]$*, where* p *is a period of* $u[1, r]$*, and* O *is size-optimal for* $u[r + 1, |u|]$*.*

Proof. Suppose that $N \in \mathcal{C}_\Sigma$ is a size-optimal automaton. We proceed by induction on the structure of N. We only consider the non-trivial cases (the remaining cases can be handled similarly).

- Suppose $N = \mathit{AppendChar}(a, L)$ and let M be a size-optimal automaton recognizing $u[2, |u|]$. Then we have:

$$|\mathit{AppendChar}(a, M)| = 1 + |M| \leq 1 + |L| = |\mathit{AppendChar}(a, L)| = |N|$$

  This implies that $\mathit{AppendChar}(a, M)$ is size-optimal.
- Suppose $N = \mathit{AppendRepeat}(L, k, P)$, where k is a natural number. Let v be the finite word recognized by L, M be a size-optimal automaton for v, and O be a size-optimal automaton for $u[|v|n, |u|]$. We have:

$$|\mathit{AppendRepeat}(M, k, O)| = 1 + |M| + |O| \leq 1 + |L| + |P| = |N|$$

  and hence $\mathit{AppendRepeat}(M, k, O)$ is size-optimal. □

Theorem 5 *Given an ultimately periodic word u with minimum prefix length l and minimum period q, at least one of the following conditions holds:*
1. $\mathit{AppendChar}(a, M)$ *is size-optimal for* u *whenever* M *is size-optimal for* $u[2, \omega]$*;*
2. *there is a multiple* r *of* q *such that* $\mathit{AppendRepeat}(M, \omega)$ *is size-optimal for* u *whenever* M *is size-optimal for* $u[1, r]$*;*
3. *there exists* $r \leq 2l + 2q$ *such that* $\mathit{AppendRepeat}(M, r/p, O)$ *is size-optimal*

for u whenever M is size-optimal for $u[1,p]$, where p is a period of $u[1,r]$, and O is size-optimal for $u[r+1,\omega]$.

Proof. Suppose that $N \in \mathcal{C}_\Sigma$ is a size-optimal automaton that recognizes u. We prove the thesis by induction on the structure of N. We only consider the most difficult case, that is, $N = \mathit{AppendRepeat}(L,k,P)$, with $k > 1$ (it is immediate to show that k cannot be equal to 1 in a size-optimal automaton). First of all, we can replace the automata L and P by equivalent size-optimal automata M and O, respectively. Let v be the finite word recognized by M and let $r = k|v|$. We have to show that $r \leq 2l + 2q$. This can be proved by contradiction assuming that $r > 2l + 2q$ and considering the substring $t = u[l+1,r]$ of u, which has partial periods q and $|v|$. The rest of the proof goes on as the proof of Theorem 3, and thus we omit its description. □

For every $n \in \mathbb{N}$ and for every $r,m,s \in \mathbb{N}^+$, we can define two classes of automata $\mathcal{S}_\Sigma(m,s), \mathcal{R}_\Sigma(n,r,m,s) \subseteq \mathcal{C}_\Sigma$. This will help to make the search space finite when building size-optimal automata for infinite words. The definitions of $\mathcal{S}_\Sigma(m,s)$ and $\mathcal{R}_\Sigma(n,r,m,s)$ are given by induction on m and n, respectively:

$$\begin{aligned}
\mathcal{S}_\Sigma(1,s) = {} & \{\mathit{AppendChar}(a)\} \\
& \cup \{\mathit{AppendRepeat}(M,k) \ : \ kp \leq s, M \in \mathcal{F}_\Sigma(p)\} \\
\mathcal{S}_\Sigma(m+1,s) = {} & \mathcal{S}_\Sigma(m,s) \ \cup \ \{\mathit{AppendChar}(a,M) \ : \ M \in \mathcal{S}_\Sigma(m,s)\} \\
& \cup \{\mathit{AppendRepeat}(M,k,O) \ : \ kp \leq s, \\
& \quad M \in \mathcal{F}_\Sigma(p), O \in \mathcal{S}_\Sigma(m,s)\}
\end{aligned}$$

$$\begin{aligned}
\mathcal{R}_\Sigma(0,r,m,s) = {} & \{\mathit{AppendRepeat}(M,\omega) \ : \ M \in \mathcal{S}_\Sigma(m,s)\} \\
\mathcal{R}_\Sigma(n+1,r,m,s) = {} & \mathcal{R}_\Sigma(n,r,m,s) \\
& \cup \{\mathit{AppendChar}(a,M) \ : \ M \in \mathcal{R}_\Sigma(n,r,m,s)\} \\
& \cup \{\mathit{AppendRepeat}(M,k,O) \ : \ kp \leq r, \\
& \quad M \in \mathcal{F}_\Sigma(p), O \in \mathcal{R}_\Sigma(n,r,m,s)\}
\end{aligned}$$

As proved by the following proposition, we can assume the parameters n,r,m,s to be bounded by suitable functions linear in the prefix length and in the period of the given ultimately periodic word.

Proposition 6 *For every ultimately periodic word u with minimum prefix length l and minimum period q, there is $M \in \mathcal{R}_\Sigma(l+q, 2l+2q, q, 2q)$ which is size-optimal for u.*

Proof. We can recursively apply Theorem 5 and end up with a size-optimal automaton N for u such that $N = N_n$, where

$$\begin{array}{rl} & N_0 = \mathit{AppendRepeat}(M_0, \omega), \text{ and} \\ \forall\, 1 \le i \le n & (N_i = \mathit{AppendRepeat}(M_i, k_i, N_{i-1}) \text{ or} \\ & N_i = \mathit{AppendChar}(a, N_{i-1})), \end{array}$$

for suitable natural numbers k_i and size-optimal automata M_i, with $0 \le i \le n$.

It is easy to see that $n \le l+q$. By contradiction, if $n > l+q$, then $|N| \ge l+q+2$, which implies that N is not size-optimal. Moreover, Theorem 5 implies that, for every $1 \le i \le n$, $k_i|v_i| \le 2l + 2q$, where v_i is the string recognized by M_i. As for the automaton M_0, by Theorem 4, we can assume that $M_0 = Q_m$, where

$$\begin{array}{rl} & Q_1 = \mathit{AppendRepeat}(R_1, h_1) \text{ or } Q_1 = \mathit{AppendChar}(a), \text{ and} \\ \forall\, 2 \le i \le m & (Q_i = \mathit{AppendRepeat}(R_i, h_i, Q_{i-1}) \text{ or} \\ & Q_i = \mathit{AppendChar}(a, Q_{i-1})). \end{array}$$

Using an argument similar to the one we used to establish the bound $n \le l+q$, one can easily show that $m \le q$. It remains to show that for every $1 \le i \le m$, $h_i|w_i| \le 2q$, where w_i is the string recognized by R_i. Suppose, by contradiction, that $h_i|w_i| > 2q$. First, note that both q and $|w_i|$ are periods of $w_i^{h_i}$. Then, since $h_i \ge 2$, we have that $|w_i^{h_i}| \ge max(2q, 2w_i) \ge q + |w_i|$. Since q is the minimum period of u, Lemma 1 implies that $|w_i|$ is a multiple of q. Hence, the h_i-repetition of w_i is useless and $w_i^{h_i}$ can be replaced by w_i, which contradicts the hypothesis that N is size-optimal. This proves that $N \in \mathcal{R}_\Sigma(l+q, 2l+2q, q, 2q)$. □

Putting all the above results together, we can solve the size-optimization problem as we solved the complexity-optimization one, provided that we restrict the search space to $\mathcal{C}_\Sigma$. We shall use an additional auxiliary procedure $\mathit{BestRepeat}(M_{\mathit{fin}}, P, i, j)$, which receives as input a matrix M_{fin}, that contains size-optimal automata recognizing substrings of a given word u, a matrix P, that contains the periods of the substrings of u, and two indices i, j, and it returns a pair (N, k), where N is a size-optimal automaton that recognizes a repeating pattern (not necessarily the minimum one) of $u[i, j]$ and h is the number of repetitions of that pattern in $u[i, j]$.

```
BestRepeat($M_{fin}, P, i, j$)
1: $p \leftarrow P(i, j)$
2: $N \leftarrow M_{fin}(i, i + p - 1)$
3: $k \leftarrow (j - i + 1)/p$
4: for all $h \in [2, (j - i + 1)/p]$ do
5:     if $(j - i + 1) \mod hp = 0$ and $|N| > |M_{fin}(i, i + hp - 1)|$ then
6:         $N \leftarrow M_{fin}(i, i + hp - 1)$
```

```
7:         $k \leftarrow (j - i + 1)/(hp)$
8:     **end if**
9: **end for**
10: **return** $(N, k)$
```

Below we report the algorithm *SizeOptimalAutomaton*, which has almost the same structure as *ComplexityOptimalAutomaton*. The matrix $M_{rep}(m, i, j)$, where m, i, j range over the interval $[1, q]$, is used to store size-optimal automata from $\mathcal{S}_\Sigma(m, 2q)$ which recognize substrings of repeating patterns of u, namely, strings of the form $u[l + i, l + hq + j]$, with $h \in \mathbb{N}$. Furthermore, we assume M_{rep} to be initialized with dummy automata of very large size.

SizeOptimalAutomaton(v, w)

```
1: $l \leftarrow |v|$
2: $q \leftarrow |w|$
3: $u \leftarrow \mathit{PrefixOfUltimatelyPeriodic}(v, w, 3l + 3q)$
4: $(P(i, j))_{i,j} \leftarrow \mathit{PeriodsOfAllSubstrings}(u)$
5: **for all** $i \in [1, 3l + 3q]$ **do**
6:     $M_{fin}(i, i) \leftarrow \mathit{AppendChar}(u[i])$
7: **end for**
8: **for all** $n \in [2, 3l + 3q]$ **in increasing order do**
9:     **for all** $i \in [1, 3l + 3q - n - 1]$ **in increasing order do**
10:        $N \leftarrow \mathit{BestSize}(\mathit{AppendChar}(u[i], M_{fin}(i + 1, i + n - 1)),$
11:                    $\mathit{AppendRepeat}(\mathit{BestRepeat}(M_{fin}, P, i, i + n - 1)))$
12:        **for all** $r \in [1, n - 1]$ **in increasing order do**
13:            $N \leftarrow \mathit{BestSize}(N, \mathit{AppendRepeat}(\mathit{BestRepeat}(M_{fin}, P, i, i + r - 1),$
14:                                        $M_{fin}(i + r, i + n - 1)))$
15:        **end for**
16:        $M_{fin}(i, i + n - 1) \leftarrow N$
17:    **end for**
18: **end for**
19: **for all** $i \in [1, q]$ **in increasing order do**
20:     $M_{rep}(1, i, i) \leftarrow \mathit{AppendChar}(u[i])$
21:     **for all** $s \in [1, 2q]$ **in increasing order do**
22:         $j \leftarrow \mathit{Normalize}(i + s, 0, q)$
23:         $M_{rep}(1, i, j) \leftarrow \mathit{BestSize}(M_{rep}(1, i, j),$
24:                           $\mathit{AppendRepeat}(\mathit{BestRepeat}(M_{fin}, P, i, i + s)))$
25:     **end for**
26: **end for**
27: **for all** $m \in [2, q]$ **in increasing order do**
28:     **for all** $i \in [1, q]$ **in increasing order do**
29:         **for all** $j \in [1, q]$ **in increasing order do**
30:             $h \leftarrow \mathit{Normalize}(i + 1, 0, q)$
31:             $M_{rep}(m, i, j) \leftarrow \mathit{BestSize}(M_{rep}(m - 1, i, j),$
32:                               $AppendChar(u[i], M_{rep}(m-1, h, j)))$
33:       end for
34:       for all $s \in [1, 2q]$ in increasing order do
35:          $C_{new} \leftarrow BestRepeat(M_{fin}, P, i, i+s)$
36:          $k \leftarrow Normalize(i+s+1, 0, q)$
37:          for all $h \in [1, q]$ in increasing order do
38:             $j \leftarrow Normalize(i+s+h, 0, q)$
39:             $M_{rep}(m, i, j) \leftarrow BestSize(M_{rep}(m, i, j),$
40:                               $AppendRepeat(C_{new}, M_{rep}(m-1, k, j)))$
41:          end for
42:       end for
43:    end for
44: end for
45: for all $i \in [1, q]$ in increasing order do
46:    $M_{inf}(l+i) \leftarrow AppendRepeat(M_{rep}(q, i, Normalize(i+q-1)), \omega)$
47: end for
48: for all $n \in [1, l+q]$ in increasing order do
49:    for all $i \in [1, l+q]$ in increasing order do
50:       $N \leftarrow BestSize(M_{inf}(i), AppendChar(u[i], M_{inf}(Normalize(i+1, l, q))))$
51:       for all $r \in [i, i+2l+2q-1]$ in increasing order do
52:          $N \leftarrow BestSize(M_{inf}, AppendRepeat(BestRepeat(M_{fin}, P, i, i+r),$
53:                               $M_{inf}(Normalize(r+1, l, q))))$
54:       end for
55:       $M_{inf}(i) \leftarrow N$
56:    end for
57: end for
58: return $M_{inf}(1)$
```

Lines 19–44 are used to fill the matrix M_{rep} with appropriate automata. This is done in an inductive way, by first computing $M_{rep}(1, i, j)$ (lines 19–26) and then computing $M_{rep}(m, i, j)$, given $M_{rep}(m-1, i, j)$, (lines 27–44). The overall complexity is $\Theta((|v| + |w|)^4)$.

9 Conclusions and further work

In this paper, we dealt with optimization problems for automaton-based representations of time granularities in a systematic way. Such problems, which are extremely relevant from a computational point of view, have often been overlooked in the literature. We started by establishing some basic properties of repeating patterns of strings. Then, we showed how finite and ultimately periodic strings can be naturally encoded in terms of Restricted Labeled Single-String Automata (RLA) and we provided efficient algorithms for their manipulation. In particular, we devised a suitable measure of automata complexity, that takes the nesting of secondary transitions into account, and we developed algorithms for granule conversions that operate in worst-case linear time with respect to such a measure. Next, we focussed on the problem of minimizing automaton-based representations with respect to either this complexity measure or the traditional measure that only considers the number of states (size) of the automaton. By exploiting dynamic programming, we gave polynomial-time algorithms that respectively compute complexity-optimal and size-optimal automata from a string-based specification of a time granularity. While the former computes automata which are complexity optimal with respect to the whole class of RLA, the latter confines itself to the restricted class of RLA in $\mathcal{C}_\Sigma$.

We believe it possible to further improve such algorithms by exploiting subtle relationships between repeating patterns of strings and secondary transitions of optimal RLA. As a matter of fact, we conjecture that the loops determined by the secondary transitions of a complexity-optimal RLA can be related to *maximal repetitions* in the recognized word (a maximal repetition of u is a periodical substring $u[i,j]$ whose minimum period increases as soon as $u[i,j]$ is prolonged to the right, e.g., $u[i,j+1]$, or to the left, e.g., $u[i-1,j]$). Moreover, we are exploring the possibility of generalizing the size-optimization algorithm in order to produce optimal automata with respect to the whole class of RLA (instead of the subclass $\mathcal{C}_\Sigma$ only). We believe that such a task could be accomplished by introducing a suitable operator, in addition to *AppendChar* and *AppendRepeat*, which collapses *non-distinguishable* states of RLA (at the moment, the major stumbling block is the problem of finding an appropriate definition of RLA distinguishable states).

Acknowledgments

We would like to thank the anonymous reviewers for their useful comments and suggestions. The work of the second and third author has been partially funded by the bilateral project *Temporal Logics in Computer and Information Sciences*, supported by the Italian Ministero degli Affari Esteri and the National Research Foundation of South Africa, under the joint Italy/South Africa Science and Technological Agreement, the European INTAS project *Algebraic and Deduction Methods in Non Classical Logics and their Applications to Computer Science*, and the Italian national project *Constraints and Preferences as a Unifying Formalism for System Analysis and Solution of Real-Life Problems.*

References

[1] C. Bettini, C.E. Dyreson, W.S. Evans, R.T. Snodgrass, and X.S. Wang. A glossary of time granularity concepts. In O. Etzion, S. Jajodia, and S. Sripada, editors, *Temporal Databases: Research and Practice*, volume 1399 of *LNCS*, pages 406–413. Springer, 1998.

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

[3] C. Bettini, X. S. Wang, and S. Jajodia. Solving multi-granularity temporal constraint networks. *Artificial Intelligence*, 140(1-2):107–152, 2002.

[4] H. Calbrix, M. Nivat, and A. Podelski. Ultimately periodic words of rational ω-languages. In *Proceedings of the 9th International Conference on Mathematical Foundations of Programming Semantics*, volume 802 of *LNCS*, pages 554–566. Springer, 1994.

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

[6] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition.* The MIT Press, 2001.

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

[8] U. Dal Lago, A. Montanari, and G. Puppis. Time granularities, calendar algebra, and automata. Research Report UDMI/2003/04, Dipartimento di Matematica e Informatica, Università di Udine, Italy, 2003.

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

[10] C.E. Dyreson, W.S. Evans, H. Lin, and R.T. Snodgrass. Efficiently supporting temporal granularities. *IEEE Transactions on Knowledge and Data Engineering*, 12(4):568–587, 2000.

[11] J. Euzenat and A. Montanari. Time granularity. In M. Fisher, D. Gabbay, and L. Vila, editors, *Handbook of Temporal Reasoning in Artificial Intelligence*, pages 59–118. Elsevier, 2005.

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

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

[14] M. Franceschet and A. Montanari. Temporalized logics and automata for time granularity. *Theory and Practice of Logic Programming*, 4(5-6):621–658, 2004.

[15] D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in

strings. *SIAM Journal on Computing*, 6:323–350, 1977.

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

[17] 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.

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

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

[20] R.T. Snodgrass, editor. *The TSQL2 Temporal Query Language.* Kluwer Academic Publishers, 1995.

[21] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 389–455. Springer, 1997.

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

Appendix

Lemma 2 The relation $\dashv$ respects the extension of strings to the right, that is, $(v \cdot a) \dashv (u \cdot a)$ holds if and only if both $v \dashv u$ and $u[|v|+1] = a$ hold (see Figure 3).

Proof. The proof is trivial. If $v \cdot a$ is a border of $u \cdot a$, then both $v = u[1, |v|] = u[|u| - |v| + 1, |u|]$ and $u[|v|+1] = a$ hold. For the converse, if v is a border of u and $u[|v|+1] = a$ holds, then $v \cdot a = u[1, |v|+1] = u[|u| - |v| + 1, |u|] \cdot a$ and hence $v \cdot a$ is a border of $u \cdot a$. $\square$

Lemma 3 The relation $\dashv$ is linear to the left, that is, whenever both $v \dashv u$ and $w \dashv u$ hold, then we have $v = w$ or $v \dashv w$ or $w \dashv v$.

Proof. Let u, v, and w be three finite strings of length n, m, and r, respectively. By definition, from $v \dashv u$, it follows that $v = u[1, m] = u[n - m + 1, n]$, and, from $w \dashv u$, it follows that $w = u[1, r] = u[n - r + 1, n]$. If r is less than m, then we have $w = u[1, r] = u[1, m][1, r] = v[1, r]$ and $w = u[n - r + 1, n] = u[n - m + 1, n][m - r + 1, m] = v[m - r + 1, m]$. Hence w is a border of v. The other cases can be proved in a similar way. $\square$

Corollary 1 Let u be a finite string and let $u_1, \ldots, u_n$ be the (unique) sequence of finite strings such that $\varepsilon = u_1 \dashv\vdash \ldots \dashv\vdash u_n \dashv\vdash u$. If $v \dashv u$, then there is $1 \leq k \leq n$ such that $v = u_k$.

Proof. The proof is by induction on n. The case $n = 1$ is trivial, since $v \dashv u$ implies $v = \varepsilon = u_1$. For $n > 1$, we distinguish two cases: either v is a maximum border of u, or v is a non-maximum border of u. In the former case, we simply let $k = n$. In the latter case, we let $w = u_n \dashv\vdash u$ and, by Lemma 3, we know that $v \dashv w$.By applying the inductive hypothesis, we immediately obtain $v = u_k$, for some $1 \leq k < n$. □

Theorem 1 Let u be a finite string and let $u_1, \ldots, u_n$ be the (unique) sequence of finite strings such that $\varepsilon = u_1 \dashv\vdash \ldots \dashv\vdash u_n \dashv\vdash u$. For any given v, the following two conditions are equivalent:

1. $(v \cdot a) \dashv\vdash (u \cdot a)$,
2. there is a $1 \leq k \leq n$ such that $u_k = v$, $u[|u_k| + 1] = a$, and $u[|u_h| + 1] \neq a$ for all $h > k$.

Proof. As for the implication from *1* to *2*, if $v \cdot a$ is the maximum border of $u \cdot a$, then, by Lemma 2, v is a border of u. From Corollary 1, we know that $v = u_k$ for a suitable $k > 0$, and, again by Lemma 2, $u[|u_k| + 1] = a$. Furthermore, for every $h > k$, u_h is longer than u_k and hence $u_h \cdot a$ cannot be a border of $u \cdot a$. Since u_h is a border of u and $u_k \cdot a$ is the maximum border of $u \cdot a$, this implies that $u[|u_h| + 1] \neq a$. Conversely, let k be the largest index such that $u[|u_k| + 1] = a$. Clearly $u_k \cdot a$ is a border of $u \cdot a$. If there would be a border $v \cdot a$ of $u \cdot a$ with $|v \cdot a| > |u_k \cdot a|$, then, by Lemma 2, v would be a border of u and, by Corollary 1, there would be an integer h such that $u_h = v$. Moreover, from Lemma 3 we know that u_k is a border of v, and hence h should be greater that k. This implies that $u[|u_h| + 1] = a$, which is against the hypothesis of k being the largest index such that $u[|u_k| + 1] = a$. □