Towards Compact and Tractable Automaton-based Representations of Time Granularity

Ugo Dal Lago, Angelo Montanari, Gabriele Puppis

Towards Compact and Tractable Automaton-based Representations of Time Granularities

Ugo Dal Lago[1], Angelo Montanari[2], and Gabriele Puppis[2]

[1] Dipartimento di Scienze dell'Informazione, Università di Bologna
Mura Anteo Zamboni 7, 40127 Bologna, Italy
dallago@cs.unibo.it

[2] Dipartimento di Matematica e Informatica, Università di Udine
via delle Scienze 206, 33100 Udine, Italy
{montana,puppis}@dimi.uniud.it

Abstract. Different approaches to time granularity have been proposed in the database literature to formalize the notion of calendar, based on algebraic, logical, and string-based formalisms. In this paper, we further develop an alternative approach based on automata, originally proposed in [4], which makes it possible to deal with infinite time granularities in an effective (and efficient) way. In particular, such an approach provides an effective solution to fundamental problems such as equivalence and conversion of time granularities. We focus our attention on two kinds of optimization problems for automaton-based representations, namely, computing the smallest representation and computing the most tractable representation, that is, the one on which crucial algorithms (e.g., granule conversion algorithms) run fastest. We first introduce and compare these two minimization problems; then, we give a polynomial time algorithm that solves the latter.

1 Introduction

The notion of time granularity comes into play in a variety of problems involving time representation and management in database applications, including temporal database design, temporal data conversion, temporal database interoperability, temporal constraint reasoning, data mining, and time management in workflow systems. Different approaches to time granularity have been proposed in the database literature, based on algebraic [1, 9], logical [3], and string-based [11] formalisms. We restrict our attention to the latter.

The string-based formalism eases access to and manipulation of data associated with different granularities, making it possible to solve some basic problems about time granularities, such as the equivalence problem, in an effective way. 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 is dealt with by the automaton-based approach to time granularity, that 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 Simple Single-String Automata (Simple SSA for short), thus making it possible to (re)use well-known results from automata theory. Simple SSA were originally proposed by Dal Lago and Montanari to model infinite periodical granularities [4]. Furthermore, they showed that regularities of modeled granularities can be naturally expressed by extending Simple SSA with counters (let us call 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 [5], we proved that SSA provide an efficient solution to the fundamental problems of equivalence, namely, the problem of establishing whether two different representations define the same granularity, and granule conversion, namely, the problem of relating granules of a given granularity to those of another one. To this end, we introduced a suitable variant of SSA, called Restricted Labeled Single-String Automata (RLA for short), and we showed that these automata are at least as expressive as the string-based formalism, better fitting for direct algorithmic manipulation. As an example, granule conversion problems can be solved in polynomial time with respect to the size of the involved RLA.

The algorithmic flavor of automaton-based representations of time granularity suggests an alternative point of view on their role: RLA can be used not only as a formalism for the direct specification of time granularities, but also as a low-level formalism into which high-level time granularity specifications can be mapped. From this point of view, the problem of reducing as much as possible the complexity of basic algorithms becomes even more crucial. In [5], we defined a suitable set of algorithms mapping expressions of Calendar Algebra (the high-level formalism for modeling time granularities developed by Ning et al. in [9]) to equivalent RLA-based representations. In this paper, we focus our attention on minimization problems for RLA.

There exist at least two possible notions of minimization. According to the first one, minimizing means computing the smallest representation of a given time granularity; according to the second one, minimizing means computing the most tractable representation of a given granularity, that is, the one on which crucial algorithms run fastest. The former kind of automaton-based representation is called a *size-optimal* representation, while the latter is called a *complexity-optimal* representation. These two criteria are clearly not equivalent, since the smallest representation is not necessarily the most tractable one, and vice versa. Furthermore we claim that both problems yield non-unique solutions. In the following, we tackle the complexity-minimization problem by using dynamic programming: we state some closure properties of RLA with respect to concatenation, iteration, and repetition of words, and we show how to compute complexity-optimal automata from smaller (optimal) ones in a bottom-up fashion. The resulting algorithm runs 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 give a definition 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 focus our attention on the automaton-based approach to time granularity. We define RLA and we state some basic properties of them. In Section 4 we briefly describe some polynomial algorithms which can be used to efficiently solve the equivalence and granule conversion problems for RLA-based representations of time granularies. In Section 5 we introduce the size-minimization and complexity-minimization problems, we point out important aspects about their solutions, and we give an intuitive explanation of the computation of complexity-optimal automata. In Section 6, we discuss the details of the proposed solution; in particular, we show how a complexity-optimal automaton recognizing a given ultimately periodic word can be effectively built up from a suitable representation of the repetitions of the word. In Section 7 we outline future research directions, with a special emphasis on possible improvements on the proposed complexity-minimization algorithm and on promising strategies to efficiently solve the size-minimization problem. (Reference [6] is an extended version of this work, including all proof details.)

2 The String-based Model of Time Granularities

Since in many applications different time granularities can be used to specify the validity intervals of different facts [1], database systems need the ability of properly relating granules belonging to different time granularities. Such an ability presupposes the formalization of the notion of granularity. In this section, we first give a formal definition of time granularity, which captures a reasonably large class of temporal structures; then, we specialize such a definition in order to allow a finite representation and an efficient manipulation of the associated data.

Definition 1. *Given a set T of temporal instants and a total order $<$ on T, a* time granularity *on the temporal domain $(T, <)$ is a total function $G : \mathbb{Z} \rightarrow 2^T$ such that, for every pair of integers x and y, $x < y$ implies*

$$\forall\, t_x \in G(x).\ \forall\, t_y \in G(y).\ t_x < t_y .$$

Each non-empty set $G(x)$, with $x \in \mathbb{Z}$, is called a *granule* and each integer in the set $\{x \in \mathbb{Z} : G(x) \neq \emptyset\}$ is called a *label*. Note that 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, like, for instance, `BusinessDay`, `BusinessWeek`, and `BusinessMonth`. Figure 1 depicts some of these granularities.

In the following, we assume granularity labels to belong to the set $\mathbb{N}^+$ (as a matter of fact, most applications assume the existence of a first granule). It is immediate to see that the set of all functions satisfying Definition 1 becomes uncountable as soon as the underlying temporal domain becomes infinite. As

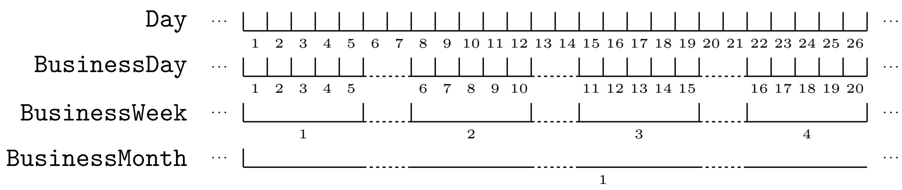

Fig. 1. Some examples of time granularities.

a consequence, it is not possible to deal with all of them by means of a finitary formalism. However, the problem of mastering temporal structures for time granularity can be tackled in an effective way by restricting to periodical granularities. In [11], Wijsen shows that such granularities can be naturally expressed in terms of ultimately periodic words 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 the following, we assume the reader to be familiar with basic terminology and notation on finite and infinite strings [10]. In particular, we will often write a generic string u as $u[1]u[2]u[3]\ldots$, where $u[i]$ denotes the i-th element of the string, and we will 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 will denote by S^∞ the set $S^\omega \cup S^*$, where S^ω (respectively, S^*) stands for the set of all and only the infinite (respectively, finite) strings over S.

Definition 2. *Given a word $u \in \{\blacksquare, \square, \wr\}^\omega$ containing infinitely many occurrences of non-separator symbols, we say that u* represents *G if, for every pair of positive integers x and y, $x \in G(y)$ if and only if $u[x+y-1] = \blacksquare$ and $u[1, x+y-2]$ contains exactly $y-1$ occurrences of $\wr$.*

As an example, the infinite word $\blacksquare\blacksquare\blacksquare\blacksquare\blacksquare\square\square \wr \blacksquare\blacksquare\blacksquare\blacksquare\blacksquare\square\square \wr \ldots$ represents the granularity `BusinessWeek` over the temporal domain of days.

In order to *finitely* model time granularities, Wijsen introduces the notion of *granspec*. A granspec is an ordered pair (u, v) of finite strings such that $u, v \in \{\blacksquare, \square, \wr\}^*$ and v contains at least one occurrence of a non-separator symbol. Strings u and v are respectively called the *prefix* and the *repeating pattern* of the ultimately periodic string $u \cdot v^\omega$ representing the (periodical) time granularity. For instance, the granularity `BusinessWeek` $\blacksquare\blacksquare\blacksquare\blacksquare\blacksquare\square\square \wr \blacksquare\blacksquare\blacksquare\blacksquare\blacksquare\square\square \wr \ldots$ can be encoded by the granspec $(\varepsilon, \blacksquare\blacksquare\blacksquare\blacksquare\blacksquare\square\square\wr)$. Furthermore, to solve the equivalence problem for (representations of) time granularities, Wijsen proposed a suitable *canonical form* of granspecs, which turns out to be a sort of minimum representation of periodical granularities. However, it is worth mentioning that, whenever the granularity to be represented has a long period and/or a long prefix, the granspec formalism produces lengthy canonical granspecs. As a consequence, computations on time granularities represented by granspecs may take a great deal of time. For example, if (u, v) is a granspec representing months of the Gregorian Calendar in terms of days, we have that $|u| + |v| \geq 146097$. In the following section, we introduce the automaton-based approach, which yields more succinct representations of time granularities.

3 From Strings to Automata

The idea of viewing granularities as ultimately periodic strings 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 [2]. The basic idea underlying the automaton-based approach to time granularity is simple: we take an automaton M recognizing a *single* word $u \in \{\blacksquare, \square, \wr\}^\omega$ 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 differ from finite automata and Büchi automata as they accept single words instead of sets of words. As a matter of fact, RLA can also be viewed as a variant of SSA [4], in which counters over discrete domains are exploited to obtain succinct representations of time granularities.

Before formalizing the notion of RLA, we give an intuitive explanation of the structure and behavior of automata belonging to this class. In order to simplify the notation and the formalization of useful properties, RLA label states instead of transitions. The set of states of an RLA, denoted by S, 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, there are two kinds of transitions, respectively called *primary* and *secondary* transitions. Intuitively, primary transitions are defined in the standard way, while secondary transitions have been introduced to succinctly represent repetitions. At any point of the computation, at most one (primary or secondary) transition is taken according to an appropriate rule envisaging the state at which the automaton lies and the value of a *counter*.

Figure 2 depicts two RLA, that respectively recognize the words $(\blacksquare\square^6\wr)^\omega$ and $(\blacksquare\wr(\square\wr)^6)^\omega$, both representing mondays in terms of days (the former associates the labels $1, 2, 3, \ldots$ with the granules, while the latter associates the labels $1, 8, 15, \ldots$ with them). States in S_Σ are represented by 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. This simple example provides an intuitive idea of how RLA allow one to compactly encode repeating patterns in granularities by means of counters and transitions.

Definition 3. *A* Restricted Labeled Single-String Automaton *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* states *(let* $S = S_\Sigma \cup S_\varepsilon$*);*
- Σ *is a finite* alphabet*;*
- $\Omega : S_\Sigma \to \Sigma$ *is the* labeling function*;*
- $\delta : S \rightharpoonup S$*, is a partial function, called* primary transition function*;*

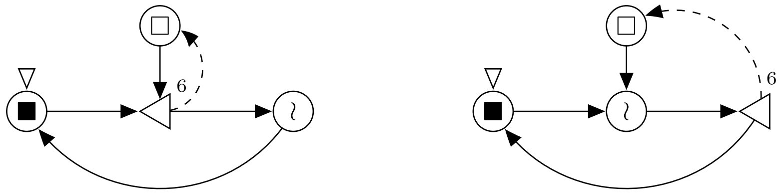

Fig. 2. Two RLA that represent mondays in terms of days.

- $\gamma : S_\varepsilon \to S$ *is a total function, called* secondary transition function*, such that:*
  1. *for every* $s \in S_\varepsilon$*,* $(\gamma(s), s)$ *belongs to the reflexive and transitive closure* δ^* *of* δ*; the least* $n \in \mathbb{N}$ *such that* $(\gamma(s), s) \in \delta^n$ *is called the* γ*-degree of* s *and* $\Gamma_M \subseteq S_\varepsilon \times S$ *is a relation such that* $(s, r) \in \Gamma_M$ *iff* $r = \delta^i(\gamma(s))$ *with* i *less than or equal to the* γ*-degree of* s*;*
  2. *the reflexive and transitive closure* Γ_M^* *of* Γ_M *must be antisymmetric;*
- $s_0 \in S$ *is the* initial state*;*
- $C_0 : S_\varepsilon \to \mathbb{N}$ *is the* initial valuation.

Conditions i) and ii) on the secondary transition function enforce the existence of a partial order Γ_M^* on states of M. Such an order immediately suggests an induction principle, called γ-*induction*, which may be used in both formal definitions and proofs.

The definition of the computation of an RLA is based on the notion of configuration. For any finite set S of states, a *valuation* C on S is any function $C : S \to \mathbb{N}$. In the following, we denote as $\mathcal{C}_M$ the class $\mathbb{N}^{S_\varepsilon}$ of all the valuations for the counters of an RLA $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, C_0)$. A *configuration* is a pair (s, C), with $s \in S$ and $C \in \mathcal{C}_M$. The transitions of M are taken according to a partial function $\Delta_M : S \times \mathcal{C}_M \rightharpoonup S \times \mathcal{C}_M$ such that $\Delta_M(s, C) = (t, D)$ if and only if one of the following three conditions holds:

- $s \in S_\Sigma \ \wedge \ t = \delta(s) \ \wedge \ \forall\, r.\ D(r) = C(r)$ (namely, whenever the automaton lies in a labeled state, then it always makes a primary transition);
- $s \in S_\varepsilon \ \wedge \ C(s) \neq 0 \ \wedge \ t = \gamma(s) \ \wedge \ D(s) = C(s) - 1 \ \wedge \ \forall\, r \neq s.\ (D(r) = C(r))$ (namely, whenever the automaton lies in a non-labeled state and the corresponding counter is positive, then it makes a secondary transition);
- $s \in S_\varepsilon \ \wedge \ C(s) = 0 \ \wedge \ t = \delta(s) \ \wedge \ D(s) = C_0(s) \ \wedge \ \forall\, r \neq s.\ (D(r) = C(r))$ (namely, whenever the automaton lies in a non-labeled state and the corresponding counter is 0, then it makes a primary transition and it reinitializes the counter).

The *computation* of M is the maximum (possibly infinite) sequence $\rho \in (S \times \mathcal{C}_M)^\infty$ such that $\rho[1] = (s_0, C_0)$ and $\Delta_M(\rho[i]) = \rho[i+1]$ for every $i \geq 1$. From the computation ρ of M, it is easy to extract a sequence of states $\rho_\Sigma \in S_\Sigma^\infty$ by discarding states belonging to S_ε and valuations. We say that M *recognizes* the word u if and only if $u = \Omega(\rho_\Sigma)$, where $\Omega(\rho_\Sigma)$ is the sequence obtained by applying the labeling function Ω to (each element of) the sequence of states ρ_Σ. Thus, RLA recognize either finite words or ultimately periodic words (namely, those words which result from concatenating a finite prefix to a repeating pattern). Note that the computation of M may be an infinite sequence, even if the recognized word is finite. However we can overcome this clumsy situation by discarding useless states and transitions of RLA.

As already mentioned, the main feature of RLA is the way they encode repeating patterns of words. As a matter of fact, it is possible to provide a formal characterization of the words recognized by RLA in terms of repetitions of smaller substrings. Precisely, given a RLA $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, C_0)$, one can show that it recognizes the word

$$u = \Omega\left(\rho_{s_0} \cdot \rho_{\delta(s_0)} \cdot \rho_{\delta^2(s_0)} \cdot \ldots\right)$$

where each ρ_s is defined to be

- s, whenever $s \in S_\Sigma$;
- $(\rho_{\gamma(s)} \cdot \rho_{\delta(\gamma(s))} \cdot \rho_{\delta^2(\gamma(s))} \cdot \ldots \cdot \rho_{\delta^{n-1}(\gamma(s))})^{C_0(s)}$, where n is the γ-degree of s, whenever $s \in S_\varepsilon$.

As a consequence, any word recognized by an RLA can be represented using expressions as $(\blacksquare^5\square^2\wr)^\omega$, $((\blacksquare^2\square)^2\square\wr)^\omega$, ... denoting nested repetitions.

4 Granularity Equivalence and Granule Conversions

In this section, we briefly discuss the equivalence and the granule conversion problems. The decidability of the former problem implies 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 relevance of the granule conversion problem has been advocated by several authors, e.g., [1], even though in most solutions it has been only partially worked out in a rather complex way.

To explain our solutions, we first address a simpler problem, which arises very often when dealing with representation of time granularities as well as with infinite strings in general, namely, the problem of finding the n-th occurrence of a given symbol in a string. From the point of view of the theory of automata, this problem can obviously be 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 (of the RLA in our case) until the n-th occurrence of the symbol is recognized. Nevertheless, we can improve this straightforward solution by taking advantage of the definition of RLA. For instance, if we are searching for an occurrence of a symbol $a \in \Sigma$ in the word u recognized by the RLA $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, C_0)$ and $\Omega(\rho_{s_0})$ contains no occurrences of symbol a, then we can avoid processing the first $|\rho_{s_0}|$ symbols in u. Similarly, if $s_0 \in S_\varepsilon$ and $\Omega(\rho_{s_0})$ contains at least an occurrence of a, but $\Omega(\rho_{\gamma(s_0)})$ does not, then we can start searching for an occurrence of a in u from the position $(1 + |\rho_{\gamma(s_0)}|)$. For every state $s \in S$ and every symbol $a \in \Sigma$, the length of ρ_s and the number of occurrences of a in $\Omega(\rho_s)$ can be computed in polynomial time *with respect to the number of states* by exploiting the definition of ρ_s. Furthermore, such values can be pre-computed and stored into appropriate data structures for M. On the grounds of the above observations, 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 in 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 for each symbol $a \in \Sigma$.

In spite of the simplicity of this idea, *SeekAtOccurrence* turns out to be rather complex and the formal analysis of its complexity is even more involved [5]. However, it is not difficult to show that the worst-case time for $SeekAtOccurrence(M, s, C, A, n, counter)$ is asymptotically equivalent to a suitable *complexity measure*, defined in terms of the nesting structure of the transition functions of M. We use $\|M\|$ to denote such a measure, which is defined, according to the principle of γ-induction, as follows:

$$\|M\| = max\{\mathbf{C}^M_{s_0,t} : (s_0, t) \in \delta^*\},$$

where, for each pair of states $(s, t) \in \delta^*$, $\mathbf{C}^M_{s,t}$ is defined to be

- 1, if $s = t$;
- $1 + \mathbf{C}^M_{\delta(s),t}$, if $s \in S_\Sigma$ and $s \neq t$;
- $max\{1 + \mathbf{C}^M_{\delta(s),t}, \mathbf{C}^M_{\gamma(s),s}\}$, if $s \in S_\varepsilon$ and $s \neq t$.

As for relationships between the complexities of automaton-based and string-based representations, there exist a number of cases that account for the compactness and tractableness 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.

We now give an intuitive account of how to decide whether or not two given RLA represent the same granularity. Details of the algorithm, which exploits noticeable properties of equivalent representations and extensively uses *SeekAtOccurrence*, are given in [5]. The basic ingredients are the following ones. First, it holds that two RLA M and N represent the same granularity if and only if ultimately periodic words u and v, recognized respectively by M and N and having prefix lengths p_u and p_v and period lengths q_u and q_v, are *G-aligned*. Two ultimately periodic words u and v are said to be G-aligned if and only if all occurrences of the filler symbol in u and v lie at the same positions and are interleaved by the same number of occurrences of the separator symbol. Such a characterization of equivalent representations can be exploited by showing that a sufficient condition for u and v to be G-aligned is that two prefixes of u and v (not shorter than $max(p_u + q_u, p_v + q_v) + lcm(q_u, q_v)$) are G-aligned. Algorithm *SeekAtOccurrence* can be used to check the G-alignment property on words recognized by two RLA M and N in time $O((\|M\| + \|N\|)n)$, where n bounds the number of occurrences of $\blacksquare$ in the prefix and period of u and v.

Consider now the problem of converting temporal intervals from a given granularity to a coarser or finer one. RLA can be exploited to solve many conversion problems in polynomial time with respect to the number of states of the involved automata. In particular, we can define two functions mapping intervals of temporal points to intervals of labels of a given granularity covering the input interval, and vice versa. It is worth pointing out that such functions are similar to the conversion operators introduced by Snodgrass et al. [7] and that they can be computed on RLA by exploiting the algorithm *SeekAtOccurrence*. As an example, the following two algorithms solve conversion problems by requiring only a finite number of calls to *SeekAtOccurrence*. It is not difficult to show that such algorithms, as well as many others which compute similar functions, can be executed in time $O(\|M\|)$, where M is the RLA representing the involved granularity.

$\underline{\mathit{UpConversion}(M, t_1, t_2)}$

let $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, C_0)$
$(s, C) \leftarrow (s_0, C_0)$
$\mathit{SeekAtOccurrence}(M, s, C, \{\blacksquare, \square\}, t_1 - 1, \mathit{counter}_1)$
$\mathit{SeekAtOccurrence}(M, s, C, \{\blacksquare\}, 1, \mathit{counter}_2)$
$x_1 \leftarrow \mathit{counter}_1[\wr] + \mathit{counter}_2[\wr] + 1$
$(s, C) \leftarrow (s_0, C_0)$
$\mathit{SeekAtOccurrence}(M, s, C, \{\blacksquare, \square\}, t_2, \mathit{counter}_3)$
$(s, C) \leftarrow (s_0, C_0)$
$\mathit{SeekAtOccurrence}(M, s, C, \{\blacksquare\}, \mathit{counter}_3[\blacksquare], \mathit{counter}_4)$
$x_2 \leftarrow \mathit{counter}_4[\wr]$
return (x_1, x_2)

$\underline{\mathit{DownConversion}(M, x_1, x_2)}$

let $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, C_0)$
$(s, C) \leftarrow (s_0, C_0)$
$\mathit{SeekAtOccurrence}(M, s, C, \{\wr\}, x_1 - 1, \mathit{counter}_1)$
$\mathit{SeekAtOccurrence}(M, s, C, \{\blacksquare\}, 1, \mathit{counter}_2)$
$t_1 \leftarrow \mathit{counter}_1[\blacksquare] + \mathit{counter}_2[\blacksquare] + \mathit{counter}_1[\square] + \mathit{counter}_2[\square]$
$(s, C) \leftarrow (s_0, C_0)$
$\mathit{SeekAtOccurrence}(M, s, C, \{\wr\}, x_2, \mathit{counter}_3)$
$(s, C) \leftarrow (s_0, C_0)$
$\mathit{SeekAtOccurrence}(M, s, C, \{\blacksquare\}, \mathit{counter}_3[\blacksquare], \mathit{counter}_4)$
$t_2 \leftarrow \mathit{counter}_4[\blacksquare] + \mathit{counter}_4[\square]$
return (t_1, t_2)

5 Optimality of Automaton-based Representations

In the previous section we briefly summarized the main features of two basic algorithms working on RLA M, whose worst-case time complexity linearly depends on $\|M\|$. It immediately follows that it is worth to minimize $\|M\|$. Furthermore, there exists a widespread recognition of the fact that state minimization is a crucial problem in classical automata theory as well as in the theory of reactive systems. Another goal of practical interest is thus the minimization of the number of states of M (let us denote it by $|M|$), so that smaller representations, in place of bigger ones, can be used. The former problem is called complexity-minimization problem, while the latter is called size-minimization problem.

Size and complexity of an RLA are obviously related one to the other; however, corresponding problems are not equivalent at all. In particular, the size-minimization problem seems to be harder than the complexity-minimization problem and it will not be discussed in detail in this paper. Furthermore, optimal automata are not guaranteed to be unique (up to isomorphisms) as it happens, for instance, for Deterministic Finite Automata. As an example, Figure 3 depicts two size-optimal automata (M and N) and two complexity-optimal automata (M and O) recognizing the finite word ■■□■■□■■□.

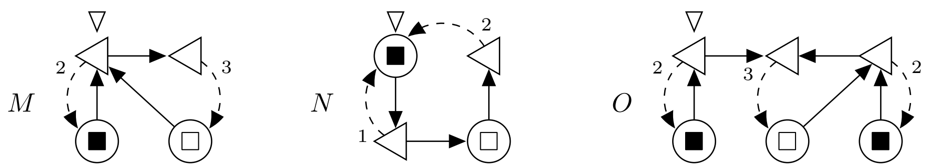

Fig. 3. Size-optimal and complexity-optimal automata.

Automata minimization 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 this paper, we cope with the minimization problem for RLA by using dynamic programming, that is, by computing an optimal automaton starting from smaller (optimal) automata in a bottom-up fashion. The key point of such a solution is the proof that the 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 of compound automata with that of their components. In the next section we will take advantage of these results to give an optimal substructure property for RLA.

The class of RLA is closed with respect to the operations of concatenation, repetition, and iteration of words. Given two RLA M and N, which respectively recognize a (finite) word u and a (not necessarily finite) word v, let *Concatenate*(M,N), *Iterate*(M), and *Repeat*(M,k) respectively be the *concatenation of M and N*, which recognizes the word $u \cdot v$, the *iteration of M*, which recognizes the word u^ω, and the *k-repetition of M*, which recognizes the word u^k. The resulting automata can be computed as follows:

- the automaton *Concatenate*(M,N) can be obtained in the usual way by linking the final state of M, namely, the state reached at the end of the computation of M, to the initial state of N by means of a primary transition;

- the automaton *Iterate*(M) can be obtained by linking the final state of M to the initial state of M by means of a primary transition;
- the automaton *Repeat*(M, k) can be obtained by introducing a new non-labeled state s_{loop} and by adding (i) a primary transition from the final state of M to s_{loop}, (ii) a secondary transition from s_{loop} to the initial state of M, and (iii) a counter on s_{loop}, with initial valuation equal to k.

Moreover, the complexity of these automata can be given in terms of the complexities of the component automata as follows:

- *Concatenate*(M, N) has complexity $max\{\|M\|, n + \|N\|\}$, where n is the cardinality of the set of states reachable from the initial state of M by means of primary transitions only;
- *Iterate*(M) has complexity $\|M\|$;
- *Repeat*(M, k) has complexity $\|M\| + 1$.

As a matter of fact, we can actually give the status of algorithms running in linear time to *Concatenate*, *Iterate*, and *Repeat*, as it can be easily checked.

Finally, let Σ be a finite alphabet and let $\mathcal{B}_\Sigma$ be the set $\{M_a : a \in \Sigma\}$, where M_a is the single-state RLA recognizing $a \in \Sigma$. We denote by $\mathcal{C}_\Sigma$ the class of all the RLA which can be obtained from $\mathcal{B}_\Sigma$ by applying the operations of *Concatenate*, *Iterate*, and *Repeat*. $\mathcal{C}_\Sigma$ is *properly* included in the class of all the RLA, that is, there exist some RLA, including size-optimal and complexity-optimal ones (e.g., the automaton M in Figure 3), which cannot be generated from automata in $\mathcal{B}_\Sigma$ by applying the operations of concatenation, iteration, and k-repetition. 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. This property can be used to prove that a complexity-optimal automaton for a given string can be generated by appropriately composing smaller (complexity-optimal) automata using the operators *Concatenate*, *Iterate*, and *Repeat*. Unfortunately, similar properties do not hold for the size of RLA.

6 Computing Complexity-optimal Automata

6.1 Sharing Free Automata

For every RLA $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, C_0)$ and for every pair of states r, s such that $(r, s) \in \delta^*$, let $\Delta^M_{r,s}$ denote the set $\{\delta^i(r) : 0 \leq i \leq n\}$, where n is the least natural number such that $s = \delta^n(r)$.

Definition 4. *Given an RLA* $M = (S_\Sigma, S_\varepsilon, \Sigma, \Omega, \delta, \gamma, s_0, C_0)$ *and a state* $s \in S_\varepsilon$*,* s *is said to be* sharing *if there is a state* $t \notin \Delta^M_{\gamma(s),s} \setminus \{s_0\}$ *such that the set* $\Delta^M_{t,s} \cap \Delta^M_{\gamma(s),s}$ *contains states other than* s *itself.* M *is sharing-free if* S_ε *does not contain sharing states.*

As a matter of fact, any automaton in $\mathcal{C}_\Sigma$ is sharing free. The following lemma shows that sharing states can be eliminated by replicating states, without increasing complexity.

Lemma 1. *For every RLA* M*, there exists an equivalent sharing-free RLA, denoted* *SharingFree*(M)*, such that* $\|M\| = \|$*SharingFree*$(M)\|$.

6.2 An Optimal Substructure Property

In this section, we prove that RLA satisfy the optimal substructure property. Lemma 1 implies that for any (finite or ultimately periodic) word $u \in \Sigma^\infty$, there is at least one sharing free complexity-optimal automaton M which recognizes u. In fact, we are going to show that we can choose M in such a way that it belongs to $\mathcal{C}_\Sigma$ and it is decomposable into complexity-optimal automata.

As a preliminary step, we characterize repeating patterns of words through the notions of period, partial period, and border.

Definition 5. *A word u has* period p *if there is a positive integer k such that $u = u[1,p]^k$.* The period *of u is the minimum period of u. By analogy we define* the prefix length *and* the period *of an ultimately periodic word u to be the integers l and q such that $u = u[1,l] \cdot u[l+1, l+q]^\omega$ and $l+q$ is minimum. Furthermore, p is said to be a* partial period *of u provided u is a prefix of $u[1,p]^\omega$. Finally a* border *of a finite word u is a non-empty string v different from u such that v is both a prefix and a suffix of u.*

It is worth noticing that $u[1,q]$ is a (maximum) border of u if and only if $p = |u| - q$ is a (minimum) partial period of u (see Figure 4).

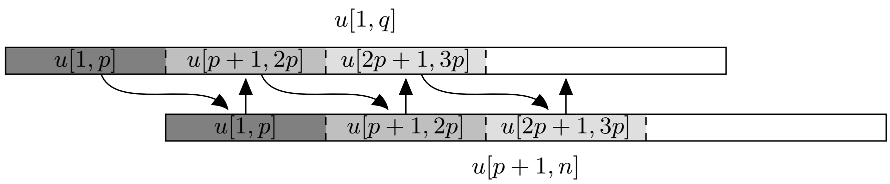

Fig. 4. Relationship between partial periods and borders.

The following two theorems state optimal substructure properties for finite and ultimately periodic words, respectively. Notice that both theorems provide only a *finite* number of ways to build a complexity-optimal automaton for u from (optimal) automata for substrings of u.

Theorem 1. *Given a finite word u such that $|u| > 1$, one of the following conditions holds:*

1. *for every pair of complexity-optimal automata M and N recognizing respectively the prefix $u[1]$ and the suffix $u[2,n]$ of u, Concatenate(M,N) is a complexity-optimal automaton recognizing u;*
2. *there exists an integer $r \in [1, |u|-1]$ such that whenever M and N are two complexity-optimal automata recognizing respectively the prefix $u[1,p]$ (with p being the period of $u[1,r]$) and the suffix $u[r+1,|u|]$ of u, then Concatenate(Repeat$(M, \frac{r}{p}), N)$ is a complexity-optimal automaton recognizing u.*

3. *for every complexity-optimal automaton M recognizing $u[1,p]$, with $p < |u|$ being the period of u, Repeat$(M, \frac{n}{p})$ is a complexity-optimal automaton recognizing u;*

Theorem 2. *Given an ultimately periodic word u with minimum prefix length l and minimum period q, one of the following conditions holds:*
1. *$l > 0$ and for every pair of complexity-optimal automata M and N recognizing respectively the prefix $u[1]$ and the suffix $u[2,\omega]$ of u, Concatenate(M,N) is a complexity-optimal automaton recognizing u;*
2. *$l > 0$ and there is an integer $r \in [1, 2l+2q]$ such that whenever M and N are two complexity-optimal automata recognizing respectively the prefix $u[1,p]$ (with p being the period of $u[1,r]$) and the suffix $u[r+1,\omega[$ of u, then Concatenate(Repeat$(M, \frac{r}{p}), N)$ is a complexity-optimal automaton recognizing u.*
3. *$l = 0$ and for every complexity-optimal automaton M recognizing $u[1,q]$, Iterate(M) is a complexity-optimal automaton recognizing u;*

Theorems 1 and 2 suggest a simple dynamic programming algorithm which, given a finite string u or a string-based representation of an ultimately periodic word u, computes in polynomial time a complexity-optimal RLA recognizing u. This algorithm heavily uses information on periods of all the substrings of u. For any finite string v (or any finite prefix v of a given ultimately periodical word), the periods of all the substrings of v can be efficiently computed in time $\Theta(|v|^2)$ by exploiting noticeable properties of periods and borders (the approach is somehow similar to the one used by Knuth, Morris, and Pratt in order to compute the prefix function of a pattern in the context of string-matching problems [8]). In particular, it turns out that the length $q(j)$ of the maximum border of $v[1,j]$ satisfies the equations $q(1) = 0$ and, for every $j > 1$, $q(j) = max(\{0\} \cup \{l : v[l] = v[j] \wedge l-1 \in q^+(j-1)\})$, where q^+ denotes the transitive closure of the function q. Since to each maximum border corresponds a minimum partial period, it turns out that the minimum partial periods of all the prefixes of v can be computed in linear time. The above mentioned bound easily follows.

7 Further Work

In this paper we gave a polynomial time algorithm that determines a complexity-optimal representation for RLA. We believe that such an algorithm can actually be improved, by exploiting subtle relationships between repeating patterns of strings and secondary transition functions of complexity-optimal RLA. As a matter of fact, we conjecture that loops of primary and secondary transition functions 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., $u[i-1,j]$.

Another interesting research direction is the development of an algorithm that efficiently solves the size-minimization problem. To this end, we conjecture that size-optimal automata can be built up from smaller components, as we did for complexity-optimal ones, via concatenation, repetition, iteration, and a new operator 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).

References

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

[2] 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 *Lecture Notes in Computer Science*, pages 554–566. Springer, 1994.

[3] C. Combi, M. Franceschet, and A. Peron. A logical approach to represent and reason about calendars. In *Proceedings of the 9th International Symposium on Temporal Representation and Reasoning*, pages 134–140. IEEE Computer Society Press, 2002.

[4] 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 *Lecture Notes in Computer Science*, pages 279–298. Springer, July 2001.

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

[6] U. Dal Lago, A. Montanari, and G. Puppis. Towards compact and tractable automaton-based representations of time granularities. Technical Report 17, Dipartimento di Matematica e Informatica, Università degli Studi di Udine, Italy, July 2003.

[7] 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, July/August 2000.

[8] D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6:323–350, 1977.

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

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

[11] 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.