Calendars, Time Granularities, and Automata

Ugo Dal Lago and Angelo Montanari

Dipartimento di Matematica e Informatica, Università di Udine
Via Delle Scienze 206, 33100 Udine, Italy
dallago@sci.uniud.it, montana@dimi.uniud.it

Abstract. 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 inter-operability, temporal constraint reasoning, data mining, and time management in workflow systems. According to a commonly accepted view, any *time granularity* can be viewed as the partitioning of a temporal domain in groups of elements, where each group is perceived as an indivisible unit (a granule). Most granularities of practical interest are modeled as infinite sequences of time granules, that present a repeating pattern and, possibly, temporal gaps within and between granules. Even though conceptually clean, this definition does not address the problem of providing infinite granularities with a finite representation to make it possible to deal with them in an effective way. In this paper, we present an automata-theoretic solution to such a problem that revises and extends the string-based model recently proposed by Wijsen [13]. We illustrate the basic features of our formalism and discuss its application to the fundamental problems of equivalence and classification of time granularities.

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 inter-operability, temporal constraint reasoning, data mining, and time management in workflow systems [6]. As an example, in a federated database system different databases may use different time granularities to store temporal information. When query processing involves pieces of information belonging to distinct databases, the system needs to integrate these different time granularities in a principled way. Such an integration presupposes the formalization of the mathematical relations between time granularities, a problem which is closely related to the classical problem of supporting calendars. Roughly speaking, a calendar is a human abstraction of time. Defining calendars corresponds to explicit time entities and relations which are relevant to specific problems to be solved. Calendars have intrinsic and extrinsic characteristics [11]. Intrinsic characteristics, such as the duration of time units and their relationships, define the semantics of a calendar, while extrinsic characteristics, such as the language in which time

© Springer-Verlag Berlin Heidelberg 2001

values are expressed and the format of time constants, vary depending on the orientation of the user. In this paper, we focus on representation methods that allow one to model intrinsic characteristics of calendars. Time granularity emerged as a formal tool to systematically deal with (intrinsic characteristics of) calendars in databases. According to a commonly accepted perspective [3], any *time granularity* can be viewed as the partitioning of a temporal domain in groups of elements, where each group is perceived as an indivisible unit (a granule). Most granularities of practical interest are modeled as infinite sequences of time granules, that present a repeating pattern and, possibly, temporal gaps within and between granules. Even though conceptually clean, this definition overlooks the problems involved in the implementation of time granularities in an actual computing system. In order to represent and deal with time granularity in an effective way, any formalism must/should satisfy the following requirements:

1. **Suitable to algorithmic manipulation.** The formalism must provide infinite granularities with a finite representation. Furthermore, data structures, which are used to actually store information, should ease access to and manipulation of time granularities.
2. **Powerful**. The set of all possible time granularities is not countable. Consequently, every representation formalism is bound to be incomplete. The class of granularities captured by the formalism should be large enough to be of practical interest.
3. **Compact.** The formalism should exploit regularities exhibited by the considered granularity to make their representation as compact as possible.

A number of representation formalisms for dealing with time granularities in databases and knowledge-based systems have been proposed in the literature (a comprehensive and an up-to-date survey can be found in [6]). The most significant ones are the formalism of collection expressions [7], that of slice expressions [8], and the Calendar Algebra [9]. All these formalisms make it possible to express granularities of practical interest, including infinite periodic granularities. They are based on the same idea: one can represent time granularities by means of symbolic terms built up from a finite set of basic granularities using a finite set of operators. The different sets of granularity operators provided by the three formal systems and their relationships are investigated in [2], where it is proved that Calendar Algebra actually subsumes the other two formalisms. A common limitation of all these approaches is that they focus on expressiveness issues, and almost completely ignore the problem of establishing whether or not the proposed formal systems are suitable for direct algorithmic manipulation.

A new approach to the representation and manipulation of infinite time granularities has been recently proposed by Wijsen [13]. According to such an approach, infinite granularities are modeled as infinite strings over a suitable finite alphabet. Furthermore, whenever the granularity is (ultimately) periodic, it can be finitely represented as an ordered pair, whose first element is a finite prefix, called offset, and the second element is the finite repeating pattern (obviously, this solution produces lengthy descriptions whenever the periodic granularity to be represented has a long period and/or prefix). The resulting string-based model is then used to formally state and solve the problems of granularity equivalence and minimality.

In this paper, we present an automata-theoretic solution to the problem of representing and manipulating time granularities that revises and extends Wijsen's one. We define a new class of automata, called *single-string automata*, and we explain how they can be used to model time granularities. In particular, we show how regularities of the modeled granularity can be exploited to make descriptions more compact. Then, we show how to extend single-string automata to deal with granularities which have a quasi-periodic structure. Finally, we concentrate on the problems of equivalence and classification of time granularities. The *equivalence* problem is the problem of deciding whether two different representations define the same time granularity. Studying decidability and complexity of this problem is quite important in several respects. As an example, the decidability of the equivalence problem directly implies the possibility of effectively testing the semantic equivalence of two different time granularity representations, making it possible to use smaller, and more tractable, representations in place of bigger ones. The *classification* problem is the problem of relating the granules of a given granularity to those of another one. It can be viewed as the counterpart of the granularity conversion problem as defined in [6]; however, since single-string automata allow us to keep track of the internal structure of granules, we will be able to establish finer connections between granules belonging to different granularities. We give two algorithms that respectively solve the equivalence and classification problems, and we analyze their performance.

The rest of the paper is organized as follows. In Section 2 we briefly summarize the main features of the string-based model for infinite time granularities. We introduce single-string automata in Section 3 and refine them in Section 4. The application of the proposed formalism to deal with granularities of practical interest is discussed in Section 5. Finally, in Section 6, we present the equivalence and classification algorithms. Conclusions provide an assessment of the work and outline future research directions.

2 A String-Based Model for Infinite Time Granularities

In this section, we first introduce a simple formalization of the notion of time granularity and then present the basic characteristics of the string-based model for infinite time granularities proposed by Wijsen [13].

We assume the temporal domain to be (isomorphic to) the set of natural numbers $\mathbb{N}$, with the usual ordering relation $<$. Some of the granularity systems proposed in the literature assume $\mathbb{Z}$, instead of $\mathbb{N}$, as the temporal domain. This choice makes it possible to avoid the introduction of a first granule and to move backward and forward with respect to the reference granule. Most time granularity applications, however, restrict themselves to the case of natural numbers, e.g., this is the case of satisfiability checking algorithms for quantitative temporal constraints with multiple granularities described in [4]. Furthermore, it is not difficult to show that the automata-theoretic formalization of time granularity can be generalized to integer numbers by exploiting well-known results in the field [10]. According to Wijsen, we further assume that any given granularity groups into another one. In fact, we view a granularity as a formal tool defining how two temporal domains, both isomorphic to $\mathbb{N}$, group one into the other. This approach can be easily characterized in terms of *granularity systems* as defined in [6].

Given the temporal domain $\langle \mathbb{N}, < \rangle$, a *time granularity* (or simply a *granularity*) is an equivalence relation $\sim$ on $N \subseteq \mathbb{N}$ such that, for every pair C_1, C_2 of $\sim$-classes, either it holds that, for all $i \in C_1$ and $j \in C_2$, $i < j$ or it holds that, for all $i \in C_1$ and $j \in C_2$, $j < i$ [13]. N is called the (set of) *support* of the time granularity $\sim$. It is immediate to see that $<$ naturally induces a linear order on the set of $\sim$-classes. Unlike other formalizations of time granularity [6], this definition does not introduce any notion of index set; however, such a notion can be easily derived from it, as shown in [13].

In order to deal with time granularities in an effective way, one needs to provide them with a *finite* representation which is as compact as possible and captures a reasonably large class of granularities. Furthermore, a suitable representation formalism must guarantee that the complexity of the algorithms for granularity manipulation remains under control. We restrict ourselves to the case in which the support N is infinite. If N is finite, indeed, the solution to the representation problem is trivial. In [13], Wijsen proposes a string-based model for infinite time granularities that allows one to represent in a compact way a broad class of granularities, which includes almost all granularities of practical interest. In the following, we briefly describe the main features of such a string-based model.

We assume the reader to be familiar with the basic terminology on finite and infinite strings (if this is not the case, a good reference is [12]). Let v be an infinite string. We will often write v as $v(1)v(2)v(3)\ldots$, where $v(i)$ is the i-th element of the string. In order to model time granularities, Wijsen introduces an alphabet of three symbols $\Sigma = \{\blacksquare, \square, \wr\}$. The symbols $\blacksquare$, $\square$, and $\wr$ are respectively called *filler*, *gap*, and *separator*. Furthermore, we alternatively refer to both $\blacksquare$ and $\square$ as *placeholders*. Let Σ^* (resp. Σ^ω) be the set of all finite (resp. infinite) strings over Σ. Furthermore, let $\Sigma^+ = \Sigma^* - \{\epsilon\}$, where ϵ is the empty string. Given $t \in \Sigma^\omega$, we denote by $\square(t,k)$ the number of occurrences of placeholders in the first k positions of t. Any infinite string $t \in \Sigma^\omega$, with an infinite number of placeholders, naturally induces a time granularity, that we denote by $gran(t)$. The granularity $gran(t)$ can be formally defined as follows: for all $i, j \in \mathbb{N}$, $(i,j) \in gran(t)$ if and only if there exist two integers k and l such that (i) $t(k) = t(l) = \blacksquare$; (ii) $k - \square(t,k) = l - \square(t,l)$; (iii) $\square(t,k) = i$ and $\square(t,l) = j$. Condition (*i*) states that a filler occurs at positions k and l; condition (*ii*) imposes that there are not separators between $t(k)$ and $t(l)$; condition (*iii*) associates string positions with the corresponding time points by neglecting all the occurrences of the separator. As an example, the string $v = \blacksquare\ \blacksquare\ \wr\ \square\ \blacksquare\ \blacksquare\ \wr\ \square\ \blacksquare\ \blacksquare\ \wr\ \ldots$ represents the granularity $gran(v) = \{\{1,2\},\{4,5\},\{7,8\},\ldots\}$.

In order to finitely model granularities, Wijsen introduces the notion of *granspec*: a granspec is a pair (u, w), with $u \in \Sigma^*$ and $w \in \Sigma^+$. The strings u and v are respectively called the *offset* and the *pattern* of (u, w). Furthermore, we say that uw^ω is the *trace* produced by (u, w); sometimes, we will indicate uw^ω as $(u, w)^\infty$. Two granspecs (u_1, w_1) and (u_2, w_2) are said to be trace-equivalent, denoted by $(u_1, w_1) \equiv_T (u_2, w_2)$, if and only if $u_1 w_1^\omega = u_2 w_2^\omega$. It is easy to check that $(u_1, w_1) \equiv_T (u_2, w_2)$ implies $gran(u_1 w_1^\omega) = gran(u_2 w_2^\omega)$, but not vice versa.

A time granularity is said to be *regular* if it is of the form $gran((u, w)^\infty)$, where (u, w) is a granspec, that is, regular time granularities are represented by ultimately periodic infinite strings, which can be expressed by means of pairs of finite strings. It is not difficult to show that for every ultimately periodic string $t \in \Sigma^\omega$, there exist infinitely many granspecs (u, w) such that $t = (u, w)^\infty$, and hence, from $(u_1, w_1) \equiv_T (u_2, w_2)$, it does not follow that $(u_1, w_1) = (u_2, w_2)$. This allows us to conclude that, while the equality of two granspecs obviously implies the equality of the corresponding regular granularities, the vice versa does not hold, that is, $(u_1, w_1) = (u_2, w_2)$ implies that $gran((u_1, w_1)^\infty) = gran((u_2, w_2)^\infty)$, but $gran((u_1, w_1)^\infty) = gran((u_2, w_2)^\infty)$ does not imply that $(u_1, w_1) = (u_2, w_2)$.

Finally, two granspecs (u_1, w_1) and (u_2, w_2) are said to be *G-equivalent*, denoted by $(u_1, w_1) \equiv_G (u_2, w_2)$, if and only if $gran(u_1 w_1^\omega) = gran(u_2 w_2^\omega)$. From the above arguments, we have that the G-equivalence of two granspecs (u_1, w_1) and (u_2, w_2) cannot be reduced to the equality (u_1, w_1) and (u_2, w_2). In [13], Wijsen proposes an algorithm to determine whether or not two given granspecs are G-equivalent, which is based on a suitable *canonical form* for granspecs. This canonical form is obtained in two steps: in the first step, we select the subset of granspecs, called *aligned* granspecs, such that, in the produced trace, every separator is directly preceded by a filler; in the second step, *canonical* granspecs are defined as a proper subclass of aligned ones.

A granspec (u, w) is said to be aligned if and only if it satisfies the following conditions: (i) for every $i \in \mathbb{N}$, if $(u, w)^\infty(i) = \wr$, then there exists $j > i$ $(u, w)^\infty(j) = \blacksquare$; (ii) $(u, w)^\infty(1) \neq \wr$; (iii) for every $i \in \mathbb{N}$, with $i > 1$, if $(u, w)^\infty(i) = \wr$, then $(u, w)^\infty(i - 1) = \blacksquare$. A precise characterization of aligned granspecs is provided by the following proposition:

Proposition 1. *A granspec (u, w) is aligned if and only if the following five statements hold:*

1. *$u(1) \neq \wr$;*
2. *for every i such that $1 < i \leq |u|$, if $u(i) = \wr$, then $u(i - 1) = \blacksquare$;*
3. *if $w(1) = \wr$, then $u \neq \varepsilon$ and the last symbol of u and w is $\blacksquare$;*
4. *for every i such that $1 < i \leq |w|$, if $w(i) = \wr$, then $w(i - 1) = \blacksquare$;*
5. *if $w = \square^i$ for some $i > 0$, then $u \neq u' \wr \square^j$ for every u' and $j \geq 0$.*

The following theorem guarantees that any granspec can be turned into a granspec that satisfies the alignment conditions.

Theorem 1. *For every granspec (u_1, w_1), there exists an aligned granspec (u_2, w_2) such that $(u_1, w_1) \equiv_G (u_2, w_2)$.*

The next theorem states the equivalence of $\equiv_T$ and $\equiv_G$ for aligned granspecs.

Theorem 2. *Let (u_1, w_1) and (u_2, w_2) be two aligned granspecs. If $(u_1, w_1) \equiv_G (u_2, w_2)$, then $(u_1, w_1) \equiv_T (u_2, w_2)$.*

Since two aligned granspecs can be distinct even if they are G-equivalent, a further normalization step is needed. A granspec (u, w) is said to be *canonical* if and only if it satisfies the following conditions: (i) it is aligned; (ii) if $w = v^k$ for a finite string v, then $w = v$ and $k = 1$; (iii) if $u \neq \varepsilon$, then the last symbol of u is distinct from the last symbol of w. The following theorem guarantees that a canonical form can always be achieved.

Theorem 3. *For every granspec (u_1, w_1), there is a canonical granspec (u_2, w_2) such that $(u_1, w_1) \equiv_G (u_2, w_2)$.*

The last theorem states the desired equivalence of $=$ and $\equiv_G$ for canonical granspecs.

Theorem 4. *Let (u_1, w_1) and (u_2, w_2) be two canonical granspecs. If $(u_1, w_1) \equiv_G (u_2, w_2)$, then $(u_1, w_1) = (u_2, w_2)$.*

In the next section, we introduce an alternative automata-based model for (ultimately) periodic time granularities and we prove that it is as expressive as the string-based one, often producing more compact representations.

3 Single-String Automata

A (deterministic) single-string automaton is a quadruple (S, Σ, δ, q_0), where S is a finite set whose elements are called *states*, Σ is a finite *alphabet*, δ is a (total) *transition function* from S to $\Sigma \times S$, and $q_0 \in S$ is said to be the *initial state*. Given a single-string automaton $M = (S, \Sigma, \delta, q_0)$, we say that $(s, t) \in S^\omega \times \Sigma^\omega$ is a *run* of M if and only if $s(1) = q_0$ and $\forall i \geq 1 \;\; \delta(s(i)) = (t(i), s(i+1))$. The run of M is uniquely identified among the set of pairs in $S^\omega \times \Sigma^\omega$; this immediately follows from the fact that δ is a single-valued function. Given a single-string automaton M and its run (s, t), we say that t is the string *generated by M* and we write $t = M^\infty$. Unlike usual classes of automata, single-string automata generate only one string and this does not cause any gain in expressivity, as stated by the following result, whose easy proof is left to the reader.

Proposition 2. *Let M be a single-string automaton and let $v \in \Sigma^\omega$ be the string generated by M. Then there is a Büchi automaton M' that generates $\{v\}$.*

The structure of strings generated by single-string automata is very simple, as stated by the following theorem.

Theorem 5. *Let Σ be an alphabet and let $v \in \Sigma^\omega$. Then v is ultimately periodic if and only if v is the string generated by a single-string automaton.*

Proof. Let $v \in \Sigma^\omega$ be an ultimately periodic string. Then we can write v as uw^ω, for some $u \in \Sigma^*$ and $v \in \Sigma^+$. Suppose $u = a_1a_2 \dots a_n$ and $w = b_1b_2 \dots b_m$. We can build a single-string automaton $M = (S, \Sigma, \delta, q_0)$ that generates exactly v as follows:

– $S = \{(1,i) | i \in [1,n]\} \cup \{(2,i) | i \in [1,m]\}$
– $\delta : S \to \Sigma \times S$ is defined letting

$$\delta((k,i)) = \begin{cases} (a_i, (1, i+1)) \;\; se \;\; i \leq n-1 \;\; e \;\; k=1 \\ (a_n, (2,1)) \;\; se \;\; i = n \;\; e \;\; k=1 \\ (b_i, (2, i+1)) \;\; se \;\; i \leq m-1 \;\; e \;\; k=2 \\ (b_n, (2,1)) \;\; se \;\; i = m \;\; e \;\; k=2 \end{cases}$$

– $q_0 = (1,1)$, if $u \neq \varepsilon$, and $q_0 = (2,1)$, otherwise.

We can easily verify that the string generated by M is v.

Conversely, let $M = (S, \Sigma, \delta, q_0)$ be a single-string automaton, with $S = \{q_0, q_1, \dots, q_m\}$ and $\Sigma = \{d_1, d_2, \dots, d_n\}$, and let (s,t) be the run of M. First, we can observe that s is an infinite string made up from a finite alphabet. Then, let $\{i_n\}_{n\in\mathbb{N}}$ be a sequence such that $s = q_{i_1}q_{i_2}q_{i_3}\dots$ and let q_{i_j} be a state that appears in s more than once, that is, $s = q_{i_1} \dots q_{i_j} \dots q_{i_{k-1}} q_{i_j} q_{i_{k+1}} \dots$. From the definition of δ, it follows that $s = q_{i_1} \dots q_{i_{j-1}} (q_{i_j} \dots q_{i_{k-1}})^\omega$. Let $\{l_n\}_{n\in\mathbb{N}}$ be a sequence such that $t = d_{l_1}d_{l_2}d_{l_3}\dots$. It is immediate to verify that $t = d_{l_1} \dots d_{l_{j-1}} (d_{l_j} \dots d_{l_{k-1}})^\omega$, which is exactly the thesis. □

3.1 Equivalence and Minimality of Single-String Automata

We say that two single-string automata are equivalent if and only if they generate the same string. We now outline an algorithm (Algorithm 3.1) to decide whether two given single-string automata are equivalent or not. The proof of Theorem 5 suggests us an effective procedure to build up two strings u and v such that uw^ω is the string generated by a given single-string automaton M. The equivalence of two single-string automata can thus be reduced to the equality of two ultimately periodic strings. The correctness of Algorithm 3.1 rests on the following pair of propositions (the trivial proof of Proposition 3 is left to the reader).

Proposition 3. *Let $v_1 = u_1w_1^\omega$ be an ultimately periodic string, where $w_1 = b_1 \dots b_n$. Furthermore, let $u_2 = u_1b_1$, $w_2 = b_2 \dots b_nb_1$, and $v_2 = u_2w_2^\omega$. It holds that $v_1 = v_2$.*

Proposition 4. *Let $v_1 = u_1w_1^\omega$ and $v_2 = u_2w_2^\omega$ be two ultimately periodic strings such that $|u_1| = |u_2|$. Then $v_1 = v_2$ if and only if*

$$u_1 = u_2 \qquad w_1^{\frac{n}{|w_1|}} = w_2^{\frac{n}{|w_2|}},$$

where n is the least common multiple of $|w_1|$ and $|w_2|$.

Proof. Let us assume that $v_1 = u_1w_1^\omega$, $v_2 = u_2w_2^\omega$, $|u_1| = |u_2|$, and $v_1 = v_2$. The way in which v_1 and v_2 are defined and the hypothesis $|u_1| = |u_2|$ together imply that $u_1 = u_2$. Furthermore, from the definition of n, we can conclude that $|w_1^{\frac{n}{|w_1|}}| = |w_2^{\frac{n}{|w_2|}}|$. From $w_1^{\frac{n}{|w_1|}} \neq w_2^{\frac{n}{|w_2|}}$, indeed, it would follow that $u_1 w_1^{\frac{n}{|w_1|}} \neq u_2 w_2^{\frac{n}{|w_2|}}$, that is, v_1 and v_2 would differ in their prefix of length $|u_1| + n$, and thus $v_1 \neq v_2$ (contradiction).

Conversely, let us assume that $v_1 = u_1 w_1^\omega$, $v_2 = u_2 w_2^\omega$, $u_1 = u_2$, and $w_1^{\frac{n}{|w_1|}} = w_2^{\frac{n}{|w_2|}}$. It immediately follows that

$$v_1 = u_1 w_1^\omega = u_2 w_1^\omega = u_2 \left(w_1^{\frac{n}{|w_1|}}\right)^\omega = u_2 \left(w_2^{\frac{n}{|w_2|}}\right)^\omega = u_2 w_2^\omega = v_2,$$

which is the thesis. □

Proposition 4 can be reformulated in a different way, taking into account the internal structure of strings to be tested for equality.

Proposition 5. *Let $v_1 = u_1 w_1^\omega$ and $v_2 = u_2 w_2^\omega$ be two ultimately periodic strings such that $|u_1| = |u_2|$. Then $v_1 = v_2$ if and only if*

$$u_1 = u_2 \qquad w_1 = s^{\frac{|w_1|}{n}} \qquad w_2 = s^{\frac{|w_2|}{n}},$$

where n is the greater common divisor of $|w_1|$ and $|w_2|$ and s is a string.

Proof. If $v_1 = u_1 w_1^\omega$, $v_2 = u_2 w_2^\omega$, $u_1 = u_2$, $v_1 = s^{\frac{|w_1|}{n}}$ and $v_2 = s^{\frac{|w_2|}{n}}$, then $v_1 = u_1 s^\omega = u_2 s^\omega = v_2$. Conversely, let us assume that $v_1 = v_2$; from $|u_1| = |u_2|$ it follows that $u_1 = u_2$. We can obviously write $w_1 = s_1 s_2 \ldots s_{\frac{|w_1|}{n}}$ and $w_2 = t_1 t_2 \ldots t_{\frac{|w_2|}{n}}$, where $|t_i| = |s_i| = n$ for each common index i. The integers $\frac{|w_1|}{n}$ and $\frac{|w_2|}{n}$ are relative prime and thus, using well-known results about finite fields, we have that

$$\left\{0, \ldots, \tfrac{|w_1|}{n} - 1\right\} = \left\{\left(\tfrac{|w_2|}{n} m\right) \ mod \ \tfrac{|w_1|}{n} \mid 0 \leq m < \tfrac{|w_1|}{n}\right\};$$
$$\left\{0, \ldots, \tfrac{|w_2|}{n} - 1\right\} = \left\{\left(\tfrac{|w_1|}{n} m\right) \ mod \ \tfrac{|w_2|}{n} \mid 0 \leq m < \tfrac{|w_2|}{n}\right\}.$$

This concludes the proof, because these relations, together with the fact that $w_1^{\frac{|w_2|}{n}} = w_2^{\frac{|w_1|}{n}}$, imply that $s_i = t_j$, for any pair i, j, and thus $w_1 = s^{\frac{|w_1|}{n}}$ and $w_2 = s^{\frac{|w_2|}{n}}$. □

As for minimality, it can be reduced to the minimality of the representation of the generated string as well. An algorithm for determining such a minimal representation can be easily obtained by exploiting the following pair of propositions (code omitted).

Proposition 6. *Let $v_1 = u_1 w_1^\omega$ be an ultimately periodic string, where $|u_1| \geq 1$, $u_1 = b_1, \ldots, b_n$, $w_1 = a_1 \ldots a_m$, and $b_n = a_m$. Furthermore, let $u_2 = b_1, \ldots, b_{n-1}$, $w_2 = b_n, a_1, \ldots a_{m-1}$, and $v_2 = u_2 w_2^\omega$. It holds that $v_1 = v_2$.*

Proposition 7. *Let $v = u w_1^\omega$, with $u = b_1, \ldots, b_n$ and $w_1 = a_1 \ldots a_m$, be an ultimately periodic string such that (if $|u| > 0$) $b_n \neq a_m$. Then, there exists a (unique) prefix w_2 of w_1 such that $w_1 = w_2^k$, with $k \geq 1$, and there exists no proper prefix w_3 of w_2 such that $w_1 = w_3^h$, with $h > k$.*

Algorithm 3.1 Determine if two ultimately periodic strings are equal on the basis of their finite representations (U_1, W_1) and (U_2, W_2).

```
1: if $|U_1| \geq |U_2|$ then
2:    for $i \leftarrow 1$ to $|U_1| - |U_2|$ do
3:       $U_2 \leftarrow U_2 W_2[1]$
4:       $W_2 \leftarrow W_2[2, \ldots, |W_2|] W_2[1]$
5: else
6:    for $i \leftarrow$ to $|U_2| - |U_1|$ do
7:       $U_1 \leftarrow U_1 W_1[1]$
8:       $W_1 \leftarrow W_1[2, \ldots, |W_1|] W_1[1]$
9: if $U_1 \neq U_2$ then
10:    print("Input strings are not equal.")
11: else
12:    for $i \leftarrow 1$ to $LCM(|W_1|, |W_2|)$ do
13:       if $W_1[mod_{|W_1|}(i-1)+1] \neq W_2[mod_{|W_2|}(i-1)+1]$ then
14:          print("Input strings are not equal.")
15:    print("Input strings are equal.")
```

3.2 Granularities and Single-String Automata

In this section, we show how single-string automata can be used to model (ultimately periodic) time granularities. Let $\Sigma = \{\blacksquare, \blacktriangleleft, \square\}$. Any infinite string $t \in \Sigma^\omega$ naturally induces a time granularity, that we denote by $gran'(t)$, such that for all $i, j \in \mathbb{N}$, $(i, j) \in gran'(t)$ if and only if the following pair of conditions holds: (i) $t(i), t(j) \in \{\blacksquare, \blacktriangleleft\}$; (ii) $t(k) \neq \blacktriangleleft$ for any $k \in [m, M-1]$, where $m = \min\{i, j\}$ and $M = \max\{i, j\}$. As an example, the string $v = \blacksquare \blacktriangleleft \square \blacksquare \blacktriangleleft \square \blacksquare \blacktriangleleft \ldots$ represents the granularity $gran'(v) = \{\{1, 2\}, \{4, 5\}, \{7, 8\}, \ldots\}$. This formalism turns out to be simpler than the one described in Section 2, as shown by the following proposition.

Proposition 8. *Let $\Sigma = \{\blacksquare, \blacktriangleleft, \square\}$ and $v_1, v_2 \in \Sigma^\omega$. If both v_1 and v_2 cannot be written as $u\square^\omega$, with $u \in \Sigma^*$, it holds that $gran'(v_1) = gran'(v_2)$ if and only if $v_1 = v_2$.*

Proof. Let us assume that neither v_1 nor v_2 is equal to $u\square^\omega$, with $u \in \Sigma^*$, and that $gran'(v_1) = gran'(v_2)$. Further, we assume that $v_1 \neq v_2$ and that j is the least integer such that $v_1(j) \neq v_2(j)$. We distinguish three relevant cases:

- If either $v_1(j)$ or $v_2(j)$ is $\square$, then we have a contradiction: it suffices to notice that if we replace $\square$ by $\blacksquare$ or $\blacktriangleleft$ in any string $v \in \Sigma^\omega$, $gran'(v)$ changes.
- If $v_1(j)$ and $v_2(j)$ are $\blacksquare$ and $\blacktriangleleft$, respectively, then we can proceed as follows: let j' be the least integer greater than j such that at least one symbol between $v_1(j')$ and $v_2(j')$ is different from $\square$. Checking every possible value for $v_1(j')$ and $v_2(j')$, we immediately obtain a contradiction.
- If $v_1(j)$ and $v_2(j)$ are $\blacktriangleleft$ and $\blacksquare$, respectively, then we can proceed as in the previous case.

Conversely, if $v_1, v_2 \in \Sigma^\omega$ and $v_1 = v_2$, then it trivially holds that $gran'(v_1) = gran'(v_2)$. $\square$

The string M^∞ generated by a single-string automaton M on the alphabet $\Sigma = \{\blacksquare, \blacktriangleleft, \square\}$ naturally induces a granularity. A granularity R such that $R = \mathit{gran}'(M^\infty)$, where M is a single-string automaton, is said to be *auto-regular*. Notice that the hypothesis of Proposition 8 that neither v_1 nor v_2 is equal to $u\square^\omega$ is not restrictive, because we assume the set of support of granularities to be infinite.

The following theorem relates auto-regular granularities to regular ones (cf. Section 2).

Theorem 6. *The class of regular granularities and the class of auto-regular granularities coincide.*

Proof. Let $\mathit{gran}((u_1, w_1)^\infty)$, with $u_1 \in \{\blacksquare, \square, \wr\}^*$ and $w_1 \in \{\blacksquare, \square, \wr\}^+$, be a regular granularity. The corresponding pair of strings $u_2 \in \{\blacksquare, \blacktriangleleft, \square\}^*$ and $w_2 \in \{\blacksquare, \blacktriangleleft, \square\}^*$ can be obtained as follows.

- We obtain u_2 from u_1 and w_1 by executing the following two steps:
  1. For each occurrence of $\blacksquare$ in u_1, we scan the subsequent positions in u_1w_1 until we reach an occurrence of either $\blacksquare$ or $\wr$ (if any); then:
     - if we stop on the symbol $\blacksquare$, both occurrences of $\blacksquare$ are left unchanged;
     - if we stop on the symbol $\wr$, we substitute $\blacktriangleleft$ for $\blacksquare$;
     - if neither an occurrence of $\blacksquare$ nor an occurrence of $\wr$ can be found after the given occurrence of $\blacksquare$, then we substitute $\blacktriangleleft$ for $\blacksquare$;
  2. We erase every occurrence of $\wr$ in u_1.
- We obtain w_2 from w_1 by executing the same sequence of steps. In this case, however, we consider the string w_1w_1 instead of the string u_1w_1 when searching for symbols following any given occurrence of $\blacksquare$.

It is easy to see that such a transformation preserves the generated granularity, that is, $\mathit{gran}((u_1, w_1)^\infty) = \mathit{gran}'((u_2, w_2)^\infty)$. From Theorem 5, it follows that there exists a single-string automaton M such that $M^\infty = (u_2, w_2)^\infty$, and thus $\mathit{gran}((u_1, w_1)^\infty) = \mathit{gran}'(M^\infty)$.

Conversely, suppose that $\mathit{gran}'(M^\infty)$ is an auto-regular granularity. From Theorem 5, it follows that M^∞ is an ultimately periodic string on $\{\blacksquare, \blacktriangleleft, \square\}$). Let $(u_1, w_1)^\infty$ be such a string. We obtain the corresponding strings $u_2 \in \{\blacksquare, \square, \wr\}^*$ and $w_2 \in \{\blacksquare, \square, \wr\}^+$ by substituting the string $\blacksquare\wr$ for every occurrence of $\blacktriangleleft$ in u_1 e w_1. It is not difficult to show that $\mathit{gran}'(M^\infty) = \mathit{gran}((u_2, w_2)^\infty)$. $\square$

4 Single-String Automata Can Be Improved

Single-string automata can be naturally represented as graphs. As an example, the single-string automaton $(\{q_0, \ldots, q_4\}, \{a, b\}, \{(q_0, (a, q_1)), (q_1, (b, q_2)), (q_2, (a, q_3)), (q_3, (b, q_4)), (q_4, (a, q_1))\}, q_0)$ can be represented as depicted in Fig. 1. As another example, let $\Sigma = \{a, b\}$. A single-string automaton that generates the infinite string $(b(ab)^5)^\omega$ is the automaton modeled by the graph depicted in Fig. 2. The number of states of the graph is really too large with respect to the inherently simple structure of the generated string. In order to reduce the size of the automaton/graph, one can think of adding "priority" transitions that are

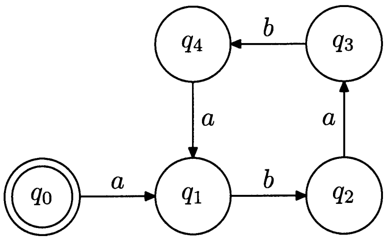


Fig. 1. A sample single-string automaton.

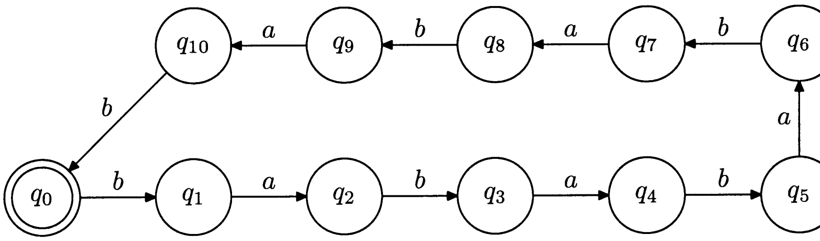


Fig. 2. A single-string automaton generating $(b(ab)^5)^\omega$.

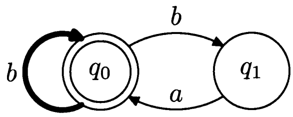


Fig. 3. A compact automaton for $(b(ab)^5)^\omega$.

taken only when particular conditions hold. The structure of the resulting automaton is depicted in Fig. 3, where the priority transition is the bold one. In the following, we will extend the notion of single-string automata by systematically exploring such an idea.

4.1 Extended Single-String Automata

Let A and B be two sets, $f : A \rightarrow A \times B$, and $\sim$ be an equivalence relation on A. We say that $\sim$ *respects* f if and only if, for all $a_1, a_2 \in A$, if $f(a_1) = (c_1, b_1)$, $f(a_2) = (c_2, b_2)$, and $a_1 \sim a_2$, then $c_1 \sim c_2$ and $b_1 = b_2$.

Furthermore, let I be a finite set of variables. We denote by L_I any logical language satisfying the following two conditions: (i) free variables of every formula in L_I belong to I, and (ii) functional and relational symbols in L_I have a natural interpretation in $\mathbb{N}$. Models of L_I formulas can be thought of as elements of $\mathbb{N}^n$, with $n = |I|$.

An *extended single-string automaton* is a 7-tuple $(S, I, \Sigma, \gamma, \delta, q_0, n_0)$, where S is a finite set of *states*; I is a finite set of *variables* ($\mathbb{N}^{|I|}$ will be denoted as C);

Σ is an *alphabet*; $\gamma : S \rightharpoonup L_I \times C^C \times \Sigma \times S$ is the (partial) *primary transition function*; $\delta : S \to C^C \times \Sigma \times S$ is the (total) *secondary transition function*; $q_0 \in S$ is the *initial state*; $n_0 \in C$. The run of an extended single-string automaton is unique, as for single-string automata, but its definition turns out to be more complex. Given an extended single-string automaton $M = (S, I, \Sigma, \gamma, \delta, q_0, n_0)$, we say that $(s, c, t) \in S^\omega \times C^\omega \times \Sigma^\omega$ is a *run* of M if and only if

- $s(1) = q_0$;
- $c(1) = n_0$;
- for all $i \geq 1$
  - if $\gamma(s(i)) = (\phi, \psi, h, k)$ and $c(i) \models \phi$, then the primary transition is taken and the following three conditions hold:

    $$c(i+1) = \psi(c(i)), \quad s(i+1) = k, \quad t(i) = h;$$

  - otherwise, the secondary transition $\delta(s(i)) = (\psi, h, k)$ is taken and the following three conditions hold:

    $$c(i+1) = \psi(c(i)), \quad s(i+1) = k, \quad t(i) = h.$$

Given an extended single-string automaton M and its run (s, c, t), we say that t is the string *generated* by M, and we write $t = M^\infty$.

As an example, the extended single-string automaton that generates the infinite string $(b(ab)^5)^\omega$ is depicted in Fig. 4, where bold arrows represent the primary transition function and thin arrows represent the secondary transition function. Moreover, n_0 is (0).

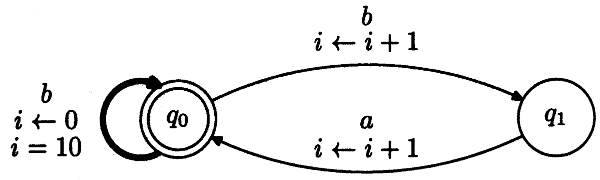


Fig. 4. An extended single-string automaton generating $(b(ab)^5)^\omega$.

Extended single-string automata do not satisfy all the properties that single-string automata satisfy, even though they can be viewed as a natural extension to single-string automata. In particular, the equivalence of two extended single-string automata is not decidable, that is, it is impossible to give an algorithm that states whether or not two extended single-string automata generate the same string. Moreover, it is not clear which is the class of strings which is captured by extended single-string automata. These problems are caused by our definition of extended single-string automata, which is very general and not restrictive at all. Extended single-string automata, however, allow us to solve the representation problem which has been discussed at the beginning of this section, provided that we find a finitary representation for an expressive enough subset of C^C. This is what we will do in the next section.

4.2 Reducible Extended Single-String Automata

In this section, we impose some restrictions on the definition of extended single-string automata in order to obtain a number of important decidability and characterization results. First, notice that every extended single-string automaton $M = (S, I, \Sigma, \gamma, \delta, q_0, n_0)$ is endowed with a finite number of states. However, it is clear that we implicitly ask the automaton to keep track of values of variables in I. More precisely, given an extended single-string automaton M and its run (s, c, t), we can define a function $\mu : S \times C \to S \times C \times \Sigma$ such that, for every $i \geq 0$, the following condition holds:

$$\mu(s(i), c(i)) = (s(i+1), c(i+1), t(i)).$$

We say that an extended single-string automaton M is *reducible* if we can define an equivalence relation $\sim$ of finite index on $S \times C$ that respects μ. The following theorem characterizes the expressiveness of reducible extended single-string automata.

Theorem 7. *If v is the string generated by a reducible extended single-string automata M, then there exists a single-string automaton M' that generates exactly v.*

Proof. Let $M = (S, I, \Sigma, \delta, \gamma, q_0, n_0)$ be a reducible extended single-string automata, let $C = \mathbb{N}^{|I|}$, and let $\mu : S \times C \to S \times C \times \Sigma$ be the function associated with M. By definition, there exists an equivalence relation $\sim$ of finite index on $S \times C$ that respects μ. We can build an equivalent single-string automaton $M' = (S', \Sigma, \delta', q_0')$ as follows:
- $S' = (S \times C)/\sim$. Since the index of $\sim$ is finite, S' is finite.
- If $\mu(x, y) = (x', y', h)$, then $\delta'([(x, y)]_\sim) = ([(x', y')]_\sim, h)$. The transition function δ' is well defined, because $\sim$ respects μ.
- $q_0' = [(q_0, n_0)]_\sim$.

Let us show that M and M' generate the same infinite string. Let us suppose that this is not the case, that is, let v and v' be the strings generated by M e M', respectively, and let i be the least natural number such that $v(i) \neq v'(i)$. If $i = 1$, we easily obtain a contradiction from the definitions of μ and q_0'. Otherwise $(i > 1)$, we have a contradiction noting that $v(i-1) = v'(i-1)$ and $\sim$ respects μ. Hence, we can conclude that $v = v'$, which is the thesis. □

Theorem 7 provides a precise characterization of the expressiveness of reducible extended single-string automata by showing that they are as powerful as single-string automata.

Let us denote by ξ a function that, given a reducible extended single-string automaton, returns an equivalent single-string automata. Obviously, ξ is not a computable function and, as a consequence, we cannot reduce the equivalence of two reducible extended single-string automata to the equivalence of two single-string automata. To make ξ computable, we have to impose further restrictions. This argument is summarized by the following theorem.

Theorem 8. *Let $\mathcal{C}$ be a set of reducible extended single-string automata such that $\xi|_{\mathcal{C}}$ is computable. Then, the equivalence of two automata in $\mathcal{C}$ is decidable.*

5 A Practical Formalism for Time Granularity

In this section, we define a class $\mathcal{D}$ of reducible extended single-string automata and we show that it satisfies the conditions of Theorem 8. We then show that $\mathcal{D}$ allows us to represent time granularities related to actual calendars in a compact way. In the next section, we will provide equivalence and classification algorithms on $\mathcal{D}$ and analyze their performance.

The class $\mathcal{D}$ is obtained by imposing the following restrictions on the definition of extended single-string automata[1]:

– the language L_I, that we use to define γ, consists of formulas of the following form:
$$f_1 \wedge f_2 \wedge \ldots \wedge f_n,$$
where, for every i, f_i is $t_1 = t_2$ or $t_1 \neq t_2$, with t_1, t_2 integer constants or expressions of the form $mod_c(v)$ (where c is a constant and $v \in I$);

– the functions that we use to define γ and δ must be of the following form:
$$(g_1, g_2, \cdots, g_{|I|}),$$
where, for every i, $g_i : \mathbb{N} \to \mathbb{N}$ is a constant function or is obtained by composing a constant number of instances of the successor function.

Given an automaton $M = (S, I, \Sigma, \delta, \gamma, q_0, n_0)$, with $I = \{v_1, \cdots, v_{|I|}\}$, belonging to $\mathcal{D}$, we first determine, for every $v_i \in I$, the least common multiple $lcm(v_i)$ of all constants c that appear in terms of the form $mod_c(v_i)$ which are used to define γ. Then, we define an equivalence relation $\sim_M$ on $S \times C$ by letting $(s_1, (i_1, \cdots, i_{|I|})) \sim_M (s_2, (j_1, \cdots, j_{|I|}))$ if and only if the following two relations hold:
$$s_1 = s_2 \text{ and } i_k \equiv_{lcm(v_k)} j_k \text{ \textit{for} } k \in [1, |I|].$$

It is not difficult to show that the following propositions hold.

Proposition 9. *$\sim_M$ is an equivalence of finite index.*

Proposition 10. *$\sim_M$ respects the function μ associated with M.*

From the definition of the equivalence relation $\sim_M$, it easily follows that the function $\xi|_{\mathcal{D}}$ is computable and hence, by Theorem 8, that the equivalence of two automata in $\mathcal{D}$ is decidable.

We conclude the section by providing an automaton $M \in \mathcal{D}$ that generates the string in $\{\blacksquare, \blacktriangleleft, \square\}^\omega$ representing the granularity which corresponds to the days-to-months relation of the Gregorian Calendar. Such a granularity is represented in Calendar Algebra by the following expression:

$$\begin{aligned}\texttt{pseudomonth} &= Alter^{12}_{11,-1}(\texttt{day}, Alter^{12}_{9,-1}(\texttt{day}, Alter^{12}_{6,-1}(\texttt{day},\\ &\quad Alter^{12}_{4,-1}(\texttt{day}, Alter^{12}_{2,-3}(\texttt{day}, Group_{31}(\texttt{day}))))))\\ \texttt{month} &= Alter^{12*400}_{2+12*399,1}(\texttt{day}, Alter^{12*100}_{2+12*99,-1}(\texttt{day},\\ &\quad Alter^{12*4}_{2+12*3,1}(\texttt{day}, \texttt{pseudomonth}))),\end{aligned}$$

[1] It is worth noting that these restrictions closely resemble those proposed by Alur and Henzinger in [1] to obtain decidable real-time logics.

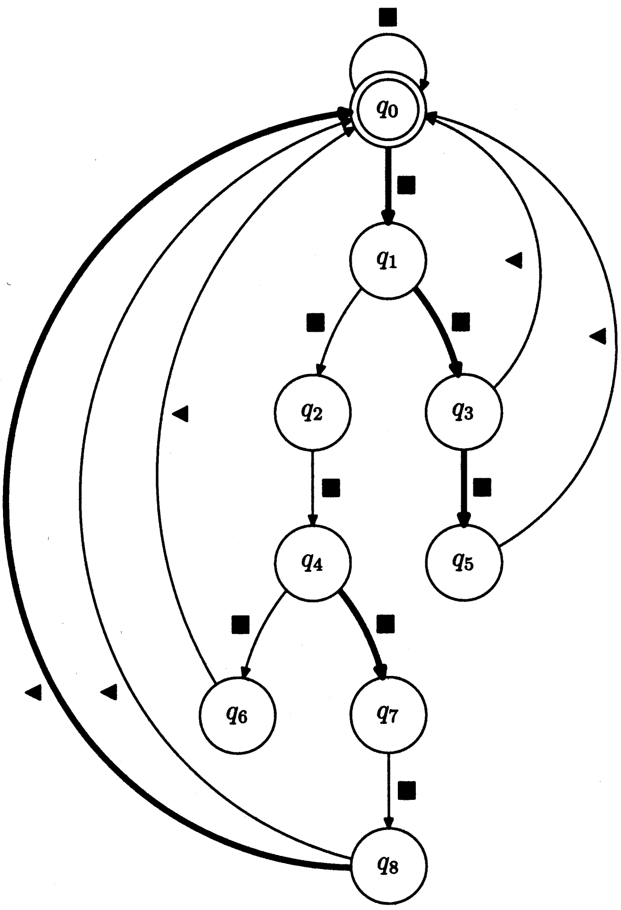


Fig. 5. A $\mathcal{D}$-automaton capturing the Gregorian Calendar.

while the $\mathcal{D}$-automaton M that captures such a granularity is the one depicted in Fig. 5. For the sake of readability, we have not reported the complete specification of M in the figure. The complete specification is obtained by adding $I = \{i, j, k\}$, $n_0 = (1, 1, 0)$ and the two transition functions δ and γ which are defined as reported in Tables 1 and 2.

6 Equivalence and Classification Algorithms

In this section, we evaluate the formalism that we introduced in the previous section by analyzing how the resulting representations influence the complexity of the algorithms for equivalence checking and classification.

Table 1. Function γ

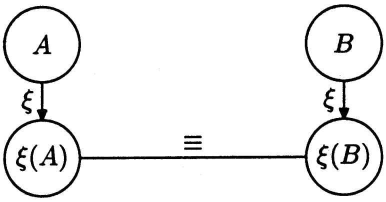
S	S	$\mathbb{N}^{\mathbb{N}}$	L_I
q_0	q_1	$-$	$mod_{27}(i) = 0$
q_1	q_2	$-$	$mod_{12}(j) = 2$
q_2	q_4	$-$	$mod_4(k) = 0$ $\wedge mod_{400}(k) \neq 100$ $\wedge mod_{400}(k) \neq 200$ $\wedge mod_{400}(k) \neq 300$
q_4	q_7	$-$	$mod_{12}(j) \neq 4$ $\wedge mod_{12}(j) \neq 6$ $\wedge mod_{12}(j) \neq 9$ $\wedge mod_{12}(j) \neq 11$
q_8	q_0	$j \leftarrow 1$ $k \leftarrow k+1$ $i \leftarrow 1$	$mod_{12}(j) = 0$

Table 2. Function δ

S	S	$\mathbb{N}^{\mathbb{N}}$
q_0	q_0	$i \leftarrow i+1$
q_1	q_2	$-$
q_2	q_4	$-$
q_3	q_0	$i \leftarrow 1;\ j \leftarrow j+1$
q_4	q_6	$-$
q_5	q_0	$i \leftarrow 1;\ j \leftarrow j+1$
q_6	q_0	$i \leftarrow 1;\ j \leftarrow j+1$
q_7	q_8	$-$
q_8	q_0	$i \leftarrow 1;\ j \leftarrow j+1$

6.1 Equivalence

The first problem we face when we introduce a new representation method is the *equivalence problem*: given two instances of a formalism, we want to establish whether the two instances denote the same object or not. In our case, we know that the equivalence problem is decidable in $\mathcal{D}$, but we do not know how the algorithm we gave behaves in terms of time and space complexities. The diagram depicted in Fig. 6 shows the high-level behaviour of our algorithm. From a



$$\begin{tikzcd} A \arrow[d, "\xi"'] & B \arrow[d, "\xi"'] \\ \xi(A) \arrow[r, dash, "\equiv"] & \xi(B) \end{tikzcd}$$

Fig. 6. High-level behaviour of our equivalence algorithm.

complexity point of view, the critical step is the computation of $\xi(A)$ and $\xi(B)$. The informal procedure we gave in the last section is not efficient, because it does not avoid the creation of useless states. Algorithm 6.1 solves this problem, provided that S' is implemented in an efficient way. As an example, if we implement S' as an hash table, we can obtain an average running time of $O(|I|)$ for insertion and lookup on S' (line 5, 9 and 10). δ' is never looked up during the computation, so we can implement it by simpler data structures. Subroutines *nextstate*, *nextconfig* and *nextchar* have an obvious meaning and we can assume

Algorithm 6.1 Given $M = (S, I, \Sigma, \delta, \gamma, q_0, n_0) \in \mathcal{D}$, build an equivalent $M' = (S', \Sigma, \delta', q'_0)$ in an efficient way.

```
1: for $j \leftarrow 1$ to $|I|$ do
2:     Compute the lcm $N[j]$ of all $c$ such that $mod_c(I[j])$ is used to define $\gamma$
3: $q'_v \leftarrow (q_0, mod_N(n_0))$
4: $S' \leftarrow \{q'_v\}$
5: $q'_0 \leftarrow q'_v$
6: $q'_n \leftarrow (nextstate(q'_v), mod_N(nextconfig(q'_v))$
7: $\delta' \leftarrow \{q'_v \rightarrow (q'_n, nextchar(q'_v)\}$
8: while $q'_n \notin S$ do
9:     $S' \leftarrow S' \cup \{q'_n\}$
10:    $q'_v \leftarrow q'_n$
11:    $q'_n \leftarrow (nextstate(q'_v), mod_N(nextconfig(q'_v))$
12:    $\delta' \leftarrow \{q'_v \rightarrow (q'_n, nextchar(q'_v)\}$
```

that they take $O(|I|)$ time. The overall complexity of Algorithm 6.1 is therefore $O(|S'| \cdot |I|)$.

In designing a reducible extended single-string automaton M that represents a granularity R, one must guarantee that the number of useful states in $\xi(M)$ is comparable to the number of states of the minimum single-string automaton that represents R. In such a case, the equivalence problem has a complexity which is comparable to the one of Wijsen's formalism.

6.2 Classification

Let R be a time granularity. We say that a granule $C \in R$ has *ordinal number* i if C is the i–th element of R in terms of the linear order induced on R by the linear order on natural numbers.

With reference to the graph representation of single-string automata, we interpret each single-string automaton $M = (S, \Sigma, \delta, q_0)$ as a graph, whose nodes correspond to the states of M. An interesting property of such a graph is that there exists at most one node $q \in S$ such that q is reachable from q_0 and there exist exactly two states $q', q'' \in S$ that are reachable from q_0 and whose only outgoing edge enters q. We call q the *loop state* (if such a node q does not exist, the loop state is q_0). The loop state of a single-string automaton $M = (S, \Sigma, \delta, q_0)$ turns out to be very useful in many situations; hence, although it can be computed in a natural way from S and δ, we often assume that it is part of the definition of M.

A problem that frequently arises in practical situations is the *classification problem*: given a granularity R and a natural number n, decide whether there exists a granule $C \in R$ such that $n \in C$ and, if this is the case, compute C and its ordinal number[2].

[2] Notice that such a formulation of the classification problem makes sense only if there exists an infinite number of granules. In the case in which we have only finitely many

If we represent granularities as single-string automata, we can easily solve the classification problem on $M = (S, \Sigma, \delta, q_0)$ and $n \in \mathbb{N}$ by executing the following steps:

1. determine the loop state (if it has not been included in the definition of the automaton);
2. starting from the initial state, make exactly n transitions keeping track of the state following the last occurrence of ◀ and of the current ordinal number (the ordinal number of the current granule);
3. if the last symbol is not □ (if it is □, then n does not belong to any granule), come back to the state following the last occurrence of ◀ and proceed until the next ◀ is reached; for any occurrence of ■, write the corresponding natural number.

Algorithm 6.2 implements such a procedure using a number of suitable heuristics. In particular, notice that lines 1–16 can be "factorized" over different instances of the algorithm involving the same automaton, but different natural numbers.

The above classification algorithm can be easily extended to automata in $\mathcal{D}$, by adapting the notions of loop state, state equality, and transition; alternatively, we can preprocess the input automaton M with Algorithm 6.1 and work with its single-string equivalent M'. In any case, the running time is proportional to $|M'|$, where $|M'|$ is the size of the graph corresponding to M'.

7 Conclusions and Further Work

In this paper, we proposed a new automata-theoretic approach to the representation and manipulation of time granularities. We proved that it is expressive enough to capture a relevant class of practical granularities, and that it generates compact representations which are suitable to be manipulated by algorithms. We explicitly defined two algorithms that respectively solve the equivalence checking and classification problems. It is possible to show that many practical problems about time granularities can be reduced to these two problems or to minor variants of them. The formalism of (extended) single-string automata is not, however, a high-level formalism, and the descriptions it produces are not as readable as, for instance, Calendar Algebra expressions. For this reason, we believe it useful to think of translations from high-level formalisms, such as Calendar Algebra, to our formalism.

The main contribution of this paper lies in the automata-theoretic account of time granularity that it provides. We believe it possible to further exploit the expressive power of automata over infinite strings to improve our ability of dealing with time granularities. As an example, moving from deterministic automata to nondeterministic ones will allow us to manage situations in which a given repeating pattern can start at a finite number of different time points (finite union of single-string automata) or situation in which the repeating pattern can start at an arbitrary time point (infinite union of single-string automata).

granules, the last one necessarily includes infinite natural numbers and thus cannot be computed.

Algorithm 6.2 Solve the classification problem on $M = (S, \Sigma, \delta, q_0)$ and $n \in \mathbb{N}$ assuming q_l is the loop state of M.

```
1: $i_f \leftarrow 0$; $j_f \leftarrow 0$; $q \leftarrow q_0$
2: while $q \neq q_l$ do
3:    $(q, c) \leftarrow \delta(q)$
4:    if $c = \blacktriangleleft$ then
5:       $j_f \leftarrow j_f + 1$
6:    $i_f \leftarrow i_f + 1$
7: $i_s \leftarrow 0$; $j_s \leftarrow 0$
8: repeat
9:    $(q, c) \leftarrow \delta(q)$
10:   if $c = \blacktriangleleft$ then
11:      $j_s \leftarrow j_s + 1$
12:   $i_s \leftarrow i_s + 1$
13: until $q = q_l$
14: if $n \leq i_f + i_s$ then
15:   $q, q_p \leftarrow q_0$; $j_p \leftarrow 1$; $h_p \leftarrow 1$
16:   for $h \leftarrow 1$ to $n - 1$ do
17:      $(q, c) \leftarrow \delta(q)$
18:      if $c = \blacktriangleleft$ then
19:         $q_p \leftarrow q$; $h_p \leftarrow h + 1$; $j_p \leftarrow j_p + 1$
20: else
21:   $d \leftarrow div_{i_s}(n - i_f)$; $r \leftarrow mod_{i_s}(n - i_f)$; $q \leftarrow q_l$; $j_p \leftarrow j_f + (d - 1) * j_s + 1$
22:   for $h \leftarrow i_f + (d - 1) * i_s + 1$ to $n - 1$ do
23:      $(q, c) \leftarrow \delta(q)$
24:      if $c = \blacktriangleleft$ then
25:         $q_p \leftarrow q$; $h_p \leftarrow h + 1$; $j_p \leftarrow j_p + 1$
26: $(q, c) \leftarrow \delta(q)$
27: if $c \neq \square$ then
28:   $print$("$C \in R$ with ordinal number", $j_p$)
29:   $(q, c) \leftarrow \delta(q_p)$; $h \leftarrow h_p$
30:   if $c \neq \square$ then
31:      $print(h)$
32:   while $c \neq \blacktriangleleft$ do
33:      $h \leftarrow h + 1$; $(q, c) \leftarrow q$
34:      if $c \neq \square$ then
35:         $print(h)$
36: else
37:   $print$("There is not $C \in R$ such that $n \in C$.")
```

References

1. R. Alur and T. Henzinger. Real-time logics: Complexity and expressiveness. *Information and Computation*, (104):35–77, 1993.
2. C. Bettini and R. De Sibi. Symbolic Representation of User-Defined Time Granularities. *Annals of Mathematics and Artificial Intelligence*, 30(1-4), 2001.

3. C. Bettini, C. E. Dyreson, W. S. Evans, R. T. Snodgrass, and X. S. Wang, A glossary of time granularity concepts. In: *Temporal Databases: Research and Practice*, O. Etzion, S. Jajodia, and S. Sripada (Eds.), LNCS 1399, pages 406–413, Springer, 1998.
4. C. Bettini, X. S. Wang, and S. Jajodia. Satisfiability of quantitative temporal constraints with multiple granularities. In: *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, LNCS 1330, pages 435–449, Springer, 1997.
5. C. Bettini, X. S. Wang, and S. Jajodia. A general framework for time granularity and its application to temporal reasoning. *Annals of Mathematics and Artificial Intelligence*, 22(1-2):29–58, 1998.
6. C. Bettini, X. S. Wang, and S. Jajodia. *Time Granularities in Databases, Data Mining, and Temporal Reasoning.* Springer, 2000.
7. B. Leban, D. McDonald, and D. Foster. A representation for collections of temporal intervals. In: *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 367–371, AAAI Press, 1986.
8. M. Niezette and J. Stevenne. An efficient symbolic representation of periodic time. In: *Proceedings of the International Conference on Information and Knowledge Management*, pages 161–168, ACM Press, 1992.
9. P. Ning, X. S. Wang, and S. Jajodia. An algebraic representation of calendars. In: *Proceedings of the AAAI Workshop on Spatial and Temporal Granularity*, C. Bettini and A. Montanari (Eds.), pages 1–8, AAAI Press, 2000.
10. D. Perrin and P.E. Schupp. Automata on the integers, recurrence distinguishability, and the equivalence and decidability of monadic theories. In: *Proceedings of the IEEE Symposium on Logic in Computer Science (LICS)*, pages 301–304, IEEE, 1986.
11. M. Soo and R. Snodgrass. Mixed Calendar Query Language Support for Temporal Constants. *The MultiCal Project, Release 1.1*, Department of Computer Science, The University of Arizona, Tucson, AZ, September 1995.
12. W. Thomas. Languages, Automata, and Logic. In: *Handbook of formal languages, Vol. 3*, G. Rozemberg and A. Salomaa (Eds.), pages 389–455, Springer, 1997.
13. J. Wijsen. A string-based model for infinite granularities. In: *Proceedings of the AAAI Workshop on Spatial and Temporal Granularity*, C. Bettini and A. Montanari (Eds.), pages 9–16, AAAI Press, 2000.