An Algebraic Characterization of Data and Timed Languages

Patricia Bouyer[1 †], Antoine Petit[1 † ‡], Denis Thérien[2 §]

[1] LSV, CNRS UMR 8643, ENS de Cachan,
61 Av. du Président Wilson,
94235 Cachan Cedex, France
{bouyer,petit}@lsv.ens-cachan.fr
[2] School of Computer Science, McGill University,
3480 University,
Montréal, QC, Canada, H3A 2A7
denis@cs.mcgill.ca

Abstract. Algebra offers an elegant and powerful approach to understand regular languages and finite automata. Such framework has been notoriously lacking for timed languages and timed automata. We introduce the notion of monoid recognizability for data languages, which include timed languages as special case, in a way that respects the spirit of the classical situation. We study closure properties and hierarchies in this model, and prove that emptiness is decidable under natural hypotheses. Our class of recognizable languages properly includes many families of deterministic timed languages that have been proposed until now, and the same holds for non-deterministic versions.

1 Introduction

The class of regular languages can be characterized in various ways: finite automata, rational expressions, monadic second order logic, extended temporal logics, finite monoids... [RS97]. All these characterizations constitute not only one of the cornerstones of theoretical computer science but also form the fundamental basis for much more practical research on verification (see *e.g.* [CGP99]). Among all these equivalences, the simplest is undoubtedly the *purely algebraic* one claiming that a word language is regular if and only if it is monoid recognizable *i.e.* it is the inverse image by a morphism of some subset of a finite monoid.

In the framework of timed languages, very useful to specify and verify real-time systems, the situation is far from being so satisfactory. The original class of timed automata, proposed by Alur and Dill [AD94] has a decidable emptiness

† Research partly supported by the French project RNRT "Calife"
‡ Research partly supported by the French-India project CEPIPRA n^{o}2102 − 1
§ Research supported by NSERC, FCAR, and the von Humboldt Foundation

problem, but is not closed under complement. Several logical characterizations [Wil94,HRS98] or even Kleene-like theorems [ACM97,Asa98,BP99,BP01] have been proposed for the whole class of timed automata but no purely algebraic one. Interesting subclasses of timed automata, closed under complement, have been proposed and often logically characterized. For instance, (recursive) event clocks automata [AFH94] are closed under complement and can be characterized in a nice logical way [HRS98]. But once again, even if a related notion of counter-free timed languages has been defined, no algebraic characterization exists.

For the first time, at least to our knowledge, we propose in this paper a purely algebraic characterization for timed languages. In fact, we deal with a more general framework than timed languages, the so-called *data languages*. We consider a finite alphabet of actions Σ and a set of data $\mathcal{D}$ (this set of data could be some time domain but also anything else). A data word is thus a sequence of pairs (a, d) where $a \in \Sigma$ and $d \in \mathcal{D}$.

We propose to use a finite fixed number of registers to store the data. When a new letter (a, d) is read, the data is kept or not depending only on the letter a and on the value of the current computation in the finite monoid M. Then the new value of the computation is calculated from the previous value, the current letter a and some *finite and bounded* information from the registers. Hence, the precise values of the registers are never used by the monoid. Only a finite bounded amount of information is needed to decide whether a data word is in the language or not.

We obtain in this way, for any set of actions Σ and set of data $\mathcal{D}$, a class of so-called "monoid recognizable" data languages. This class is closed under boolean operations. As first result, which shows the interest of our approach, the choice of the monoid is fundamental. More precisely, we prove that, like in the formal language case, if two monoids are such that none of them divides the other, then the corresponding class of data languages are incomparable. We then study the exact power of the number of registers.

We next define a notion of deterministic data automata and, as one of our two main theorems, we prove that a data language is monoid recognizable if and only if it is accepted by some data automaton. Note that the translation from monoid to automaton and vice versa is simple and very close to what happens in formal language theory, which emphasizes once more the elegance of the proposed approach.

We then focus on the problem of deciding emptiness of languages recognized by data automata, or equivalently, monoid recognizable. We propose a simple and nice condition related to the registers and the data domain under which emptiness is decidable. More precisely, under this condition, we propose our second main result: an algorithm to transform a data automaton $\mathcal{A}$ into a finite automaton recognizing the classical formal language of those words of Σ^* that can be obtained from a data word accepted by $\mathcal{A}$ by erasing the data. The idea of this construction is similar to the region automaton construction of Alur and Dill [AD94].

If the set of data $\mathcal{D}$ is a time domain, our recognizable data languages contain all the timed languages recognized by deterministic timed automata [AD94] or their deterministic extensions [DZ98,CG00]. But our class also contains a lot of timed languages which cannot be recognized by any timed automata (even non-deterministic ones).

We also briefly study two possible extensions of our model. First, we consider non-deterministic data automata (or equivalently a non-deterministic notion of monoid recognizability). Then we get a larger class of data languages, still closed under union and intersection but not anymore by complementation. On the contrary, this new class is closed by concatenation and iteration. Once again, emptiness can be decided, by an algorithm similar to the one used in the deterministic case. Second, we show that if we extend the power of the registers and allow computations to be performed on them, then what monoid is used to recognize the language becomes essentially irrelevant.

This paper contains only sketches of proofs. The complete proofs are available in the technical report [BPT01].

2 Basic Definitions

If Z is any set, Z^* denotes the set of finite sequences of elements in Z. We consider throughout this paper a finite alphabet Σ and an unrestricted set of data $\mathcal{D}$. Among the elements of $\mathcal{D}$, we distinguish a special initial value, denoted by $\perp$. A *data word* over Σ and $\mathcal{D}$ is a finite sequence $(a_1, d_1) \ldots (a_p, d_p)$ of $(\Sigma \times \mathcal{D})^*$. A data language is a set of data words. If $k \geq 1$ is the number of registers, a *register update*, or simply an *update*, $\overline{up}$, is defined by a subset of $\{1, \ldots, k\}$. This update induces a function up from $\mathcal{D}^k \times \mathcal{D}$ into $\mathcal{D}^k$ mapping $((d_i)_{i=1..k}, d)$ to $((d'_i)_{i=1..k})$ where $d'_i = d$ if $i \in \overline{up}$ and $d'_i = d_i$ otherwise. In the sequel, we will simply write $((d'_i)_{i=1..k}) = up((d_i)_{i=1..k}, d)$. If $\sim$ is an equivalence defined on $\mathcal{D}^k$ and if $\theta \in \mathcal{D}^k$, we denote $\overline{\theta}$ the class of θ modulo $\sim$.

3 Monoid Recognizability

A first naive attempt to define a notion of monoid recognizability for data languages could simply consider a morphism from the free monoid $(\Sigma \times \mathcal{D})^*$ to some finite monoid M. But the corresponding class of languages would be very restricted. Since the image of $\Sigma \times \mathcal{D}$ would be finite, the simple language $\{(a, d)(a, d') \mid d \neq d'\}$ would not be monoid recognizable as soon as $\mathcal{D}$ is infinite. Similar unsuccessful attempts have been studied in details [ABB80] in the framework of formal languages over an infinite alphabet. We thus need to have a finite mechanism to take into account the data. But, in order to maintain the relevance of the monoid, this mechanism should be very simple and in particular unable to perform any computation. We propose to use, as mechanism, a finite fixed number of registers to store the data. These registers are used through a notion of updates as defined in the previous section. Note that such updates decide to store or not a data independently of this data or of the values of the registers.

Definition 1. *Let L be a data language over Σ and $\mathcal{D}$, let M be a finite monoid. We say that M* recognizes *L if there exists a subset P of M, an integer k, for each pair $(m, a) \in M \times \Sigma$ an update $up_{m,a}$ over k registers, an equivalence of finite index $\sim$ on $\mathcal{D}^k$, and a morphism $\varphi : (\Sigma \times \mathcal{D}^k/\!\sim)^* \longrightarrow M$ such that: a data word $(a_1, d_1) \dots (a_n, d_n)$ is in L if and only if the sequences $(\theta_i)_{i=0\dots n}$ and $(m_i)_{i=0\dots n}$ defined by:*

$$\begin{cases} \theta_0 = \perp^k \\ \theta_{i+1} = up_{m_i,a_{i+1}}(\theta_i, d_{i+1}) \end{cases} \quad \textit{and} \quad \begin{cases} m_0 = 1_M \\ m_{i+1} = m_i \ \varphi(a_{i+1}, \overline{\theta_{i+1}}) \end{cases}$$

satisfy $m_n \in P$.

When a new letter (a_{i+1}, d_{i+1}) is read, the data is kept or not (as determined by $up_{m_i,a_{i+1}}$) depending thus only on the letter a_{i+1} and on the value m_i of the current computation in the finite monoid M. Then the new value m_{i+1} of the computation is calculated from the previous value m_i, the current letter a_{i+1} and some *finite and bounded* information, $\overline{\theta_{i+1}}$, from the registers. Hence, the precise values of the registers are never used by the monoid. Only a finite bounded amount of information is needed to decide whether a data word is in the language or not. Note that, the sequences $(\theta_i)_{i=0\dots n}$ and $(m_i)_{i=0\dots n}$ associated with a given data word are unique. A data language is said to be *monoid recognizable* if there exists some finite monoid recognizing it.

Note that the definition proposed in [KF94], also based on registers, is much more restrictive than ours. In particular their languages have the following property, if $(a_1, d_1) \dots (a_n, d_n)$ is in the language then for any bijection ν from $\mathcal{D}$ into $\mathcal{D}$, $(a_1, \nu(d_1)) \dots (a_n, \nu(d_n))$ has also to be in the language. We do not require at all such a property.

Example 1. The data language $L = \{(a, d)(a, d') \mid d \neq \perp, d \neq d'\}$ over $\{a\}$ and $\mathcal{D}$ is recognized by the finite monoid $M = \{1, y, y^2, 0\}$ with $y^3 = 0$ and $0x = x0 = 0$ for any $x \in M$. We use two registers and we define two classes over $\mathcal{D}^2$: $\overline{\theta_{\neq}} = \{(d, d') \mid d \neq d'\}$ and $\overline{\theta_=} = \mathcal{D}^2 \setminus \overline{\theta_{\neq}}$. We define a morphism $\varphi : (\{a\} \times \{\overline{\theta_{\neq}}, \overline{\theta_=}\})^* \longrightarrow M$ by $\varphi(a, \overline{\theta_{\neq}}) = y$, $\varphi(a, \overline{\theta_=}) = 0$. Our updates are $up_{1,a} = \{1\}$ and if $z \in M \setminus \{1\}$, $up_{z,a} = \{2\}$. With these definitions, L is accepted by M (with $P = \{y^2\}$). As an example of computation, consider the data word $(a, d)(a, d')$ with $d \neq \perp$ and $d \neq d'$.

In the monoid M	$1_M \xrightarrow[d]{a}$	$y \xrightarrow[d']{a}$	y^2
Two registers	$\begin{pmatrix} \perp \\ \perp \end{pmatrix}$	$\begin{pmatrix} d \\ \perp \end{pmatrix}$	$\begin{pmatrix} d \\ d' \end{pmatrix}$
Equivalence classes	$\overline{\theta_=}$	$\overline{\theta_{\neq}}$	$\overline{\theta_{\neq}}$

We must notice that the registers do not compute anything. For example, taking $\mathcal{D} = \mathbb{Q}$, with only one register we could have computed the difference $d' - d$

instead of putting the data d' in a second register. But this is not allowed in our model.

Example 2. The data language $\{(a, d_1) \dots (a, d_n)(a, d) \mid d \notin \{d_1, \dots, d_n\}\}$ over $\{a\}$ and $\mathcal{D}$ (where $\mathcal{D}$ is infinite) is not recognized by any finite monoid. Intuitively, an unbounded number of data should be stored, which is not allowed.

Remark 1. This definition of monoid recognizability is a quite natural extension of the monoid recognizability in formal language theory. Namely, if $\mathcal{D}$ reduces to $\{\perp\}$, then Σ and $\Sigma \times \mathcal{D}$ are in bijection and a formal language is recognizable if and only if its image is a monoid recognizable data language. It can also be shown that if $\mathcal{D}$ is finite and if $L \subseteq (\Sigma \times \mathcal{D})^*$ is a monoid recognizable data language, then L is also a recognizable formal language.

If M is a finite monoid and k an integer, the set of data languages over Σ and $\mathcal{D}$ recognized by M using k registers, is denoted by $\mathcal{L}_{M,k}(\Sigma, \mathcal{D})$, or simply $\mathcal{L}_{M,k}$. We also set $\mathcal{L}_M = \bigcup_k \mathcal{L}_{M,k}$ and $\mathcal{L}_k = \bigcup_M \mathcal{L}_{M,k}$. The set of recognizable data languages has many interesting closure properties.

Proposition 1. *The set $\mathcal{L}_{M,k}$ is closed under complementation. If $L_1 \in \mathcal{L}_{M_1,k_1}$ and $L_2 \in \mathcal{L}_{M_2,k_2}$, then $L_1 \cup L_2$ and $L_1 \cap L_2$ are in $\mathcal{L}_{M_1 \times M_2, k_1+k_2}$.*

From the algebraic point of view, the soundness of our definition is assessed by the following result. It shows essentially that increasing the number of registers cannot help if the monoid is not powerful enough. Hence the structure of the monoid is really fundamental and plays a role similar to what happens in the framework of formal languages.
If M and M' are monoids, recall that M *divides* M' if M is a quotient of a submonoid of M' [Pin86].

Proposition 2. *If M and M' are finite monoids such that neither M divides M' nor M' divides M, then $\mathcal{L}_M \neq \mathcal{L}_{M'}$.*

Proof. Let L be a language on Σ and define

$$L_{\mathcal{D}} = \{(a_1, d_1) \dots (a_n, d_n) \mid a_1 \dots a_n \in L \text{ and } d_i \in \mathcal{D}\}$$

Let M be a finite monoid. Then, we will prove that

$$L \text{ is recognized by } M \iff L_{\mathcal{D}} \text{ is recognized by } M$$

This will establish the result as the hypothesis on M and M' implies that the class of formal languages they respectively recognize are incomparable.
We prove the two implications separately:
Assume that L is recognized by M. There exists a morphism $\varphi : \Sigma^* \longrightarrow M$ and some $P \subseteq M$ such that $L = \varphi^{-1}(P)$. Taking $k = 0$, it is easy to see that M recognizes $L_{\mathcal{D}}$.
Assume that $L_{\mathcal{D}}$ is recognized by M with k, $\sim$, φ as in Definition 1. In particular, if $a_1 \dots a_n$ is in Σ^*, the image of the data word $(a_1, \perp) \dots (a_n, \perp)$ in $(\Sigma \times \mathcal{D}^k/\!\sim)^*$ is

$(a_1, \overline{\perp^k}) \dots (a_n, \overline{\perp^k})$. We define a morphism $\psi : \Sigma^* \longrightarrow M$ by $\psi(a) = \varphi((a, \overline{\perp^k}))$. Then,

$$\begin{aligned} a_1 \dots a_n \in L &\iff (a_1, \perp) \dots (a_n, \perp) \in L_{\mathcal{D}} \\ &\iff \varphi((a_1, \overline{\perp^k}) \dots (a_n, \overline{\perp^k})) \in P \\ &\iff \psi(a_1 \dots a_n) \in P \end{aligned}$$

Thus, M recognizes the language L and the conclusion easily follows. □

The following statements make precise the relative role of the monoid and of the registers. For example, each additional register strictly increases the class of languages being recognized, as in timed automata each additional clock increases also the power of the automata [HKWT95]. On the other hand, if the monoid and the alphabet are fixed, then the hierarchy on registers collapse.

Proposition 3. *1. If M is a fixed finite monoid, the sequence $(\mathcal{L}_{M,k}(\Sigma, \mathcal{D}))_k$ collapses, more precisely, $\mathcal{L}_{M,2^{|\Sigma \times M|}-1} = \mathcal{L}_{M,2^{|\Sigma \times M|}}$.*
2. The sequence $(\mathcal{L}_k(\Sigma, \mathcal{D}))_k$ is strictly monotonic.
3. Let M_0 be the finite monoid $\{1, 0, x\}$ with $x^2 = x$. For each integer k, there exists a finite alphabet Σ_k and a language L_k over Σ_k such that $L_k \in \mathcal{L}_{M_0,k}(\Sigma_k, \mathcal{D}) \setminus \bigcup_{k'<k} \mathcal{L}_{k'}$.

Proof (Sketch).

1. Updates are parameterized by a pair of $M \times \Sigma$, thus, considering a data language recognized by a finite monoid M with k registers, we define the function

$$\begin{aligned} \lambda : \{1 \dots k\} &\longrightarrow \{up_{m,a} \mid (m, a) \in M \times \Sigma\} \\ i \quad &\longmapsto \{up_{m,a} \mid (m, a) \in M \times \Sigma \text{ and } i \in up_{m,a}\} \end{aligned}$$

   and the equivalence $i \cong j \iff \lambda(i) = \lambda(j)$. Intuitively, if $i \cong j$, the registers i and j play the same role, in that they are updated exactly at the same steps. Thus, we can keep only one register for each class. Moreover, the class of registers i such that $\lambda(i) = \emptyset$ is not useful.
2. The data language $L_k = \{(a, d_1) \dots (a, d_n) \mid i \equiv j \mod (k-1) \implies d_i = d_j\}$ over $\{a\}$ and $\mathcal{D}$ ($\mathcal{D}$ is supposed to be infinite) is recognized by a finite monoid with k registers, but is recognized by no finite monoid with strictly less than k registers.
3. We define $\Sigma_k = \{a_0, a_1, \dots, a_{k-1}\}$ and for each $i = 1 \dots k-1$, we define a function μ_i such that for each data word $u \in (\Sigma_k \times \mathcal{D})^*$, $\mu_i(u)$ is the data d (if it exists) such that $u = u' \ (a_i, d) \ u''$ where u'' does not contain any a_i. If this data does not exists, $\mu_i(u)$ is $\perp$. We define the data language

$$\begin{aligned} L'_k = \{u \ (a_0, d'_1) \dots (a_0, d'_n) \mid\ & u \in ((\Sigma_k \setminus \{a_0\}) \times \mathcal{D})^* \\ & \text{and for each } j, \ d'_j \in \cup_{i=1}^{k-1} \{\mu_i(u)\} \ \ \} \end{aligned}$$

   The language L'_k is recognized by M_0 using k registers but is recognized by no finite monoid using strictly less than k registers. □

4 Data Automata

In this section, we define a notion of recognizability by data automata and prove its equivalence with monoid recognizability.

Definition 2. *A* data automaton *over Σ and $\mathcal{D}$ is a tuple $\mathcal{A} = (Q, q_0, F, k, \sim, T)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, k is an integer, $\sim$ is an equivalence relation of finite index defined on $\mathcal{D}^k$ and $T \subseteq (Q \times \mathcal{D}^k/\!\sim \times \Sigma \times \mathcal{U} \times \mathcal{D}^k/\!\sim \times Q)$ is a finite set of transitions ($\mathcal{U}$ is a set of updates) such that the following determinism hypothesis holds: for each tuple $(q, g, a) \in Q \times \mathcal{D}^k/\!\sim \times \Sigma$, there is a (unique) update up such that any transition $(q, g, a, up', g', q') \in T$ satisfies $up' = up$ and if (q, g, a, up, g', q_1') and (q, g, a, up, g', q_2') are in T, then $q_1' = q_2'$.*

A data word $(a_1, d_1) \ldots (a_n, d_n)$ is accepted by the data automaton $\mathcal{A}$ if there exists a path in $\mathcal{A}$

$$q_0 \xrightarrow[d_1]{g_1, a_1, up_1, g_1'} q_1 \xrightarrow[d_2]{g_2, a_2, up_2, g_2'} q_2 \ldots \xrightarrow[d_n]{g_n, a_n, up_n, g_n'} q_n$$

such that the sequence $(\theta_i)_{i=0\ldots n}$ defined by

$$\theta_0 = \perp^k \quad \text{and} \quad \theta_{i+1} = up_{i+1}(\theta_i, d_{i+1})$$

satisfies $\overline{\theta_{i-1}} = g_i$ for $1 \leq i \leq n$, $\overline{\theta_i} = g_i'$ for $1 \leq i \leq n$ and $q_n \in F$.
The set of data words that are accepted by $\mathcal{A}$ is denoted by $L(\mathcal{A})$.

Example 3. The data language described in Example 1,

$$L = \{(a, d)(a, d') \mid d \neq \perp,\ d \neq d'\}$$

is recognized by the following data automaton ($\overline{\theta_=}$ and $\overline{\theta_{\neq}}$ are defined in Example 1):

$$\longrightarrow q_0 \xrightarrow{\overline{\theta_=},\ a,\ \{r_1\},\ \overline{\theta_{\neq}}} q_1 \xrightarrow{\overline{\theta_{\neq}},\ a,\ \{r_2\},\ \overline{\theta_{\neq}}} q_2 \longrightarrow$$

We claim that this notion of recognizability by data automata is equivalent to the notion of monoid recognizability in the following sense:

Theorem 1. *Let L be a data language over Σ and $\mathcal{D}$. Then L is recognized by a data automaton if and only if it is recognized by a finite monoid.*

We thus have a result similar to the formal language case. As it appears below, the transformations from monoids to automata and from automata to monoids are very close to the ones used in formal languages. We believe that this similarity emphasizes the appropriateness of our approach.

Proof (Sketch). **If Implication.** First, assume that $L \subseteq (\Sigma \times \mathcal{D})^*$ is recognized by a finite monoid M, with the notations of Definition 1. We construct a data automaton over Σ and $\mathcal{D}$, $\mathcal{A} = (Q, q_0, F, k, \sim, T)$, as follows:

- k and $\sim$ comes from the monoid recognizer,
- $Q = M$, $q_0 = 1_M$ and $F = P$,
- $T = \{(m, g, a, up_{m,a}, g', m') \mid m \in M,\ g, g' \in \mathcal{D}^k\!/\!\sim,\ a \in \Sigma,\ m' = m\varphi(a, g')\}$.

One can prove that $\mathcal{A}$ is a valid deterministic data automaton and that the language accepted by $\mathcal{A}$ is precisely L.

Only If Implication. Now, assume that $L \subseteq (\Sigma \times \mathcal{D})^*$ is recognized by the data automaton $\mathcal{A} = (Q, q_0, F, k, \sim, T)$. We define M as the set of functions from $Q \times \mathcal{D}^k\!/\!\sim$ into itself. We claim that L is recognized by M. The morphism $\varphi : (\Sigma \times \mathcal{D}^k\!/\!\sim)^* \longrightarrow M$ is induced by the function $(a, g') \longmapsto \Big[(q, g) \mapsto (q', g')\Big]$ where q' is the unique state for which there exists a transition (q, g, a, up, g', q') in $\mathcal{A}$ (the unicity of q' comes from the determinism of $\mathcal{A}$). For any $m \in M$, suppose $m((q_0, g_0)) = (q, g)$ (g_0 is the equivalence class of $\perp^k$). Because of determinism again, for any a, there is a unique up such that there exists a transition $(q, g, a, up, -, -)$ and we define $up_{m,a} = up$. We finally define $P = \{m \mid m((q_0, \overline{\perp^k})) \in F \times \mathcal{D}^k\!/\!\sim\}$. It is then possible to prove that $L(M) = L$. The equivalence between monoids and automata is thus proved. $\square$

We can notice that the translations from monoids to automata and vice-versa do not change neither the set of updates, nor the number of registers and the equivalence.

We say that a data language is *recognizable* if it is recognized by some data automaton (which is equivalent to being recognized by a finite monoid).

5 Decidability of the Emptiness Problem

We first note that the general class of recognizable data languages is undecidable: we can easily simulate a two counter machine [Min67] using a data automaton. We propose a condition that determines a class of data automata for which the emptiness problem is decidable.

As a preliminary, given a register update up, we define a relation on $\mathcal{D}^k\!/\!\sim$, denoted by $\xrightarrow{up}$, in the following way:

$$\overline{\theta} \xrightarrow{up} \overline{\theta'} \quad \textit{iff} \quad \exists v \in \overline{\theta},\ \exists d \in \mathcal{D},\ up(v, d) \in \overline{\theta'}$$

In order to capture decidability in our model, we define the following condition:

$$\textsc{Condition } (\dagger)\text{:}\ \overline{\theta} \xrightarrow{up} \overline{\theta'} \quad \textit{iff} \quad \forall v \in \overline{\theta},\ \exists d \in \mathcal{D},\ up(v, d) \in \overline{\theta'}$$

We will prove that this simple condition ensures the decidability of the emptiness problem. The principle of the proof of this result is very similar to the one of region construction as defined by Alur and Dill [AD94].

Theorem 2. *Let L be a recognizable data language over Σ and $\mathcal{D}$. Assume L is recognized by the finite monoid M with an equivalence and updates that satisfy the condition (†). Then the emptiness of L is decidable in complexity* PSPACE.

Proof (Sketch). Let $L \subseteq (\Sigma \times \mathcal{D})^*$ be a recognizable data language. We assume that M is a monoid which recognizes L and that k, $\sim$, φ and $up_{m,a}$ satisfy the condition (†) and are correctly defined in order to recognize L. As in the proof of Theorem 1, we construct a data automaton $\mathcal{A}$ whose transitions are

$$m \xrightarrow{g,\ a,\ up_{m,a},\ g'} m\varphi(a, g')$$

Of course, $L = L(\mathcal{A})$. From $\mathcal{A}$, we construct a finite automaton $\mathcal{B} = (Q, I, F, T)$ where $Q = M \times \mathcal{D}^k/\!\sim$, $I = (1_M, \overline{\bot^k})$, $F = P \times \mathcal{D}^k/\!\sim$ and T is defined by

$$((m, g), a, (m', g')) \in T \iff m \xrightarrow{g,\ a,\ up_{m,a},\ g'} m' \text{ and } g \xrightarrow{up_{m,a}} g'$$

We can prove that, as condition (†) holds, this finite automaton accepts

$$\mathrm{U{\small NDATA}}(L) = \{a_1 \dots a_n \mid \exists d_1, \dots, d_n,\ (a_1, d_1) \dots (a_n, d_n) \in L\}$$

□

It remains to study the decidability of condition (†).
We first define

$$\begin{cases} \widehat{up}(\overline{\theta}) = \{v' \mid \exists v \in \overline{\theta},\ \exists d \in \mathcal{D},\ v' = up(v, d)\} \\ \widehat{up}^{-1}(\overline{\theta'}) = \{v \mid \exists d \in \mathcal{D},\ up(v, d) \in \overline{\theta'}\} \end{cases}$$

With these definitions, it is obvious that

$$\mathrm{C{\small ONDITION}}\ (\dagger) \iff \left[\widehat{up}(\overline{\theta}) \cap \overline{\theta'} \neq \emptyset \implies \widehat{up}^{-1}(\overline{\theta'}) \cap \overline{\theta} = \overline{\theta}\right]$$

Since our updates do not compute anything, from a given equivalence class $\overline{\theta}$, the sets $\widehat{up}(\overline{\theta})$ and $\widehat{up}^{-1}(\overline{\theta})$ can be obtained "easily". Indeed, $\widehat{up}$ has just for effect to put in all the registers of up any value in $\mathcal{D}$ and $\widehat{up}^{-1}$ is just a projection on the registers which are not updated by up. Therefore, as soon as we are able to decide both the emptiness of $\widehat{up}(\overline{\theta}) \cap \overline{\theta'}$ and the equality $\widehat{up}^{-1}(\overline{\theta'}) \cap \overline{\theta} = \overline{\theta}$, the condition (†) becomes decidable.
Note that it is in particular the case when the equivalence $\sim$ is given by a set of linear inequations.

6 Extensions of the Model

6.1 Non-Deterministic Models

Up to now, we only considered models that are deterministic, *i.e.* for each data word, there is a unique possible execution on it. Now, we will consider a non-deterministic version of the models. We thus define non-deterministic data automata as in Definition 2, but without the determinism condition. We also say that a finite monoid M non-deterministically recognizes a data language L whenever there exists, k, $\sim$, φ as in Definition 1, but for each $(m, a) \in M \times \Sigma$, there exists a finite set of updates $U_{m,a}$ (instead of a unique update $up_{m,a}$) such that the conditions in Definition 1 hold. Some properties which are true for deterministic data automata are also true for non-deterministic data automata:

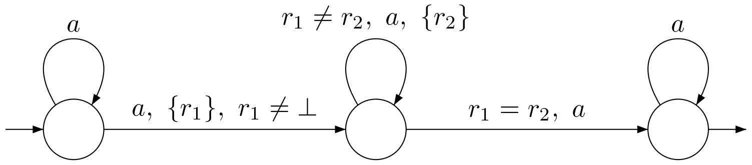
Proposition 4. *Let L be a data language over Σ and $\mathcal{D}$. Then,*

- *L is non-deterministically recognized by a finite monoid if and only if it is recognized by a non-deterministic data automaton.*

We say that L is nd-recognizable *whenever L is accepted by some non-deterministic data automaton.*

- *Condition (†) ensures the decidability of the emptiness problem, i.e. if L is recognized by a non-deterministic data automaton that satisfies the condition (†), then we can test for its emptiness.*
- *The class of nd-recognizable data languages is strictly more expressive than the class of recognizable data languages.*

Proof (Sketch for the last point). Consider the data language L accepted by the following non-deterministic data automaton:



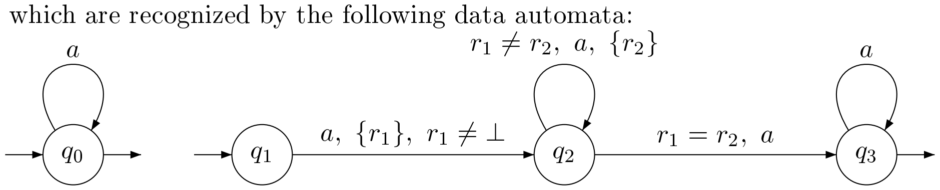
We have that $L = \{(a, d_1) \ldots (a, d_n) \mid \exists 1 \leq i < j < n,\ d_i = d_j \neq \perp\}$. Moreover, one can prove that L is not recognized by any (deterministic) data automaton. □

Corollary 1. *The class of recognizable data languages is not closed under concatenation.*

Proof. Consider the previous data language L. Although it is not recognizable, this language is the concatenation of the two following recognizable data languages:

$$\{(a, d_1) \ldots (a, d_p) \mid d_i \in \mathcal{D}\} \ \text{ and } \ \{(a, d_0) \ldots (a, d_n) \mid \exists 1 \leq j < n,\ d_j = d_0\}$$

which are recognized by the following data automata:



□

Proposition 5. *The class of nd-recognizable data languages is closed under union, intersection, concatenation and finite iteration. It is not closed under complementation.*

6.2 More General Updates

The updates used in the model are very simple, we can only "write a data in a memory", but we cannot perform any calculation. So, the question is: does all what precedes generalizes to models in which updates can perform calculations. In this section, an update is now a general function $up : \mathcal{D}^k \times \mathcal{D} \longrightarrow \mathcal{D}^k$.

Considering the simple updates of registers, we showed that the monoid played a very important role: "different" monoids do not recognize the same data languages. Extending the updates, the relevance of the monoid is lost.

Proposition 6. *Let L be a language over the finite alphabet Σ. Assume that L is recognized by a finite monoid M. Then the data language*

$$L_M = \{(a_1, m_1) \dots (a_n, m_n) \mid a_1 \dots a_n \in L\}$$

over Σ and M is recognized by the monoid $N = \{1, x, y\}$ with $zx = x$ and $zy = y$.

Proof. We assume that $L \subseteq \Sigma^*$ is recognized by M. There exists a morphism $\varphi : \Sigma^* \longrightarrow M$, a subset $P \subseteq M$ such that $L = \varphi^{-1}(P)$. Let us now define $k = 1$ (there is only one register) and $\mathcal{D} = M$. Then, for each $z \in N$, for each $a \in \Sigma$, we define $up_{z,a} : M \times M \longrightarrow M$ by $up_{z,a}(m, d) = m\varphi(a)$. We define also a morphism $\psi : (\Sigma \times M)^* \longrightarrow N$ by:

$$\psi(a, m) = \begin{cases} x \text{ if } m \in P \\ y \text{ if } m \in M \setminus P \end{cases}$$

Then, using this construction, we can prove that N recognizes the data language L_M. □

However, allowing more general updates like functions $\mathcal{D}^k \times \mathcal{D} \longrightarrow \mathcal{D}^k$, the results on equivalence between monoids and automata and on decidability are always true, because these results do not depend on the updates.

7 Comparison with Timed Automata

One of the main motivation of this work was to find an algebraic characterization of timed languages. It is clear that if we consider as data domain $\mathcal{D}$ a classical time domain, then our data languages reduce to timed languages (since we can easily handle the monotonicity condition on time).

Proposition 7. *Let $\mathcal{A}$ be a (deterministic) timed automata with n clocks. There exists a (deterministic) data automaton with $2n + 2$ registers which recognizes the same language.*

Proof (Sketch). We assume that the definition of a (deterministic) timed automaton is known, otherwise, we refer to [AD94].
Let us consider a deterministic timed automaton $\mathcal{A}$ with n clocks, $\{x_1, \dots, x_n\}$. Without loss of generality, we can assume that there exists an equivalence on

$\mathcal{D}^n$, namely $\equiv$, such that if g is a guard appearing in $\mathcal{A}$, then g is an equivalence class of $\equiv$. A clock x_0 is added to the set of clocks and represents the universal time, *i.e.* x_0 is never reset in $\mathcal{A}$. We construct a (deterministic) data automaton $\mathcal{B}$ with $2n+2$ registers in the following way:
The set of states of $\mathcal{B}$ is $Q \times \mathcal{F}$ where Q is the set of states of $\mathcal{A}$ and $\mathcal{F}$ is the set of functions $f : \{x_0, \ldots, x_n\} \longrightarrow \{0, \ldots, 2n+1\}$ such that for all $0 \leq i \leq n$, $f(x_i) \in \{i, n+1+i\}$. Intuitively, the value of the clock x_i will be alternatively kept by the two registers i and $n+1+i$.
The equivalence $\sim$ in $\mathcal{B}$ is defined by:

$$(\theta_i)_{0\leq i\leq 2n+1} \sim (\theta'_i)_{0\leq i\leq 2n+1}$$
$$\Updownarrow$$
$$\begin{cases} \forall f \in \mathcal{F},\ (\theta_{f(x_0)} - \theta_{f(x_i)})_{1\leq i\leq n} \equiv (\theta'_{f(x_0)} - \theta'_{f(x_i)})_{1\leq i\leq n}, \\ \quad\left(\theta_0 < \theta_{n+1} \iff \theta'_0 < \theta'_{n+1}\right), \\ \quad\left(\theta_0 > \theta_{n+1} \iff \theta'_0 > \theta'_{n+1}\right) \end{cases}$$

Consider a transition in $\mathcal{A}$:

$$q \xrightarrow{\ g,\ \ a,\ \ C := 0\ } q'$$

For each function f in $\mathcal{F}$, we construct transitions in $\mathcal{B}$ in the following way:

$$q, f \xrightarrow{\ \overline{\theta},\ \ a,\ \ \alpha,\ \ \overline{\theta'}\ } q', f'$$

where

- $\overline{\theta}$ is any equivalence class of $\sim$,
- $\alpha = \{0, 1, \ldots, 2n+1\} \setminus \{f(x_0), \ldots, f(x_n)\}$,
- $f' \in \mathcal{F}$ is such that

$$f'(x_0) = \begin{cases} 0 & \text{if } f(x_0) = n+1 \\ n+1 & \text{if } f(x_0) = 0 \end{cases}$$
$$f'(x_i) = \begin{cases} f(x_i) & \text{if } x_i \notin C \\ n+1+i & \text{if } x_i \in C \text{ and } f(x_i) = i \\ i & \text{if } x_i \in C \text{ and } f(x_i) = n+1+i \end{cases}$$

- $\overline{\theta'}$ is any equivalence class of $\sim$ such that

$$(\beta_i)_{0\leq i\leq 2n+1} \in \overline{\theta'} \implies (\beta_{f'(x_0)} - \beta_{f(x_i)})_{1\leq i\leq n} \in g \text{ and } \beta_{f'(x_0)} > \beta_{f(x_0)}$$

The timed automaton $\mathcal{B}$ that we just constructed is deterministic and recognizes the same timed language as $\mathcal{A}$. $\square$

Hence any timed language accepted by some deterministic timed automaton (as defined by [AD94]) is also recognized by a data automaton with the timed domain as data domain.
Conversely, data automata allow to recognize a much larger class of languages. Indeed all the languages accepted by the extension of timed automata proposed in [CG00] are also recognized by data automata. And even, for example, the language $\{(a,\tau)(a,2\tau)\dots(a,n\tau) \mid \tau \in \mathbb{Q}_+\}$ is recognized by a data automaton whereas it is known that this language cannot be recognized by a timed automaton, even in the extension proposed by [DZ98].
We can also define more exotic languages which are monoid recognizable as for instance the set $\{(a,t_1)\dots(a,t_n) \mid \forall i,\ t_i$ is a prime number$\}$. Namely, it suffices to consider a monoid with 2 elements, 1 register and an equivalence relation of index 2. The first class contains all the prime numbers and the second class all the others. Note that the condition (†) holds.

8 Conclusion

We have proposed in this paper a notion of monoid recognizability for data languages. We also gave an automaton characterization of this notion. Hence, the picture for data languages is rather close to the one for classical formal languages. As an instance of our results, we can deal with timed languages.
This theory has now to be developed. For instance, a notion of aperiodic data language can naturally be defined and has to be studied.
In the timed framework, any timed language recognized by deterministic timed automata is monoid recognizable. But the exact relations with the numerous sets of timed languages that have been proposed in the literature, see for instance [HRS98], have to be investigated.
Another interesting direction will also consist in understanding the exact relation between the power of the monoid and the power of the updates. In this paper, we have investigated the two extreme cases. If updates on registers can only choose to store or to skip a data, then the structure of the monoid is crucial. On the contrary, if the updates can do heavy computations, then the monoid is nearly useless. All cases in between have still to be studied.

References

[ABB80] Autebert, J.-M., Beauquier, J., and Boasson, L. *Langages sur des alphabets infinis*. Discrete Applied Mathematics, vol. 2:1–20, 1980.

[ACM97] Asarin, E., Caspi, P., and Maler, O. *A Kleene Theorem for Timed Automata*. In *Proc. 12th IEEE Symp. Logic in Computer Science (LICS'97)*, pp. 160–171. IEEE Computer Society Press, June 1997.

[AD94] Alur, R. and Dill, D. *A Theory of Timed Automata*. Theoretical Computer Science, vol. 126(2):183–235, 1994.

[AFH94] Alur, R., Fix, L., and Henzinger, T. A. *Event-Clock Automata: a Determinizable Class of Timed Automata*. In *Proc. 6th Int. Conf. Computer*

Aided Verification (CAV'94), vol. 818 of *Lecture Notes in Computer Science*, pp. 1–13. Springer-Verlag, June 1994.

[Asa98] Asarin, E. *Equations on Timed Languages*. In *Hybrid Systems: Computation and Control*, vol. 1386 of *Lecture Notes in Computer Science*. Springer-Verlag, Apr. 1998.

[BDFP00] Bouyer, P., Dufourd, C., Fleury, E., and Petit, A. *Are Timed Automata Updatable?*. In *Proc. 12th Int. Conf. Computer Aided Verification (CAV'2000)*, vol. 1855 of *Lecture Notes in Computer Science*, pp. 464–479. Springer-Verlag, July 2000.

[BP99] Bouyer, P. and Petit, A. *Decomposition and Composition of Timed Automata*. In *Proc. 26th Int. Coll. Automata, Languages, and Programming (ICALP'99)*, vol. 1644 of *Lecture Notes in Computer Science*, pp. 210–219. Springer-Verlag, July 1999.

[BP01] Bouyer, P. and Petit, A. *A Kleene/Büchi-like Theorem for Clock Languages*. Journal of Automata, Languages and Combinatorics, 2001. To appear.

[BPT01] Bouyer, P., Petit, A., and Thérien, D. *An Algebraic Characterization of Data and Timed Languages*. Research report LSV-01-1, Laboratoire Spécification et Vérification, Ecole Normale Supérieure de Cachan, 2001.

[CG00] Choffrut, C. and Goldwurm, M. *Timed Automata with Periodic Clock Constraints*. Journal of Automata, Languages and Combinatorics, vol. 5(4):371–404, 2000.

[CGP99] Clarke, E., Grumberg, O., and Peled, D. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[DZ98] Demichelis, F. and Zielonka, W. *Controlled Timed Automata*. In *Proc. 9th Int. Conf. Concurrency Theory (CONCUR'98)*, vol. 1466 of *Lecture Notes in Computer Science*, pp. 455–469. Springer-Verlag, Sep. 1998.

[HKWT95] Henzinger, T. A., Kopke, P. W., and Wong-Toi, H. *The Expressive Power of Clocks*. In *Proc. 22nd Int. Coll. Automata, Languages, and Programming (ICALP'95)*, vol. 944 of *Lecture Notes in Computer Science*, pp. 335–346. Springer-Verlag, July 1995.

[HRS98] Henzinger, T. A., Raskin, J.-F., and Schobbens, P.-Y. *The Regular Real-Time Languages*. In *Proc. 25th Int. Coll. Automata, Languages, and Programming (ICALP'98)*, vol. 1443 of *Lecture Notes in Computer Science*, pp. 580–591. Springer-Verlag, July 1998.

[KF94] Kaminski, M. and Francez, N. *Finite-Memory Automata*. Theoretical Computer Science, vol. 134:329–363, 1994.

[Min67] Minsky, M. *Computation: Finite and Infinite Machines*. Prentice Hall Int., 1967.

[Pin86] Pin, J.-E. *Varieties of Formal Languages*. North Oxford, London et Plenum, New-York, 1986.

[RS97] Rozenberg, G. and Salomaa, A., eds. *Handbook of Formal Languages*. Springer-Verlag, 1997.

[Wil94] Wilke, T. *Specifying Timed State Sequences in Powerful Decidable Logics and Timed Automata*. In *Proc. 3rd Int. Symp. Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'94)*, vol. 863 of *Lecture Notes in Computer Science*, pp. 694–715. Springer-Verlag, Sep. 1994.