An Algebraic Approach to Data Languages 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]. Following the terminology of Henzinger *et al.* [HRS98], we thus get a *fully decidable* class of languages, *i.e.* a class of languages closed under boolean operations and for which emptiness is decidable. 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. Aside of its simplicity, this equivalence leads to several beautiful theorems making a bridge between formal languages and algebra. A most famous example is

† 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

due to Schützenberger who showed that the class of languages recognized by aperiodic monoids coincides with the class of star-free languages [Sch65]. Note that this result, together with a theorem of Kamp [Kam68], yields an algorithm to decide whether a recognizable language can be defined by a linear temporal logic formula.

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 problem, but is not closed under complement. Several logical characterizations [Wil94,HRS98] or even Kleene-like theorems [ACM97,Asa98,BP99,ACM01,BP01] have been proposed for the whole class of timed automata but no purely algebraic one. Besides, 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, as 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 $d \in \mathcal{D}$. As we will explain in details in Section 3, the monoid recognizability for data languages can not be obtained through the simple notion of morphism, as is the case for regular formal languages. We propose in this paper another mechanism, based on registers. We obtain in this way, for any set of actions Σ and any set of data $\mathcal{D}$, a class of so-called “monoid recognizable” data languages.

Note that similar situations arose in other contexts. For example, it has been shown in [Bar89] that the class NC^1 of languages recognized by boolean circuits of logarithmic depth can be characterized in algebraic terms, using the notion of *programs of polynomial length* instead of morphisms. Another example is the algebraic characterization of PSPACE using the *leaf languages* approach [HLS$^+$93]. The class of monoid recognizable languages is closed under boolean operations. In this class, two hierarchies naturally occur, depending on which monoid and how many registers are used. 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. This implies that increasing the number of registers cannot help if the monoid is not powerful enough. On the contrary, if the monoid M is fixed, then the number of registers can be bounded by some constant depending only on Σ and M.

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 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].

Hence the class of data languages recognized by monoids, where the condition above holds, forms a fully decidable class of data languages. 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 article is a long version of [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 k*-register update*, or simply an *update*, $\overline{up}$, is defined by a subset of $\{1, \ldots, k\}$. This update induces an application 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

INtuitively, the principle of monoid recognizability consists in mapping the words of a free monoid Γ^* (where Γ can be either finite or infinite) into a finite monoid M and to define a language by the set of words which are mapped on a given subset F of M. Of course, "interesting" mappings will allow to deduce properties of the language from properties of a monoid which recognizes it.

In the formal language case, the best known and most studied method to define monoid recognizability is to use simply a morphism φ from Γ^* into M. In such a framework, to decide if a word $w \in \Gamma^*$ belongs or not to the language L, it is sufficient to run the procedure of Figure 1. Then, it can be shown that a word language is monoid recognizable if and only if it is regular. Apart from its simplicity, this equivalence leads to several beautiful theorems making a bridge between formal languages and algebra [Pin86].

% Initialization
$m := 1$
% Computation
While not end of w do
    Read the next letter a of the word
    Compute $\varphi(a)$
    Compute $m := m\varphi(a)$
Endwhile
% Output
If $m \in F$ then output *"yes"* else output *"no"*

Fig. 1. The mechanism for formal languages

Unfortunately, using such a simple mechanism for data languages is hopeless if we want an interesting class of languages. Indeed, 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.
Hence, we need some kind of auxiliary memory to take care of the values of the data. This will lead of course to a more complicated mechanism than morphisms. Here, we propose to use a finite number of registers as auxiliary memory. Roughly, and intuitively, a data word w will be in the language if and only if the procedure of Figure 2 answers *"yes"*.
We need now to precise how the registers are updated and how the successive values in the monoid are computed.

% Initialization
$m := 1$
all the registers are set to $\perp$
% Computation
While not end of w do
    Read the next letter (a, d) of the word
    Update the registers with the new data d using a and m
    Compute the new value m in the monoid from the old value, a and the registers
Endwhile
% Output
If $m \in F$ then output *"yes"* else output *"no"*

Fig. 2. A mechanism that uses registers

In order to maintain the relevance of the monoid, the whole mechanism has to be very simple and, in particular, has to be unable to perform any computation. To this purpose, we first use the notion of updates as defined in the previous section. Then, the new value in the monoid does not depend on the exact data stored in the registers but only on a *finite and bounded* information from these registers.
All this leads to the formal definition of a k-registers-mechanism:

Definition 1. *A* k-registers-mechanism *over a finite monoid* M *is a triple* $\rho = [(up_{m,a})_{m \in M, a \in \Sigma}, \sim, \varphi]$ *where:*

- *for each* $(m, a) \in M \times \Sigma$*,* $up_{m,a}$ *is a* k*-registers update,*
- $\sim$ *is an equivalence of finite index on* $\mathcal{D}^k$*,*
- φ *is a morphism from* $(\Sigma \times \mathcal{D}^k / \sim)^*$ *into* M*.*

Note that if $k = 0$ a k-registers-mechanism reduces to a morphism from Σ^* into M.
If ρ is a k-registers-mechanism over a finite monoid M and $w = (a_1, d_1)(a_2, d_2) \ldots (a_n, d_n)$ is a data word of $(\Sigma \times \mathcal{D})^*$, the computation of ρ over w is simply the element of M given by the computation depicted on Figure 3 (where θ is an array of size k corresponding to the k registers and $\overline{\theta}$ denotes the equivalence class by $\sim$ of the registers θ).
The output of this computation depicted on Figure 3 is denoted by $\rho(w)$.
In the following, if $w = (a_1, d_1)(a_2, d_2) \ldots (a_n, d_n)$ is a data word of $(\Sigma \times \mathcal{D})^*$, the value of θ at step i of the loop is denoted by θ_i and the value of m at step i is denoted by m_i.
>From this definition of k-registers-mechanism, we can now define the notion of data language recognized by a monoid M.

Definition 2. *Let* L *be a data language over* Σ *and* $\mathcal{D}$ *and let* M *be a finite monoid. We say that* M recognizes L *if there exists a subset* F *of* M *and a* k*-registers-mechanism* $\rho = [(up_{m,a})_{m \in M, a \in \Sigma}, \sim, \varphi]$ *such that:*

$$L = \rho^{-1}(F)$$

```
% Initialization
m := 1
∀1 ≤ j ≤ k, θ[j] := ⊥
% Computation
For i := 1 to n do
        θ := up_{m,a_i}(θ, d_i)
        m := mφ(a_i, θ̄)
Endfor
% Output
Output m
```

Fig. 3. Computation in a k-registers-mechanism

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 $(a_1, d_1) \ldots (a_n, d_n)$ is in the language then for any bijection ν from $\mathcal{D}$ into $\mathcal{D}$, $(a_1, \nu(d_1)) \ldots (a_n, \nu(d_n))$ has also to be in the language. We do not require at all such a property.

Example 1. The data language $L = \{(a,d)(a,d') \mid d \neq \bot, d \neq d'\}$ 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 $x \in M$.
To this aim, we use two registers. Thus, we define the 2-registers-mechanism $\rho = [(up_{m,a})_{m \in M, a \in \Sigma}, \sim, \varphi]$ in the following way:

- The updates are $up_{1,a} = \{1\}$ and if $z \in M \setminus \{1\}, up_{z,a} = \{2\}$.
- $\sim$ has two equivalence classes, namely $\overline{\theta_{\neq}} = \{(d,d') \mid d \neq d'\}$ and $\overline{\theta_{=}} = \mathcal{D}^2 \setminus \overline{\theta_{\neq}}$.
- The morphism $\varphi : (\{a\} \times \{\overline{\theta_{\neq}}, \overline{\theta_{=}}\})^* \longrightarrow M$ is defined by $\varphi(a, \overline{\theta_{\neq}}) = y$ and $\varphi(a, \overline{\theta_{=}}) = 0$.

With these definitions, L is accepted by M using ρ (with $F = \{y^2\}$).
As an example of computation, consider the data word $(a,d)(a,d')$ with $d \neq \bot$ and $d \neq d'$.

In the monoid M	$1_M \xrightarrow[d]{a}$	$y \xrightarrow[d']{a}$	y^2
Values of the two registers	$\begin{pmatrix} \bot \\ \bot \end{pmatrix}$	$\begin{pmatrix} d \\ \bot \end{pmatrix}$	$\begin{pmatrix} d \\ d' \end{pmatrix}$
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. 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 $\mathcal{L}_k = \bigcup_M \mathcal{L}_{M,k}$.

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 $\mathcal{L}_{M_1 \times M_2, k_1+k_2}$.*

Proof. **The set $\mathcal{L}_{M,k}$ is closed under complementation.**
Let $L \in \mathcal{L}_{M,k}$ be a data language. Assume ρ is a k-registers-mechanism and F is a subset of M such that $L = \rho^{-1}(F)$ (as in Definition 2). Let $(a_1, d_1) \dots (a_p, d_p)$ be a data word. Then the following equivalence holds:

$$(a_1, d_1) \dots (a_p, d_p) \in L \iff \rho((a_1, d_1) \dots (a_p, d_p)) \in F$$

Thus,

$$(a_1, d_1) \dots (a_p, d_p) \notin L \iff \rho((a_1, d_1) \dots (a_p, d_p)) \in M \setminus F$$

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 $\mathcal{L}_{M_1 \times M_2, k_1 + k_2}$.
Let $L_1 \in \mathcal{L}_{M_1,k_1}$ and $L_2 \in \mathcal{L}_{M_2,k_2}$.
Assume that for $i = 1, 2$, $\rho_i = [(up^{(i)}_{m,a})_{m \in M, a \in \Sigma}, \sim_i, \varphi_i]$ is a k_i-registers-mechanism and $F_i \subseteq M_i$ is a subset of M_i such that $L_i = \rho_i^{-1}(F_i)$.
We define $k = k_1 + k_2$, $M = M_1 \times M_2$ with the monoid law

$$(m_1, m_2) \cdot (m_1', m_2') = (m_1 m_1', m_2, m_2')$$

We also define the equivalence $\sim$ on $\mathcal{D}^k$ by

$$\theta_1\theta_2 \sim \theta_1'\theta_2' \iff \theta_1 \sim_1 \theta_1' \text{ and } \theta_2 \sim_2 \theta_2'$$

and the morphism $\varphi(a, \overline{\theta_1.\theta_2}) = (\varphi_1(a, \overline{\theta_1}), \varphi_2(a, \overline{\theta_2}))$. We finally define for each $m \in M$ and each $a \in \Sigma$ the k-register update $up_{m,a}$ by

$$up_{m,a} = up^{(1)}_{m,a} \cup (k_1 + up^{(2)}_{m,a})$$

The language $L_1 \cup L_2$ is then recognized by M using the mechanism $\rho = [(up_{m,a})_{m \in M, a \in \Sigma}, \sim, \varphi]$ for $F = (F_1 \times M_2) \cup (M_1 \times F_2)$ whereas $L_1 \cap L_2$ is recognized by M using ρ for $F = F_1 \times F_2$. $\square$

>From the algebraic point of view, the soundness of our definition is assessed by the following result, which shows that the structure of the monoid is really fundamental and plays a role similar to what happens in the framework of formal languages. Note that, in particular, this result proves that increasing the number of registers cannot help if the monoid is not powerful enough.
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 the data language

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

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 $F \subseteq M$ such that $L = \varphi^{-1}(F)$. It is easy to see that M recognizes $L_{\mathcal{D}}$ (using no register).
Assume that $L_{\mathcal{D}}$ is recognized by M using the k-registers-mechanism $\rho = [(up_{m,a})_{m\in M, a\in\Sigma}, \sim, \varphi]$ and $F \subseteq M$. 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 $(\Sigma \times \mathcal{D}^k/\sim)^*$, considering the computation of Figure 3, 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,

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

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. The sequence $(\mathcal{L}_k(\Sigma, \mathcal{D}))_k$ is strictly monotonic.*
2. If M is a fixed finite monoid, the sequence $(\mathcal{L}_{M,k}(\Sigma, \mathcal{D}))_k$ collapses, more precisely, $\mathcal{L}_{M,2^{|\Sigma\times M|}-1} = \mathcal{L}_{M,2^{|\Sigma\times M|}}$.

Proof. 1. We will prove that the data language

$$L_k = \{(a, d_1) \ldots (a, d_n) \mid i \equiv j \mod (k-1) \implies d_i = d_j\}$$

over $\{a\}$ and $\mathcal{D}$ ($\mathcal{D}$ is supposed to be infinite) is recognized by a finite monoid using k registers, but is recognized by no finite monoid using strictly less than k registers.

To prove that L_k is recognized by a finite monoid with k registers, we first define the finite monoid M as the one containing 1, 0 and generated by

$$\{x_P \mid P \text{ partition of } \{0, \ldots, k-1\}\}$$

such that for every partitions $P_1, \ldots, P_{2k-1}$,

$$\begin{cases} x_{P_1} \ldots x_{P_{k-1}} x_{P_k} \ldots x_{P_{2k-1}} = x_{P_1} \ldots x_{P_{k-1}} \\ \qquad\qquad\qquad\qquad \text{if } P_{k+l} \text{ is compatible with } \{l, l+1\} \\ x_{P_1} \ldots x_{P_{k-1}} x_{P_k} \ldots x_{P_k+l} = 0 \\ \qquad\qquad\qquad\qquad \text{if } P_{k+l} \text{ is not compatible with } \{l, l+1\} \end{cases}$$

We also define a k-register-mechanism $\rho = [(up_{m,a})_{m \in M, a \in \Sigma}, \sim, \varphi]$ defining separately the three components.
The equivalence relation $\sim$ on $\mathcal{D}^k$ is defined by: for each partition P of $\{0, \ldots, k-1\}$,

$$\overline{\theta_P} = \{\theta \in \mathcal{D}^k \mid i \text{ and } j \text{ are equivalent in } P \iff \theta[i] = \theta[j]\}$$

The morphism φ is defined by $\varphi(a, \overline{\theta_P}) = x_P$. We finally define updates $up_{x_{P_1} \ldots x_{P_{k+l}}, a} = \{l\}$ and $up_{0,a} = \emptyset$. Using all this, L is recognized by M.

We now want to prove that L is not recognized by any finite monoid with strictly less than k registers. Assume it is. Let $(a_1, d_1) \ldots (a_p, d_p)$ be a data word, we define the sequence $(\theta_i)_{i=0 \ldots p}$ as in Figure 3. We then say that the data word $(a_1, d_1) \ldots (a_p, d_p)$ is read on the path $(\overline{\theta_1}, \ldots, \overline{\theta_k})$. For each finite sequence $c = (\overline{\theta_1}, \ldots, \overline{\theta_k})$, we define the set

$$E(c) = \{(a_1, d_1) \ldots (a_k, d_k) \text{ read on the path } c\}$$

For each such path, there exists an integer $n(c)$ such that the data indexed by $n(c)$ is not kept in mind at the end of the path (*i.e.* if θ_k is the last data vector, it does not contain the data indexed by $n(c)$) – there exists such an integer $n(c)$ because if $l < k$, l registers can not keep in mind k data. We define for each $1 \leq i \leq k$ a partition P_i of $(\Sigma \times \mathcal{D})^k$ by: $P_i = \bigsqcup_j H_i^{(j)}$ such that $w,\ w' \in (\Sigma \times \mathcal{D})^k$ are in the same $H_i^{(j)}$ if and only if they only differ from the i^{th} character. We now restrict the data to a finite subset D of $\mathcal{D}$ such that $|D| > |\{c \text{ path of length } k\}|$. In this case there exists a path c_0 such that

$$|E(c_0)| \geq \frac{|D|^k}{|\{c \text{ path of length } k\}|}$$

For each i, we have that $|\{j \mid H_i^{(j)}$ is a part of $P_i\}| = |D|^{k-1}$. Thus, as $|D| > |\{c$ path of length $k\}|$, there exists two data words $(a, d_1) \ldots (a, d_{n(c_0)}) \ldots (a, d_k)$ and $(a, d_1) \ldots (a, d'_{n(c_0)}) \ldots (a, d_k)$ in $E(c_0)$ such that:

$$\begin{array}{ccccccccccc}
(a,d_1) & & \ldots & & (a,d_{n(c_0)}) & & \ldots & & (a,d_k) & \\
 & \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow \\
 & \theta_1 & & \theta_{n(c_0)-1} & & \theta_{n(c_0)} & & \theta_{k-1} & & \theta_k \\
 & = & & = & & \sim & & \sim & & = \\
 & \theta'_1 & & \theta'_{n(c_0)-1} & & \theta'_{n(c_0)} & & \theta'_{k-1} & & \theta'_k \\
 & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \\
(a,d_1) & & \ldots & & (a,d'_{n(c_0)}) & & \ldots & & (a,d_k) &
\end{array}$$

As $\theta_k = \theta'_k$, if the data word $(a, d_1) \ldots (a, d_{n(c_0)}) \ldots (a, d_k)w$ is in L then the data word $(a, d_1) \ldots (a, d'_{n(c_0)}) \ldots (a, d_k)w$ is also in L. However, it is not the case. Thus L is not recognized by any finite monoid with strictly less than k registers.

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

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

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.

More formally, let $(a_1, d_1) \ldots (a_p, d_p)$ be a data word. An execution of this word on M is defined by a sequence $(a_1, \theta_1) \ldots (a_p, \theta_p)$ as in Figure 3. We can note that for each $1 \leq h \leq p$, for each $i,\ j,\ i \cong j \implies \theta_h[i] = \theta_h[j]$ (where $\theta_h[i]$ denotes the i^{th} component of the data vector θ_h). We define a function $\mu : \{1 \ldots k\} \longrightarrow \{1 \ldots |\cong|\}$ such that $\mu(i) = \mu(j) \iff i \cong j$ and if $\lambda(i) = \emptyset$, then $\mu(i) = 1$. We can note that $|\cong| \leq 2^{|\Sigma \times M|}$. We define on $\mathcal{D}^h$ (or on $\mathcal{D}^{h-1}$ if there exists i such that $\lambda(i) = \emptyset$) an equivalence relation $\approx$ in the following way: $\beta \approx \beta' \iff \theta \sim \theta'$ if $\theta_i = \beta_{\mu(i)}$. The equivalence $\approx$ has a finite index and we can define a morphism $\psi : (\Sigma \times \mathcal{D}^h/_{\approx}) \longrightarrow M$ by $\psi(a, \overline{\beta}) = \varphi(a, \overline{\theta})$ (following the definition of $\approx$). The data language L is recognized by M using the h-register-mechanism $[(\mu(up_{m,a}))_{m \in M, a \in \Sigma}, \approx, \psi]$.

□

Remark 2. This proposition shows in particular that for a fixed monoid and a fixed alphabet, the number of registers can be bounded. This result becomes of course false if only the monoid is fixed.

For instance, let M_0 be the finite monoid $\{1, 0, x\}$ with $x^2 = x$. For any integer k, let us define $\Sigma_k = \{a_0, a_1, \ldots, a_{k-1}\}$. For any data word $u \in (\Sigma_k \times \mathcal{D})^*$ and any

$i = 1 \dots k-1$, we set $\mu_i(u)$ as the data d (if it exists) such that $u = u'\ (a_i, d)\ u''$ where u'' does not contain any a_i; and we set $\mu_i(u) = \perp$ if the data does not exist. For each k, we define now the data language

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

We claim that

$$L_k \in \mathcal{L}_{M_0,k}(\Sigma_k, \mathcal{D}) \setminus \bigcup_{k'<k} \mathcal{L}_{k'}$$

Indeed, define the k-registers-mechanism $\rho = [(up_{m,a})_{m \in M, a \in \Sigma}, \sim, \varphi]$ where $\sim$ is defined by the two equivalence classes

$$\overline{\theta} = \{\theta \in \mathcal{D}^k \mid \exists 1 \leq i \leq k-1 \text{ such that } \theta_0 = \theta_i\} \text{ and } \overline{\theta'} = \mathcal{D}^k \setminus \overline{\theta}$$

The updates are defined by $up_{z,a_i} = \{i\}$ if $0 \leq i \leq k-1$ and $z \in M$. The morphism $\varphi : (\mathcal{D} \times \{\overline{\theta}, \overline{\theta'}\})^* \longrightarrow M$ is finally defined by $\varphi(a_0, \overline{\theta}) = x$, $\varphi(a_0, \overline{\theta'}) = 0$ and $\varphi(a_i, -) = x$ if $1 \leq i \leq k-1$. Using this construction, M_0 recognizes L_k.

To prove that L_k is not recognized by any monoid using strictly less than k registers can be done without difficulty using a construction similar to the one presented in the proof of the previous Proposition.

4 Data Automata

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

Definition 3. *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 hypotheses *hold:*

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

$$q_0 \xrightarrow[d_1]{g_1, a_1, up_1, g'_1} q_1 \xrightarrow[d_2]{g_2, a_2, up_2, g'_2} q_2 \quad \dots \quad q_{n-1} \xrightarrow[d_n]{g_n, a_n, up_n, g'_n} q_n$$

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

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

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,

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

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

$$\begin{tikzcd} {} \arrow[r] & q_0 \arrow[rr, "{\overline{\theta_=},\ a,\ \{r_1\},\ \overline{\theta_{\neq}}}"] & & q_1 \arrow[rr, "{\overline{\theta_{\neq}},\ a,\ \{r_2\},\ \overline{\theta_{\neq}}}"] & & q_2 \arrow[r] & {} \end{tikzcd}$$

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. **If Implication.** First, assume that $L \subseteq (\Sigma \times \mathcal{D})^*$ is recognized by a finite monoid M using the k-register-mechanism $\rho = [(up_{m,a})_{m \in M, a \in \Sigma}, \sim, \varphi]$ and the accepting set $F \subseteq M$. 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 k-registers-mechanism,
- $Q = M$ and $q_0 = 1_M$
- $T = \{(m, g, a, up_{m,a}, g', m') \mid m \in M,\ g, g' \in \mathcal{D}^k/\!\sim,\ a \in \Sigma,\ m' = m\varphi(a, g')\}$.

We will prove that $\mathcal{A}$ is a valid deterministic data automaton and that $L(\mathcal{A}) = L$. First, note that if m is a state of $\mathcal{A}$, if g is a given equivalence class and if a is an action, there is a unique update up such that $\mathcal{A}$ has a transition $(q, g, a, up, -, -)$, namely $up = up_{m,a}$. Moreover if g' is an equivalence class, the state m' such that $(m, g, a, up_{m,a}, g', m')$ is a transition of $\mathcal{A}$ is uniquely determined. Thus, $\mathcal{A}$ satisfies the determinism hypothesis of Definition 3.

Assume the data word $w = (a_1, d_1) \dots (a_p, d_p)$ is in L. The sequences $(\theta_i)_{i=0\dots n}$ and $(m_i)_{i=0\dots n}$ defined by:

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

satisfy $m_n \in F$. Consider the following run in $\mathcal{A}$

$$\begin{array}{ccccccc} 1_M & \xrightarrow[d_1]{\overline{\theta_0}, a_1, up_{1_M, a_1}, \overline{\theta_1}} & m_1 & \xrightarrow[d_2]{\overline{\theta_1}, a_2, up_{m_1, a_2}, \overline{\theta_2}} & m_2 \ \dots & \xrightarrow[d_n]{\overline{\theta_{n-1}}, a_n, up_{m_{n-1}, a_n}, \overline{\theta_n}} & m_n \\ \theta_0 & & \theta_1 & & \theta_2 & & \theta_n \end{array}$$

It is a valid accepting path for w in $\mathcal{A}$ because $\theta_{i+1} = up_{m_i, a_{i+1}}(\theta_i, d_{i+1})$ and $m_{i+1} = m_i \varphi(a_{i+1}, \overline{\theta_{i+1}})$.

Conversely, suppose $w = (a_1, d_1) \dots (a_p, d_p)$ is in $L(\mathcal{A})$. Consider the run

$$\begin{array}{ccccccc} 1_M & \xrightarrow[d_1]{g_1, a_1, up_1, g_1'} & m_1 & \xrightarrow[d_2]{g_2, a_2, up_1, g_2'} & m_2 \ \dots & \xrightarrow[d_n]{g_n, a_n, up_n, g_n'} & m_n \\ \theta_0 & & \theta_1 & & \theta_2 & & \theta_n \end{array}$$

where $\theta_0 = \perp^k$ and $\theta_{i+1} = up_{m_i, a_{i+1}}(\theta_i, d_{i+1})$ satisfy $\overline{\theta_i} = g_{i+1}$ and $\overline{\theta_{i+1}} = g'_{i+1}$. Thus, we have that $\theta_0 = \perp^k$, $\theta_{i+1} = up_{m_i, a_{i+1}}(\theta_i, d_{i+1})$, $m_0 = 1_M$, $m_{i+1} = m_i \varphi(a_{i+1}, \overline{\theta_{i+1}})$ satisfy $m_n \in F$ and $w \in L$.

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 applications 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 application $(a, g') \longmapsto \Big[(q, g) \mapsto (q', g')\Big]$ 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 $F = \{m \mid m((q_0, \overline{\perp^k})) \in F \times \mathcal{D}^k/\sim\}$. We note L' the data language accepted by M using the k-registers-mechanism $[(up_{m,a})_{m \in M, a \in \Sigma}, \sim, \varphi]$ and the set F.

Assume $w = (a_1, d_1) \dots (a_p, d_p)$ is in L using the following computation:

$$\begin{array}{ccccccc} q_0 & \xrightarrow[d_1]{g_1, a_1, up_1, g_1'} & q_1 & \xrightarrow[d_2]{g_2, a_2, up_1, g_2'} & q_2 \ \dots & \xrightarrow[d_n]{g_n, a_n, up_n, g_n'} & q_n \\ \theta_0 & & \theta_1 & & \theta_2 & & \theta_n \end{array}$$

where $\theta_0 = \perp^k$ and $\theta_{i+1} = up_{i+1}(\theta_i, d_{i+1})$. It satisfies $\overline{\theta_i} = g_{i+1}$, $\overline{\theta_{i+1}} = g_{i+1}$ and $q_n \in F$. Let m_0 be the identity function on $Q \times \mathcal{D}^k/\sim$. Let m_{i+1} be the composition of m_i with $\varphi(a_{i+1}, \overline{\theta_{i+1}}) = \varphi(a_{i+1}, g'_{i+1})$, *i.e.* of m_i with $[(q, \overline{\alpha}) \longmapsto (q', g'_{i+1})]$ for the unique q' such that a transition exists $(q, \overline{\alpha}, a_{i+1}, up, g'_{i+1})$. Assuming inductively that $m_i((q_0, \perp^k)) = (q_i, \overline{\theta_i})$, we thus get $q' = q_{i+1}$ and $m_{i+1}((q_0, \overline{\perp^k})) = (q_{i+1}, \overline{\theta_i + 1})$ and thus $w \in L'$.

Conversely, assume that $w = (a_1, d_1) \dots (a_p, d_p)$ is in L'. Hence the sequences defined by

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

satisfy the property that $m_n((q_0, \overline{\perp^k})) = (q, \overline{\theta})$ for some $q \in F$. We define $(q_i, \overline{\theta_i})$ by $(q_i, \overline{\theta_i}) = m_i((q_0, \overline{\perp^k}))$ and we claim that

$$\begin{array}{ccccccc} q_0 & \xrightarrow[d_1]{\overline{\theta_0}, a_1, up_{m_0, a_1}, \overline{\theta_1}} & q_1 & \xrightarrow[d_2]{\overline{\theta_1}, a_2, up_{m_1, a_2}, \overline{\theta_2}} & q_2 \;\cdots & \xrightarrow[d_n]{\overline{\theta_{n-1}}, a_n, up_{m_{n-1}, a_n}, \overline{\theta_n}} & q_n \\ \theta_0 & & \theta_1 & & \theta_2 & & \theta_n \end{array}$$

is a valid accepting path in $\mathcal{A}$. Hence, $w \in L$.

The equivalence between monoids and automata is now 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 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 (for example $\mathbb{N}$ or $\mathbb{Q}^+$ or $\mathbb{R}^+$), then timed languages reduce to our data languages (since we can easily handle the monotonicity condition on time).

Proposition 4. *Let $\mathcal{A}$ be a (deterministic) timed automata with n clocks over a timed domain $\mathcal{D}$. 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, $\{x_1, \ldots, x_n\}$. A clock x_0 is added to the set of clocks and represents the universal time, *i.e.* x_0 is never reset in $\mathcal{A}$. 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 $\equiv$ ($\equiv$ can for example be the region equivalence). 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:

$$
\begin{gathered}
(\theta_i)_{0 \le i \le 2n+1} \sim (\theta'_i)_{0 \le i \le 2n+1} \\
\Updownarrow \\
\begin{cases}
\forall f \in \mathcal{F}, \left(\begin{array}{c} (\theta_{f(x_0)} - \theta_{f(x_i)})_{1 \le i \le n} \equiv (\theta'_{f(x_0)} - \theta'_{f(x_i)})_{1 \le i \le n} \\ \text{or} \\ (\theta_{f(x_0)} - \theta_{f(x_i)})_{1 \le i \le n} < 0 \text{ and } (\theta'_{f(x_0)} - \theta'_{f(x_i)})_{1 \le i \le n} < 0 \end{array} \right) \\
\qquad\qquad \big(\theta_0 < \theta_{n+1} \iff \theta'_0 < \theta'_{n+1}\big) \\
\qquad\qquad \big(\theta_0 > \theta_{n+1} \iff \theta'_0 > \theta'_{n+1}\big)
\end{cases}
\end{gathered}
$$

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

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

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

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

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

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

The data automaton $\mathcal{B}$ that we just constructed is deterministic and recognizes the same 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 time 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 $\{(a, \tau)(a, 2\tau) \ldots (a, n\tau) \mid \tau \in \mathbb{Q}_+\}$ 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.

6 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:

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

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

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

If $\rho = [(up_{m,a})_{m \in M, a \in \Sigma}, \sim, \varphi]$ is a k-registers-mechanism, we say that ρ satisfies the condition $(\dagger)$ whenever condition $(\dagger)$ holds for every classes of $\sim$ and for every update $up_{m,a}$.
We will prove that this simple condition ensures the decidability of the emptiness problem. The principle of the proof of this result is 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 using the k-registers-mechanism ρ such that ρ satisfies the condition $(\dagger)$. Then the emptiness of L is decidable in complexity* PSPACE.

Proof. Let $L \subseteq (\Sigma \times \mathcal{D})^*$ be a recognizable data language. We assume that M is a monoid which recognizes L using a k-register-mechanism $\rho = [(up_{m,a})_{m \in M, a \in \Sigma}, \sim, \varphi]$ that satisfies condition $(\dagger)$. 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 $Q = M \times \mathcal{D}^k/_\sim$, $I = (1_M, \overline{\perp^k})$, $F = P \times \mathcal{D}^k/_\sim$ (P is the acceptance set for the monoid recognizability) 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 will prove that, as condition (†) holds, this finite automaton accepts

$$\textsc{Undata}(L) = \{a_1 \dots a_n \mid \exists d_1, \dots, d_n,\ (a_1, d_1) \dots (a_n, d_n) \in L\}$$

Assume that $\mathcal{A}$ accepts the data word $w = (a_1, d_1) \dots (a_n, d_n)$. The following path accepts w:

$$\begin{array}{ccccccc} m_0 & \xrightarrow[d_1]{g_1, a_1, up_{m_0,a_1}, g_1'} & m_1 & \xrightarrow[d_2]{g_2, a_2, up_{m_1,a_2}, g_2'} & m_2 \dots & \xrightarrow[d_n]{g_n, a_n, up_{m_{n-1},a_n}, g_n'} & m_n \\ \theta_0 & & \theta_1 & & \theta_2 & & \theta_n \end{array}$$

In particular, for each i, $\theta_{i+1} = up_{m_i,a_{i+1}}(\theta_i, d_{i+1})$, $\overline{\theta_i} = g_i$ and $\overline{\theta_{i+1}} = g_i'$. Hence, for each i, $g_i \xrightarrow{up_{m_i,a_{i+1}}} g_i'$ and there is a transition $((m_i, g_i), a_{i+1}, (m_{i+1}, g_i'))$ in $\mathcal{B}$. Thus, the following path of $\mathcal{B}$ accepts the word $\textsc{Undata}(w) = a_1 \dots a_n$.

$$(m_0, g_1) \xrightarrow{a_1} (m_1, g_1') = (m_1, g_2) \xrightarrow{a_2} (m_2, g_2') = (m_2, g_3) \dots \xrightarrow{a_n} (m_n, g_n')$$

Conversely, if $a_1 \dots a_n$ is a word accepted by $\mathcal{B}$ through the path

$$(m_0, g_0) \xrightarrow{a_1} (m_1, g_1) \xrightarrow{a_2} (m_2, g_2) \dots \xrightarrow{a_n} (m_n, g_n)$$

It means that for each i, $(m_i, g_i, a_{i+1}, up_{m_i,a_{i+1}}, g_{i+1}, m_{i+1})$ is transition of $\mathcal{A}$ and that $g_i \xrightarrow{up_{m_i,a_{i+1}}} g_{i+1}$. We define $\theta_0 = \perp^k$ and inductively θ_{i+1} by: as $\overline{\theta_i} = g_i$, there exists d_{i+1} such that $up_{m_i,a_{i+1}}(\theta_i, d_{i+1}) \in g_{i+1}$; we thus define θ_{i+1} as $up_{m_i,a_{i+1}}(\theta_i, d_{i+1})$. Hence the following path accepts $(a_1, d_1) \dots (a_n, d_n)$ in $\mathcal{A}$:

$$\begin{array}{ccccccc} m_0 & \xrightarrow[d_1]{g_0, a_1, up_{m_0,a_1}, g_1} & m_1 & \xrightarrow[d_2]{g_1, a_2, up_{m_1,a_2}, g_2} & m_2 \dots & \xrightarrow[d_n]{g_{n-1}, a_n, up_{m_{n-1},a_n}, g_n} & m_n \\ \theta_0 & & \theta_1 & & \theta_2 & & \theta_n \end{array}$$

The proof is now complete : $\mathcal{B}$ accepts $\textsc{Undata}(L)$.

Thus L is empty if and only if $\textsc{Undata}(L)$ is empty. We can decide emptiness by applying the usual non-deterministic algorithm to the constructed automaton. Since this automaton has $|M \times \mathcal{D}^k/\!\sim|$ states, the algorithm can be implemented in space $\log(|M \times \mathcal{D}^k/\!\sim|)$, which is polynomial in the size of the input. $\square$

We will now show that condition (†) can often be easily decided.
We define $\widehat{up}(\overline{\theta}) = \{v' \mid \exists v \in \overline{\theta},\ \exists d \in \mathcal{D},\ v' = up(v, d)\}$ and $\widehat{up}^{-1}(\overline{\theta'}) = \{v \mid \exists d \in \mathcal{D},\ up(v, d) \in \overline{\theta'}\}$. With these definitions,

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

Thus, if $\widehat{up}$, $\widehat{up}^{-1}$, $\cap$ and $=$ are computable, then condition (†) is also computable. Moreover, the updates we use do not compute anything, and the condition can be further simplified. If X is a subset of $\{1 \dots k\}$, we define π_X the projection over the components of X. If up is an update (*i.e.* $up \subseteq \{1 \dots k\}$), we define $\times_{up}$ as following product: if R is a subset of $\mathcal{D}^{k-|up|}$ and if R' is a subset of $\mathcal{D}^{|up|}$,

$R \times_{up} R' = \{v \in \mathcal{D}^k \mid \pi_{\overline{up}}(v) \in R \text{ and } \pi_{up}(v) \in R'\}$ where $\overline{up} = \{1 \dots k\} \setminus up$. Condition $(\dagger)$ is then equivalent to:

$$\Big((\pi_{\overline{up}}(\overline{\theta}) \times_{up} \mathcal{D}^{|up|}) \cap \overline{\theta'} \neq \emptyset\Big) \implies \pi_{\overline{up}}(\overline{\theta}) \subseteq \pi_{\overline{up}}(\overline{\theta'})$$

Thus, if $\sim$ is defined in such a way that:

1. we can compute $\pi_{\overline{up}}(\overline{\theta})$
2. we can compute $\pi_{\overline{up}}(\overline{\theta}) \times_{up} \mathcal{D}^{|up|}$
3. we can decide $(\pi_{\overline{up}}(\overline{\theta}) \times_{up} \mathcal{D}^{|up|}) \cap \overline{\theta'} \neq \emptyset$
4. we can decide $\pi_{\overline{up}}(\overline{\theta}) \subseteq \pi_{\overline{up}}(\overline{\theta'})$

then we can decide $(\dagger)$!
We can note that all the operations from the previous list are elementary operations on equivalence classes.

7 Extensions of the Model

7.1 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 define a non deterministic k-register-mechanism as a triple $[(U_{m,a})_{m\in M, a\in\Sigma}, \sim, \varphi]$ where the only difference with Definition 1 is that $U_{m,a}$ is a set of updates instead of simply a unique update. We hence say that a finite monoid M non-deterministically recognizes a data language L whenever there exists a non-deterministic k-register-mechanism that recognizes L in the same way as Definition 2. Some properties which are true for deterministic data automata are also true for non-deterministic data automata:

Proposition 5. *Let L be a data language over Σ and $\mathcal{D}$. Then,*

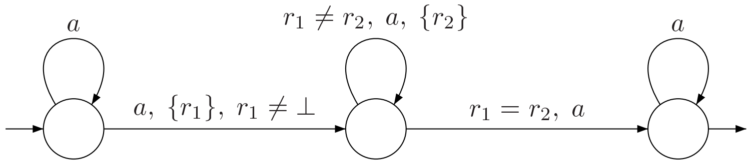
– *L is non-deterministically recognized by a finite monoid if and only if it is recognized by a non-deterministic data automaton.*

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

– *Condition $(\dagger)$ ensures the decidability of the emptiness problem, i.e. if L is recognized by a non-deterministic data automaton that satisfies the condition $(\dagger)$, 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. The two first points of the theorem can be proved in the same way as Theorem 1 and Theorem 2. We just need to present the proof of the last point, namely that there exists a data language which is nd-recognizable but not recognizable.

Consider the data language L accepted by the following non-deterministic data automaton:



We have that $L = \{(a, d_1) \dots (a, d_n) \mid \exists 1 \leq i < j < n,\ d_i = d_j \neq \bot\}$. Moreover, one can prove that L is not recognized by any (deterministic) data automaton. Suppose L is accepted by a data automaton $\mathcal{A}$ using k registers. There are finitely many paths of length $k+1$ in $\mathcal{A}$. For each such path c, we define

$$E(c) = \{(a, d_1) \dots (a, d_{k+1})\text{read through the path } c \text{ and } i \neq j \implies d_i \neq d_j\}$$

There exists an integer $n(c)$ such that the data indexed by $n(c)$ (of a data word read through the path c) is not kept in mind at the end of the path c. Using a combinatorial argument (see the proof of Proposition 3), there exists two data words in some $E(c)$ that differ only from the data $n(c)$.

$$\begin{array}{ccccccc}
q_0 \xrightarrow{(a,d_1)} & q_1 \ \cdots & q_{n(c)-1} \xrightarrow{(a,d_{n(c)})} & q_{n(c)} & \cdots\ q_k & \xrightarrow{(a,d_{k+1})} & q_{k+1} \\
\theta_0 & \theta_1 & \theta_{n(c)-1} & \theta_{n(c)} & \theta_k & & \theta_{k+1} \\
= & = & = & \sim & \sim & & = \\
\theta'_0 & \theta'_1 & \theta'_{n(c)-1} & \theta'_{n(c)} & \theta'_k & & \theta'_{k+1} \\
q_0 \xrightarrow{(a,d_1)} & q_1 \ \cdots & q_{n(c)-1} \xrightarrow{(a,d'_{n(c)})} & q_{n(c)} & \cdots\ q_k & \xrightarrow{(a,d_{k+1})} & q_{k+1}
\end{array}$$

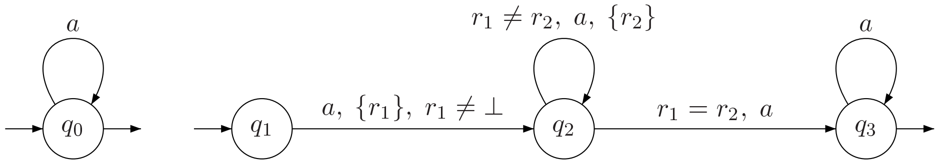
As $\theta_k = \theta'_k$, for each w, $(a, d_1) \dots (a, d_{n(c)}) \dots (a, d_{k+1})$ is in L if and only if $(a, d_1) \dots (a, d'_{n(c)}) \dots (a, d_{k+1})$ is in L. Of course, this is not true. Hence, L is not accepted by any (deterministic) data automaton. □

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:

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

which are recognized by the following data automata:



$\square$

Proposition 6. *The class of nd-recognizable data languages is closed under union, intersection, concatenation and finite iteration. It is not closed under complementation.*

Proof. For union and intersection, the classical construction suffices. Let next L_1 and L_2 be data languages that are accepted by data automata with respectively k_1 and k_2 registers. Then we will prove that $L_1 \cdot L_2$ is accepted by a data automaton with $max(k_1, k_2)$ registers. We define two integers $k = max(k_1, k_2)$ and $l = min(k_1, k_2)$. Intuitively, we want to put sequentially the two automata. But just before going inside the second automaton, the values of the registers should contain the data $\perp$, but they do not. The trick is to duplicate the second automaton many times: Assume $\mathcal{A}_i = (Q_i, q_{0,i}, F_i, k_i, \sim_i, T_i)$ for $i = 1, 2$. If P is a subset of $\{1, \ldots, l\}$, we first define π_P as $\pi_P(\theta) = \theta[P \leftarrow \perp]$ and a new equivalence on $\mathcal{D}^k$ by:

$$\theta \equiv \theta' \iff \forall P \subseteq \{1 \ldots l\},\ \pi_P(\theta)_{|k_1} \sim_1 \pi_P(\theta')_{|k_1} \text{ and } \pi_P(\theta)_{|k_2} \sim_2 \pi_P(\theta')_{|k_2}$$

where $*_{|k_i}$ denotes the k_i first components of $*$.
With this definition of $\equiv$, if $\theta \equiv \theta'$, for each $P \subseteq \{1 \ldots l\}$, $\pi_P(\theta) \equiv \pi_P(\theta')$. We refine the automata $\mathcal{A}_i$ using the equivalence $\equiv$. If g is a class with respect to $\equiv$, $\pi_P^{-1}(g)$ is a finite union of classes with respect to $\equiv$.
We thus define $\mathcal{A} = (Q, q_0, F, k, \equiv, T)$ by:

$$\begin{cases} Q &= Q_1 \cup (Q_2 \times \mathcal{P}(\{1 \ldots l\})) \\ q_0 &= q_{0,1} \\ F &= F_2 \times \mathcal{P}(\{1 \ldots l\}) \\ T &= T_1 \cup \{(f, true, a, up, \pi_P^{-1}(g'), (q, P)) \mid f \in F_1,\ (q_0, g, a, up, g', q) \in T_2, \perp^{k_2} \in g \\ & \qquad\qquad\qquad\qquad \text{and } P = \{\text{registers not updated by } up\}\} \\ & \quad \cup \{((q, P), \pi_P^{-1}(g), a, up, \pi_{P'}^{-1}(g'), (q', P')) \mid (q, g, a, up, g', q') \in T_2 \\ & \qquad\qquad\qquad\qquad \text{and } P' = P \cap \{\text{registers not updated by } up\}\} \end{cases}$$

The data automaton $\mathcal{A}$ recognizes the data language $L(\mathcal{A}_1) \cdot L(\mathcal{A}_2)$: assume a data word w is in $L(\mathcal{A}_1) \cdot L(\mathcal{A}_2)$, $w = uv$ where $u \in L(\mathcal{A}_1)$ and $v \in L(\mathcal{A}_2)$ and consider runs of u in $\mathcal{A}_1$ and of v in $\mathcal{A}_2$ respectively:

$$\begin{array}{ccc} q_0 \longrightarrow \ldots \longrightarrow q_n & \text{and} & q'_0 \xrightarrow[d'_1]{g_1, a'_1, up_1, g'_1} q'_1 \xrightarrow[d'_2]{g_2, a'_2, up_2, g'_2} q'_2 \ldots \\ \theta_0 \qquad\qquad\quad \theta_n & & \theta'_0 \qquad\qquad\qquad \theta'_1 \qquad\qquad\qquad \theta'_2 \end{array}$$

In $\mathcal{A}$, their concatenation corresponds to the following path:

$$\begin{array}{l} q_0 \longrightarrow \ldots \longrightarrow q_n \xrightarrow[d'_1]{true, a'_1, up_1, \pi_{P_1}^{-1}(g'_1)} (q'_1, P_1) \xrightarrow[d'_2]{\pi_{P_1}^{-1}(g_2), a'_2, up_2, \pi_{P_2}^{-1}(g'_2)} (q'_2, P_2) \ldots \\ \theta_0 \qquad\qquad\quad \theta_n \qquad\qquad\qquad\qquad\quad \theta'_{1,P_1} \qquad\qquad\qquad\qquad\qquad\quad \theta'_{2,P_2} \end{array}$$

where $\theta'_i = \pi_{P_i}(\theta'_{i,P_i})$ because $P_{i+1} = P_i \cap \{\text{registers not updated by } up_{i+1}\}$ and $\theta'_0 = \perp^k$. This is an accepting path for w in $\mathcal{A}$. Hence, $L(\mathcal{A}_1) \cdot L(\mathcal{A}_2) \subseteq L(\mathcal{A})$.

Conversely, if $w \in L(\mathcal{A})$, assume w is read in $\mathcal{A}$ through the following path:

$$\begin{array}{llllll} q_0 \longrightarrow \ldots \longrightarrow q_n & \xrightarrow[d'_1]{true, a'_1, up_1, \pi^{-1}_{P_1}(g'_1)} & (q'_1, P_1) & \xrightarrow[d'_2]{\pi^{-1}_{P_1}(g_2), a'_2, up_2, \pi^{-1}_{P_2}(g'_2)} & (q'_2, P_2) \ldots \\ \theta_0 \qquad\qquad\quad \theta_n & & \theta'_1 & & \theta'_2 \end{array}$$

This path can be cut into two parts: one in $\mathcal{A}_1$ and an other in $\mathcal{A}_2$ ($P_0 = \{a, \ldots, k\}$):

$$\begin{array}{llllll} q_0 \longrightarrow \ldots \longrightarrow q_n & \text{and} & q'_0 \xrightarrow[d'_1]{g_1, a'_1, up_1, g'_1} & q'_1 & \longrightarrow \quad \ldots & \longrightarrow q'_p \\ \theta_0 \qquad\qquad\quad \theta_n & & \pi_{P_0}(\theta_n) & \pi_{P_1}(\theta'_1) & \pi_{P_2}(\theta'_2) & \end{array}$$

Those paths are accepting paths for respectively u and v such that $|u| = n$ and $w = uv$. Hence, $u \in L(\mathcal{A}_1)$ and $v \in L(\mathcal{A}_2)$ and thus $w \in L(\mathcal{A}_1) \cdot L(\mathcal{A}_2)$.
The proof is now complete: $\mathcal{A}$ recognizes the concatenation $L(\mathcal{A}_1) \cdot L(\mathcal{A}_2)$.

A similar construction to the previous one can handle iteration.

Finally, the data language L considered in the proof of Proposition 5 is nd-recognizable. We can prove that the complement of this data language, namely

$$\overline{L} = \{(a, d_1) \ldots (a, d_n) \mid \forall 1 \leq i < j < n,\ d_i \neq d_j\}$$

is recognized by no finite monoid (of course when $\mathcal{D}$ is infinite). The proof uses similar arguments as the ones of the proof of Proposition 5 or of Proposition 3.
$\square$

7.2 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 7. *Let L be a language over the finite alphabet Σ. Assume that L is recognized by a finite monoid M. Then the data language $L_M = \{(a_1, m_1) \ldots (a_n, m_n) \mid a_1 \ldots a_n \in L\}$ 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. $\square$

Remark 3. We can note that allowing more general updates enlarges the class of languages that can be recognized by a monoid. For example, let L be the data language

$$\{(a, p_1) \dots (a, p_n) \mid \forall i,\ p_i \text{ prime number and } i \neq j \implies p_i \neq p_j\}$$

over $\{a\} \times \mathbb{N}$. This data language is not recognizable, but is recognized by a monoid using more general updates (like, for example, $up((\theta_i)_{i=1,2}, d) = (d, \theta_2 * d)$).

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.

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. And, in this framework, our results can be seen to be the first purely algebraic characterization for timed languages.

This theory has now to be developped. For instance, a notion of aperiodic data language can naturally be defined and has to be studied. Besides, a logical characterization of the monoid recognizable data languages would be very interesting. To try to adapt the work of Wilke [Wil94] on timed automata to data automata is in particular a challenging direction.

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.

Besides the case of time, there is another, probably more theoretical, instance of data languages which could be worth to study: the case where the set of data $\mathcal{D}$ is finite. Then $\Sigma \times \mathcal{D}$ is simply a free monoid and we have seen that a data language is monoid recognizable if and only if it is a recognizable formal language (see Remark 1). But, given a finite monoid M, it remains to characterize the class of data languages that are recognized by M and, in particular, to compare it with the class of formal languages recognized by M.

At least, 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

[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.

[ACM01] Asarin, E., Caspi, P., and Maler, O. *Timed Regular Expressions*, 2001. Submitted to the Journal of the Association for Computing Machinery (JACM).

[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.

[Bar89] Barrington, D. *Bounded-width branching programs recognize exactly those languages in NC^1*. Journal of Computer and Systems Science, vol. 38(1):150–164, 1989.

[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 (JALC), 2001. To appear.

[BPT01] Bouyer, P., Petit, A., and Thérien, D. *An Algebraic Characterization of Data and Timed Languages*. In *Proc. 12th Int. Conf. Concurrency Theory (CONCUR'01)*, Lecture Notes in Computer Science. Springer-Verlag, 2001. To appear.

[CG00] Choffrut, C. and Goldwurm, M. *Timed Automata with Periodic Clock Constraints*. Journal of Automata, Languages and Combinatorics (JALC), 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.

[HLS+93] Hertrampf, U., Lauteman, C., Schwentick, T., Vollmer, H., and Wagner, K. *On the Power of Polynomial Time Bit-Reductions*. In *Proc. 8th Conference on Structures in Complexity Theory*, pp. 200–207. IEEE Computer Society Press, 1993.

[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.

[Kam68] KAMP, J. A. W. *Tense Logic and the Theory of Linear Order*. Ph.D. thesis, UCLA, Los Angeles, CA, USA, 1968.

[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.

[Sch65] SCHÜTZENBERGER, M.-P. *On Infinite Monoids Having only Trivial Subgroups*. Information and Control, vol. 8:190–194, 1965.

[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.