An Algebraic Approach to Data Languages and Timed Languages

Patricia Bouyer [a,1], Antoine Petit [a,1,2], Denis Thérien [b,3]

[a] *LSV, CNRS UMR 8643 & ENS de Cachan,*
61, avenue du Président Wilson,
94235 Cachan Cedex, France
Email: `{bouyer,petit}@lsv.ens-cachan.fr`

[b] *School of Computer Science, McGill University,*
3480 University,
Montréal, QC, Canada, H3A 2A7
Email: `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 includes 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.

[1] Research partly supported by the French project RNRT "Calife"
[2] Research partly supported by the French-Indian project CEFIPRA n^{o}2102 − 1
[3] Research supported by NSERC, FCAR, and the von Humboldt Foundation

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 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.

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,ACM02,BP02] 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-clock 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.

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 cannot 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 data 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, two different varieties of monoids recognize two different sets of data languages. 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 three possible extensions of our model. First, we extend in a natural way the set of operations on registers that we can perform (registers can be erased or swapped). The model obtained using this larger class of updates is not more expressive than the original model, but the new operations are very natural and useful macros to represent systems. We then 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. Finally, 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 $(a_1, d_1) \dots (a_p, d_p)$ 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*, is an application up from $\mathcal{D}^k \times \mathcal{D}$ into $\mathcal{D}^k$, such that there exists a set $I_{up} \subseteq \{1, \dots, k\}$ and up maps $((d_i)_{i=1..k}, d)$ onto $((d'_i)_{i=1..k})$ where $d'_i = d$ if $i \in I_{up}$ and $d'_i = d_i$ if $i \notin I_{up}$. In the sequel, on the pictures, an update up will be precisely written as the set I_{up}.

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 presented as Algorithm 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].

Algorithm 1 The mechanism for formal languages

```
#   % Initialization
#   m := 1
#   % Computation
#   While not end of w do
#           Read the next letter a of the word
#           Compute φ(a)
#           Compute m := mφ(a)
#   Endwhile
#   % Output
#   If m ∈ F then output "yes" else output "no"
```

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 described in Algorithm 2 answers *"yes"*.

Algorithm 2 A mechanism that uses registers

```
#   % Initialization
#   m := 1
#   all the registers are set to ⊥
#   % 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 ∈ F then output "yes" else output "no"
```

We need now to precise how the registers are updated and how the successive values in the monoid are computed.

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-register mechanism:

Definition 1 *A* k-register 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-register 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-register mechanism reduces to a morphism from Σ^* into M.

If ρ is a k-register mechanism over a finite monoid M and 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 computation of ρ over w yields the element of M given by the computation described in Algorithm 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 is denoted by $\rho(w)$.

Algorithm 3 Computation in a k-register mechanism

% *Initialization*
$m := 1$
$\forall 1 \leq j \leq k,\ \theta[j] := \perp$
% *Computation*
For $i := 1$ *to* n *do*
    $\theta := up_{m,a_i}(\theta, d_i)$
    $m := m\varphi(a_i, \overline{\theta})$
Endfor
% *Output*
Output m

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-register 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-register mechanism $\rho = [(up_{m,a})_{m \in M, a \in \Sigma}, \sim, \varphi]$ such that:*

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

A data language is said to be *monoid recognizable* if there exists some finite monoid recognizing it.

Example 3 *The data language $L = \{(a,d)(a,d') \mid n \geq 1,\ d \neq \perp, d \neq d'\}$ 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 $x \in M$.*

To this aim, we use two registers. Thus, we define the 2-register mechanism $\rho = [(up_{m,a})_{m \in M, a \in \Sigma}, \sim, \varphi]$ in the following way:

- *The updates are $up_{1,a}$ such that $I_{up_{1,a}} = \{1\}$ and if $z \in M \setminus \{1\}$, $up_{z,a}$ such that $I_{up_{z,a}} = \{2\}$.*
- *$\sim$ 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 $d \neq \perp$ and $d \neq d'$.

In the monoid M	$1_M \xrightarrow[d]{a}$	$y \xrightarrow[d']{a}$	y^2
Values of the two registers	$\begin{pmatrix} \perp \\ \perp \end{pmatrix}$	$\begin{pmatrix} d \\ \perp \end{pmatrix}$	$\begin{pmatrix} d \\ d' \end{pmatrix}$
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 4 *The data language $\{(a,d_1)\ldots(a,d_n)(a,d) \mid n \geq 1,\ d \notin \{d_1, \ldots, d_n\}\}$ 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.*

Proposition 5 *Assume $\mathcal{D}$ reduces to $\{\perp\}$. A formal language is recognizable if and only if its image is a monoid recognizable data language.*

Assume $\mathcal{D}$ is finite. If a data language is monoid recognizable, then it is also a recognizable formal language. The converse also holds.

Proof. The first property is obvious, as Σ and $\Sigma \times \mathcal{D}$ are then in bijection.

For the second property, we assume that $\mathcal{D}$ is a finite set of data and that $L \subseteq (\Sigma \times \mathcal{D})^*$ is a data language recognized by the finite monoid M, using the k-register mechanism $\rho = [(up_{m,a})_{m \in M, a \in \Sigma}, \sim, \varphi]$. The morphism $\varphi : (\Sigma \times \mathcal{D}^k/\sim)^* \longrightarrow M$ can be extended in a natural way into a morphism $\overline{\varphi} : (\Sigma \times \mathcal{D}^k)^* \longrightarrow M$. Note that $(\Sigma \times \mathcal{D}^k)$ is finite. We define the morphism

$$\begin{aligned} \psi : (\Sigma \times \mathcal{D})^* &\longrightarrow (M \times \mathcal{D}^k)^{(M \times \mathcal{D}^k)} \\ (a,d) &\longmapsto \left((m,\theta) \longmapsto (m',\theta') \text{ such that } \begin{cases} \theta' = up_{m,a}(\theta,d) \\ m' = m.\varphi(a,\theta') \end{cases} \right) \end{aligned}$$

and F' as the set of functions $\sigma : (M \times \mathcal{D}^k) \longrightarrow (M \times \mathcal{D}^k)$ such that, if 1_M is the neutral element of M, $\sigma((1_M, \perp^k))$ is of the form (f, θ) where f is in F, the accepting set for L, and $\theta \in \mathcal{D}^k$. It is easy to see that the morphism ψ "simulates" the mechanism ρ.

Conversely, assume $L \subseteq (\Sigma \times \mathcal{D})^*$ is regular. There exists a finite monoid M, $F \subseteq M$ and a morphism $\psi : (\Sigma \times \mathcal{D})^* \longrightarrow M$ such that $L = \psi^{-1}(F)$. Using a mechanism with one register, that is always updated, we can easily see that L is a monoid recognizable data 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 6 *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-register mechanism and F is a subset of M such that $L = \rho^{-1}(F)$ (as in Definition 2). Let $(a_1, d_1) \ldots (a_p, d_p)$ be a data word. Then the following equivalence holds:

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

Thus,

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

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-register 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 classical product. 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 $up_{m,a}$ such that

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

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.

Let L be a language on Σ. We define the data language

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

Lemma 7 *Let M be a finite monoid and L a language defined on the alphabet Σ. Then, L is recognized by M $\iff$ $L_{\mathcal{D}}$ is recognized by M.*

Proof. 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-register 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. □

This establishes that the role of the monoid is fundamental, in the sense that two different varieties of monoids (see [Pin86] for a definition of this notion) recognize two different sets of data languages. This is an easy implication of the previous lemma and of the variety theorem [Pin86].

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 data 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 8 *(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, $\mathcal{L}_{M,2^{|M\times\Sigma|}+1} = \mathcal{L}_{M,2^{|M\times\Sigma|}}$.

Proof.

(1) Assume that $\mathcal{D}$ is an infinite set of data. We will prove that the data language

$$L_k = \{(a, d_1), \dots, (a, d_n) \mid \forall i, j \;\; i \equiv j \bmod k-1 \implies d_i = d_j\}$$

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

Intuitively, L_k can be recognized by a finite monoid with k registers as follows. Reading a data word $(a, d_1) \dots (a, d_n)$, the first $k-1$ data are recorded in the first $k-1$ registers. Then the data d_k is put in the last register and the equality $d_k = d_1$ is tested through the equivalence relation on $\mathcal{D}^k$. Hence the next data d_{k+1} is put in the last register and the equality $d_{k+1} = d_2$ is once again tested through the equivalence relation on $\mathcal{D}^k$. The process continues until the last data letter is read or an error occurs. The main difficulty is to know with which register the last one has to be compared. This is done thanks to the monoid which roughly keeps this information through the whole computation.

More formally, we first define the monoid M as follows. The elements of M are:

$$\begin{aligned}
&0\\
&m^i, \text{ where } i \in \{0, \ldots, k-1\}\\
&n^i I, \text{ where } i \in \{1, \ldots, k-1\} \text{ and } I \subseteq \{1, \ldots, k\}
\end{aligned}$$

and the composition law is given by:

$$\begin{aligned}
&m^k = 1\\
&n^i I \cdot m^j = 0\\
&m^j \cdot n^i I = \begin{cases} 0 \text{ if } j+i \bmod k-1 \notin I\\ m^{j+i \bmod k-1} \text{ otherwise}\end{cases}\\
&n^i I \cdot n^j J = n^{i+j \bmod k-1} H \text{ where } H = \{h \in J \mid h-j \in I\}
\end{aligned}$$

It is left to the reader to verify that this composition law is indeed associative.

We can now define the k-register mechanism $\rho = [(up_{m,a})_{m \in M, a \in \Sigma}, \sim, \varphi]$.

- The updates are:

$$\begin{aligned}
&up_{m^i,a} \text{ such that } I_{up_{m^i,a}} = \{i+1\} \text{ for } 0 \leq i \leq k-2\\
&up_{x,a} \;\text{ such that } I_{up_{x,a}} = \{k\} \text{ for any } x \text{ which is not of the form } m^i
\end{aligned}$$

- The equivalence relation $\sim$ on $\mathcal{D}^k$ is such that:

$$(\theta_i)_{1\leq i\leq k} \sim (\theta'_i)_{1\leq i\leq k} \text{ if } \begin{cases} \text{either } \theta_k = \theta'_k = \perp\\ \text{or } \theta_k \neq \perp \text{ and } \theta'_k \neq \perp\\ \quad \text{and } \{i \mid \theta_i = \theta_k\} = \{i \mid \theta'_i = \theta'_k\}\end{cases}$$

  and has thus $1 + 2^{k-1}$ equivalence classes.

- The application φ is defined by:

$$\varphi(a,\overline{\theta}) = \begin{cases} m \text{ if } \theta_k = \perp \\ n\{i \mid \theta_i = \theta_k\} \text{ if } \theta_k \neq \perp \end{cases}$$

We claim that the monoid M defined above recognizes the language L_k with the k-register mechanism ρ and $\{m^i \mid 0 \leq i \leq k-1\}$ as set of final elements. A computation of ρ on a data word $(a,d_1)\ldots(a,d_{k-1})(a,d_k)(a,d_{k+1})$ is given by the following picture.

$$\begin{array}{ccccccccc} & (a,d_1) & & \ldots\ldots\ (a,d_{k-1}) & & (a,d_k) & & (a,d_{k+1}) & \\ \begin{pmatrix}\perp\\ \cdot\\ \cdot\\ \cdot\\ \perp\end{pmatrix} & & \begin{pmatrix} d_1\\ \perp \\ \cdot\\ \cdot\\ \perp\end{pmatrix} & & \begin{pmatrix} d_1\\ \cdot\\ \cdot\\ d_{k-1}\\ \perp\end{pmatrix} & & \begin{pmatrix} d_1\\ \cdot\\ \cdot\\ d_{k-1}\\ d_k\end{pmatrix} & & \begin{pmatrix} d_1\\ \cdot\\ \cdot\\ d_{k-1}\\ d_{k+1}\end{pmatrix} \\ 1 & & m & & m^{k-1} & & \alpha_k & & \alpha_{k+1}\end{array}$$

From the definition of the elements of ρ, it holds

$$\begin{aligned} \alpha_k &= m^{k-1} \cdot \varphi(a, \overline{(d_1,\ldots,d_{k-1},d_k)}) \\ &= m^{k-1} \cdot n\{i \mid d_k = d_i\} \\ &= \begin{cases} 0 \text{ if } k \bmod k-1 = 1 \notin \{i \mid d_k = d_i\} \text{ i.e. } d_k \neq d_1 \\ m \text{ otherwise} \end{cases} \end{aligned}$$

and

$$\begin{aligned} \alpha_{k+1} &= m \cdot \varphi(a, \overline{(d_1,\ldots,d_{k-1},d_{k+1})}) \\ &= m \cdot n\{i \mid d_{k+1} = d_i\} \\ &= \begin{cases} 0 \text{ if } 2 \bmod k-1 = 2 \notin \{i \mid d_{k+1} = d_i\} \text{ i.e. } d_{k+1} \neq d_2 \\ m^2 \text{ otherwise} \end{cases} \end{aligned}$$

We will now prove that L_k is not recognized by any finite monoid with at most $k-1$ registers. Assume the contrary and let M be some finite monoid and let $\rho = [(up_{m,a})_{m\in M, a\in\Sigma}, \sim, \varphi]$ be a h-register mechanism (with $h \leq k-1$) recognizing L_k.

For any data word $w = (a,d_1)\ldots(a,d_k)$ of length k, we define the sequence $(\theta_i)_{0\leq i\leq k}$ as in Algorithm 3. The data word w is thus said to be read on the path $c = (\overline{\theta_1},\ldots,\overline{\theta_k})$. For such a given path c, we denote by $E(c)$ the set of all the data words read on c. Since the vectors θ_i are of size h, there exists for any path c an integer $n(c) \in \{1, \dots, k\}$ such that, for any data word $(a, d_1) \dots (a, d_k)$ read on the path c, the last data vector θ_k does not contain the data indexed by $n(c)$.

Now let us consider a subset D of $\mathcal{D}$ (which is infinite by hypothesis) of size $\beta > N^k$ where N is the number of equivalence classes of $\sim$. The set $(\{a\} \times D)^k$ has β^k elements. Since the number of pathes is N^k, there exists some path c such that $E(c)$ contains at least $\frac{\beta^k}{N^k} > \beta^{k-1}$ data words of $(\{a\} \times D)^k$. Let us consider the equivalence relation $\approx$ on the elements of $(\{a\} \times D)^k$ defined as follows:

$$u \approx v \text{ if } \forall i \neq n(c),\ u_i = v_i$$

Since D is of size β, $\approx$ has β^{k-1} equivalence classes. Hence the set $E(c)$ contains at least two data words

$$(a, d_1) \dots (a, d_{n(c)}) \dots (a, d_k)$$

and

$$(a, d_1) \dots (a, d'_{n(c)}) \dots (a, d_k)$$

of $(\{a\} \times D)^k$ which are $\approx$-equivalents. The computations of ρ on these two words are represented by the following figure.

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

As $\theta_k = \theta'_k$, for any data word w, if the data word $(a, d_1) \dots (a, d_{n(c)}) \dots (a, d_k) w$ is in L then the data word $(a, d_1) \dots (a, d'_{n(c)}) \dots (a, d_k) w$ is also in L. However, this is impossible since $d_{n(c)} \neq d'_{n(c)}$. 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 $k+1$ registers using the mechanism $[(up_{m,a})_{m \in M, a \in \Sigma}, \sim, \varphi]$, and assuming that k is greater than the powerset of the cardinality of $M \times \Sigma$, we get that at least two registers are updated in the same way by all the updates. More precisely, defining

the application

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

λ cannot be injective. There exists thus two integers $i \neq j$ such that $\lambda(i) = \lambda(j)$. We assume, without loss of generality, that i and j are respectively k and $k+1$, and we define the equivalence relation $\approx$ on the set $\mathcal{D}^k$ by

$$(d_1, \dots, d_k) \approx (d'_1, \dots, d'_k) \iff (d_1, \dots, d_k, d_k) \sim (d'_1, \dots, d'_k, d'_k)$$

We get easily that L is recognized by M using the k-register mechanism $[(up'_{m,a})_{m \in M, a \in \Sigma}, \approx, \psi]$ where $I_{up'_{m,a}} = I_{up_{m,a}} \setminus \{k+1\}$ and ψ is defined from φ in an obvious way. We then get that $\mathcal{L}_{M,k+1} = \mathcal{L}_{M,k}$ as soon as $k \geq 2^{|M \times \Sigma|}$.

□

Remark 9 *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 be the finite monoid $\{1, 0, x\}$ with $x^2 = x$. For any integer k, let us define $\Sigma_k = \{a_0, a_1, \dots, a_{k-1}\}$. 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

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

We claim that

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

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

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

The updates up_{z,a_i} are defined by $I_{up_{z,a_i}} = \{i\}$ 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 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 10 *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

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

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 11 *The data language described in Example 3,*

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

is recognized by the following data automaton ($\overline{\theta_=}$ and $\overline{\theta_{\neq}}$ are defined in Example 3):

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

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

Theorem 12 *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-register 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 10.

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:

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

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

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

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

$$\begin{array}{ccccccc} 1_M & \xrightarrow[d_1]{g_1, a_1, up_1, g'_1} & m_1 & \xrightarrow[d_2]{g_2, a_2, up_1, g'_2} & m_2 \ldots & \xrightarrow[d_n]{g_n, a_n, up_n, g'_n} & m_n \\ \theta_0 & & \theta_1 & & \theta_2 & & \theta_n \end{array}$$

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-register mechanism $[(up_{m,a})_{m \in M, a \in \Sigma}, \sim, \varphi]$ and the set F.

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

$$\begin{array}{ccccccc} q_0 & \xrightarrow[d_1]{g_1, a_1, up_1, g'_1} & q_1 & \xrightarrow[d_2]{g_2, a_2, up_1, g'_2} & q_2 \ldots & \xrightarrow[d_n]{g_n, a_n, up_n, g'_n} & q_n \\ \theta_0 & & \theta_1 & & \theta_2 & & \theta_n \end{array}$$

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 there exists a transition of the form

$(q, \overline{\alpha}, a_{i+1}, up, g'_{i+1})$. 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 13 *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.*

Sketch of proof. 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 to represent the universal time, *i.e.* x_0 is never reset in $\mathcal{A}$. There exists an equivalence relation edfined on $\mathcal{D}^n$, namely $\equiv$, such that if g is a guard appearing in $\mathcal{A}$, then g is an union of equivalence classes ($\equiv$ 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:

$$(\theta_i)_{0\leq i\leq 2n+1} \sim (\theta'_i)_{0\leq i\leq 2n+1}$$

$$\Updownarrow$$

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

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 $\sim$,
- α is such that $I_\alpha = \{0, 1, \ldots, 2n+1\} \setminus \{f(x_0), \ldots, f(x_n)\}$,

- $f' \in \mathcal{F}$ 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 data automaton $\mathcal{B}$ that we just constructed is deterministic and recognizes the same data (or timed) language as $\mathcal{A}$. □

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 the recognition of 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 $\{(a,t_1)\ldots(a,t_n) \mid \forall i,\ t_i \text{ is a prime number}\}$. 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:

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

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

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

This condition is quite natural: it specifies that two equivalent register vectors have the same future behaviours (condition (†) is a bisimulation relation w.r.t. the transition relation defined by $\xrightarrow{up}$). Figure 1 illustrates this decidability condition.

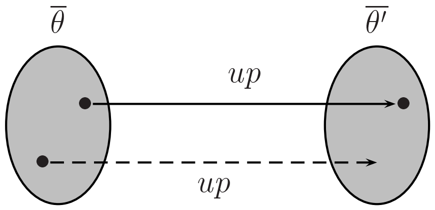


Fig. 1: Decidability condition (†)

If $\rho = [(up_{m,a})_{m\in M, a\in\Sigma}, \sim, \varphi]$ is a k-register mechanism, we say that ρ satisfies the condition (†) whenever condition (†) 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 14 *Let L be a recognizable data language over Σ and $\mathcal{D}$. Assume L is recognized by the finite monoid M using the k-register mechanism ρ such that ρ satisfies the condition (†). 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 (†). As in the proof of Theorem 12, 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 $Q = M \times \mathcal{D}^k\!/\!\sim$, $I = (1_M, \overline{\bot^k})$, $F = P \times \mathcal{D}^k\!/\!\sim$ (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 $\textsc{Undata}(L)$.

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,

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

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 $\{1 \ldots k\}$, we define π_X the projection over the components of X. If up is an update, we define $\times_{up}$ as the following product: if R is a subset of $\mathcal{D}^{k-|I_{up}|}$ and if R' is a subset of $\mathcal{D}^{|I_{up}|}$, $R \times_{up} R' = \{v \in \mathcal{D}^k \mid \pi_{\overline{up}}(v) \in R \text{ and } \pi_{up}(v) \in R'\}$ where $\overline{up} = \{1 \ldots k\} \setminus I_{up}$. Condition (†) is then equivalent to:

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

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

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

then we can decide (†) !

We note that all the operations from the previous list are elementary operations on equivalence classes.

Example 15 *Let us reconsider data automata constructed from timed automata, as described in Section 5. We will prove that such data automata satisfy condition (†). Assume $\mathcal{A}$ is a timed automaton and consider one of the transitions of the corresponding data automaton,*

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

where there exists $\beta \in \overline{\theta}$ with $\alpha(\beta, d) \in \overline{\theta'}$ ($d \in \mathcal{D}$). Take now $\gamma \in \overline{\theta}$. We have that $\beta \sim \gamma$, thus

$$(\beta_{f(x_0)} - \beta_i)_{1 \leq i \leq 2n} \equiv_2 (\gamma_{f(x_0)} - \gamma_i)_{1 \leq i \leq 2n}$$

There exists a successor of γ such that

$$(d - \beta_i)_{1 \leq i \leq 2n} \equiv_2 (d' - \gamma_i)_{1 \leq i \leq 2n}$$

It is obvious that $\alpha(\gamma, d') \in \overline{\theta'}$. This proves that data automata constructed from timed automata, as in Section 5, satisfy the decidability condition (†).

7 Extensions of the Model

The data automata model we did consider can only store data in the registers, no operation can be performed before updating the registers. In this section, we study several extensions of the model defined in Section 4.

7.1 Erasing and Swapping Registers

The first extension we consider allows to erase and to swap registers. In this section, an update of the registers is then a function *up* which assigns to each register, either the data currently read, or the value of an other register, or the empty data, namely $\perp$. More formally, an update is a function *up* such that there exists an application $\sigma : \{1, \ldots, k\} \longrightarrow \{1, \ldots, k\} \cup \{\perp\} \cup \{\mathbf{c}\}$ such that

$$(\theta'_i)_{i=1,\ldots,k} = up((\theta_i)_{i=1,\ldots,k}, d) \iff \forall i, \begin{cases} \theta'_i = \perp & \text{if } \sigma(i) = \perp \\ \theta'_i = d & \text{if } \sigma(i) = \mathbf{c} \\ \theta'_i = \theta_{\sigma(i)} & \text{otherwise} \end{cases}$$

Proposition 16 *Data automata using this extended type of updates are as expressive as data automata.*

Proof. Let $\mathcal{A} = (Q, k, \Sigma, \mathcal{D}, \sim, q_0, F, T)$ be a data automaton using extended updates, as described above. We construct a data automata $\mathcal{B}$ with simple updates, as defined in Section 4, that recognizes the same data language.

We denote by $\mathcal{F}$ the set of functions $f : \{1, \ldots, k\} \longrightarrow \{0, 1, \ldots, k\}$. Intuitively, the value of the $f(i)^{\text{th}}$ register in the transformed automaton correspond to the value stored in the i^{th} register in the original data automaton. The register "0" is a particular register which is never updated and thus always contains the value $\perp$. Let us now construct the data automaton $\mathcal{B} = (Q', (k+1), \Sigma, \mathcal{D}, \equiv, q'_0, F', T')$ in the following way:

- $Q' = Q \times \mathcal{F}$,
- $q'_0 = (q_0, \mathsf{Id})$, where $\mathsf{Id}(i) = i$ for every register i,
- $F' = F \times \mathcal{F}$,
- $\equiv$ is defined as:

$$\theta \equiv \theta' \iff \forall f \in \mathcal{F},\ (\theta_{f(i)})_{i=1\ldots k} \sim (\theta'_{f(i)})_{i=1\ldots k}$$

- if $(q, g, a, up, g', q') \in T$, then for every f in $\mathcal{F}$, $((q, f), \overline{g}, a, up', \overline{g'}, (q', f'))$ is in T' if

· $(\theta_{f(i)})_{1\leq i\leq k} \in \overline{g} \implies (\theta_i)_{0\leq i\leq k} \in g$
· $(\theta_{f(i)})_{1\leq i\leq k} \in \overline{g'} \implies (\theta_i)_{0\leq i\leq k} \in g'$
· $f'(i) = f(j)$ if *up* puts the value of register j in register i. We denote I the set of i such that $f'(i)$ is defined. If the cardinality of I is k, then f' is totally defined. Otherwise, there is a register, different from 0, say h, which is not in $f'(I)$. For every register i to which *up* assigns the current value of the data, we set $f'(i) = h$. For every register i to which *up* assigns the value $\perp$, we set $f'(i) = 0$.
· *up'* writes the current value in the register h, if defined in the previous item.

From this construction, it is easy to show that $\mathcal{A}$ and $\mathcal{B}$ accept the same data language. □

Erasing or swapping registers are thus macros with no additional expressive power. However, these macros can be very useful. For instance, they are used below to simplify the proof of Proposition 19.

Note that the proof of Proposition 16 also shows that we could restrict our model to updating at most one register on each transition.

7.2 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. We also say that a data language is *nd-recognizable* whenever it is recognized by some non-deterministic data automaton. Some properties which are true for deterministic data automata are also true for non-deterministic data automata:

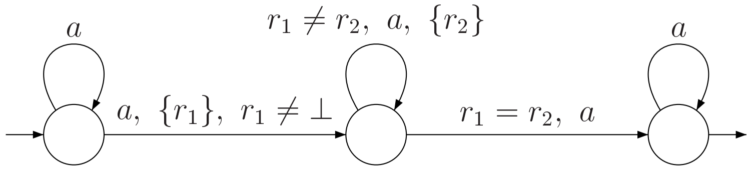
Proposition 17 • *A data language is non-deterministically recognized by a finite monoid if and only if it is nd-recognizable.*
• *Condition (†) ensures the decidability of the emptiness problem, i.e. if a data language is recognized by a non-deterministic data automaton that satisfies*

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. The two first points of the theorem can be proved in the same way as Theorem 12 and Theorem 14. 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:



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

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 stored at the end of the path c. Using a combinatorial argument (see the proof of Proposition 8), there exists two data words in some $E(c)$ that differ only on the data $n(c)$.

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

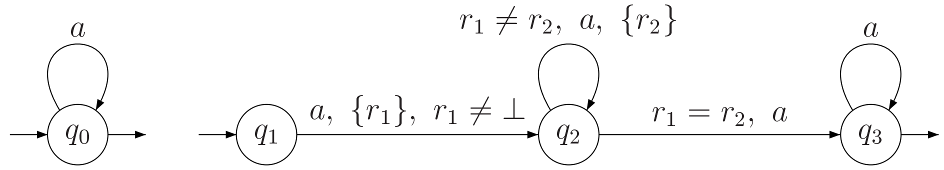
As $\theta_k = \theta'_k$, for each w, $(a, d_1) \dots (a, d_{n(c)}) \dots (a, d_{k+1})$ is in L if and only if $(a, d_1) \dots (a, d'_{n(c)}) \dots (a, d_{k+1})$ is in L. Of course, this is not true. Hence, L is not accepted by any (deterministic) data automaton. $\square$

Corollary 18 *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 19 *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 constructions suffice. 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 $k = \max(k_1, k_2)$ registers. We will use Proposition 16 to prove this result. Assume that, for $i = 1,2$, $\mathcal{A}_i = (Q_i, k_i, \Sigma, \mathcal{D}, \sim_i, q_{0,i}, F_i, T_i)$. We construct the automaton $\mathcal{A} = (Q, k, \Sigma, \mathcal{D}, \sim, q_0, F, T)$, using extended updates as in Section 7.1, such that:

- $Q = Q_1 \cup Q_2$
- $\theta \sim \theta'$ if $\theta_{|1\dots l_i} \sim_i \theta'_{|1\dots l_i}$ for $i = 1,2$
- $q_0 = q_{0,1}$
- $F = \begin{cases} F_2 & \text{if } I_2 \cap F_2 = \emptyset \\ F_1 \cup F_2 & \text{if } I_2 \cap F_2 \neq \emptyset \end{cases}$
- $q \xrightarrow{g,a,up,g'} q' \in T \iff$
$$\begin{cases} \text{either } q \xrightarrow{G,a,up,G'} q' \in T_1 \text{ with } G \implies g \text{ and } G' \implies g' \\ \text{or } q \xrightarrow{G,a,up,G'} q' \in T_2 \text{ with } G \implies g \text{ and } G' \implies g' \\ \text{or } q \in F_1 \text{ and } \exists i \xrightarrow{\overline{\perp^{k_2}},a,\overline{up},G'} q' \in T_2 \text{ with } i \in I_2,\; G' \implies g' \\ \quad \text{and } \overline{up} \text{ puts the current data in the registers of } up \\ \quad \text{and puts } \perp \text{ in the other registers}^{4} \end{cases}$$

The data automaton $\mathcal{A}$ recognizes the data language $L(\mathcal{A}_1) \cdot L(\mathcal{A}_2)$: assume that the data word w is in $L(\mathcal{A}_1) \cdot L(\mathcal{A}_2)$, we can write $w = uv$ where $u \in L(\mathcal{A}_1)$

[4] Proposition 16 is used.

and $v \in L(\mathcal{A}_2)$ and consider the executions in $\mathcal{A}_1$ for u and in $\mathcal{A}_2$ for v:

$$\begin{array}{cccccccc} q_0 \longrightarrow \dots & q_{n-1} & \xrightarrow[d_n]{g_n, a_n, up_n, g'_n} & q_n & \text{and} & q'_0 & \xrightarrow[d'_1]{g_1, a'_1, up_1, g'_1} & q'_1 \dots \\ \theta_0 & \theta_{n-1} & & \theta_n & & \theta'_0 & & \theta'_1 \end{array}$$

where for each i, $g_i \implies G_i$ and $g'_i \implies G'_i$, and $\overline{up_1}$ is defined as in T. It is an execution which accepts w in $\mathcal{A}$. Thus, $L(\mathcal{A}_1) \cdot L(\mathcal{A}_2) \subseteq L(\mathcal{A})$.

Conversely, if $w \in L(\mathcal{A})$, assume that w can be read in $\mathcal{A}$ on the following run:

$$\begin{array}{cccccc} q_0 \longrightarrow \dots & q_{n-1} & \xrightarrow[d_n]{G_n, a_n, up_n, G'_n} & q_n & \xrightarrow[d'_1]{G'_n, a'_1, \overline{up_1}, G'_1} & q'_1 \dots \\ \theta_0 & \theta_{n-1} & & \theta_n & & \theta'_1 \end{array}$$

This run can be splitted into two parts: one in $\mathcal{A}_1$ and an other in $\mathcal{A}_2$:

$$\begin{array}{cccccccc} q_0 \longrightarrow \dots & q_{n-1} & \xrightarrow[d_n]{g_n, a_n, up_n, g'_n} & q_n & \text{et} & q'_0 & \xrightarrow[d'_1]{g_1, a'_1, up_1, g'_1} & q'_1 \dots \\ \theta_0 & \theta_{n-1} & & \theta_n & & \theta'_0 & & \theta'_1 \end{array}$$

These runs accept respectively u and v such that $|u| = n$ and $w = uv$. Thus, $u \in L(\mathcal{A}_1)$ 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 $L(\mathcal{A}_1) \cdot L(\mathcal{A}_2)$.

Note that we could have constructed directly a data automata, as initially defined, to recognizes the data language $L(\mathcal{A}_1) \cdot L(\mathcal{A}_2)$, but the use of proposition 16 makes the proof much easier.

A similar construction can handle the iteration, because, in the previous construction, the number of registers is not increased.

Finally, the data language L considered in the proof of Proposition 17 is nd-recognizable. We can prove that the complement of this data language, namely

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

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 17 or of Proposition 8.
$\square$

7.3 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. The following question arises: does all that precedes generalize 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 20 *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 21 *We note that allowing more general updates enlarges the class of data languages that can be recognized by a monoid. For example, let L be the data language*

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

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 still hold, because these results do not depend on the updates.

8 Comparison with Existing Works

Up to our knowledge, there is no real existing work on the relation between algebra and timed languages. To achieve our goal, we have been led to consider the more general framework of data languages. Forgetting the internal structure of $\Gamma = \Sigma \times \mathcal{D}$, Γ can be viewed as an infinite alphabet, and thus it is much relevant to compare our work to previous works done on languages on infinite alphabets.

In a chronological order, the first work on infinite alphabets has been proposed by Autebert, Beauquier and Boasson in [ABB80]. Several notions of rational and recognizable sets of words have been proposed, among which the following: a language L defined on an infinite alphabet $\mathcal{D}$ is said H-rational if for every finite alphabet X, for every alphabetical morphism $\varphi : \mathcal{D}^* \longrightarrow X^*$, $\varphi(L)$ is regular (*i.e* accepted by a finite automaton). It is easy to see that every H-rational language is also recognizable, as a data language, but the converse is not true. Note that the authors do not propose any automaton or logic-based formalism.

An other related work is proposed by Kaminski and Francez in [KF94]. A notion of *register automata*, quite close to our formalism, is proposed. The class of languages accepted by these automata is closed under union, intersection, concatenation and finite iteration, but it is not closed under complementation. Like our model, an automaton cannot perform any calculation with the registers, but it can only store the data which are read. However, the constraints allowed in this model are restricted to the comparison of the current data with a data stored in one of the registers. A consequence of this restriction is that the letters read in the word are intuitively not very important, such an automaton can only "count" the number of times a letter appears in a word. No other formalism (algebraic or logical) is proposed for this model. Our model is thus an extension of the formalism proposed in [KF94].

The last work we can compare our work with has been done by Neven, Schwentick and Vianu in [NSV01]. The register automata proposed in [KF94] are further studied and the class of *pebble automata* is also proposed. In these automata, some letters of a word can be marked and we can impose conditions on the marked letters. Some logical formalisms are also proposed, but the hierarchies between subclasses of automata and subclasses of the logics are not comparable.

9 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 as an interesting algebraic characterization for timed languages.

A logical characterization of data languages has been proposed [Bou02], extending the work of [Wil94] on timed automata. This theory of data languages has now to be developped. For instance, a notion of aperiodic data language can naturally be defined and has to be studied, with the three points of view, namely monoids, automata and logic.

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. We have seen that a data language is monoid recognizable if and only if it is a recognizable formal language (see proposition 5). 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. Some other aspects could be of interest, like decomposition theorems *à la* Krohn-Rhodes.

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

[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 Symposium on Logic in Computer Science (LICS'97)*, pp. 160–171. IEEE Computer Society Press, 1997.

[ACM02] Asarin, E., Caspi, P., and Maler, O. *Timed Regular Expressions*. Journal of the Association for Computing Machinery (JACM), vol. 49(2):172–206, 2002.

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

[AFH94] Alur, R., Fix, L., and Henzinger, T. A. *A Determinizable Class of Timed Automata*. In *Proc. 6th International Conference on Computer Aided Verification (CAV'94)*, vol. 818 of *Lecture Notes in Computer Science*, pp. 1–13. Springer, 1994.

[Asa98] Asarin, E. *Equations on Timed Languages*. In *Proc. 1st International Workshop on Hybrid Systems: Computation and Control (HSCC'98)*, vol. 1386 of *Lecture Notes in Computer Science*, pp. 1–12. Springer, 1998.

[Bar89] Barrington, D. A. M. *Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in NC^1*. Journal of Computer and Systems Science (JCSS), vol. 38(1):150–164, 1989.

[Bou02] Bouyer, P. *A Logical Characterization of Data Languages*. Information Processing Letters (IPL), vol. 84(2):75–85, 2002.

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

[BP02] Bouyer, P. and Petit, A. *A Kleene/Büchi-like Theorem for Clock Languages*. Journal of Automata, Languages and Combinatorics (JALC), vol. 7(2):167–186, 2002.

[BPT01] Bouyer, P., Petit, A., and Thérien, D. *An Algebraic Characterization of Data and Timed Languages*. In *Proc. 12th International Conference on Concurrency Theory (CONCUR'01)*, vol. 2154 of *Lecture Notes in Computer Science*, pp. 248–261. Springer, 2001.

[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 International Conference on Concurrency Theory (CONCUR'98)*, vol. 1466 of *Lecture Notes in Computer Science*, pp. 455–469. Springer, 1998.

[HKWT95] HENZINGER, T. A., KOPKE, P. W., and WONG-TOI, H. *The Expressive Power of Clocks*. In *Proc. 22nd International Colloquium on Automata, Languages and Programming (ICALP'95)*, vol. 944 of *Lecture Notes in Computer Science*, pp. 417–428. Springer, 1995.

[HLS+93] HERTRAMPF, U., LAUTEMAN, C., SCHWENTICK, T., VOLLMER, H., and WAGNER, K. W. *On the Power of Polynomial Time Bit-Reductions*. In *Proc. 8th Structure in Complexity Theory Conference (CoCo)*, 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 International Colloquium on Automata, Languages and Programming (ICALP'98)*, vol. 1443 of *Lecture Notes in Computer Science*, pp. 580–591. Springer, 1998.

[Kam68] KAMP, J. A. *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 (TCS), vol. 134(2):329–363, 1994.

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

[NSV01] NEVEN, F., SCHWENTICK, T., and VIANU, V. *Towards Regular Languages over Infinite Alphabets*. In *Proc. 26th International Symposium on Mathematical Foundations of Computer Science (MFCS'01)*, vol. 2136 of *Lecture Notes in Computer Science*, pp. 560–572. Springer, 2001.

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

[RS97] ROZENBERG, G. and SALOMAA, A., eds. *Handbook of Formal Languages*. Springer, 1997.

[Sch65] SCHÜTZENBERGER, M.-P. *On Finite Monoids Having only Trivial Subgroups*. Information and Control, vol. 8(2):190–194, 1965.

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