# A Logical Characterization of Data Languages

Patricia BOUYER

LSV - CNRS UMR 8643 - ENS de Cachan
61, Av. du Président Wilson - 94235 Cachan Cedex - France
e-mail: `bouyer@lsv.ens-cachan.fr`

**Keywords:** real-time systems, formal languages.

## 1 Introduction

***The ideal case of formal languages.*** In the formal languages framework, the class of regular languages can be characterized in various ways: finite automata, rational expressions, monadic second-order logic, extended temporal logics, finite monoids... (see *e.g.* [RS97]). Among all these equivalences, the logical characterization is of particular interest since many subclasses of regular languages (like star-free languages or locally threshold testable languages) can be exactly described through fragments of the whole logics, like first-order logics or linear temporal logics [Pin94,PP01]. 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,BBF$^+$01]).

***The timed framework.*** In the framework of timed languages, very useful to specify and verify real-time systems [AD90,ACD93,LPY97,Yov98]..., the situation is far from being so satisfactory. The original class of timed automata, proposed by Alur and Dill [AD90] is not closed under complement. Several Kleene-like theorems [ACM97,BP99,ACM01,BP01] and logical characterizations [Wil94,HRS98] have been proposed for the whole class of timed automata. 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]. From the algebraic point of view, very few was proposed in the literature. In order to try to fill this important gap, we have introduced in [BPT01] a more general notion than timed languages, namely *data languages*.

***The data framework.*** We consider a finite alphabet of actions $\Sigma$ 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 $d \in \mathcal{D}$. In this framework, we proposed together with a notion of data automata, a notion of algebraic characterization using monoids. The languages accepted by data automata (called *recognizable*) are the same as the languages recognized through the monoids. The class of recognizable data languages, even restricted to a time domain is strictly larger than the class of timed languages accepted by some classical timed automaton. For example, the simple language $\{(a,\tau)(a,2\tau)\dots(a,n\tau) \mid \tau > 0 \text{ and } n \text{ integer}\}$ is recognized by no classical timed automaton whereas it is recognized in a simple way by a data automaton.

***Logic for data languages.*** A natural extension of this previous work is thus to make this model more "robust" by proposing a logical characterization of our data automata: in the present paper, we propose a characterization of recognizable data languages by some second-order monadic logic with data predicates. This property language is close to the logic proposed by Wilke [Wil94] to characterize timed automata, but more general. Moreover, as a large subclass of data automata (strictly containing classical timed automata) is decidable [BPT01], we get a logic which is strictly more expressive than the logic proposed by [Wil94] and whose theory is decidable.

*Outline of the paper.* In Section 2, we will present the automata model first defined in [BPT01] and give some examples; in Section 3, we will define our property language ; the three following sections are devoted to the equivalence between the data automata and our property language. Finally, in Section 7, we discuss the decidability and the expressiveness of our logical language.

## 2 The Data Automata Formalism

The first section is devoted to the data automata formalism [BPT01]: we present the formal definition and then illustrate the definition with some examples. Let $\Sigma$ be a finite alphabet and $\mathcal{D}$ a finite (or infinite) set of data ($\mathcal{D}$ can for example be a time domain like the set of positive rational numbers). We assume $\mathcal{D}$ has an initial data, $\perp$, which can be for example 0 if $\mathcal{D}$ is a time domain. A *data word* over $\Sigma$ and $\mathcal{D}$ is an element of $(\Sigma \times \mathcal{D})^*$. A *data language* over $\Sigma$ and $\mathcal{D}$ is a set of data words over $\Sigma$ and $\mathcal{D}$. If $k$ is an integer, an update over $\mathcal{D}^k$ is a subset $up$ of $\{1, \ldots, k\}$. If $\theta \in \mathcal{D}^k$ and $d \in \mathcal{D}$, we note $up(\theta, d)$ the tuple $\theta' \in \mathcal{D}^k$ such that $\theta'_i = d$ if $i \in up$ and $\theta'_i = \theta_i$ otherwise.

**Definition 1.** *A* data automaton *over $\Sigma$ 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,*
- *$k \in \mathbb{N}$ is the number of registers,*
- *$\sim$ is an equivalence relation of finite index defined on $\mathcal{D}^k$,*
- *$T \subseteq Q \times \mathcal{D}^k/\!\sim \times \Sigma \times \mathcal{U} \times \mathcal{D}^k/\!\sim \times Q$ is a finite set of transitions ($\mathcal{U}$ is a set of updates over $\mathcal{D}^k$).*

Let $\mathcal{A} = (Q, q_0, F, k, \sim, T)$ be a data automaton over $\Sigma$ and $\mathcal{D}$. A *path* in $\mathcal{A}$ is a finite sequence of consecutive transitions of $\mathcal{A}$:

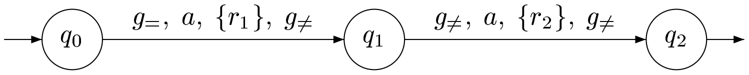
$$q_0 \xrightarrow{g_1, a_1, up_1, g'_1} q_1 \xrightarrow{g_2, a_2, up_2, g'_2} q_2 \ldots \xrightarrow{g_n, a_n, up_n, g'_n} q_n$$

A *run* over this path for the data word $u = (a_1, d_1) \ldots (a_n, d_n)$ is a sequence

$$(q_0, \theta_0) \xrightarrow[d_1]{a_1} (q_1, \theta_1) \xrightarrow[d_2]{a_2} (q_2, \theta_2) \quad \ldots \quad (q_{n-1}, \theta_{n-1}) \xrightarrow[d_n]{a_n} (q_n, \theta_n)$$

such that the sequence $(\theta_i)_{i=0\ldots n}$ defined by $\theta_0 = \perp^k$ and $\theta_{i+1} = up_{i+1}(\theta_i, d_{i+1})$ satisfies $\theta_{i-1} \in g_i$ for $1 \leq i \leq n$ and $\theta_i \in g'_i$ for $1 \leq i \leq n$. The data word is said *accepted* by $\mathcal{A}$ whenever $q_0$ is an initial state and $q_n$ is a final state. The set of data words accepted by $\mathcal{A}$ is denoted by $L(\mathcal{A})$.

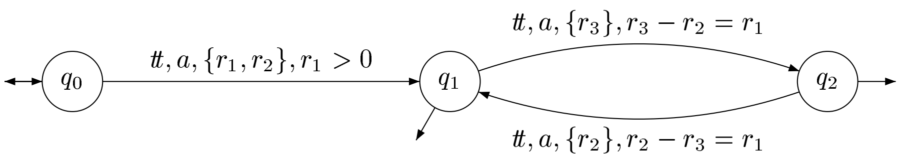
*Example 1.* The data language $L = \{(a, d)(a, d') \mid d \neq \perp,\ d \neq d'\}$ is recognized by the following data automaton with two registers (where $g_= = \{(d, d') \mid d = d' \text{ or } d = \perp\}$ and $g_{\neq} = \{(d, d') \mid d \neq d' \text{ and } d \neq \perp\}$):




In general, *registers* are like memories which keep in mind some data values.

*Example 2.* Timed automata can be transformed into equivalent data automata. Data automata, even restricted to timed languages are strictly more expressive than timed automata. See [BPT01] for a detailed presentation of these properties.

*Example 3.* The language $\{(a, \tau)(a, 2\tau) \ldots (a, n\tau) \mid \tau > 0 \text{ and } n \text{ integer}\}$ is recognized by no classical timed automaton [DZ98]. However, it is recognized by the following data automaton (the notation *tt* is a shortcut to represent any equivalence class):



## 3 Description of the Logic

Our aim is to propose a logical language which is equivalent to data automata. We first consider the logical language $\mathcal{L}_c$ consisting of every closed formula $\varphi ::= \exists X_1 \ldots \exists X_k.\psi$ where $(X_i)_{1\leq i\leq k}$ are second-order variables and $\psi$ is defined inductively by the following grammar:

$$\psi ::= x < y \mid x = y \mid \exists x.\psi \mid \forall x.\psi \mid R_a x \mid Xx \mid g(x_1,\ldots,x_\ell) \mid \neg\psi \mid \psi \wedge \psi \mid \psi \vee \psi$$

where $g$ is some subset of $\mathcal{D}^\ell$, $a \in \Sigma$ and $x, y, x_1, \ldots, x_\ell$ are first-order variables. This logical language is a second-order logic equipped with data predicates $g(x_1,\ldots,x_\ell)$. Informally, this predicate allows to check whether the set of data represented by the variables $x_1,\ldots,x_\ell$ verifies the constraint $g$ or not.

This logical language is interpreted over data words and assignments for the free variables of the formula. In the following, $\mu$ (resp. $\nu$) is a first-order (resp. second-order) variable assignment, *i.e.* a function which associates with each first-order (resp. second-order) variable an integer (resp. a set of integers). We assume that $u = (a_1,d_1)\ldots(a_n,d_n)$ is a data word, that $\mu$ and $\nu$ are assignments for at least every free variables of the formula we consider. The satisfaction relation is then defined inductively as in **Table 1**.

| | | |
|---|---|---|
| $(u;\mu.\nu) \models x < y$ | $\iff$ | $\mu(x) < \mu(y)$ |
| $(u;\mu.\nu) \models x = y$ | $\iff$ | $\mu(x) = \mu(y)$ |
| $(u;\mu.\nu) \models \exists x.\psi$ | $\iff$ | there exists $1 \leq i \leq n,\ (u;\mu[x/i].\nu) \models \psi$ |
| $(u;\mu.\nu) \models \forall x.\psi$ | $\iff$ | for every $1 \leq i \leq n,\ (u;\mu[x/i].\nu) \models \psi$ |
| $(u;\mu.\nu) \models R_a x$ | $\iff$ | $a_{\mu(x)} = a$ |
| $(u;\mu.\nu) \models Xx$ | $\iff$ | $\mu(x) \in \nu(X)$ |
| $(u;\mu.\nu) \models g(x_1,\ldots,x_l)$ | $\iff$ | $(d_{\mu(x_1)},\ldots,d_{\mu(x_l)}) \in g$ |
| $(u;\mu.\nu) \models \neg\psi$ | $\iff$ | $(u;\mu.\nu) \not\models \psi$ |
| $(u;\mu.\nu) \models \psi_1 \wedge \psi_2$ | $\iff$ | $(u;\mu.\nu) \models \psi_1$ and $(u;\mu.\nu) \models \psi_2$ |
| $(u;\mu.\nu) \models \psi_1 \vee \psi_2$ | $\iff$ | $(u;\mu.\nu) \models \psi_1$ or $(u;\mu.\nu) \models \psi_2$ |
| $(u;\mu.\nu) \models \exists X.\psi$ | $\iff$ | there exists $\alpha \subseteq \{1,\ldots,n\}$ such that $(u;\mu.\nu[X/\alpha]) \models \psi$ |

**Table 1.** ***Satisfaction relation for*** $\mathcal{L}_c$

The notation $\mu[x/i]$ means that the assignment $\mu$ is extended to the first-order variable $x$ and that $x$ is mapped onto the integer $i$. The notation $\nu[X/\alpha]$ is similar. If $\varphi$ is a closed formula, *i.e.* there is no free variable in $\varphi$, then the satisfaction relation is defined for empty assignments. In this case, we simply write $u \models \varphi$ for $(u,\emptyset.\emptyset) \models \varphi$. A data language $L$ is the set of *models* of a closed formula $\varphi$ if

$$L = \{u \in (\Sigma \times \mathcal{D})^* \mid u \models \varphi\}.$$

*Example 4.* The data language

$$\{(a_1,d_1)\ldots(a_n,d_n) \mid n \in \mathbb{N} \text{ and } \exists 1 \leq i < j \leq n,\ (a_i,d_i) = (a_j,d_j)\}$$

can be described by the formula $\exists x.\exists y.\ \big(x < y \wedge \bigvee_{a\in\Sigma}(R_a x \wedge R_a y) \wedge g(x,y)\big)$ where $g = \{(d,d') \mid d = d'\}$ is the diagonal of $\mathcal{D}^2$.

*Example 5.* The complementary of the previous language, namely the language

$$\{(a_1,d_1)\ldots(a_n,d_n) \mid n \in \mathbb{N} \text{ and } \forall 1 \leq i < j \leq n,\ (a_i,d_i) \neq (a_j,d_j)\}$$

can be described by the formula $\forall x.\forall y.(x < y \implies g_{\neq}(x,y) \vee \bigwedge_{a\in\Sigma}(\neg R_a x \vee \neg R_a y))$ where $g_{\neq} = \{(d,d') \mid d \neq d'\}$. However, this language is not accepted by any data automaton (see [BPT01]).

Even if $\mathcal{L}_c$ is a natural candidate to express data languages, the previous example shows that the logical language $\mathcal{L}_c$ appears as too much expressive. We therefore consider a restriction of this logical language.

The logical language $\mathcal{L}$ is defined as the set of closed formulae $\exists X_1 \dots \exists X_k.\psi$ where $(X_i)_{1\le i\le k}$ are second-order variables and $\psi$ is defined inductively by the grammar:

$$\psi ::= x < y \mid x = y \mid \exists x.\psi \mid \forall x.\psi \mid R_a x \mid Xx \mid g(X_1, \dots, X_\ell, x) \mid \neg\psi \mid \psi \wedge \psi \mid \psi \vee \psi$$

where $g$ is some constraint of $\mathcal{D}^\ell$ and $a \in \Sigma$.
The only difference with $\mathcal{L}_c$ stands in the predicate $g(X_1, \dots, X_\ell, x)$ whose semantics is defined as follows. Let $u = (a_1, d_1)\dots(a_n, d_n)$ be a data word, let $\mu$ be a first-order variable assignment and let $\nu$ be a second-order variable assignment, then

$$(u; \mu.\nu) \models g(X_1, \dots, X_\ell, x) \iff \left(d^{(\mu,\nu)}_{\max(X_1\le x)}, \dots, d^{(\mu,\nu)}_{\max(X_\ell\le x)}\right) \in g$$

where for every $1 \le i \le \ell$

$$d^{(\mu,\nu)}_{\max(X_i\le x)} = \begin{cases} d_\lambda \text{ if } \lambda \in \nu(X_i) \text{ and } \forall j,\ (j \in \nu(X_i) \text{ and } j \le \mu(x) \text{ implies } j \le \lambda) \\ \perp \text{ if } \forall j,\ j \in \nu(X_i) \text{ implies } \mu(x) < j \end{cases}$$

The idea behind this predicate is the following: the variable $X_i$ represents the set of dates where the register $r_i$ is updated, thus at the date represented by the variable $x$, the data contained in the registers $r_i$ verify the constraint $g$, ($\max(X_i \le x)$ represents the last date the register $r_i$ has been updated).

*Example 6.* The language $\{(a, \tau)\dots(a, n\tau) \mid \tau > 0 \text{ and } n \text{ integer}\}$ can be described by the formula

$$\begin{aligned}\exists X_0 \exists X_1 \exists X_2.\ &[\exists x_0.X_0 x_0 \wedge \forall y.((X_0 y \implies y = x_0) \wedge y \ge x_0) \wedge X_2 x_0 \wedge g_0(X_0, x_0)] \\ &\wedge \forall y \forall x. \begin{bmatrix} (X_1 x \wedge y = x + 1) \implies (X_2 y \wedge \neg X_1 y \wedge g(X_0, X_1, X_2, y)) \\ \wedge\ (X_2 x \wedge y = x + 1) \implies (X_1 y \wedge \neg X_2 y \wedge g(X_0, X_2, X_1, y)) \end{bmatrix}\end{aligned}$$

where the macro $y = x+1$ stands for $\forall z.(x < z \implies y \le z) \wedge (z < y \implies z \le x)$, $g_0 = \{d \mid d > 0\}$ and $g = \{(d_1, d_2, d_3) \mid d_2 - d_3 = d_1\}$.

**Lemma 1.** *The property language $\mathcal{L}$ is not more expressive than the property language $\mathcal{L}_c$, i.e. for any formula $\varphi$ in $\mathcal{L}$, there exists a formula $\psi$ in $\mathcal{L}_c$ which is logically equivalent to $\varphi$.*

*Proof.* It is easy to verify that the macro $g(X_1, \dots, X_\ell, x)$ stands for:

$$\bigvee_{\substack{J \subseteq \{1,\dots,\ell\} \\ J = \{j_1,\dots,j_h\} \\ j_1 < j_2 \dots < j_h}} \exists x_{j_1} \dots \exists x_{j_h}.\ \left(\bigwedge_{i\in J} \Big((X_i x_i) \wedge (x_i \le x) \wedge (\forall y.\ x_i \le y \le x \wedge X_i y \implies x_i = y)\Big) \wedge \Big(\bigwedge_{i \notin J} \forall y.(X_i y \implies x < y)\Big) \wedge \overline{g}(x_{j_1}, \dots, x_{j_h})\right)$$

where $(d_1, \dots d_h) \in \overline{g} \iff (d'_1, \dots, d'_\ell) \in g$ if $\begin{cases} d'_i = d_r \text{ whenever } i = j_r \\ d'_i = \perp \text{ whenever } i \ne j_r \text{ for every } r. \end{cases}$ □

The main result of this paper is the following theorem, which states that the property language $\mathcal{L}$ is equivalent to the data automata formalism. This theorem appears thus as an extension of Büchi's theorem [Büc62] in the formal language framework where it is shown that finite automata are equivalent to the monadic second-order logic. It is also an extension of the result of Wilke [Wil94] who characterizes the timed automata with the monadic second-order logic of distance.

**Theorem 1 (Logical Characterization).** *A data language is recognized by a data automaton if and only if it is the model of some closed formula from $\mathcal{L}$.*

The remainder of this paper is devoted to the proof of this equivalence theorem. The proof is close to the one proposed by Wilke for timed languages [Wil94]. However, it differs on several important technical points. For instance, in [Wil94], the formulae are transformed into formulae with a single first-order variable whereas here, we did choose to use some prenex form of the formulae.

## 4 A "simplified form" for the formulae of $\mathcal{L}$

An *atomic formula* is a formula of the form $x < y$, $x = y$, $R_a x$, $Xx$ or $g(Y_1, \ldots, Y_\ell, x)$. An atomic formula $g(Y_1, \ldots, Y_\ell, x)$ is called an *atomic data predicate*. In a classical way (see *e.g.* [CL93]), every closed formula of $\mathcal{L}$ can be transformed in prenex form:

$$\varphi = \exists X_1 \ldots \exists X_k . Q_1 x_1 \ldots Q_p x_p \left( \bigvee_i \bigwedge_j p_{i,j} \right)$$

where for all $i$, $Q_i \in \{\exists, \forall\}$ and for all $i,\ j$, $p_{i,j}$ is an atomic formula or its negation.
We consider a formula $\varphi$ in such a prenex form. For any second-order variable $Y$ appearing in $\varphi$, there exists an integer $1 \leq i_Y \leq k$ such that $Y = X_{i_Y}$. We then define the equivalence $\sim_\varphi$ on $\mathcal{D}^k$ by:

$$(\theta_i)_{i=1\ldots k} \sim_\varphi (\theta'_i)_{i=1\ldots k} \iff \text{for every atomic data predicate } g(Y_1, \ldots, Y_\ell, x) \text{ of } \varphi, \\ \Big( (\theta_i)_{i=i_{Y_1}\ldots i_{Y_\ell}} \in g \iff (\theta'_i)_{i=i_{Y_1}\ldots i_{Y_\ell}} \in g \Big)$$

Note that this equivalence is of finite index. If $\theta \in \mathcal{D}^k$, the equivalence class of $\theta$ (w.r.t $\sim_\varphi$) is denoted by $cl_\varphi(\theta)$.
Let $g$ be an equivalence class of $\sim_\varphi$, an atomic data predicate of the form $g(X_1, \ldots X_k, x)$ is said to be $\varphi$-*primitive*. It is immediate to verify that every atomic data predicate in $\varphi$ can be transformed into a disjunction of $\varphi$-primitive atomic data predicates.

We thus have obtained the following result:

**Lemma 2.** *Every closed formula $\varphi$ from $\mathcal{L}$ can be transformed into a formula of the form:*

$$\exists X_1 \ldots \exists X_k . Q_1 x_1 \ldots Q_p x_p \left( \bigvee_i \bigwedge_j p_{i,j} \right)$$

*where any $p_{i,j}$ is an atomic formula or its negation and any atomic data predicate is $\varphi$-primitive.*

## 5 From Logic to Data Automata...

The idea is to transform a formula of our logical language $\mathcal{L}$ into a formula of a second-order monadic logic over an (other) finite alphabet for which we will construct an equivalent finite automaton (using Büchi's theorem [Büc62]). We will then transform this finite automaton into a data automaton which will be proved to be equivalent to our initial formula.

Let $\varphi$ be a formula of $\mathcal{L}$. From Lemma 2, we may assume that $\varphi$ is of the form $\exists X_1 \ldots \exists X_k . Q_1 x_1 \ldots Q_p x_p . \bigvee_i \bigwedge_j p_{i,j}$ where for each $h$, $Q_h \in \{\exists, \forall\}$ and for each $i, j$, $p_{i,j}$ is an atomic formula or its negation and every atomic data predicate is $\varphi$-primitive. Let $\psi = \bigvee_i \bigwedge_j p_{i,j}$. All the variables of $\psi$ are free and belong to $\{X_1, \ldots, X_k\} \cup \{x_1, \ldots, x_p\}$. We transform the formula $\psi$ into a formula $\overline{\psi}$ defined inductively by:

$$\begin{array}{|lcl|lcl}
\overline{x < y} & = & x < y & \overline{g(X_1 \ldots X_k, x)} & = & R_g x \\
\overline{x = y} & = & x = y & \overline{\psi_1 \wedge \psi_2} & = & \overline{\psi_1} \wedge \overline{\psi_2} \\
\overline{R_a x} & = & R_a x & \overline{\psi_1 \vee \psi_2} & = & \overline{\psi_1} \vee \overline{\psi_2} \\
\overline{Xx} & = & R_X x & \overline{\neg \psi} & = & \neg \overline{\psi}
\end{array}$$

These formulae interpret over words on the alphabet $(\Sigma \times \mathcal{D}^k/_{\sim_\varphi} \times \mathcal{P}(\{X_1 \ldots X_k\}))^*$ using free variable assignments in the following way ($\mu$ and $\nu$ are resp. first-order and second-order variable assignments and $(a_1, g_1, \mathcal{X}_1) \ldots (a_n, g_n, \mathcal{X}_n)$ is in $(\Sigma \times \mathcal{D}^k/_{\sim_\varphi} \times \mathcal{P}(\{X_1 \ldots X_k\}))^*$):

$$\begin{array}{rcl}
((a_1, g_1, \mathcal{X}_1) \ldots (a_n, g_n, \mathcal{X}_n), \mu.\nu) \models R_g x & \iff & g_{\mu(x)} = g \\
((a_1, g_1, \mathcal{X}_1) \ldots (a_n, g_n, \mathcal{X}_n), \mu.\nu) \models R_X x & \iff & X \in \mathcal{X}_{\mu(x)} \\
((a_1, g_1, \mathcal{X}_1) \ldots (a_n, g_n, \mathcal{X}_n), \mu.\nu) \models R_a x & \iff & a_{\mu(x)} = a
\end{array}$$

The other formulae are interpreted as usual.

Let $u = (a_1, d_1)\dots(a_n, d_n) \in (\Sigma \times \mathcal{D})^*$ be a data word and let $\nu$ be an assignment for the second-order variables $X_1, \dots, X_k$ (for each $X_i$, $\nu(X_i)$ is thus a finite set of integers). We associate with $u$ the words $u'_\nu$ and $u_\nu$ defined as follows:

$$u'_\nu = (a_1, d_1, \theta_1, \mathcal{X}_1)\dots(a_n, d_n, \theta_n, \mathcal{X}_n) \in (\Sigma \times \mathcal{D} \times \mathcal{D}^k \times \mathcal{P}(\{X_1 \dots X_k\}))^*$$

with $\theta_0 = \perp^k$, $\theta_{i+1}[j] = \begin{cases} \theta_i[j] & \text{if } i \notin \nu(X_j) \\ d_i & \text{if } i \in \nu(X_j) \end{cases}$ and $\mathcal{X}_i = \{X_j \mid i \in \nu(X_j)\}$, and

$$u_\nu = (a_1, cl_\varphi(\theta_1), \mathcal{X}_1)\dots(a_n, cl_\varphi(\theta_n), \mathcal{X}_n) \in (\Sigma \times \mathcal{D}^k/_{\sim_\varphi} \times \mathcal{P}(\{X_1 \dots X_k\}))^*.$$

The relation between $u$, $\nu$ and $u_\nu$ is stated by the following lemma:

**Lemma 3.** $(u, \nu) \models Q_1 x_1 \dots Q_p x_p.\psi \iff (u_\nu, \nu) \models Q_1 x_1 \dots Q_p x_p.\overline{\psi}$

*Proof.* Let $\mu$ be a valuation for the first-order variables. We will prove that for each atomic formula $\rho$, $(u, \mu.\nu) \models \rho$ if and only if $(u_\nu, \mu.\nu) \models \overline{\rho}$. We will then get the announced equivalence because $\psi$ is a boolean combination of atomic formulae. Let us prove this equivalence.

- if $\rho = R_a x$, then $\overline{\rho} = R_a x$ and $(u, \mu.\nu) \models R_a x \iff a_{\mu(x)} = a \iff (u_\nu, \mu.\nu) \models R_a x$.
- if $\rho = Xx$, then $\overline{\rho} = R_X x$ and

$$(u_\nu, \mu.\nu) \models R_X x \iff X \in \mathcal{X}_{\mu(x)} \iff \mu(x) \in \nu(X) \iff (u, \mu.\nu) \models Xx.$$

- if $\rho = g(X_1, \dots, X_k, x)$, then $\overline{\rho} = R_g x$ and

$$(u, \mu.\nu) \models g(X_1, \dots, X_k, x) \iff \left(d^{(\mu,\nu)}_{\max(X_1 \leq x)}, \dots, d^{(\mu,\nu)}_{\max(X_k \leq x)}\right) \in g.$$

  But, by construction $\theta_{\mu(x)} = \left(d^{(\mu,\nu)}_{\max(X_1 \leq x)}, \dots, d^{(\mu,\nu)}_{\max(X_k \leq x)}\right)$. Thus, we get that

$$(u_\nu, \mu.\nu) \models R_g x \iff cl_\varphi(\theta_{\mu(x)}) = g. \qquad \square$$

Applying the fundamental result of Büchi [Büc62], we can construct a finite automaton $\overline{\mathcal{A}}$ over the alphabet $(\Sigma \times \mathcal{D}^k/_{\sim_\varphi} \times \mathcal{P}(\{X_1 \dots X_k\}))^*$ that recognizes the language defined by the formula $\exists X_1 \dots \exists X_k.Q_1 x_1 \dots Q_p x_p.\overline{\psi}$. From $\overline{\mathcal{A}}$, we construct a data automaton $\mathcal{A}$ using one register for each second-order variable $X_i$ as follows.

$$\text{if } q \xrightarrow{a,g,\mathcal{X}} q' \text{ is a transition of } \overline{\mathcal{A}} \text{ then } q \xrightarrow{*,a,up,g} q' \text{ is a transition of } \mathcal{A}$$

where $up = \{i \mid X_i \in \mathcal{X}\}$ and $*$ is any equivalence class of $\sim_\varphi$. Initial and final states of $\mathcal{A}$ correspond to initial and final states of $\overline{\mathcal{A}}$. The automaton $\mathcal{A}$ is precisely the one we were looking for:

**Lemma 4.** *A data word $u$ is accepted by $\mathcal{A}$ if and only if $u \models \exists X_1 \dots \exists X_k.Q_1 x_1 \dots Q_p x_p.\psi$.*

*Proof.* We assume that $u = (a_1, d_1)\dots(a_n, d_n)$ is accepted by $\mathcal{A}$ through the path

$$q_0 \xrightarrow{*,a_1,up_1,g_1} q_1 \xrightarrow{*,a_2,up_2,g_2} \dots \xrightarrow{*,a_{n-1},up_{n-1},g_{n-1}} q_{n-1} \xrightarrow{*,a_n,up_n,g_n} q_n$$

Let $\nu$ be the second-order variable assignment defined by $\nu(X_i) = \{j \mid i \in up_j\}$. The word $u_\nu$ that corresponds to $u$ can be read (on the corresponding path) in $\overline{\mathcal{A}}$. Thus, there exists $\nu'$, an assignment for the second-order variables, such that

$$(u_\nu, \nu') \models Q_1 x_1 \dots Q_p x_p.\overline{\psi}$$

Let $\mu$ be a first-order variable assignment, we will prove that for any atomic formula $\rho$, $(u_\nu, \mu.\nu') \models \rho \iff (u_\nu, \mu.\nu) \models \rho$. Indeed,

$$
\begin{array}{lllll}
\cdot\ (u_\nu, \mu.\nu') \models R_g x & \iff & cl_\varphi(\theta_{\mu(x)}) = g & \iff & (u_\nu, \mu.\nu) \models R_g x \\
\cdot\ (u_\nu, \mu.\nu') \models R_a x & \iff & a_{\mu(x)} = a & \iff & (u_\nu, \mu.\nu) \models R_a x \\
\cdot\ (u_\nu, \mu.\nu') \models R_X x & \iff & X \in \mathcal{X}_{\mu(x)} & \iff & \mu(x) \in \nu(X) \iff (u_\nu, \mu.\nu) \models R_X x
\end{array}
$$

Hence, we get that $(u_\nu, \nu) \models Q_1 x_1 \ldots Q_p x_p.\overline{\psi}$. Thus, from Lemma 3, $(u, \nu) \models Q_1 x_1 \ldots Q_p x_p.\psi$.

Conversely, assume that $(u, \nu) \models Q_1 x_1 \ldots Q_p x_p.\psi$. It implies that $(u_\nu, \nu) \models Q_1 x_1 \ldots Q_p x_p.\overline{\psi}$. Thus, $u_\nu$ can be read in $\overline{\mathcal{A}}$. The word $u$ can be read on the corresponding path in $\mathcal{A}$. □

We thus have proved the first implication of Theorem 1.

## 6 ... and Back

We will now prove that any data language accepted by some data automaton can be defined using a formula of $\mathcal{L}$.
Let $\mathcal{A} = (Q, q_0, F, k, \sim, T)$ be a data automaton. We number the set of transitions of $\mathcal{A}$:

$$T = \{t_i = (q_i, g_i, a_i, up_i, g'_i, q'_i) \mid i = 1 \ldots q\}.$$

The macro $x + 1$ has been defined in Example 6; The macro $x \leq y$ stands for $(x < y \vee x = y)$. We use several second-order variables which play two different roles:

- for each transition $t_i$ $(1 \leq i \leq q)$, we use a second-order variable $X_i$ which will contain all the positions where the transition $t_i$ is taken,
- for each register $r_j$ $(1 \leq j \leq k)$, we will use a second-order variable $Y_j$ which will contain all the positions where the register $j$ is updated.

Let $\varphi$ be the formula $\exists X_1 \ldots \exists X_q.\exists Y_1 \ldots \exists Y_k.\Big(\varphi_{pos} \wedge \varphi_{init} \wedge \varphi_{final} \wedge \varphi_{succ} \wedge \varphi_{up} \wedge \varphi_{act} \wedge \varphi_{guards}\Big)$ where

- $\varphi_{pos} = \forall x.\Big(\bigwedge_{i \neq j} (\neg X_i x \vee \neg X_j x) \wedge \bigvee_i X_i x\Big)$

  $\varphi_{pos}$ expresses that each position corresponds exactly to one transition,
- $\varphi_{init} = \exists x.\Big(\Big(\bigvee_{i \text{ s.t. } q_i = q_0} X_i x\Big) \wedge \forall y.\,(x \leq y)\Big)$

  $\varphi_{init}$ expresses that we start a computation in an initial state,
- $\varphi_{final} = \exists x.\Big(\Big(\bigvee_{i \text{ s.t. } q'_i \in F} X_i x\Big) \wedge \forall y.\,(y \leq x)\Big)$

  $\varphi_{final}$ expresses that we end a computation in a final state,
- $\varphi_{succ} = \forall x.\forall y.\Big(y = x + 1 \implies \bigvee_{(i,j) \text{ s.t. } q'_i = q_j} (X_i x \wedge X_j y)\Big)$

  $\varphi_{succ}$ expresses that, in a path, two successive transitions have a common state,
- $\varphi_{up} = \forall x.\Big(\bigwedge_i \Big(X_i x \implies \bigwedge_{j \text{ s.t. } j \in up_i} Y_j x\Big)\Big) \wedge \forall x.\Big(\bigwedge_j \Big(Y_j x \implies \bigvee_{i \text{ s.t. } j \in up_i} X_i x\Big)\Big)$

  $\varphi_{up}$ expresses that each transition updates the correct registers,
- $\varphi_{act} = \forall x.\,(\bigwedge_i X_i x \implies R_{a_i} x)$

  $\varphi_{act}$ expresses that each transition fires the right action,
- $\varphi_{guards} = \forall x.\,(\bigwedge_i (X_i x \implies g_i(Y_1, \ldots, Y_k, x)))$

  $\varphi_{guards}$ expresses that when we take a transition, the registers verify the right constraint.

We now prove the following result:

**Lemma 5.** *For each data word $u$, $u$ is accepted by $\mathcal{A}$ if and only if $u \models \varphi$.*

Assume now that a data word $u = (a_1, d_1) \ldots (a_n, d_n)$ is accepted by the data automaton $\mathcal{A}$. There exists a run in $\mathcal{A}$ which accepts $u$:

$$q_0 \xrightarrow{g_1, a_1, up_1, g_1'} q_1 \xrightarrow{g_2, a_2, up_2, g_2'} \ldots \xrightarrow{g_n, a_n, up_n, g_n'} q_n$$

We define the second-order variable assignment $\nu$ by $\begin{cases} \nu(X_i) = \{j \mid q_{j-1} \xrightarrow{g_j, a_j, up_j, g_j'} q_j = t_i\}, \\ \nu(Y_i) = \{j \mid i \in up_j\}. \end{cases}$

It is without any difficulty to prove that

$$(u, \nu) \models \varphi_{pos} \wedge \varphi_{init} \wedge \varphi_{final} \wedge \varphi_{succ} \wedge \varphi_{up} \wedge \varphi_{act} \wedge \varphi_{guards} \qquad (*)$$

Conversely, assume that there exists some second-order variable assignment $\nu$ such that $(*)$ holds. Then we can construct a path

$$q_0 \xrightarrow{g_1, a_1, up_1, g_1'} q_1 \xrightarrow{g_2, a_2, up_2, g_2'} \ldots \xrightarrow{g_n, a_n, up_n, g_n'} q_n$$

such that $q_{i-1} \xrightarrow{g_i, a_i, up_i, g_i'} q_i$ is the transition $t_j$ if and only if $i \in \nu(X_j)$. It is easy to prove that the data word $u$ can be read on this path in $\mathcal{A}$ and that it is then accepted by the automaton.
This concludes the proof of our main result, Theorem 1.

## 7 Decidability of the Logic

Assume that $\varphi$ is a formula of $\mathcal{L}$. As we wrote in Section 4, $\varphi$ defines an equivalence relation $\sim_\varphi$ on $\mathcal{D}^k$ for some $k$ (depending on $\varphi$). Assume that the following condition holds:

$$\text{if } up \subseteq \{1, \ldots, k\},\ \theta_1 \sim \theta_2 \in \mathcal{D}^k,\ d_1 \in \mathcal{D}, \text{ there exists } d_2 \in \mathcal{D} \text{ s.t. } up(\theta_1, d_1) \sim up(\theta_2, d_2) \qquad (\ddagger)$$

for some refinement $\sim$ of the equivalence relation $\sim_\varphi$. Then a data automaton with guards verifying condition $(\ddagger)$ belongs to a decidable class of automata (see [BPT01] for a proof). Thus, using what precedes, we get the following theorem:

**Theorem 2.** *The restriction of the logic $\mathcal{L}$ to formulae for which condition* $(\ddagger)$ *holds is decidable.*

Even if in general the condition $(\ddagger)$ is not decidable, it holds for large classes of natural updates and equivalence relations on $\mathcal{D}^k$.
Using Example 6, it is easy to verify that the formula proposed to describe the timed language $L = \{(a, \tau) \ldots (a, n\tau) \mid \tau > 0 \text{ and } n \text{ integer}\}$ verifies condition $(\ddagger)$. For example you can consider the equivalence relation $\sim$ defined on $\mathcal{D}^3$ by:

$$(\theta_i)_{i=1,2,3} \sim (\theta_i')_{i=1,2,3} \iff \begin{cases} \theta_1 = \theta_3 - \theta_2 \iff \theta_1' = \theta_3' - \theta_2' \\ \theta_1 = \theta_2 - \theta_3 \iff \theta_1' = \theta_2' - \theta_3' \\ \theta_i > 0 \iff \theta_i' > 0 \text{ for each } i = 1, 2, 3 \end{cases}$$

Thus, as $L$ is accepted by no classical timed automaton, there is no formula in the logic presented in [Wil94] equivalent to the formula of Example 6. Moreover, every classical timed automaton can be transformed into an equivalent data automaton which verifies condition $(\ddagger)$. The decidable fragment of our logic is thus strictly more expressive than the monadic logic of distance proposed in [Wil94], even if we restrict to timed behaviors.

## 8 Conclusion

In [BPT01], we proposed a class of languages, containing the timed languages, for which we have defined some kind of recognizability using on the one hand automata and on the other hand finite monoids. This new class of languages seems to be a nice candidate to be a robust model for real-time systems. To improve it, it is important to see if it can be characterized through an interesting logic. That's precisely this last point which motivated this paper. Moreover, a large subclass of the logic we propose has a decidable theory and this decidable subclass is strictly more expressive than the monadic logic of distance proposed in [Wil94]. Further studies will consist in studying the property language $\mathcal{L}_c$ and also in proposing some kind of Kleene's Theorem for recognizable data languages.

**Acknowledgments:** I would like to thank Antoine Petit for his careful reading of drafts of this paper.

## References


[ACD93] ALUR, R., COURCOUBETIS, C., and DILL, D. *Model-Checking in Dense Real-Time*. Information and Computation, vol. 104(1):2–34, 1993.

[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).

[AD90] ALUR, R. and DILL, D. *Automata for Modeling Real-Time Systems*. In *Proc. 17th International Colloquium on Automata, Languages, and Programming (ICALP'90)*, vol. 443 of *Lecture Notes in Computer Science*, pp. 322–335. Springer-Verlag, 1990.

[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.

[BBF+01] BÉRARD, B., BIDOIT, M., FINKEL, A., LAROUSSINIE, F., PETIT, A., PETRUCCI, L., and SCHNOEBELEN, P. *Systems and Software Verification - Model-Checking Techniques and Tools*. Springer-Verlag, 2001.

[Büc62] BÜCHI, J. R. *On a Decision Method in Restricted Second Order Arithmetic*. In *Proc. (1960) Int Congr. Logic, Methodology and Philosophy of Science*, pp. 1–11. Stanford University Press, 1962.

[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)*, vol. 2154 of *Lecture Notes in Computer Science*, pp. 248–261. Springer-Verlag, 2001.

[CGP99] CLARKE, E., GRUMBERG, O., and PELED, D. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[CL93] CORI, R. and LASCAR, D. *Logique mathématique*. Masson, 1993.

[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.

[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.

[LPY97] LARSEN, K. G., PETTERSSON, P., and YI, W. UPPAAL *in a Nutshell*. Journal of Software Tools for Technology Transfer, vol. 1(1–2):134–152, Oct. 1997.

[Pin94] PIN, J.-E. *Logic on Words*. Bulletin of the European Association of Theoretical Computer Science, 1994. Vol. 54, pp. 145–165.

[PP01] PERRIN, D. and PIN, J.-E. *Infinite Words*, 2001. Can be found on `http://www.liafa.jussieu.fr/~jep/Resumes/InfiniteWords.html`.

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

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

[Yov98] YOVINE, S. *Model-Checking Timed Automata*. In *School on Embedded Systems*, vol. 1494 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.