On the use of guards for logics with data

Thomas Colcombet[1], Clemens Ley[2], Gabriele Puppis[2]

[1] CNRS/LIAFA
[2] Department of Computer Science, Oxford University

Abstract. The notion of orbit finite data monoid was recently introduced by Bojańczyk as an algebraic object for defining recognizable languages of data words. Following Büchi's approach, we introduce the new logic 'rigidly guarded MSO' and show that the data languages definable in this logic are exactly those recognizable by orbit finite data monoids. We also establish, following this time the approach of Schützenberger, McNaughton and Papert, that the first-order variant of this logic defines exactly the languages recognizable by aperiodic orbit finite data monoids. Finally, we give a variant of the logic that captures the larger class of languages recognized by non-deterministic finite memory automata.

1 Introduction

Data languages are languages over an infinite alphabet – the letters of which are called data values – which are closed under permutation of the data values. This invariance under permutation makes any property concerning the data values, other than equality, irrelevant. Some examples of data languages are:

- *The sets of words containing exactly k distinct data values.*
- *The sets of words where the first and last data values are equal.*
- *The sets of words with no consecutive occurrences of the same data value.*
- *The sets of words where each data value occurs at most once.* ($\star$)

The intention behind data values in data words (or data trees, ...) is to model, e.g. the id's in a database, or the process or users numbers in the log of a system. Those numbers are used as identifiers, and we are interested only in comparing them by equality. The invariance under permutation of data languages captures this intention. Data words can also be defined to have both a data value and a letter from a finite alphabet at each position. This is more natural in practice, and does not make any difference in the results to follow.

The paper aims at understanding better how the classical theory of regular languages can be extended to data languages. The classical theory associates regular languages to finite state automata or, equivalently, to finite monoids. For instance, important properties of regular languages can be detected by exploiting equivalences with properties of the monoid – see, e.g. Straubing's book [14] or Pin's survey [11] for an overview of the approach.

Recently, Bojańczyk introduced the notion of *data monoids* [3] as a framework for algebraic characterizations of data languages. Bojańczyk focused on the languages of data words recognizable by *orbit finite data monoids*, an analog of

finite monoids for data languages. All the above examples but $\star$ are recognizable with this definition. Our main objective is to understand better the expressive power of the orbit finite data monoid model by comparing it with automaton-based models and logical formalisms for data words.

In terms of logic, there is a natural way to define logics for data words. It is sufficient for this to use a predicate $x \sim y$ meaning that the data values at positions x and y are equal. In particular, one may think that the monadic (second-order) logic with this new predicate is a good candidate to equivalently specify recognizable languages, i.e., would play the role of monadic logic in the standard theory of regular languages. However, this is not the case, as monadic logic happens to be much too expressive. One inclusion indeed holds: every language of data words recognized by an orbit finite monoid is definable in monadic logic. However, the converse does not hold, as witnessed by the formula

$$\forall\, x, y.\; x \neq y \to x \not\sim y, \tag{$\star\star$}$$

which defines the above (non-recognizable) language $\star$. More generally, it has been shown that monadic logic (indeed, even first-order logic with data equality) has an undecidable satisfiability problem and it can express properties not implementable by automaton models, such as finite memory automata (FMA, described below) [10]. We naturally aim at answering the following question:

> *Is there a natural variant of monadic logic which defines precisely the data languages recognizable by orbit finite data monoids?*

We answer this question positively, introducing *rigidly guarded MSO* (abbreviating *monadic second-order logic with rigidly guarded data equality tests*). This logic allows testing equality of two data values only when the two positions are related in an injective way (we say rigid). That is, data equality tests are allowed only in formulas of the form $\varphi(x, y) \wedge x \sim y$, where φ is rigid, i.e., defines a partial injection. For instance, it is easy to check whether there are two consecutive positions sharing the same data value, e.g., by the formula $\exists\, x, y.\; (x = y+1) \wedge x \sim y$. The guard $(x = y + 1)$ is rigid since x uniquely determines y, and vice versa. However, it is impossible to describe the language $\star$ in this logic. In particular, the above formula $\star\star$ can be rewritten as $\neg\exists\, x, y.\; (x \neq y) \wedge x \sim y$, but this time the guard $x \neq y$ is not rigid: for a given x, there can be several y such that $x \neq y$. It may seem a priori that the fact that rigidity is a semantic property is a severe drawback. This is not the case since (i) rigidity can be enforced syntactically, and (ii) rigidity is decidable for formulas in our logic.

To validate the robustness of our approach, we also answer to the following question inspired from the seminal Schützenberger-McNaughton-Papert result:

> *Does the rigidly guarded FO logic (i.e., the first-order fragment of rigidly guarded MSO) correspond to aperiodic orbit finite data monoids?*

We answer this question positively as well. We finally consider data languages recognizable by finite memory automata and we prove that a natural variant of rigidly guarded MSO, called $\exists$*backward-rigidly guarded MSO* captures the class of data languages recognized by non-deterministic finite memory automata.

Overall, we don't claim that data languages recognizable by orbit finite data monoids are the counterpart to the notion of regular languages in the standard theory, since this model is rather expressively weak (see related work below). However, in this restricted framework, we are able to recover several of the major results which hold for usual regular languages.

Related work. This work is highly related to the well known theory of regular languages. We refer by this to the key equivalence between monadic logic and regular languages due to Büchi [5], and the Schützenberger-McNaughton-Papert result that characterizes the subclass of first-order definable languages [13, 9].

The other branch of related work is the one concerned with languages of data words. The first contribution in this direction is due to Kaminski and Francez [7], who introduced finite memory automata (FMA for short). These automata possess a fixed finite set of registers that can store data values. At each step such an automaton can compare the current data value with the values stored in the registers, and can decide to store this value in some register (forgetting the previous content of the register). This model of automaton, in its non-deterministic form, has a decidable emptiness problem and an undecidable universality problem (decidability of the latter problem is recovered in the deterministic variant).

Recently, the deterministic model of FMA has been modified by requiring a stricter policy in the use of registers [2]. This modification does not affect the expressive power of the model, but, as opposed to the original model, the new model can be efficiently minimized. In [1] partial results on relating automata to logics are also given. The question of characterizing the first-order logic definable language among the languages recognized by deterministic FMA is still open.

Many other automaton models for data languages have been studied in the literature (see [12] for a survey). These include pebble automata [10] and data automata [4], the latter introduced as a mean of capturing decidable logics over data words. The algebraic theory for these models has not been developed, nor is there an exact characterization of definability in logics for any of these models.

Contribution and structure of the paper. These are our contributions:

1. We show how infinite orbit finite data monoids can be finitely represented.
2. We introduce a new logic called "rigidly guarded MSO" – a natural weakening of MSO logic with data equality tests. Although the syntax of our logic is based on a semantic property, one can decide whether a formula belongs to the logic or not.
3. We show that satisfiability of rigidly guarded MSO formulas is decidable.
4. We show that rigidly guarded MSO is as expressive as orbit finite data monoids, and that its first-order fragment corresponds precisely to aperiodic orbit finite data monoids.
5. We give a decidable variant of rigidly guarded MSO that captures the data languages recognized by non-deterministic finite memory automata and has a decidable satisfiability problem. We also provide a decidable logic for data trees along the same lines.

Section 2 gives preliminaries on data languages and data monoids, and explains how to define representations of data monoids with finitely many orbits. Section 3 introduces rigidly guarded MSO and its first-order fragment. Section 4 describes the translation from rigidly guarded MSO (resp., FO) formulas to orbit finite data monoids (resp., aperiodic orbit finite data monoids) recognizing the same languages. Section 5 describes the converse translation, namely, from (aperiodic) orbit finite data monoids to rigidly guarded MSO (resp., FO) formulas. Section 6 introduces a variant of rigidly guarded MSO that captures precisely the class of languages recognized by non-deterministic finite memory automata. Finally, Section 7 provides an assessment of the results and related open problems.

Acknowledgments. We would like to thank Michael Benedikt and Anca Muscholl for the many helpful remarks on the paper.

2 Data Monoids

In this paper, D will usually denote an infinite set of *data values* (e.g., $d, e, f, \ldots$) and A will denote a finite set of *symbols* (e.g., $a, b, c, \ldots$). A *data word* over the alphabet $D \times A$ is a finite sequence $u = (d_1, a_1) \ldots (d_n, a_n)$ in $(D \times A)^*$. The domain of u, denoted $\mathcal{D}om(u)$, is $\{1, \ldots, n\}$.

Given a set $C \subseteq D$ of data values, a *(data) renaming* on C is a permutation on C that is the identity for all but finitely many values of C. We denote by G_C the set of all renamings on C. Renamings are naturally extended to tuples of data values, data words, sets of data words, and so on. A *data language* over $D \times A$ is a set of data words in $(D \times A)^*$ that is closed under renamings in G_D.

Recall that a monoid is an algebraic structure $\mathcal{M} = (M, \cdot)$ where $\cdot$ is an associative product on M and M contains an identity $1_\mathcal{M}$. A monoid is *aperiodic* if for all elements s, there is n such that $s^n = s^{n+1}$. We say that the set G_C of renamings *acts* on a monoid $\mathcal{M} = (M, \cdot)$ if there is a function $\hat{}$ that maps every renaming $\tau \in G_C$ to an automorphism $\hat{\tau}$ on $\mathcal{M}$. That is, for all renamings $\tau, \pi \in G_C$ and all elements $s, t \in M$, we have (i) $\widehat{\tau \circ \pi} = \hat{\tau} \circ \hat{\pi}$, (ii) $\hat{\tau}_{\mathrm{id}}(s) = s$, where τ_{id} is the identity function on C, (iii) $\hat{\tau}(s) \cdot \hat{\tau}(t) = \hat{\tau}(s \cdot t)$, and (iv) $\hat{\tau}(1_\mathcal{M}) = 1_\mathcal{M}$. For example, consider the free monoid $((D \times A)^*, \cdot)$ consisting of all finite words over $D \times A$ equipped with the operation of juxtaposition (the empty word ε playing the role of the identity). The group G_D of data renamings acts on the free monoid when the action is defined by $\hat{\tau}((d_1, a_1) \ldots (d_n, a_n)) = (\tau(d_1), a_1) \ldots (\tau(d_n), a_n)$.

We say that a renaming τ is a *stabilizer* of an element s of a monoid $\mathcal{M}$ acted upon by G_C, if $\hat{\tau}(s) = s$. A set $C' \subseteq D$ of data values *supports* an element s if all renamings that are the identity on C' are stabilizers of s. It is known that the intersection of two sets that support s is a set that supports s as well [3, 6]. Hence we can define *the memory* of s, denoted $\mathrm{mem}(s)$, as the intersection of all sets that support s. Note that there are finite monoids whose elements have infinite memory (see [3] for an example). On the other hand, monoids that are homomorphic images of the free monoid contains only elements with finite memory. As we are interested in homomorphic images of the free monoid, we

will consider monoids whose elements have finite memory (this property is called *finite support axiom* and the resulting algebraic objects *data monoids*).

Definition 1. *A* data monoid $\mathcal{M} = (M, \cdot, \hat{\ })$ *over* C *is a monoid* $(M, \cdot)$ *that is acted upon by* G_C*, in which every element has finite memory.*

Unless otherwise stated, data monoids are defined over the set D of data values.

The *orbit* of an element s of $\mathcal{M} = (M, \cdot, \hat{\ })$ is the set of all elements $\hat{\tau}(s)$ with $\tau \in G_D$. Note that two orbits are either disjoint or equal. We say that $\mathcal{M}$ is *orbit finite* if it contains finitely many orbits. It is easy to see that if two elements are on the same orbit, then their memories have the same size. Hence an orbit finite data monoid has a uniform bound on the size of the memories (this is not true for arbitrary data monoids).

A *morphism* between two data monoids $\mathcal{M} = (M, \cdot, \hat{\ })$ and $\mathcal{N} = (N, \odot, \check{\ })$ is a monoid morphism that commutes with the action of renamings.A data language $L \subseteq (D \times A)^*$ is *recognized* by a morphism $h : (D \times A)^* \to \mathcal{M}$ if the membership of a word $u \in (D \times A)^*$ in L is determined by the element $h(u)$ of $\mathcal{M}$, namely, if $L = h^{-1}(h(L))$. As L is closed under renamings, $h(L)$ is a union of orbits.

Finite presentations of data monoids. Since orbit finite data monoids are infinite objects, we need suitable representations that ease algorithmic manipulation. The basic idea is to consider the restriction of an orbit finite data monoid to a finite set of data values:

Definition 2. *Given a data monoid* $\mathcal{M} = (M, \cdot, \hat{\ })$ *and* $C \subseteq D$*, we define the* restriction *of* $\mathcal{M}$ *to* C *to be* $\mathcal{M}|_C = \big(M|_C, \cdot|_C, \hat{\ }|_C\big)$*, where* $M|_C$ *consists of all elements* $s \in M$ *such that* $\mathrm{mem}(s) \subseteq C$*,* $\cdot|_C$ *is the restriction of* $\cdot$ *to* $M|_C$*, and* $\hat{\ }|_C$ *is the restriction of* $\hat{\ }$ *to* G_C *and* $M|_C$*.*

Note that $s \cdot t \in M|_C$ and $\hat{\tau}(s) \in M|_C$ for all $s, t \in M|_C$ and $\tau \in G_C$. Hence, if C is finite, $\mathcal{M}|_C$ is a finite data monoid.[3] Hereafter, we denote by $\|\mathcal{M}\|$ the maximum cardinality of the memories of the elements of an orbit finite data monoid $\mathcal{M}$.

Proposition 1. *Let* $\mathcal{M}$*,* $\mathcal{M}'$ *be orbit finite data monoids such that* $\|\mathcal{M}\| = \|\mathcal{M}'\|$ *and let* $C \subseteq D$ *be of cardinality at least* $2\|\mathcal{M}\|$*. If* $\mathcal{M}|_C$ *and* $\mathcal{M}'|_C$ *are isomorphic, then so are* $\mathcal{M}$ *and* $\mathcal{M}'$*.*

The above proposition shows that the restriction of an orbit finite data monoid $\mathcal{M}$ over a *sufficiently large* finite set C uniquely determines $\mathcal{M}$. A more careful analysis shows that many operations on orbit finite data monoids (e.g., the product of two such monoids, the quotient with respect to a congruence) can be performed at the level of the finite restriction. This allows us to effectively compute the results of algebraic operations on orbit finite data monoids.

Term-based presentations of data monoids. We have just shown how we can represent an infinite data monoid by a finite one. It is also possible to give

[3] One has to keep in mind that data monoids over finite sets do not satisfy the same properties as those over infinite sets. For instance, the Memory Theorem, as stated in [3], does not hold for data monoids over finite sets.

a more explicit presentation of orbit finite data monoids using what we call a *term-based presentation system*. Each element is a term of the form $o(d_1, \ldots, d_k)$ in which o is an *orbit name* (belonging to some fixed set) of a fixed arity k, and $d_1, \ldots, d_k$ are distinct values. Those terms are furthermore considered modulo an equivalence relation, and equipped with a binary operation. Such a presentation is valid if the binary operation is associative over the equivalence classes, and if the data values respect the renaming policy required for data monoids. Under those suitable assumptions, the equivalence classes of terms equipped with the associative operation as product and the natural renaming operations form a data monoid. Furthemore, if there are finitely many orbit names, then the represented data monoid is orbit finite. We also show that conversely, every orbit finite data monoid can be represented by such a term-based representation, using finitely many orbit names.

This kind of presentation ease algorithmic manipulations of the elements of the data monoid, and are heavily used in the proofs. Some open questions are directly related to this presentation such as: is it possible to get rid of the equivalence relation for recognizing a language of data words?

3 Rigidly guarded logics

From now on, we abbreviate monadic second-order logic with data equality tests by *MSO*. MSO formulas are built up from atoms of the form $x < y$, $a(x)$, $x \in X$, or $x \sim y$, using boolean connectives and existential quantifications over first-order variables (e.g., $x, y, \ldots$) and monadic second-order variables (e.g., $X, Y, \ldots$). The meaning of an atom $x \sim y$ is that the data values at the two positions that correspond to the interpretation of the variables x and y must be the same. The meaning of the other predicates is as usual.

Here we introduce a new logic called "rigidly guarded MSO". We say that a formula $\varphi(x, y)$ with two free first-order variables x, y is *rigid* if for all data words $u \in (D \times A)^*$ and all positions x (resp., y) in u, there is *at most one* position y (resp., x) in u such that $u \vDash \varphi(x, y)$. *Rigidly guarded MSO* is obtained from MSO by enforcing the following restriction: every data equality test of the form $x \sim y$ must be guarded by a rigid formula $\varphi(x, y)$. Precisely, the formulas of rigidly guarded MSO are build up using the following grammar:

$$\varphi \;:=\; \exists\, x.\, \varphi \mid \exists\, Y.\, \varphi \mid x < y \mid a(x) \mid x \in Y \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi_{\text{rigid}}(x, y) \wedge x \sim y$$

where $a \in A$ and $\varphi_{\text{rigid}}(x, y)$ is a *rigid* formula that is generated by the same grammar. *Rigidly guarded FO* is the first-order fragment of rigidly guarded MSO.

The notion of rigidity is a semantic property, and this may seem problematic. However, we can enforce rigidity syntactically as follows. Instead of a guard $\varphi_{\text{rigid}}(x, y)$ in a formula, one uses the new guard

$$\widetilde{\varphi}_{\text{rigid}}(x, y) \;=^{\text{def}}\; \varphi_{\text{rigid}}(x, y) \wedge \forall\, x', y'.\, \varphi_{\text{rigid}}(x', y') \to (x = x' \leftrightarrow y = y')\ .$$

It is easy to check that $\widetilde{\varphi}_{\texttt{rigid}}$ is always rigid, and that furthermore, if φ_{rigid} is rigid then it is equivalent to $\widetilde{\varphi}_{\texttt{rigid}}$. This trick allows us to enforce rigidity

syntactically. We will also see in Corollary 2 below that one can decide if a formula respects the rigidity assumption in all its guards (the problem being undecidable when data tests are not guarded).

We remark that in this logic, the similar constructions $\varphi_{\text{rigid}}(x,y) \wedge x \not\sim y$, $\varphi_{\text{rigid}}(x,y) \to x \sim y$, and $\varphi_{\text{rigid}}(x,y) \to x \not\sim y$ can be derived. This is thanks to the Boolean equivalences $\varphi \to \varphi'$ iff $\varphi \to (\varphi \wedge \varphi')$, $\varphi \wedge \neg\varphi'$ iff $\neg(\varphi \to \varphi')$, and $\varphi \to \neg\varphi'$ iff $\neg(\varphi \wedge \varphi')$.

Example 1. Let us consider the language $L_{\geq k}$ of all data words that contain at least k different data values. If $k = 1$ we just need to check the non-emptiness of the word by the sentence $\exists\, x.\ \texttt{true}$. For $k = 2$ it is sufficient to test for the existence of two distinct *consecutive* data values, using for instance the formula $\exists\, x, y.\ (x + 1 = y) \wedge x \not\sim y$. For $k > 2$, one can proceed by induction as follows. One first observes that if a word has at least k distinct data values, then there is a minimal factor witnessing this property, say $[x, y]$. A closer inspection reveals that, in this case, $[x + 1, y - 1]$ is a maximal factor that uses exactly $k - 2$ data values and thus belongs to the language $L_{\geq k-2} \setminus L_{\geq k-1}$. By induction, the fact that $[x+1, y-1]$ is a maximal factor that belongs to $L_{\geq k-2} \setminus L_{\geq k-1}$ is expressible in rigidly guarded FO by a formula $\varphi(x, y)$. Furthermore, this formula $\varphi(x, y)$ is *rigid* according to its semantic definition. We conclude that the language $L_{\geq k}$ is defined by the formula $\exists\, x, y.\ \varphi(x, y) \wedge x \not\sim y$.

To simplify the notation, it is sometimes convenient to think of a first-order variable x as a second-order variable X interpreted as a singleton set. Therefore, by a slight abuse of notation, we shall often write variables in uppercase letters, without explicitly saying whether these are first-order or second-order variables (their correct types can be inferred from the atoms they appear in). As usual, we write $\varphi(X_1, \ldots, X_m)$ whenever we want to make explicit that the free variables of φ are among $X_1, \ldots, X_m$. Moreover, given a formula $\varphi(X_1, \ldots, X_m)$, a data word $u \in (D \times A)^*$, and some unary predicates $U_1, \ldots, U_m \subseteq \mathcal{D}om(u)$, we write $u \vDash \varphi(U_1, \ldots, U_m)$ whenever φ holds on u by interpreting the free variables $X_1, \ldots, X_m$ with the predicates $U_1, \ldots, U_m$.

As usual, given a formula $\varphi(\bar{X})$ with some free (first-order or monadic second-order) variables $X_1, \ldots, X_m$, one can see it as defining the language

$$[\![\varphi]\!] \;=\; \{\langle u, U_1, \ldots, U_m\rangle \;:\; u \vDash \varphi(U_1, \ldots, U_m)\} \;\subseteq\; (D \times A \times B^m)^*$$

where B denotes the binary alphabet $\{0, 1\}$ and $\langle u, U_1, \ldots, U_m\rangle$ is the word over the alphabet $D \times A \times B^m$ that has letter $(d, a, b_1, \ldots, b_m)$ at position i iff (d, a) is the i-th letter of u, and for all $j = 1 \ldots m$, b_j is 1 if $i \in U_j$, and 0 otherwise.

4 From the logic to data monoids

In this section, we show that every data language defined by a rigidly guarded MSO sentence is recognized by an orbit finite data monoid. Our proof follows the classical technique for showing that MSO definable languages over standard words can be recognized by monoids. Namely, we show that each construction

in the logic corresponds to a closure under some operation on recognizable languages: disjunction corresponds to union, negation corresponds to complement, existential quantification corresponds to projection, etc.

The principle of the proof is to establish that, given a rigidly guarded MSO formula $\varphi(\bar{X})$, the language $[\![\varphi]\!]$ is recognized by an orbit finite data monoid. Though this statement is true, it cannot be used – as it is the case in the standard theory – as an induction hypothesis. The problem is that the operation that corresponds to existential quantification (i.e. projection) transforms an orbit finite data monoid into a data monoid which is not orbit finite, in general. That is why our induction hypothesis is stronger, and states that $[\![\varphi]\!]$ is recognized by an orbit finite data monoid via a *projectable* morphism, to be defined below (we write $s \doteq t$ whenever the elements s and t are in the same orbit):

Definition 3. *Let h be a morphism from the free data monoid $(D \times A \times B^m)^*$ to a data monoid $\mathcal{M} = (M, \cdot, \hat{\ })$. We say that h is* projectable *if for all data words $u \in (D \times A)^*$ and all tuples of predicates $\bar{U} = (U_1, \dots, U_m)$ and $\bar{V} = (V_1, \dots, V_m)$,*

$$h(\langle u, \bar{U} \rangle) \doteq h(\langle u, \bar{V} \rangle) \quad \textit{implies} \quad h(\langle u, \bar{U} \rangle) = h(\langle u, \bar{V} \rangle) \ .$$

We now state the theorem, which is at the same time our induction hypothesis:

Theorem 1. *For all rigidly guarded MSO formulas $\varphi(\bar{X})$, the language $[\![\varphi]\!]$ is effectively recognized by an orbit finite data monoid with a projectable morphism.*

From the above theorem we obtain, in particular, the following key corollaries:

Corollary 1. *Every data language definable in rigidly guarded MSO (resp., rigidly guarded FO) is recognizable by an orbit finite data monoid (resp., aperiodic orbit finite data monoid).*

Note that the claim for the first-order case is deduced using the result that every language recognized by an orbit finite data monoid and definable in FO (without any rigidity assumption) is recognized by an aperiodic orbit finite data monoid [3]. That is why Theorem 1 needs not to consider the first-order case.

Corollary 2. *The satisfiability problem for rigidly guarded MSO logic is decidable. Moreover, one can decide whether a formula belongs to the rigidly guarded MSO logic, and in this case whether the formula is rigid.*

The proof of Theorem 1 is by structural induction on the rigidly guarded MSO formulas: the translation of the atomic formulas $x < y$, $a(x)$, $x \in Y$ are easy (at least towards non-aperiodic monoids) and the translations of the Boolean connectives are as in the classical case.

The translation of the existential closures uses a powerset construction on orbit finite data monoids. Since data monoids are in general infinite objects, the standard powerset construction would yield infinitely many orbits even if the original data monoid has finitely many of them. In our case, the construction remembers all possible elements of the original monoid, but since the morphism is projectable, one never has to store more than one element per orbit. Indeed,

whenever another element in the same orbit is encountered, it has to be equal to the one already present: this limitation allows us to preserve orbit finiteness.

The most technical part concerns the translation of the rigidly guarded data tests $\varphi(x,y) \wedge x \sim y$. The rigidity assumption on the guard $\varphi(x,y)$ is crucial for this result: if $\varphi(x,y)$ were not rigid, then the data monoid recognizing $[\![\varphi(x,y) \wedge x \sim y]\!]$ would still be orbit finite, but the morphism would in general not be projectable. The proof that $[\![\varphi(x,y) \wedge x \sim y]\!]$ is recognized via a projectable morphism requires a bit of analysis since rigidity is a semantic assumption and hence one cannot directly deduce from it a property for the data monoid. However, one can use the rigidity property for "normalizing" the data monoid, allowing the construction to go through.

5 From data monoids to the logic

Having shown that every language defined by a rigidly guarded MSO (resp., FO) formula is recognized by an orbit finite data monoid (resp., by an aperiodic orbit finite data monoid), we now show the converse.

Theorem 2. *Given an orbit finite data monoid $\mathcal{M}$, a morphism h from the free data monoid to $\mathcal{M}$, and an orbit o, one can compute a rigidly guarded MSO sentence φ that defines the data language $L = h^{-1}(o)$. Moreover, if $\mathcal{M}$ is aperiodic, then φ is a rigidly guarded FO sentence.*

This is the most technical result of the paper. Note that in the classical theory of regular languages, the analogous of Theorem 2 (at least the part involving only MSO) is straightforward: indeed, a monoid can be used as an automaton, and it is sufficient to write an MSO formula that guesses a run of such an automaton and checks that it is valid and accepting. We cannot use such a proof for data monoids: not only there is no equivalent automaton model, but furthermore, the above approach is intrinsically not compatible with the notion of rigidity.

Our proof follows a structure similar to Schützenberger's proof that languages recognized by aperiodic monoids are definable by star-free expressions (i.e., in FO logic). The proof relies on an induction on the structure of ideals of the data monoid, the so called *Green's relations* [11]. This requires specific study of this theory for orbit finite data monoids. Such a study was initiated by Bojańczyk [3], but we had to develop several new tools for our proof to go through (these tools concern the size of $\mathcal{H}$-classes and the analysis of the memory inside the $\mathcal{J}$-classes). As opposed to the classical case, the proof is significantly more involved for MSO compared to FO.

6 Logics for finite memory automata

In this section, we try to see how guards as introduced above can help constructing decidable logics. We consider languages recognized by *finite memory automata* (FMA) [7]. These extend finite state automata by a finite set of registers, storing values from an infinite alphabet D. Data words are processed from

left to right. At each step the automaton compares (up to equality) the current input value with the values that are stored in its registers. Based on this information, the automaton decides whether or not to store the input value in one of its registers, updates its state, and moves one symbol to the right.

The deterministic variant of FMA can be viewed as the natural automaton counterpart of orbit finite data monoids. However, deterministic FMA are more expressive than orbit finite data monoids, as witnessed by the language

$$L \stackrel{\text{def}}{=} \{d_1 \dots d_n \ : \ n \in \mathbb{N},\ d_1 = d_i \text{ for some } 1 < i \leq n\}$$

which is recognizable by deterministic FMA, but not by orbit finite data monoids. Moreover, unlike classical finite automata, non-deterministic FMA are even more expressive than deterministic FMA, as witnessed by the language

$$L' \stackrel{\text{def}}{=} \{d_1 \dots d_n \ : \ n \in \mathbb{N},\ d_i = d_j \text{ for some } 1 \leq i < j \leq n\}.$$

It thus comes natural to look for logical characterizations of data languages recognizable by deterministic (resp., non-deterministic) FMA.

A natural attempt at finding a logic for FMA consists in relaxing the notion of rigidity. One could imagine using backward-rigid guards for data tests. These are formulas $\varphi(x, y)$ that determine the leftmost position $\min(x, y)$ from the rightmost position $\max(x, y)$ (but possibly not the other way around). Formulas of *backward-rigidly guarded MSO* are built up using the grammar:

$$\varphi \coloneqq \exists\, x.\, \varphi \mid \exists\, Y.\, \varphi \mid x < y \mid a(x) \mid x \in Y \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi_{\text{backward}}(x, y) \wedge x \sim y$$

where $\varphi_{\text{backward}}$ is a backward-rigid formula generated from the same grammar (as usual, we can enforce backward-rigidity syntactically). For example the above language L can be easily defined by the backward-rigidly guarded MSO sentence $\exists\, x, y.\, (x < y \wedge \forall\, z.\, x \leq z) \wedge x \sim y$. One can translate backward-rigidly guarded MSO formulas to equivalent deterministic FMA, but not the other way around:

Proposition 2. *Every language definable in backward-rigidly guarded MSO is recognizable by deterministic FMA. There is a language recognized by a deterministic FMA which cannot be defined in backward-rigidly guarded MSO.*

We do not have a candidate logic that corresponds precisely to deterministic FMA. However, we are able to characterize the larger class of languages recognized by non-deterministic FMA. The logic for this class is obtained from backward-rigidly guarded MSO by allowing the guards to use additional second-order variables (which however needs to be quantified existentially in the outermost part of the formula). The logic, abbreviated $\exists$*backward-rigidly guarded MSO*, consists of the formulas $\exists\, \bar{Z}.\, \varphi$, with φ is generated by the grammar

$$\varphi \coloneqq \exists x.\, \varphi \mid \exists Y.\, \varphi \mid x{<}y \mid a(x) \mid x{\in}Y \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi_{\exists\text{backward}}(x, y, \bar{Z}) \wedge x{\sim}y$$

where $\varphi_{\exists\text{backward}}$ is a formula from the same grammar that determines $\min(x, y)$ from $\max(x, y)$ and $\bar{Z}$, and where the quantifications are over variables different from $\bar{Z}$ (the variables $\bar{Z}$ are quantified only in the outermost part of $\exists\, \bar{Z}.\, \varphi$).

Theorem 3. *A language is definable in* $\exists$*backward-rigidly guarded MSO iff it is recognizable by non-deterministic FMA.*

As it happens for rigidly guarded MSO logic, one can derive from Theorem 3 the following decidability results:

Corollary 3. *The satisfiability problem for ∃backward-rigidly guarded MSO is decidable. Moreover, one can decide whether a formula belongs to the ∃backward-rigidly guarded MSO, and in this case whether the formula is ∃backward-rigid.*

It is also easy to generalize both Theorem 3 and Corollary 3 to data *tree* languages recognized by non-deterministic finite memory tree automata [8]. For this we use a natural variant of ∃backward-rigidly guarded MSO on data trees. The guarded tests in this case are of the form

$$\varphi_{\exists\,\text{downward}}(x, y, \bar{Z}) \ \wedge \ \varphi'_{\exists\,\text{downward}}(x, z, \bar{Z}) \ \wedge \ y \sim z$$

where $\varphi_{\exists\,\text{downward}}(x, y, \bar{Z})$ (resp. $\varphi'_{\exists\,\text{downward}}(x, z, \bar{Z})$) is a formula in the logic that determines the position y (resp., z) from an ancestor x in the data tree and the second-order variables $\bar{Z}$. This logic happens to be equivalent with the natural extension of non-deterministic FMA to trees.

Finally, it is natural to look for effective characterizations of data languages that are both recognizable by non-deterministic FMA and definable in (unrestricted) FO. However, it is known that such characterization cannot be achieved: in [1] it has been shown that the problem of determining whether a language recognized by a non-deterministic FMA is definable in FO is undecidable. The problem of characterizing FO within the class of languages recognizable by deterministic FMA is still open.

7 Conclusion and future work

We have shown that the algebraic notion of orbit finite data monoid corresponds to a variant of the logic MSO which is – and this is of course subjective – natural. It is natural in the sense that it only relies on a single and understandable principle: guarding data equality tests by rigidly definable relations.

We believe that this notion of guard is interesting by itself. Of course, it is not the first time that guards are used to recover some decidability properties from a too expressive logic. What is more original in the present context is the equivalence with the algebraic object, which shows that this approach is in some sense maximal: it is not just a particular technique among others for having decidability, but it is sufficient for completely capturing the expressiveness of the very natural algebraic model.

Another contribution of the present work is the development of the structural understanding of orbit finite data monoids. By structural understanding, we refer to Green's relations. These relations form a major tool in most involved proofs concerning finite monoids. The corresponding study of Green's relations for orbit finite data monoids was already a major argument in the proof of [3], and it had to be developed even further in the present work.

Finally, we proved that a variant of the same logic captures the larger class of data languages recognized by non-deterministic NFA.

We are only at the beginning of understanding the various notions of recognizability for data languages. However, several interesting questions were raised during our study. Some of them concern the fine structure of the logic:

> *The nesting level of guards seems to be a robust and relevant parameter in our logic. Can we understand it algebraically? Can we decide it?*

Other questions concern the more general model of FMA:

> *Can we characterize among the languages recognized by deterministic FMA the ones recognizable by orbit finite data monoids? Can we give a logic equivalent to deterministic FMA and characterize its FO fragment?*

References

[1] M. Benedikt, C. Ley, and G. Puppis. Automata vs. logics on data words. In *Proceedings of the 19th EACSL Annual Conference on Computer Science Logic (CSL)*, volume 6247 of *Lecture Notes in Computer Science*, pages 110–124. Springer, 2010.

[2] M. Benedikt, C. Ley, and G. Puppis. What you must remember when processing data words. In *Proceedings of the 4th Alberto Mendelzon International Workshop on Foundations of Data Management (AMW)*, volume 619 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

[3] M. Bojańczyk. Data monoids. In *Proceedings of the 28th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 9 of *LIPIcs*, pages 105–116. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[4] M. Bojańczyk, A. Muscholl, T. Schwentick, L. Segoufin, and C. David. Two-variable logic on words with data. In *Proceedings of the 21th IEEE Symposium on Logic in Computer Science (LICS)*, pages 7–16. IEEE Computer Society, 2006.

[5] R.J. Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 6(1-6):66–92, 1960.

[6] M. Gabbay and A.M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363, 2002.

[7] M. Kaminski and N. Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.

[8] M. Kaminski and T. Tan. Tree automata over infinite alphabets. In *Proceedings of the Pillars of Computer Science, Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, volume 4800 of *Lecture Notes in Computer Science*, pages 386–423. Springer, 2008.

[9] R. McNaughton and S. Papert. *Counter-free Automata*. M.I.T. Research Monograph. Elsevier MIT Press, 1971.

[10] F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Transactions on Computational Logic*, 5(3):403–435, 2004.

[11] J.E. Pin. Mathematical foundations of automata theory. Available on: `http://www.liafa.jussieu.fr/~jep/MPRI/MPRI.html`, 2010.

[12] T. Schwentick. Automata for XML - a survey. *Journal of Computer and System Sciences*, 73(3):289–315, 2007.

[13] M.P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965.

[14] H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhauser Verlag, 1994.