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

$$\forall\, x, y.\ x \neq y \rightarrow x \nsim y, \tag{$\star\star$}$$

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 $u \models \varphi(x, y)$. *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:

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

where $a \in A$ and $\varphi_{\mathrm{rigid}}(x, y)$ 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 $\varphi_{\mathrm{rigid}}(x, y)$ in a formula, one uses the new guard

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

It is easy to check that $\widetilde{\varphi}_{\mathtt{rigid}}$ is always rigid, and that furthermore, if φ_{rigid} is rigid then it is equivalent to $\widetilde{\varphi}_{\mathtt{rigid}}$. 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 $\varphi_{\mathrm{rigid}}(x,y) \wedge x \nsim y$, $\varphi_{\mathrm{rigid}}(x,y) \rightarrow x \sim y$, and $\varphi_{\mathrm{rigid}}(x,y) \rightarrow x \nsim y$ can be derived. This is thanks to the Boolean equivalences $\varphi \rightarrow \varphi'$ iff $\varphi \rightarrow (\varphi \wedge \varphi')$, $\varphi \wedge \neg\varphi'$ iff $\neg(\varphi \rightarrow \varphi')$, and $\varphi \rightarrow \neg\varphi'$ 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 $\exists\, x.\, \mathtt{true}$. For $k = 2$ it is sufficient to test for the existence of two distinct *consecutive* data values, using for instance the formula $\exists\, x,y.\ (x+1 = y) \wedge x \nsim y$. 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 $\exists\, x,y.\ \varphi(x,y) \wedge x \nsim y$.

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

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

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

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

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:

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

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

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

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* $\exists$*backward-rigidly guarded MSO is decidable. Moreover, one can decide whether a formula belongs to the* $\exists$*backward-rigidly guarded MSO, and in this case whether the formula is* $\exists$*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 $\exists$backward-rigidly guarded MSO on data trees. The guarded tests in this case are of the form

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

where $\varphi_{\exists\,\mathrm{downward}}(x,y,\bar{Z})$ (resp. $\varphi'_{\exists\,\mathrm{downward}}(x,z,\bar{Z})$) 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.

A Proofs for Section 2 (Data monoids)

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 $\mathcal{M}'$.*

Proof. Let $\mathcal{M} = (M, \cdot, \hat{\ })$ and $\mathcal{M}' = (M', \odot, \check{\ })$ and let f_C be a data monoid isomorphism from $\mathcal{M}|_C$ to $\mathcal{M}'|_C$. We show how extend f_C to an isomorphism from $\mathcal{M}$ to $\mathcal{M}'$. Given $s \in M$, we let τ be any renaming such that $\hat{\tau}(s) \in M|_C$ (note that such a renaming exists since $|C| \geq |\mathrm{mem}(s)|$) and we accordingly define

$$f(s) \;=\; \hat{\tau}^{-1}(f_C(\hat{\tau}(s))).$$

We prove that the function f is well-defined, namely, that $f(s)$ does not depend on the choice of the renaming τ. To do that, we consider two renamings τ and σ such that $\hat{\tau}(s), \hat{\sigma}(s) \in M|_C$, we define $t = \hat{\tau}^{-1}(f_C(\hat{\tau}(s)))$ and $t' = \hat{\sigma}^{-1}(f_C(\hat{\sigma}(s)))$, and we prove that $t = t'$. Consider the data renaming $\pi \coloneqq \sigma \circ \tau^{-1}$. As in this case $\sigma = \pi \circ \tau$, we get

$$t' \;=\; \check{\sigma}^{-1}(f_C(\hat{\sigma}(s))) \;=\; \check{\sigma}^{-1}(f_C(\hat{\pi}(\hat{\tau}(s)))).$$

Similarly, as $f_C \circ \hat{\pi} = \check{\pi} \circ f_C$ and as $\tau^{-1} = \sigma^{-1} \circ \pi$

$$\check{\sigma}^{-1}(f_C(\hat{\pi}(\hat{\tau}(s)))) \;=\; \check{\sigma}^{-1}(\check{\pi}(f_C(\hat{\tau}(s))) \;=\; \check{\tau}^{-1}(f_C(\hat{\tau}(s))) \;=\; t$$

which proves that the function f is well defined.

We claim that f is a bijection from M to M'. Surjectivity of f is straightforward, since for every element $s' \in M'$, there exists a renaming τ such that $\hat{\tau}(s') \in M'|_C$ and hence, if we let $s = \hat{\tau}^{-1}(f_C^{-1}(\hat{\tau}(s')))$, we have $f(s) = s'$. The proof that f is injective is analogous to the proof that f is well defined, and thus omitted.

Commutativity with renamings. Now, we show that f commutes with the action of renamings. Given an element $s \in M$ and a renaming $\pi \in G_D$, we choose a renaming τ such that both $\hat{\tau}(s)$ and $\hat{\tau}(\hat{\pi}(s))$ belong to $M|_C$ (note that such a renaming exists since $|C| \geq |\mathrm{mem}(s) \cup \mathrm{mem}(\hat{\pi}(s))|$). We also define the renaming $\sigma = \tau \circ \pi \circ \tau^{-1}$. Note that, by construction, we have $\hat{\sigma}(\hat{\tau}(s)) = \hat{\tau}(\hat{\pi}(s))$. Finally, by exploiting the definition of f and the fact that f_C is a data monoid morphism from $\mathcal{M}|_C$ to $\mathcal{M}'|_C$, we obtain

$$\begin{aligned}
f(\hat{\pi}(s)) &= \check{\tau}^{-1}(f_C(\hat{\tau}(\hat{\pi}(s)))) \\
&= \check{\tau}^{-1}(f_C(\hat{\sigma}(\hat{\tau}(s)))) \\
&= \check{\tau}^{-1}(\check{\sigma}(f_C(\hat{\tau}(s)))) \\
&= \check{\pi}(\check{\tau}^{-1}(f_C(\hat{\tau}(s)))) \\
&= \check{\pi}(f(s)).
\end{aligned}$$

Commutativity with products. We conclude the proof by showing that f is a monoid morphism from $\mathcal{M}$ to $\mathcal{M}'$. Clearly, since $M|_C$ (resp., $M'|_C$) contains the identity $1_{\mathcal{M}}$ of $\mathcal{M}$ (resp., the identity $1_{\mathcal{M}'}$ of $\mathcal{M}'$) and since f_C is a monoid morphism from $\mathcal{M}|_C$ to $\mathcal{M}'|_C$, we have that $f(1_{\mathcal{M}}) = f_C(1_{\mathcal{M}}) = 1_{\mathcal{M}'}$. Let us now consider two elements $s, t \in M$. Let τ be a renaming such that both $\hat{\tau}(s)$ and $\hat{\tau}(t)$ belong to $M|_C$ (again, such a renaming exists since $|C| \geq |\mathrm{mem}(s) \cup \mathrm{mem}(t)|$). Since f_C is a monoid morphism, we obtain

$$\begin{aligned}
f(s \cdot t) &= \check{\tau}^{-1}(f_C(\hat{\tau}(s \cdot t))) \\
&= \check{\tau}^{-1}(f_C(\hat{\tau}(s) \cdot \hat{\tau}(t))) \\
&= \check{\tau}^{-1}(f_C(\hat{\tau}(s)) \odot f_C(\hat{\tau}(t))) \\
&= \check{\tau}^{-1}(f_C(\hat{\tau}(s))) \odot \check{\tau}^{-1}(f_C(\hat{\tau}(t))) \\
&= f(s) \odot f(t).
\end{aligned}$$

We can conclude that f is a data monoid isomorphism from $\mathcal{M}$ to $\mathcal{M}'$. □

In Section 2 we informally described term-based presentation systems for orbit finite data monoids. Below, we give a formal definition of them. We denote by $T_{O,C}$ the set of all *terms* of the form $o(d_1, \dots, d_k)$, where o is an orbit name from a finite set O, with associated arity k, and $d_1, \dots, d_k$ are pairwise distinct data values from a fixed (possibly infinite) set C.

Definition 4. *Let O be a set of orbit names and let C be a (finite or infinite) set of data values. A* term-based presentation system *$\mathcal{S}$ over (O, C) consists of a set of terms $T = T_{O,C}$, a binary operation $\odot$ on T, an action $\check{}$ defined by $\check{\tau}(o(d_1, \dots, d_n)) = o(\tau(d_1), \dots, \tau(d_n))$, and a congruence $\approx$ for $\odot$ on T such that, for all terms $s, t, u \in T$ and all renamings $\tau \in G_C$,*

1. *Equivariance: $\check{\tau}(s) \odot \check{\tau}(t) = \check{\tau}(s \odot t)$,*
2. *Congruence for renamings: if $s \approx t$ then $\check{\tau}(s) \approx \check{\tau}(t)$,*
3. *Associativity up to $\approx$: $(s \odot t) \odot u \approx s \odot (t \odot u)$,*
4. *Identity: there is a distinguished term $1 \in T$ satisfying $1 \odot t = t \odot 1 = t$.*

Note that as $\approx$ is a congruence for $\odot$, $s \approx s'$ and $t \approx t'$ imply $s \odot t \approx s' \odot t'$. If $\mathcal{S} = (T, \odot, \check{}, \approx)$ is a term-based presentation system then $(T, \odot, \check{})$ is not necessarily a data monoid because associativity only holds up to $\approx$.

We say that $\mathcal{S}$ *represents* the triple $\mathcal{M} = (M, \cdot, \hat{})$ if

1. M is the set of $\approx$-equivalence classes of terms from T,
2. $\cdot$ is a binary product on M defined by $[s]_\approx \cdot [t]_\approx = [s \odot t]_\approx$,
3. $\hat{}$ maps any renaming $\tau \in G_C$ to the function on M defined by $\hat{\tau}([s]_\approx) = [\check{\tau}(s)]_\approx$

(it is easy to check that both $\cdot$ and $\hat{}$ are well defined).

Proposition 3. *Every term-based presentation system represents a data monoid. Conversely, every orbit finite data monoid is represented by a term-based presentation system.*

Proof. We prove the first claim. Let $\mathcal{S} = (T, \odot, \check{}, \approx)$ be a term-based presentation system over C and let $\mathcal{M} = (M, \cdot, \hat{})$ be the tuple represented by $\mathcal{S}$. It follows from Conditions 1. and 2. of the definition of a term-based presentation system that $(M, \cdot)$ is a monoid. We verify the other properties of data monoids:

1. We first check that $\widehat{\tau \circ \pi} = \hat{\tau} \circ \hat{\pi}$ for all data renamings $\tau, \pi \in G_C$. Let $[o(\bar{d})]$ be an element of M (we will drop the subscript $\approx$ in the rest of this proof). Then
$$\widehat{\tau \circ \pi}[o(\bar{d})] = [o((\tau \circ \pi)(\bar{d}))] = [o(\tau(\pi(\bar{d})))] = \hat{\tau}([o(\pi(\bar{d}))]) = \hat{\tau} \circ \hat{\pi}([o(\bar{d})]).$$
2. Clearly if τ_{id} is the identity data renaming then $\hat{\tau}_{\mathrm{id}}([o(\bar{d})]) = [o(\tau_{\mathrm{id}}(\bar{d}))] = [o(\bar{d})]$.
3. Let $[s], [t] \in M$ and a data renaming $\tau \in G_C$ be given. We need to check that $\hat{\tau}([s] \cdot [t]) = \hat{\tau}([s]) \cdot \hat{\tau}([t])$. We assume that $s = o(\bar{d})$, $t = p(\bar{e})$, and $s \odot t = q(\bar{f})$. Then
$$\hat{\tau}([s] \cdot [t]) = \hat{\tau}([s \odot t]) = \hat{\tau}([q(\bar{f})]) = [q(\tau(\bar{f}))] = [\check{\tau}(q(\bar{f}))] = [\check{\tau}(s \odot t)].$$
From Condition 3. of of Definition 4 we know
$$\check{\tau}(s \odot t) \approx \check{\tau}(s) \odot \check{\tau}(t).$$
We continue our calculation as follows
$$\begin{aligned}[\check{\tau}(s \odot t)] &= [\check{\tau}(s) \odot \check{\tau}(t)] = [\check{\tau}(o(\bar{d})) \odot \check{\tau}(p(\bar{e}))] = [o(\tau(\bar{d})) \odot p(\tau(\bar{e}))] \\ &= [o(\tau(\bar{d}))] \cdot [p(\tau(\bar{e}))] = \hat{\tau}([s]) \cdot \hat{\tau}([t]).\end{aligned}$$
Combining the two equations we get that $\hat{\tau}([s] \cdot [t]) = \hat{\tau}([s]) \cdot \hat{\tau}([t])$.
4. We finally show that $\hat{\tau}(1) = 1$, where 1 is the identity of $\mathcal{M}$. This follows from Condition 2. of Definition 4.

The second claim requires a more technical proof. Let $\mathcal{M} = (M, \cdot, \hat{\ })$ be an orbit finite data monoid over C. If C is infinite, then we assume without loss of generality that the data values in C are the positive natural numbers. If C is finite, then we assume that C is a prefix of $\mathbb{N}$. We first define a term-based representation system $S = (T, \odot, \check{\ }, \approx)$ and later we show that $\mathcal{S}$ represents $\mathcal{M}$.

Definition of $\mathcal{S}$. Let O be a set of orbit names that contains exactly one orbit name o for each orbit of $\mathcal{M}$. The arity of o is the arity of the orbit that o corresponds to. We define T to be the set of terms that are build up from orbit symbols in O and data values in C. Recall that the action is completely constraint in a term-based representation system: It has to be defined by $\check{\tau}(o(\bar{d})) = o(\tau(\bar{d}))$ for all data renamings $\tau \in G_C$.

The idea of the composition operation $\odot$ is to first map two terms s, t to two terms $\tilde{s}, \tilde{t}$ that are canonical some sense. Then we map $\tilde{s}, \tilde{t}$ to the monoid, where we can compute the product. Then we basically apply the reverse operation to obtain a term u that we define to be the product of s and t.

More formally, we define two functions f and g that that relate the terms in T with the elements of $\mathcal{M}$. We fix a representative m_o inside each orbit o of $\mathcal{M}$ in such a way that $\mathrm{mem}(m_o)$ is a prefix of the natural numbers, namely, $\mathrm{mem}(m_o) = \{1, \ldots, \mathrm{arity}(o)\}$. We associate with each sequence of data values $\bar{d} = d_1, \ldots, d_k$ a renaming $\sigma_{\bar{d}}$ that maps the numbers $1, \ldots, k$ to the values $d_1, \ldots, d_k$, respectively, and that is the identity on $D \setminus \{1, \ldots, k, d_1, \ldots, d_k\}$. We then define the function f from T to M such that, for every term $o(\bar{d})$,

$$f(o(\bar{d})) \;=\; \hat{\sigma}_{\bar{d}}(m_o).$$

Note that f is not injective in general. We define $\approx$ by $s \approx t$ iff $f(s) = f(t)$.

To define the composition $\odot$ we need to choose a distinguished element in every set $f^{-1}(m)$ for $m \in M$. This can be accomplished by fixing a function g from M to T such that if $m \in M$ then $g(m)$ is the term among $f^{-1}(m)$ whose data values are minimal with respect to the lexicographical order.

We now define what we meant above by canonical terms. We say that a pair of terms $o(d_1, \ldots, d_n)$ and $p(e_1, \ldots, e_m)$ is canonical, if $d_1, \ldots, d_n = 1, \ldots, n$ and if $e_i, e_j \in \{e_1, \ldots, e_m\} \setminus \{d_1, \ldots, d_n\}$ and $e_i < e_j$ then $i < j$. Clearly for every pair of terms s, t there is a data renaming $\sigma_{s,t}$ such that $\sigma_{s,t}(s), \sigma_{s,t}(t)$ are canonical.

Now we can define the product: For every pair terms s, t in T we define $s \odot t$ to be an element in $f^{-1}(f(s) \cdot f(t))$, more precisely we define

$$s \odot t \;=\; \check{\sigma}^{-1}_{s,t} \circ g\ (f \circ \hat{\sigma}_{s,t}\ (s) \cdot f \circ \hat{\sigma}_{s,t}\ (t)).$$

Finally we define the identity element $1_{\mathcal{S}}$ of $\mathcal{S}$ to be $g(1_{\mathcal{M}})$ where $1_{\mathcal{M}}$ is the identity element of $\mathcal{M}$. This completes the definition of $\mathcal{S} = (T, \odot, \check{\ }, \approx)$.

$\mathcal{S}$ is a term-based presentation system. Before we prove that $\mathcal{S}$ satisfies the conditions of Definition 4, we establish the following claim.

Claim. *Let $s, t \in T$ be terms and let τ be a data renaming. Then*

(C1) $f(\check{\tau}(t)) = \hat{\tau}(f(t))$

(C2) $f(s \odot t) = f(s) \cdot f(t)$

(C3) $\tau \circ \sigma_{s,t}\ (d) = \sigma_{\tau(s),\tau(t)}(d)$ *and* $\sigma_{\tau(s),\tau(t)} \circ \tau\ (d) = \sigma_{s,t}(d)$ *for all* $d \in \mathrm{mem}(s) \cup \mathrm{mem}(t)$.

Proof of Claim. We first prove Condition C1. Recall that if $\bar{d} = d_1, \ldots, d_k$ is a sequence of values, then $\sigma_{\bar{d}}$ is the data renaming that maps the numbers $1, \ldots, k$ to the values $d_1, \ldots, d_k$ respectively, and that is the identity on $D \setminus \{1, \ldots, k, d_1, \ldots, d_k\}$. We note that for all data renamings τ and all numbers $i \leq k$,

$$\tau \circ \sigma_{\bar{d}}\ (i) \;=\; \sigma_{\tau(\bar{d})}(i). \qquad (\star)$$

This is because for all $i \leq k$, $\tau \circ \sigma_{\bar{d}}\ (i) = \tau(d_i) = \sigma_{\tau(\bar{d})}(i)$. We can now show the claim: Let $\tau \in G_C$ and $t = o(\bar{d}) \in T$ with $\bar{d} = d_1, \ldots, d_k$ be given. Then

$$\begin{aligned}
f(\check{\tau}(o(\bar{d}))) &= f(o(\tau(\bar{d}))) && \text{(by definition of } \check{\ }\text{)}\\
&= \hat{\sigma}_{\tau(\bar{d})}(m_o) && \text{(by definition of } f\text{)}\\
&= \widehat{\tau \circ \sigma_{\bar{d}}}(m_o) && \text{(by } \star \text{ and because } \mathrm{mem}(m_o) \subseteq \{1, \ldots, k\}\text{)}\\
&= \hat{\tau} \circ \hat{\sigma}_{\bar{d}}\ (m_o) && \text{(because } \mathcal{M} \text{ is a data monoid)}\\
&= \hat{\tau}(f(o(\bar{d}))) && \text{(by definition of } f.\text{)}
\end{aligned}$$

We now prove Condition C2:

$$\begin{aligned}
f(s \odot t) &= \hat{\sigma}^{-1}_{s,t} \circ \hat{\sigma}_{s,t} \circ f\ (s \odot t) && \text{(because } \hat{\sigma}^{-1}_{s,t} \circ \hat{\sigma}_{s,t} = \text{id)}\\
&= \hat{\sigma}^{-1}_{s,t} \circ f \circ \check{\sigma}_{s,t}\ (s \odot t) && \text{(by Condition C1)}\\
&= \hat{\sigma}^{-1}_{s,t} \circ f \circ \check{\sigma}_{s,t}\ (\check{\sigma}^{-1}_{s,t} \circ g\ (f \circ \hat{\sigma}_{s,t}\ (s) \cdot f \circ \hat{\sigma}_{s,t}\ (t)))\\
& && \text{(by definition of } \odot\text{)}\\
&= \hat{\sigma}^{-1}_{s,t} \circ f\ (g\,(f \circ \check{\sigma}_{s,t}\ (s) \cdot f \circ \check{\sigma}_{s,t}\ (t)))\\
& && \text{(because } \check{\sigma}_{s,t} \circ \check{\sigma}^{-1}_{s,t} = \text{id)}\\
&= \hat{\sigma}^{-1}_{s,t}(f \circ \check{\sigma}_{s,t}\ (s) \cdot f \circ \check{\sigma}_{s,t}\ (t)) && \text{(because } f \circ g = \text{id)}\\
&= \hat{\sigma}^{-1}_{s,t}(\hat{\sigma}_{s,t} \circ f\ (s) \cdot \hat{\sigma}_{s,t} \circ f\ (t)) && \text{(by Condition C1)}\\
&= \hat{\sigma}^{-1}_{s,t} \circ \hat{\sigma}_{s,t}\ (f(s) \cdot f(t)) && \text{(because } \mathcal{M} \text{ is a data monoid)}\\
&= f(s) \cdot f(t) && \text{(because } \hat{\sigma}^{-1}_{s,t} \circ \hat{\sigma}_{s,t} = \text{id.)}
\end{aligned}$$

As for the last condition, let $d \in \text{mem}(s) \cup \text{mem}(t)$ be given and assume that $s = o(d_1, \dots, d_n)$ and that $t = p(e_1, \dots, e_m)$. First consider the case where $d_i \in \text{mem}(s)$. Note that by definition, $\sigma_{s,t}(d_i) = i$. Hence $\tau \circ \sigma_{s,t}\ (d_i) = \tau(i) = \sigma_{\tau(s),\tau(t)}(d_i)$. If $e_i \in \text{mem}(t)$ is given, then $\sigma_{s,t}(e_i) = |\{e_1, \dots, e_i\} \smallsetminus \{d_1, \dots, d_n\}|$ the the result follows in the same way. The proof that $\sigma_{\tau(s),\tau(t)} \circ \tau\ (d) = \sigma_{s,t}(d)$ is similar. □

Turning to the main proof of the proposition, we show that $\mathcal{S}$ is a presentation system by checking the conditions from Definition 4.

1. *Congruence for $\odot$:* We need to show that $s \approx s'$ and $t \approx t'$ implies that $s \odot t \approx s' \odot t'$. Assume that $s \approx s'$ and $t \approx t'$. Then, using Condition C2, we get

$$f(s \odot t) \ = \ f(s) \cdot f(t) \ = \ f(s') \cdot f(t') \ = \ f(s' \odot t')$$

2. *Equivariance:* We need to show that $\check{\tau}(s) \odot \check{\tau}(t) = \check{\tau}(s \odot t)$ for all $s, t \in T$ and $\tau \in G_C$.

$$\begin{aligned}
\check{\tau}(s) \odot \check{\tau}(t) &= \check{\sigma}^{-1}_{\check{\tau}(s),\check{\tau}(t)} \circ g\ (f \circ \hat{\sigma}_{\check{\tau}(s),\check{\tau}(t)}(\check{\tau}(s)) \cdot f \circ \hat{\sigma}_{\check{\tau}(s),\check{\tau}(t)}\ (\check{\tau}(t)))\\
&\overset{C3}{=} \check{\sigma}^{-1}_{\check{\tau}(s),\check{\tau}(t)} \circ g\ (f \circ \hat{\sigma}_{s,t}(s) \cdot f \circ \hat{\sigma}_{s,t}\ (t))\\
&\overset{C3}{=} \check{\tau} \circ \check{\sigma}^{-1}_{s,t} \circ g\ (f \circ \hat{\sigma}_{s,t}(s) \cdot f \circ \hat{\sigma}_{s,t}\ (t))\\
&= \check{\tau}(s \odot t).
\end{aligned}$$

3. *Congruence for renamings:* We need to show that if $s \approx t$ then $\check{\sigma}(s) \approx \check{\sigma}(t)$ for all $s, t \in T$ and $\sigma \in G_C$. Let $s, t \in T$ with $s \approx t$ be given and let $\sigma \in G_C$. By definition of $\approx$ this implies that $f(s) = f(t)$. As $\hat{\sigma}$ is a function it follows that $\hat{\sigma}(f(s)) = \hat{\sigma}(f(t))$. By Condition C1 it follows that $f(\check{\sigma}(s)) = f(\check{\sigma}(t))$. Hence $\check{\sigma}(s) \approx \check{\sigma}(t)$.

4. *Associativity up to* $\approx$*:* Let $s, t, u \in T$ be given. We show that $(s \odot t) \odot u = s \odot (t \odot u)$ using Condition C2 and the associativity of the product $\cdot$ of $\mathcal{M}$:
$$\begin{aligned} f((s \odot t) \odot u) &= f(s \odot t) \cdot f(u) \\ &= (f(s) \cdot f(t)) \cdot f(u) \\ &= f(s) \cdot (f(t) \cdot f(u)) \\ &= f(s) \cdot f(t \odot u) \\ &= f(s \odot (t \odot u)). \end{aligned}$$
5. *Identity:* Recall that we defined $1_\mathcal{S}$ to be $g(1_\mathcal{M})$. As $1_\mathcal{M}$ has empty memory, it follows that $g(1_\mathcal{M}) = f^{-1}(1_\mathcal{M})$. Let t be a term. Then we have
$$\begin{aligned} 1_\mathcal{S} \odot t &= \check{\sigma}^{-1}_{1_\mathcal{S},t} \circ g\ (f \circ \hat{\sigma}_{1_\mathcal{S},t}\ (1_\mathcal{S})\ \cdot\ f \circ \hat{\sigma}_{1_\mathcal{S},t}\ (t)) \\ &= \check{\sigma}^{-1}_{1_\mathcal{S},t} \circ g\ (f \circ \hat{\sigma}_{1_\mathcal{S},t}\ (f^{-1}(1_\mathcal{M}))\ \cdot\ f \circ \hat{\sigma}_{1_\mathcal{S},t}\ (t)) \\ &= \check{\sigma}^{-1}_{1_\mathcal{S},t} \circ g\ (1_\mathcal{M}\ \cdot\ f \circ \hat{\sigma}_{1_\mathcal{S},t}\ (t)) \\ &= \check{\sigma}^{-1}_{1_\mathcal{S},t} \circ g\ (f \circ \hat{\sigma}_{1_\mathcal{S},t}\ (t)) \\ &= t. \end{aligned}$$

We can conclude that $\mathcal{S}$ is a term-based presentation system.

The term-based system $\mathcal{S}$ *represents* $\mathcal{M}$*.* It remain to show that $\mathcal{S} = (T, \odot, \check{\ })$ represents the data monoid $\mathcal{M} = (M, \cdot, \hat{\ })$. By definition, $\mathcal{S}$ represents $\widetilde{\mathcal{M}} = (\widetilde{M}, \tilde{\cdot}, \tilde{\hat{\ }})$ where $\widetilde{M}$ is the set of equivalence classes of $\approx$, and the product $\tilde{\cdot}$, the action $\tilde{\hat{\ }}$, and the identity $1_{\widetilde{\mathcal{M}}}$ are defined by
$$\begin{aligned} [s]\ \tilde{\cdot}\ [t] &= [s \odot t] \\ \tilde{\hat{\tau}}([s]) &= [\check{\tau}(s)] \\ 1_{\widetilde{\mathcal{M}}} &= [1_\mathcal{S}] \end{aligned}$$

We know form the first claim of the proposition that $\widetilde{\mathcal{M}}$ is a data monoid. We need to show that $\widetilde{\mathcal{M}}$ and $\mathcal{M}$ are isomorphic. Let $h : \widetilde{\mathcal{M}} \to \mathcal{M}$ be the function defined by
$$h([s]) \quad = \quad f(s).$$

We show that h is a data monoid isomorphism, that is a bijective data monoid homomorphism. We first check that h is a homomorphism. There are three properties to check:

1. We need to show that $h([s]\ \tilde{\cdot}\ [t]) = h([s]) \cdot h([t])$. Using Conditoin C2 we can calculate
$$h([s]\ \tilde{\cdot}\ [t]) \quad = \quad h([s \odot t]) \quad = \quad f(s \odot t) \quad = \quad f(s) \cdot f(t) \quad = \quad h([s]) \cdot h([t])$$
2. Next we show that $h(1_{\widetilde{\mathcal{M}}}) = 1_\mathcal{M}$:
$$h(1_{\widetilde{\mathcal{M}}}) \quad = \quad h([1_\mathcal{S}]) \quad = \quad h(g(1_\mathcal{M})) \quad = \quad 1_\mathcal{M}.$$
3. Finally, we need to show that for all terms $s \in T$ and all renamings $\tau \in G_C$, $h(\tilde{\hat{\tau}}([s])) = \hat{\tau}(h([s]))$. This holds because of the following:
$$h(\tilde{\hat{\tau}}([s])) \quad = \quad h([\check{\tau}(s)]) \quad = \quad f(\check{\tau}(s)) \quad = \quad \hat{\tau}(f(s)) \quad = \quad \hat{\tau}(h([s])).$$

It is immediate from the definition that h is injective. It remains to show that h is surjective. Let some $m \in M$ be given. We need to show that there is some $\tilde{m} \in \tilde{M}$ such that $h(\tilde{m}) = m$. We will show that there is a term $t \in T$ such that $f(t) = m$. This is sufficient as it implies that $h([t]) = f(t) = m$.

Let o be the orbit of m and assume that it has arity k. Recall that we distinguished a representative m_o in each orbit o. As m and m_o are in the same orbit there must be a data renaming τ such that $\hat{\tau}(m_o) = m$. Also recall that we showed in (C1) above that $f \circ \check{\tau} = \hat{\tau} \circ f$. By multiplying with $\check{\tau}^{-1}$ on the right we get $f = \hat{\tau} \circ f \circ \check{\tau}^{-1}$. Let $t = \check{\tau}(o(1, \dots, k))$. Then

$$\begin{aligned} f(\check{\tau}(o(1,\dots,k))) &= \hat{\tau} \circ f \circ \check{\tau}^{-1}(\check{\tau}(o(1,\dots,k))) \\ &= \hat{\tau} \circ f(o(1,\dots,k)) \\ &= \hat{\tau}(\hat{\sigma}_{1,\dots,k}(m_o)) && \text{(by the definition of } f\text{)} \\ &= \hat{\tau}(m_o) && \text{(because } \sigma_{1,\dots,k} = \mathrm{id}\text{)} \\ &= m. \end{aligned}$$

We just proved that $f(t) = m$ and hence h is surjective. This completes the proof of the proposition. □

B Proofs for Section 4 (From the logic to data monoids)

This section is devoted to prove Theorem 1, that is, that the data language defined by a rigidly guarded MSO formula can be effectively recognized by an orbit finite data monoid via a projectable morphism. As mentioned in the main part of the paper, the proof is an induction on the structure of the formula.

For the construction to be effective, we need to compute the result of algebraic operations, such as product or projection, on orbit finite data monoids. It turns out that all the operations we need for the proof are compatible with the operation on the respective restrictions. As an example, for every pair of orbit finite data monoids $\mathcal{M}$, $\mathcal{M}'$ and for every $C \subseteq D$, we have that the product $\mathcal{M} \times \mathcal{M}'$ is an orbit finite data monoid and, moreover, $(\mathcal{M} \times \mathcal{M}')|_C = \mathcal{M}|_C \times \mathcal{M}'|_C$. It follows that one can easily compute a finite presentation of the result of an algebraic construction starting from some given representations of the input orbit finite data monoids. In view of the above arguments, in the rest of the proof, we shall often skip the details about how the representations of the various orbit finite data monoids are computed and we shall focus instead on the pure algebraic constructions. Hence, by a slight abuse of terminology, we will say that an orbit finite data monoid $\mathcal{M}'$ is *computed* from another orbit finite data monoid $\mathcal{M}$ when a presentation of $\mathcal{M}'$ can be obtained effectively from a given presentation of $\mathcal{M}$. In a similar way, since morphisms from free data monoids to orbit finite data monoids are uniquely determined by the images of the singleton data words, we say that a morphism $h' : (D' \times A')^* \to \mathcal{M}'$ can be computed from another morphism $h : (D \times A)^* \to \mathcal{M}$ if for all singleton words $u' \in (D' \times A')^*$, the image $h'(u')$ can be obtained effectively from the images $h(u)$, for $u \in (D \times A)^*$.

We begin by describing the translation of the existential closures of rigidly guarded MSO formulas:

Lemma 1. *Let $\varphi(\bar{X}, X_{m+1})$ be a formula and let $\varphi'(\bar{X}) = \exists\, X_{m+1}.\, \varphi(\bar{X}, X_{m+1})$. If $[\![\varphi]\!]$ is recognized by an orbit finite data monoid $\mathcal{M}$ via homomorphism h, then we can compute an orbit finite data monoid $\mathcal{M}'$ and a morphism h' such that $\mathcal{M}'$ recognizes $[\![\varphi']\!]$ via h'.*

Proof. For the sake of brevity, we denote the language $[\![\varphi]\!]$ over $D \times A \times B^{m+1}$ defined by the formula $\varphi(\bar{X}, X_{m+1})$ by L, and the language $[\![\varphi']\!]$ over $D \times A \times B^m$ defined by $\varphi'(\bar{X}) = \exists\, X_{m+1}.\, \varphi(\bar{X}, X_{m+1})$ by L' . We assume that L is recognized by an orbit finite data monoid $\mathcal{M}$ via a morphism h. We will exploit a variant of the powerset construction applied to the orbit finite data monoid $\mathcal{M}$ in order to obtain an orbit finite data monoid $\mathcal{M}'$ that recognizes L'. The same construction can be applied to a given restriction $\mathcal{M}|_C$ that represents $\mathcal{M}$ (see Proposition 1) in order to compute a restriction $\mathcal{M}'|_C$ that represents $\mathcal{M}'$. However, note that the cardinality of the set C must be at least twice the maximal size of the memories of the elements in $\mathcal{M}'$.

The powerset construction. Let $\mathcal{M} = (M, \cdot, \hat{\ })$. We define $\mathcal{M}' = (M', \odot, \check{\ })$ as follows:

- the elements in M' are the subsets of M that contain only pairwise orbit distinct elements, namely, those sets $S \subseteq M$ such that for all $s, s' \in S$, $s \doteq s'$ implies $s = s'$;
- the product $\odot$ in M' is defined on pairs of sets $S, T \in M'$ by

  $$S \odot T \;=\; \begin{cases} S \cdot T & \text{if for all } s, s' \in S \text{ and all } t, t' \in T,\ s \cdot t \doteq s' \cdot t' \text{ implies } s \cdot t = s' \cdot t' \\ \varnothing & \text{otherwise} \end{cases}$$

  where $S \cdot T$ denotes the set $\{s \cdot t \;:\; s \in S, t \in T\}$;
- the function $\check{}$ maps any renaming τ to the automorphism $\check{\tau}$ such that

  $$\check{\tau}(S) = \{\hat{\tau}(s) \;:\; s \in S\}$$

  for all $S \in M'$.

It is routine to check that the product $\odot$ is associative, the function $\check{}$ is a group action, the empty set $\varnothing$ is a null element in $\mathcal{M}'$, and the singleton $\{1_{\mathcal{M}}\}$ is the identity in $\mathcal{M}'$. Moreover, if n is the number of orbits of $\mathcal{M}$, then every set $S \in M'$ has cardinality at most n (indeed, if this were not the case, then S would contain two distinct elements s and s' such that $s \doteq t$, which is against the definition of M'). From this property it follows that every set $S \in M'$ has finite memory, precisely, $\mathrm{mem}(S)$ is contained in the union of the memories $\mathrm{mem}(s)$ of the finitely many elements $s \in S$. In fact, since M has at most n orbits and the memories of two elements in the same orbit have the same size, we obtain the following upper bound on the size of the memories of the elements of $\mathcal{M}'$: $|\mathrm{mem}(\mathcal{M}')| \leq n|\mathrm{mem}(\mathcal{M})|$.

Another consequence of the above construction is that $\mathcal{M}'$ has finitely many orbits. Suppose, by way of contradiction, that there exist infinitely many sets $S_1, S_2, \ldots \in M'$ that are pairwise orbit distinct. First of all, we can turn each set S_i into an ordered sequence $\bar{s}_i = (s_{i,1}, \ldots, s_{i,k_i})$ (the choice of the order in which we list the elements is arbitrary and it is not relevant here). We can then denote by M_i the union of the memories of the elements in each tuple $\bar{s}_i$, namely, we let $M_i = \mathrm{mem}(s_{i,1}) \cup \ldots \cup \mathrm{mem}(s_{i,k})$. Without loss of generality, we can assume that (i) all tuples $\bar{s}_1, \bar{s}_2, \ldots$ have the same cardinality k, (ii) all sets $M_1, M_2, \ldots$ have the same cardinality l, and (iii) for every index $1 \leq j \leq k$, the elements $s_{1,j}, s_{2,j}, \ldots$ are in the same orbit (note that we can always enforce these assumptions by restricting to infinite subsequences $\bar{s}_{i_1}, \bar{s}_{i_2}, \ldots$). Now, for every index $i > 1$, we fix a renaming π_i such that $\pi_i(M_i) = M_1$ (recall that $|M_i| = |M_1| = l$). Similarly, for every pair of indices $i > 1$ and $1 \leq j \leq k$, we fix a renaming $\tau_{i,j}$ such that $\hat{\tau}_{i,j}(\hat{\pi}_i(s_{i,j})) = s_{1,j}$ (recall that $s_{1,j}$ and $s_{i,j}$ are in the same orbit). We then consider the functions $\tau_{i,j}$ restricted to the set M_1. From the Pigeonhole Principle, we have that there exist two indices $i \neq i'$ such that, for every $1 \leq j \leq k$,

$$\tau_{i,j}|M_1 \;=\; \tau_{i',j}|M_1.$$

In particular, this implies that for every $1 \leq j \leq k$, the function $\tau_{i',j}^{-1} \circ \tau_{i,j}$ is the identity on the set M_1. Moreover, by construction, we have

$$\big(\hat{\pi}_{i'}^{-1} \circ \hat{\tau}_{i',j}^{-1} \circ \hat{\tau}_{i,j} \circ \hat{\pi}_i\big)(s_{i,j}) \;=\; \big(\hat{\pi}_{i'}^{-1} \circ \hat{\tau}_{i',j}^{-1}\big)(s_{1,j}) \;=\; s_{i',j}.$$

Since $\hat{\pi}_i(s_{i,j}) = s_{1,j}$ and $\hat{\tau}^{-1}_{i',j} \circ \hat{\tau}_{i,j}$ is the identity on the set $M_1 \supseteq \mathrm{mem}(s_{1,j})$, we have that $\big(\hat{\tau}^{-1}_{i',j} \circ \hat{\tau}_{i,j}\big)(s_{1,j}) = s_{1,j}$ and hence $\big(\hat{\pi}^{-1}_{i'} \circ \hat{\pi}_i\big)(s_{i,j}) = s_{i',j}$ for all $1 \le j \le k$. In conclusion, we have just shown that there exist two indices $i \ne i'$ and a renaming $\pi_{i,i'}$ $(= \pi^{-1}_{i'} \circ \pi_i)$ such that

$$\check{\pi}_{i,i'}(S_i) \;=\; \big\{\hat{\pi}_{i,i'}(s_{i,j}) \;:\; 1 \le j \le k\big\} \;=\; \big\{s_{i',j} \;:\; 1 \le j \le k\big\} \;=\; S_{i'}.$$

This is against the hypothesis of the sets S_i and $S_{i'}$ having different orbits. Thus $\mathcal{M}'$ is a data monoid with finitely many orbits.

The morphism. Below, we define a morphism h' from the the free data monoid $(D \times A \times B^m)^*$ to the data monoid $\mathcal{M}'$. Precisely, for every expanded data word $\langle u, U_1, \dots, U_m\rangle$, we let

$$h'(\langle u, U_1, \dots, U_m\rangle) \;=\; \big\{h(\langle u, U_1, \dots, U_m, U_{m+1}\rangle) \;:\; U_{m+1} \subseteq \mathcal{D}om(u)\big\}$$

(note that, since h is projectable, then $h'(\langle u, U_1, \dots, U_m\rangle)$ contains only pairwise orbit distinct elements and hence it is an element of the data monoid $\mathcal{M}'$).

We verify that the morphism h' is projectable. Let us consider a data word u and some tuples of predicates $\bar{U} = U_1, \dots, U_m$ and $\bar{V} = V_1, \dots, V_m$ and suppose that $h'(\langle u, \bar{U}\rangle) \doteq h'(\langle u, \bar{V}\rangle)$. This means that there is a renaming τ such that

$$h'(\langle u, \bar{V}\rangle) \;=\; \check{\tau}\big(h'(\langle u, \bar{U}\rangle)\big).$$

Moreover, by definition of h', we have

$$\big\{h(\langle u, \bar{V}, V_{m+1}\rangle) \;:\; V_{m+1} \subseteq \mathcal{D}om(u)\big\} \;=\; \big\{\hat{\tau}(h(\langle u, \bar{U}, U_{m+1}\rangle)) \;:\; U_{m+1} \subseteq \mathcal{D}om(u)\big\}.$$

Since h is projectable, we have that the two sets $h'(\langle u, \bar{U}\rangle)$ and $h'(\langle u, \bar{V}\rangle)$ coincide, which proves that h' is projectable as well.

Recognizability. We now prove that the language L' defined by $\varphi' = \exists\, X_{m+1}.\, \varphi$ is recognized by the data monoid $\mathcal{M}'$ and the morphism h'. For the sake of brevity, we let $F = h(L)$ and $F' = h'(L')$. We then consider an expanded data word $\langle u, \bar{U}\rangle \in (D \times A \times B^m)^*$ and we prove that $\langle u, \bar{U}\rangle \in L'$ iff $h'(\langle u, \bar{U}\rangle) \in F'$. The left-to-right implication is trivial, so we prove the converse implication. Suppose that $h'(\langle u, \bar{U}\rangle) \in F'$. Since $F' = h'(L')$, we know that there is an expanded data word $\langle v, \bar{V}\rangle \in L'$ such that $h'(\langle u, \bar{U}\rangle) = h'(\langle v, \bar{V}\rangle)$. We know from the definition of h' that

$$\big\{h(\langle u, \bar{U}, U_{m+1}\rangle) \;:\; U_{m+1} \subseteq \mathcal{D}om(u)\big\} \;=\; \big\{h(\langle v, \bar{V}, V_{m+1}\rangle) \;:\; V_{m+1} \subseteq \mathcal{D}om(v)\big\}$$

and from the definition of L' that $(\langle v, \bar{V}, V_{m+1}\rangle) \in L$ for some unary predicate $V_{m+1} \subseteq \mathcal{D}om(v)$. Moreover, since L is recognized by $\mathcal{M}$ via the morphism h and $\langle v, \bar{V}, V_{m+1}\rangle$ belongs to L, we have $h(\langle v, \bar{V}, V_{m+1}\rangle) \in F$ and hence $h(\langle u, \bar{U}, U_{m+1}\rangle) \in F$ for some unary predicate $U_{m+1} \subseteq \mathcal{D}om(u)$. This implies that $\langle u, \bar{U}, U_{m+1}\rangle \in L$ and hence $\langle u, \bar{U}\rangle \in L'$. □

We now turn to the translation of rigidly guarded data tests.

Lemma 2. *Given a rigid formula $\varphi(x,y)$, an orbit finite data monoid $\mathcal{M}$ and a projectable morphism h that recognizes $[\![\varphi]\!]$, one can compute an orbit finite data monoid $\mathcal{M}'$ and a projectable morphism h' that recognizes $[\![\varphi(x,y) \;\wedge\; x \sim y]\!]$.*

Proof. Let $\mathcal{M} = (M, \cdot, \hat{\ })$ be an orbit finite data monoid and let $h : (D \times A \times B^2)^* \to \mathcal{M}$ be a projectable morphism that recognizes $L = [\![\varphi(x,y)]\!]$. We first show that we can assume that h is surjective.

> ***Claim 1.*** *Given an orbit finite data monoid $\mathcal{M}$ and a morphism $h : (D \times A)^* \to \mathcal{M}$, one can compute the data sub-monoid $h((D \times A)^*)$.*

> *Proof of Claim 1.* Suppose that the orbit finite data monoid $\mathcal{M} = (M, \cdot, \hat{\ })$ is represented by its restriction $\mathcal{M}|_C$, for some finite subset C of D such that $|C| \geq 2|\mathrm{mem}(\mathcal{M})|$. Let $\mathcal{M}' = h\big((D \times A)^*\big)$ be the data sub-monoid induced by h. Clearly, we have $|\mathrm{mem}(\mathcal{M})| = |\mathrm{mem}(\mathcal{M}')|$ and hence, by Proposition 1, the data sub-monoid $\mathcal{M}'$ is uniquely determined by its restriction $\mathcal{M}'|_C$. Moreover, the domain of $\mathcal{M}'|_C$ is the finite set $h\big((A \times C)^*\big)$, which is computable from $\mathcal{M}|_C$ and $h|(A \times C)$. Finally, the product and the group action of the data sub-monoid $\mathcal{M}'|_C$ are the restrictions of the product and the group action of $\mathcal{M}$ to the finite set $h\big((A \times C)^*\big)$. This shows that $\mathcal{M}'|_C$ can be computed from $\mathcal{M}|_C$ and $h|(A \times C)$. This completes the proof of Claim 1. □

Unfortunately, the property of projectability is not straightforwardly preserved when we turn a morphism for a rigid guard $\varphi(x,y)$ to a morphism for the rigidly guarded comparison $\varphi \wedge x \sim y$. In this case, we derive from the rigidity assumption on φ a stronger notion of projectability, which is defined below and which is called 0-reduced projectability.

Hereafter, we call an element $0_{\mathcal{N}}$ of a data monoid $\mathcal{N}$ a *null element* if $0_{\mathcal{N}} \cdot m = m \cdot 0_{\mathcal{N}} = 0_{\mathcal{N}}$. It is easy to see that if a data monoid has a null element then this element is unique. Note that if a language L is accepted by a data monoid, then L is accepted by a data monoid with a null. Hence in the following we will assume that $\mathcal{M}$ has a null element.

Definition 5. *Let h be a morphism from $(D \times A \times B^2)^*$ to a data monoid $\mathcal{M} = (M, \cdot, \hat{\ })$. We say that h is* 0-reduced *if for all data words $u \in (D \times A)^*$ and all positions $x, x', y, y' \in \mathcal{D}om(u)$, the following implications hold:*

- *If $h(\langle u, \{x\}, \varnothing\rangle) = h(\langle u, \{x'\}, \varnothing\rangle)$ then $x = x'$ or $h(\langle u, \{x\}, \varnothing\rangle) = h(\langle u, \{x'\}, \varnothing\rangle) = 0_{\mathcal{M}}$.*
- *If $h(\langle u, \varnothing, \{y\}\rangle) = h(\langle u, \varnothing, \{y'\}\rangle)$ then $y = y'$ or $h(\langle u, \varnothing, \{y\}\rangle) = h(\langle u, \varnothing, \{y'\}\rangle) = 0_{\mathcal{M}}$.*

We show that the languages defined by rigid formulas are recognized by morphisms that are both 0-reduced and projectable (we shortly call them 0-reduced projectable morphisms).

> ***Claim 2.*** *Let $\varphi(x,y)$ be a rigid formula. Given an orbit finite data monoid $\mathcal{M}$ and a projectable morphism that recognizes $[\![\varphi(x,y)]\!]$, one can compute an orbit finite data monoid $\mathcal{M}'$ and a 0-reduced projectable morphism that recognizes $[\![\varphi(x,y)]\!]$.*

Proof of Claim 2. Let $\mathcal{M} = (M, \cdot, \hat{\ })$ be an orbit finite data monoid and $h : (D \times A \times B^2)^* \to \mathcal{M}$ a projectable morphism that recognizes $L = [\![\varphi(x,y)]\!]$. By Claim 1, we can assume, that h is a surjective mapping. Below, we construct a new orbit finite data monoid $\mathcal{M}'$, as a quotient of $\mathcal{M}$, and a corresponding morphism h' that recognizes the same language $L = [\![\varphi(x,y)]\!]$. As usual, the same construction can be applied effectively to a given restriction $\mathcal{M}|_C$ that represents $\mathcal{M}$, thus obtaining a corresponding representation $\mathcal{M}'|_C$ of $\mathcal{M}'$.

Collapsing bad elements. Let $F = h(L)$ and let G be the maximal set of all elements such that $M \cdot G \cdot M \cap F = \emptyset$. Intuitively, G contains those elements of $\mathcal{M}$ that cannot be extended to some elements in F by concatenating elements to the left, to the right, or both. Note that G is an ideal of $\mathcal{M}$, namely, $M \cdot G \cdot M \subseteq G$, and, furthermore, it is closed under the action of renamings, namely, $\hat{\tau}(G) \subseteq G$ for all renamings τ. We now introduce the equivalence $\approx_G$ that groups any two elements $s, t \in M$ whenever we have either $s = t$ or $s \in G$ and $t \in G$. Note that $\approx_G$ is a congruence with respect to the product of $\mathcal{M}$, namely, if $s \approx_G s'$ and $t \approx_G t'$, then $s \cdot t \approx_G s' \cdot t'$. The equivalence $\approx_G$ is also compatible with the action of renamings, namely, if $s \approx_G t$, then $\hat{\tau}(s) \approx_G \hat{\tau}(t)$ for all renamings τ. This allows us to define a data monoid $\mathcal{M}'$ as the quotient of $\mathcal{M}$ with respect to $\approx_G$, namely, as the triple $\mathcal{M}' = (M/_{\approx_G}, \odot, \check{\ })$, where

- $[s]_{\approx_G} \odot [t]_{\approx_G} = [s \cdot t]_{\approx_G}$,
- $\check{\tau}([s]_{\approx_G}) = [\hat{\tau}(s)]_{\approx_G}$

Note that both functions $\odot$ and $\check{\ }$ are well defined.
Clearly $\mathcal{M}'$ is an orbit finite data monoid. Moreover, we observe that for all $r \in M \setminus G$, the $\approx_G$-equivalence class of r is the singleton $\{r\}$. The only other element of $M/_{\approx_G}$ is the entire set G. In addition G is a null element of $\mathcal{M}'$. Hence we will also denote it by $0_{\mathcal{M}'}$.

The morphism. We now define the morphism $h' : (D \times A \times B^2)^* \to \mathcal{M}'$ that recognizes L. This is nothing but the functional composition $h_G \circ h$ of the morphism h from $(D \times A \times B^2)^*$ to $\mathcal{M}$ and the morphism h_G from $\mathcal{M}$ to $\mathcal{M}'$ defined by

$$h_G(s) = [s]_{\approx_G}.$$

Note that both h and h_G are surjective morphisms and hence h' is surjective as well. Moreover, since $h_G^{-1} \circ h_G$ is the identity on $F = h(L)$, we have

$$L = h^{-1}(h(L)) = h^{-1}(F) = h^{-1}(h_G^{-1}(h_G(F))) = (h')^{-1}(h'(L)).$$

This shows that h' recognizes the language L.

Projectability. Below, we verify that the morphism h' is projectable. Consider a data word $u \in (D \times A)^*$ and some predicates $U_1, U_2, V_1, V_2 \subseteq \mathcal{D}om(u)$ and suppose that $h'(\langle u, U_1, U_2 \rangle)$ and $h'(\langle u, V_1, V_2 \rangle)$ are in the same orbit, namely, that there is a data renaming τ such that $h'(\langle u, V_1, V_2\rangle) = \check{\tau}(h'(\langle u, U_1, U_2\rangle))$. We distinguish two cases depending on whether one element among $h'(\langle u, U_1, U_2\rangle)$ and $h'(\langle u, V_1, V_2\rangle)$ coincides with $0_{\mathcal{M}'}$ or not. If $h'(\langle u, U_1, U_2\rangle) = 0_{\mathcal{M}'}$, then we recall that $0_{\mathcal{M}'}$ has empty memory and we obtain

$$h'(\langle u, V_1, V_2\rangle) = \check{\tau}(h'(\langle u, U_1, U_2\rangle)) = \check{\tau}(0_{\mathcal{M}'}) = 0_{\mathcal{M}'} = h'(\langle u, U_1, U_2\rangle).$$

A similar conclusion can be obtained from $h'(\langle u, V_1, V_2\rangle) = 0_{\mathcal{M}'}$. In the remaining case, we assume that neither $h'(\langle u, U_1, U_2\rangle)$ nor $h'(\langle u, V_1, V_2\rangle)$ are the null element. By construction, we know that neither $h(\langle u, U_1, U_2\rangle)$ nor $h(\langle u, V_1, V_2\rangle)$ belong to the ideal G and hence $h'(\langle u, U_1, U_2\rangle) = \{h(\langle u, U_1, U_2\rangle)\}$ and also $h'(\langle u, V_1, V_2\rangle) = \{h(\langle u, V_1, V_2\rangle)\}$. Moreover, we have

$$h'(\langle u, V_1, V_2\rangle) \; = \; \check{\tau}(h'(\langle u, U_1, U_2\rangle)) \; = \; \{\hat{\tau}(h(\langle u, U_1, U_2\rangle))\}$$

and hence $h(\langle u, V_1, V_2\rangle) = \hat{\tau}(\langle u, U_1, U_2\rangle)$. Finally, since h is projectable, we obtain $h(\langle u, U_1, U_2\rangle) = h(\langle u, V_1, V_2\rangle)$ and therefore we get $h'(\langle u, U_1, U_2\rangle) = h'(\langle u, V_1, V_2\rangle)$. This shows that h' is projectable as well.

0-*Reduced projectability.* We now exploit the fact that the language L is defined by a *rigid* formula $\varphi(x, y)$ to prove that the morphism h' is also 0-reduced. Let $u \in (D \times A)^*$ be a data word and let $x, x' \in \mathcal{D}om(u)$ be two positions in it. By way of contradiction, we assume that $x \neq x'$ and $h'(\langle u, \{x\}, \varnothing\rangle) = h'(\langle u, \{x'\}, \varnothing\rangle) \neq 0_{\mathcal{M}'}$ and we prove that $\varphi(x, y)$ is not rigid (the same conclusion can be obtained from the assumption that there exist two positions $y, y \in \mathcal{D}om(u)$ such that $y \neq y'$ and $h'(\langle u, \varnothing, \{y\}\rangle) = h'(\langle u, \varnothing, \{y'\}\rangle) \neq 0_{\mathcal{M}'}$, thus proving that the morphism h' is 0-reduced). Since $h'(\langle u, \{x\}, \varnothing\rangle) = h'(\langle u, \{x'\}, \varnothing\rangle) \neq 0_{\mathcal{M}'}$, we know that $h(\langle u, \{x\}, \varnothing\rangle) = h(\langle u, \{x'\}, \varnothing\rangle) \notin G$ and hence there exist $s, t \in M$ such that $s \cdot h(\langle u, \{x\}, \varnothing\rangle) \cdot t = s \cdot h(\langle u, \{x'\}, \varnothing\rangle) \cdot t \in F$. Moreover, since h is surjective, we know that there exist two expanded data words $\langle v, U_1, U_2\rangle$ and $\langle w, V_1, V_2\rangle$ such that $h(\langle u, U_1, U_2\rangle) = s$ and $h(\langle v, V_1, V_2\rangle) = t$. Since $\mathcal{M}$ and h recognize the language L defined by the formula $\varphi(x, y)$ and $F = h(L)$, we have that $\varphi(x, y)$ is satisfied by both the sequence $\langle v, U_1, U_2\rangle\langle u, \{x\}, \varnothing\rangle\langle w, V_1, V_2\rangle$ and the sequence $\langle v, U_1, U_2\rangle\langle u, \{x'\}, \varnothing\rangle\langle w, V_1, V_2\rangle$. Finally, since $x \neq x'$, we must conclude that $\varphi(x, y)$ is not rigid. This completes the proof of Claim 2. □

We can now start with the main part of the proof of Lemma 2. Recall that $\mathcal{M} = (M, \cdot, \hat{\ })$ is the data monoid that recognizes $L = [\![\varphi(x, y)]\!]$ via the projectable morphism $h : (D \times A \times B^2)^* \to \mathcal{M}$. We denote by $\mathcal{N} = (N, \odot, \check{\ })$ the syntactic data monoid of the language defined by $x \sim y$ and by $g : (D \times A \times B^2)^* \to \mathcal{N}$ the corresponding morphism that recognizes $[\![x \sim y]\!]$. The data monoid $\mathcal{N}$ has finitely many orbits and its elements can be assumed to be terms of one the following forms:

1. $o(\varepsilon)$, which plays the role of the identity $1_{\mathcal{N}}$ in $\mathcal{N}$ and which corresponds to the image of the empty word under g;
2. $o(d)$, for any $d \in D$, which corresponds to the image of the data words expanded by a singleton predicate $U = \{x\}$ and the empty predicate $V = \varnothing$ under g;
3. $p(d)$, for any $d \in D$, which corresponds to the image of the data words expanded by the empty predicate $U = \varnothing$ and a singleton predicate $V = \{y\}$ under g;
4. $r(\varepsilon)$, with corresponds to the image of the expanded data words that satisfy $x \sim y$ under g;
5. $s(\varepsilon)$, which plays the role of the null element $0_{\mathcal{N}}$ in $\mathcal{N}$ and which corresponds to the image under g of the data words expanded with two non-empty predicates U, V that do not satisfy $x \sim y$

For example, we have $o(d) \odot p(d) = r(\varepsilon)$ and $o(d) \odot p(e) = s(\varepsilon)$, for all pairs of distinct values $d, e \in D$. Note that the morphism g is not projectable.

The 0*-collapse product.* We define the data monoid $\mathcal{M}'$ for the formula $\varphi(x,y) \wedge x \sim y$ using a suitable variant of the algebraic product of $\mathcal{M}$ and $\mathcal{N}$, which we call 0*-collapse product* (strictly speaking, the 0-collapse product is a special form of semi-direct product). Formally, we let $\mathcal{M}'$ be the triple $(M', \circledcirc, \tilde{\ })$, where

- M' consists of all pairs $(m,n) \in M \times N$ such that $m = 0_{\mathcal{M}}$ implies $n = 0_{\mathcal{N}}$;
- for every $(m,n), (m',n') \in M'$, the product $(m,n) \circledcirc (m',n')$ is either the pair $(m \cdot m', n \odot n')$ or the pair $(0_{\mathcal{M}}, 0_{\mathcal{N}})$, depending on whether $m \cdot m' \neq 0_{\mathcal{M}}$ or not;
- $\tilde{\tau}(m,n) = (\hat{\tau}(m), \check{\tau}(n))$ for all $(m,n) \in M'$ and all $\tau \in \mathit{\Gamma}_D$.

Clearly, the thus defined data monoid $\mathcal{M}'$ has finitely many orbits.

The Morphism. Accordingly, we denote by h' the morphism that maps any expanded word $\bar{u} \in (D \times A \times B^2)^*$ to either the pair $\big(h(\bar{u}), g(\bar{u})\big)$ or the pair $(0_{\mathcal{M}}, 0_{\mathcal{N}})$, depending on whether $h(\bar{u}) \neq 0_{\mathcal{M}}$ or not. Clearly, h' recognizes the language $[\![\varphi(x,y) \wedge x \sim y]\!]$.

Projectability. Below, we prove that h' is a projectable morphism. Consider a data word $u \in (D \times A)^*$ and some predicates $U_1, U_2, V_1, V_2 \subseteq \mathcal{D}om(u)$ and suppose that the elements $h'(\langle u, U_1, U_2\rangle)$ and $h'(\langle u, V_1, V_2\rangle)$ are in the same orbit. We distinguish between the case where $h(\langle u, U_1, U_2\rangle) = 0_{\mathcal{M}}$ (and hence $h(\langle u, V_1, V_2\rangle) = 0_{\mathcal{M}}$ as well) and the case where $h(\langle u, U_1, U_2\rangle) \neq 0_{\mathcal{M}}$ (and hence $h(\langle u, V_1, V_2\rangle) \neq 0_{\mathcal{M}}$ as well). In the former case, we immediately have $h'(\langle u, U_1, U_2\rangle) = (0_{\mathcal{M}}, 0_{\mathcal{N}}) = h'(\langle u, V_1, V_2\rangle)$. In the latter case, $h'(\langle u, U_1, U_2\rangle) = \big(h(\langle u, U_1, U_2\rangle), g(\langle u, U_1, U_2\rangle)\big)$ and $h'(\langle u, V_1, V_2\rangle) = \big(h(\langle u, V_1, V_2\rangle), g(\langle u, V_1, V_2\rangle)\big)$ and hence, from the definition of the action $\tilde{\ }$ of $\mathcal{M}'$, we obtain $h(\langle u, U_1, U_2\rangle) \doteq h(\langle u, V_1, V_2\rangle)$ and $g(\langle u, U_1, U_2\rangle) \doteq g(\langle u, V_1, V_2\rangle)$. Moreover, since h is projectable, we have that $h(\langle u, U_1, U_2\rangle) = h(\langle u, V_1, V_2\rangle)$. We still need to prove that that $g(\langle u, U_1, U_2\rangle) = g(\langle u, V_1, V_2\rangle)$ holds as well. To do that, we distinguish between the following subcases:

1. $U_1 = U_2 = \varnothing$. We have $g(\langle u, U_1, U_2\rangle) = 1_{\mathcal{N}}$ and hence, since $1_{\mathcal{N}}$ has empty memory and $g(\langle u, U_1, U_2\rangle) \doteq g(\langle u, V_1, V_2\rangle)$, we obtain that $g(\langle u, V_1, V_2\rangle) = 1_{\mathcal{N}}$.
2. Both U_1 and U_2 are non-empty. In this case $g(\langle u, U_1, U_2\rangle)$ must be either the null element $0_{\mathcal{N}}$ or the term $r(\varepsilon)$ (recall that this term represents all expanded data words that satisfy $x \sim y$). Both elements have empty memory and hence from $g(\langle u, U_1, U_2\rangle) \doteq g(\langle u, V_1, V_2\rangle)$ we devise $g(\langle u, U_1, U_2\rangle) = g(\langle u, V_1, V_2\rangle)$.
3. $U_1 \neq \varnothing$ and $U_2 = \varnothing$. Clearly, U_1 is a singleton of the form $\{x\}$. Similarly, since $g(\langle u, U_1, U_2\rangle) \doteq g(\langle u, V_1, V_2\rangle)$, we have that V_1 is a singleton of the form $\{x'\}$ and $V_2 = \varnothing$. We then recall that $h(\langle u, U_1, U_2\rangle) = h(\langle u, V_1, V_2\rangle) \neq 0_{\mathcal{M}}$ and that the morphism h is 0-reduced, which implies that $x = x'$. This shows that $g(\langle u, U_1, U_2\rangle) = g(\langle u, V_1, V_2\rangle)$.
4. $U_1 = \varnothing$ and $U_2 \neq \varnothing$. This case is symmetric to the previous one and can be dealt with by similar arguments.

It follows that h' is a projectable morphism. □

We are now ready to prove the theorem and the corollaries in Section 4.

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.*

Proof. As already mentioned, the proof is by structural induction on the rigidly guarded MSO formula $\varphi(\bar{X})$. As for the base cases, we observe that the languages defined by the atomic formulas $x < y$, $a(x)$, and $x \in Y$ are recognized by suitable orbit finite data monoids and projectable morphisms. As for the inductive step, we suppose to be given a formula φ (resp., two formulas φ_1 and φ_2) with m free variables $X_1, \dots, X_m$, an orbit finite data monoid $\mathcal{M}$ (resp., two orbit finite data monoids $\mathcal{M}_1$ and $\mathcal{M}_2$), and a projectable morphism $h : (D \times A \times B^m)^* \to \mathcal{M}$ (resp., two projectable morphisms $h_1 : (D \times A \times B^m)^* \to \mathcal{M}_1$ and $h_2 : (D \times A \times B^m)^* \to \mathcal{M}_2$) recognizing the languages defined by φ (resp., the languages defined by φ_1 and φ_2). We then observe that the language defined by the formula $\neg\varphi$ (resp., $\varphi_1 \wedge \varphi_2$) is recognized by the orbit finite data monoid $\mathcal{M}$ (resp., $\mathcal{M}_1 \times \mathcal{M}_2$) via the projectable morphism h (resp., $h_1 \times h_2$). As for the existential closure, Lemma 1 implies that the language defined by the formula $\exists\, X_m.\, \varphi$ is recognized by a suitable orbit finite data monoid $\mathcal{M}'$ via a projectable morphism h', both computable from $\mathcal{M}$ and h. Finally, if $m = 2$ and $\varphi(x_1, x_2)$ is a rigid formula, then we know from Lemma 2 how to compute an orbit finite data monoid $\mathcal{M}'$ and a projectable morphism h' that recognizes the language defined by $\varphi(x_1, x_2) \wedge x_1 \sim x_2$. This concludes the proof of the theorem. □

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).*

Proof. The case of rigidly guarded MSO corresponds just to Theorem 1 in the case of a sentence φ.

The case of rigidly guarded FO could be proved by establishing the aperiodicity at the same time. However, in our case, it is sufficient to remark that, according to [3], every data language definable in (non-necessarily rigidly guarded) FO is recognized by an aperiodic data monoid (in particular the syntactic one). Hence, if we consider the syntactic data monoid (we do not develop this object further here) of a language definable in rigidly guarded FO, it is aperiodic from [3], and it is finite orbit as a quotient of the finite orbit data monoid obtained from Theorem 1. □

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.*

Proof. By Theorem 1, any formula of rigidly guarded MSO can be effectively transformed into an orbit finite data monoid. Satisfiability of this formula then corresponds to language non-emptiness. This can be tested by a simple saturation argument on the algebraic object.

As for the second claim, to decide whether a formula φ belongs to the rigidly guarded MSO logic it is sufficient to check that (i) the formula φ satisfies the syntactical restrictions given by the grammar of rigidly guarded MSO formulas and (ii) all data comparisons are guarded by rigid formulas of the form $\alpha(x,y)$. Both the first and the second tasks can be performed by an induction of the structure of the formula φ. In particular, we observe that a guard $\alpha(x,y)$ is rigid iff the formula

$$\alpha^{\text{rigid?}} \overset{\text{def}}{=} \forall\, x, x', y, y'.\, \alpha(x,y) \wedge \alpha(x',y') \rightarrow (x = x' \leftrightarrow y = y')$$

holds on all words. The latter condition can be tested using the first claim of the corollary (a direct construction is also possible). □

C Proofs for Section 5 (From data monoids to the logic)

In this section we prove Theorem 2. We first outline the key ideas underlying the proof. Hereafter, $\mathcal{M}$ is an orbit finite data monoid and h is a morphism form the free data monoid $(D \times A)^*$ to $\mathcal{M}$.

The objective of the proof is to find suitable formulas that, given some positions $x \leq y$ in a word w, determine the orbit of $h(w[x,y])$ (i.e. the image of the infix $w[x,y]$ under the morphism h). The general technique is to exploit an induction on certain ideals of the orbit finite data monoid, which are induced by the so-called Green's relations (to be defined later). Roughly speaking, we first construct the desired formulas for shorter infixes of the word and then we move up towards longer infixes, until we determine the orbit of the entire word. Doing so, we need to be able to compute the orbit of an infix $w[x,y]$ on the basis of some bounded amount of information related to some factors of it (e.g. $w[x,z]$ and $w[z+1,y]$ for some z between x and y). This requires not only to compute the orbit of $h(w[x,y])$, but also its memorable values. Here "computing memorable values" means being able to locate some positions in $w[x,y]$ that carry the memorable values of the element $h(w[x,y])$. For this, one uses formulas of the form $\varphi(x,y,z_1,\dots,z_n)$ which determine the orbit of $h(w[x,y])$ and some witnessing positions $z_1,\dots,z_n$ for the memorable values. This must be done with care in order to preserve the rigidity assumptions necessary for the logic.

Definition 6. *We say that a formula $\varphi(x_1,\dots,x_n)$* determines *x_j from x_i if for all words w and positions x in w, there is y such that $w \vDash \varphi(x_1,\dots,x_n)$ and $x_i = x$ implies $x_j = y$.*
The formula $\varphi(x_1,\dots,x_n)$ is rigid *if x_i determines x_j for all i and all j among $1,\dots,n$ (note that this is consistent with our previous definition).*

Below, we formalize the notion of "computing the orbits under some guard".

Definition 7. *A formula $\varphi(x,y,z_1,\dots,z_n)$ is a* witnessing *formula if it determines all variables from x and, symmetrically, all variables from y, and whenever $w \vDash \varphi(x,y,z_1,\dots,z_n)$, then $x \leq z_i \leq y$ for all $1 \leq i \leq n$.*
A formula witnesses the orbit *o if it is a witnessing formula and*

$$w \vDash \varphi(x,y,z_1,\dots,z_n) \quad \textit{implies} \quad h(w[x,y]) = o(w[z_1],\dots,w[z_n]).$$

A family of formulas $F = (\varphi_o)_{o\in O}$ computes the orbits under the guard *$\alpha(x,y)$ if each formula φ_o witnesses the orbit o, and $\bigvee_o \exists\, \bar{z}.\ \varphi_o(x,y,\bar{z})$ is equivalent to $\alpha(x,y)$. One says that one can* compute the orbits under the guard *α if there exists such a family.*

We aim at proving that for every rigid formula $\alpha(x,y)$ (and, in particular, for the rigid formula $\alpha(x,y) = (\neg\exists\, z.\ z < x) \wedge (\neg\exists\, z.\ z > y)$), one can compute the orbits under α. The key idea is to exploit an induction on the algebraic structure of the orbit finite data monoid $\mathcal{M}$. This induction is guided by the so-called Green's relations, introduced just below.

As already noticed in [3], a relevant part of Green's theory [**?**, 11], which holds for finite monoids, can be lifted to *locally finite* monoids and, in particular, to orbit finite data monoids. The basic Green's relations $\leq_{\mathcal{R}}$, $\leq_{\mathcal{L}}$, $\leq_{\mathcal{J}}$ associated with $\mathcal{M}$ are the preorders defined by:

$$s \leq_{\mathcal{R}} t \text{ iff } s{\cdot}M \subseteq t{\cdot}M \qquad s \leq_{\mathcal{L}} t \text{ iff } M{\cdot}s \subseteq M{\cdot}t \qquad s \leq_{\mathcal{J}} t \text{ iff } M{\cdot}s{\cdot}M \subseteq M{\cdot}t{\cdot}M.$$

We denote by $\mathcal{R}$, $\mathcal{L}$, $\mathcal{J}$ the corresponding equivalence relations (e.g., $s \mathrel{\mathcal{J}} t$ iff $s \leq_{\mathcal{J}} t$ and $t \leq_{\mathcal{J}} s$) and we define an additional fourth relation $\mathcal{H}$ defined by $s \mathrel{\mathcal{H}} t$ iff $s \mathrel{\mathcal{R}} t$ and $s \mathrel{\mathcal{L}} t$. Given an element s of a data monoid $\mathcal{M}$, we denote by $R(s)$ (resp., $L(s)$, $J(s)$, $H(s)$) the $\mathcal{R}$-class (resp., $\mathcal{L}$-class, $\mathcal{J}$-class, $\mathcal{H}$-class) of s.

We also lift the above relations to orbits, namely, for each $\mathcal{K}$ among $\mathcal{R}$, $\mathcal{L}$, $\mathcal{J}$, we denote by $\leq_{\dot{\mathcal{K}}}$ the preorder relation such that $s \leq_{\dot{\mathcal{K}}} t$ iff $s \leq_{\mathcal{K}} \hat{\tau}(t)$ for some renaming $\tau \in G_D$. We do the same for the equivalence relations $\mathcal{R}$, $\mathcal{L}$, $\mathcal{J}$, $\mathcal{H}$.

The proof of Theorem 2 is an induction based on the $\dot{J}$-classes of $\mathcal{M}$:

Lemma 3 (Inductive statement). *For every $\dot{\mathcal{J}}$-class $\dot{J}$ of $\mathcal{M}$, the following claims hold:*

(C1) there is a formula $\varphi_{\dot{J}}(x,y)$ such that $w \vDash \varphi_{\dot{J}}(x,y)$ iff $h(w[x,y]) \in \dot{J}$;

(C2) for every guard $\alpha(x,y)$ such that $w \vDash \alpha(x,y)$ implies $h(w[i,j]) \geq_{\dot{\mathcal{J}}} \dot{J}$, there exists effectively a family of formulas F^{α} computing the orbits under α.

We will prove the above lemma twice: the first time under the assumption that $\mathcal{M}$ is *aperiodic*, returning formulas of the rigidly guarded *FO* logic, and the second time without the assumption of aperiodicity, returning formulas of the rigidly guarded *MSO* logic. In the aperiodic case, for computing the orbit of an infix, one uses the fact that it is sufficient to know its $\mathcal{L}$-class and its $\mathcal{R}$-class, plus the equality relationships between the memorable values in the two classes (this follows basically from the fact that the $\mathcal{H}$-classes of an aperiodic monoid are singletons). This is not true in the general (non-aperiodic) case, and different objects have to be used, which results in the use of monadic second-order variables.

Moreover, in both the aperiodic and the non-aperiodic cases a careful analysis of Green's relations and of memories of elements in an orbit finite data monoid is required. We do this in the next Section C.1. Later, we will prove the inductive invariant in the aperiodic case (Section C.2) and in the non-aperiodic case (Section C.3).

C.1 Green's theory and memorable values

Here we focus our attention on the memorable values of the elements of orbit finite data monoids.

Definition 8. *Given an element m of an orbit finite data monoid $\mathcal{M}$, we define by* $\mathrm{mem}_{\mathcal{R}}(s)$ *(resp.,* $\mathrm{mem}_{\mathcal{L}}(s)$*) to be the intersection of* $\mathrm{mem}(t)$ *for all elements t in the $\mathcal{R}$-class (resp., $\mathcal{L}$-class) of s.*

We call *$\mathcal{R}$-memorable* (resp., *$\mathcal{L}$-memorable*) values of s the values in $\mathrm{mem}_{\mathcal{R}}(s)$ (resp., $\mathrm{mem}_{\mathcal{L}}(s)$).

Our final goal is to transform data monoids into rigidly guarded formulas. For this, one needs to locate where the memorable data values come from. An important tool is the following result:

Proposition 4. *For every element s of an orbit finite data monoid $\mathcal{M}$, we have* $\mathrm{mem}(s) = \mathrm{mem}_{\mathcal{R}}(s) \cup \mathrm{mem}_{\mathcal{L}}(s)$.

Before turning to the proof Proposition 4, let us show that a similar result fails for data monoids with infinitely many data orbits. Consider the data language $L_{\text{even}} \subseteq D^*$ that consists of all words $u \in D^*$ where every value $d \in D$ occurs in u an even number of times. The syntactic data monoid of the language L_{even} consists of one element m_C for each finite subset C of D. The product corresponds to the symmetric difference of sets. It is easy to see that the memorable values of m_C are exactly the values in C, which are neither $\mathcal{L}$-memorable nor $\mathcal{R}$-memorable (the data monoid is a group).

In order to prove Proposition 4, we need to introduce a couple of other concepts. An *inverse* of an element s of a monoid, is an element t such that $s \cdot t = t \cdot s = 1_{\mathcal{M}}$. If the inverse of s exists, then it can be easily proven to be unique and hence it can be denoted by s^{-1}. A *data group* is simply a *data monoid* where all elements have an inverse. The next lemma shows that orbit finiteness is, quite surprisingly, a severe restriction for data groups.

Lemma 4. *Every orbit finite data group is finite.*

Proof. We first observe that:

> **Claim 1.** *In an orbit finite data group,* $\mathrm{mem}(s) = \mathrm{mem}(s^{-1})$.
>
> *Proof of Claim 1.* Since orbit finite data groups are locally finite, there is $n > 0$ such that $s^n = 1$ and hence $s^{-1} = s^{n-1}$. This implies $\mathrm{mem}(s^{-1}) = \mathrm{mem}(s^{n-1}) \subseteq \mathrm{mem}(s)$. By symmetry, we get $\mathrm{mem}(s) = \mathrm{mem}(s^{-1})$. □
>
> **Claim 2.** $\mathrm{mem}\, s \setminus \mathrm{mem}\, t \subseteq \mathrm{mem}(s \cdot t)$.
>
> *Proof of Claim 2.* Indeed, $\mathrm{mem}(s) = \mathrm{mem}(s \cdot t \cdot t^{-1}) \subseteq \mathrm{mem}(s \cdot t) \cup \mathrm{mem}(t^{-1}) = \mathrm{mem}(s \cdot t) \cup \mathrm{mem}(t)$. Hence, $\mathrm{mem}(s) \setminus \mathrm{mem}(t) \subseteq \mathrm{mem}(s \cdot t)$. □

Now, assume, towards a contradiction, that $\mathcal{G}$ is an infinite data group with finitely many orbits. $\mathcal{G}$ must contain an infinite orbit o. Let $H = \{h_1, h_2, \ldots\}$ be some infinite subset of o such that each element h_i has a distinguished memorable value d_i that is not memorable in any other element of H. Then, from the above claim, for all k, $\{d_1, \ldots, d_k\} \subseteq \mathrm{mem}(s_1 \cdots s_k)$. This contradicts the finite memory property. □

It is know that every $\mathcal{H}$-class H of a monoid is associated with a group $\Gamma(H)$ called the Schützenberger group [11] (in fact there exist two such groups, but we will only consider one of them here). We define $T(H)$ to be the set of all elements $t \in H$ such that $t \cdot H$ is a subset of H. For each $t \in T(H)$ we let γ_t be the transformation on H that maps $h \in H$ to $t \cdot h$. Formally the Schützenberger group $\Gamma(H)$ consists of the set of all transformations γ_t with $t \in T(H)$. The multiplication operation of the group is the functional composition $\circ$. Moreover, there is a natural way to extend the action $\hat{}$ of the data monoid $\mathcal{M}$ to an action $\tilde{}$ on $\Gamma(H)$ by simply letting $\tilde{\tau}(\gamma_s) = \gamma_{\hat{\tau}(s)}$ for all renamings $\tau \in \Gamma_{D \setminus (\mathrm{mem}_{\mathcal{R}}(H) \cup \mathrm{mem}_{\mathcal{L}}(H))}$, where $\mathrm{mem}_{\mathcal{R}}(H) = \mathrm{mem}_{\mathcal{R}}(h)$ and $\mathrm{mem}_{\mathcal{L}}(H) = \mathrm{mem}_{\mathcal{L}}(h)$ for some arbitrary element $h \in H$ (note that all elements of H have the same set of $\mathcal{R}$-memorable values and the same set of $\mathcal{L}$-memorable values). The following lemma shows that $\tilde{}$ is indeed a group action on the Schützenberger group $\Gamma(H)$.

Lemma 5. *If $\mathcal{M} = (M, \cdot, \hat{})$ is a data monoid over the set D of data values and H is an $\mathcal{H}$-class of $\mathcal{M}$, then $(\Gamma(H), \circ, \tilde{})$ is a data group over the set $D \setminus (\mathrm{mem}_{\mathcal{R}}(H) \cup \mathrm{mem}_{\mathcal{L}}(H))$. Moreover, if $\mathcal{M}$ is orbit finite, then $(\Gamma(H), \circ, \tilde{})$ is orbit finite as well.*

Proof. It is known that $(\Gamma(H), \circ)$ is a group. We only need to verify that $\tilde{}$ is an action on $\Gamma(H)$. We first show that $\Gamma(H)$ is closed under the action $\tilde{}$ of the renamings over $D \setminus (\mathrm{mem}_{\mathcal{R}}(H) \cup \mathrm{mem}_{\mathcal{L}}(H))$. By definition of $\tilde{}$, this is equivalent to verifying that H is closed under the action $\hat{}$ of renamings over $D \setminus (\mathrm{mem}_{\mathcal{R}}(H) \cup \mathrm{mem}_{\mathcal{L}}(H))$. The following proof is similar to proof of the Memory Theorem for $\mathcal{J}$-classes in [3]; however, we give a complete proof here for the sake of self-containment.

Suppose that H is the intersection of an $\mathcal{R}$-class R and an $\mathcal{L}$-class L. As a renaming is a permutation that is the identity on all but finite many values, any renaming over $D \setminus (\mathrm{mem}_{\mathcal{R}}(H) \cup \mathrm{mem}_{\mathcal{L}}(H))$ can be decomposed into a sequence of transpositions of pairs of values from $D \setminus (\mathrm{mem}_{\mathcal{R}}(H) \cup \mathrm{mem}_{\mathcal{L}}(H))$. Therefore, in order to prove the closure of $H = R \cap L$ under the action of the renamings over $D \setminus (\mathrm{mem}_{\mathcal{R}}(H) \cup \mathrm{mem}_{\mathcal{L}}(H))$, it is sufficient to prove a similar closure property for the transpositions π_{de} of pair of elements $d, e \notin \mathrm{mem}_{\mathcal{R}}(H) \cup \mathrm{mem}_{\mathcal{L}}(H)$. We first show that R is closed under such transpositions. Let d and e be two values outside $\mathrm{mem}_{\mathcal{R}}(H)$ and let π_{de} be their transposition. Since $d, e \notin \mathrm{mem}_{\mathcal{R}}(H)$, we know that there exist two elements $s, t \in R$ such that d is not memorable in s and b is not memorable in t. Let f be a value outside $\mathrm{mem}(s) \cup \mathrm{mem}(t)$. By definition of memory, we know that $\hat{\pi}_{df}$, where π_{df} is the transposition of d and f, is a stabilizer of s and, similarly, $\hat{\pi}_{ef}$ is a stabilizer of t. Now, consider an element s' that is $\mathcal{R}$-equivalent to s. There must exist u and u' in $\mathcal{M}$ such that $s \cdot u = s'$ and $s' \cdot u' = s$. Since $\hat{}$ commutes with the product of $\mathcal{M}$, we obtain $\hat{\pi}_{df}(s') = \hat{\pi}_{df}(s \cdot u)$ and hence $\hat{\pi}_{df}(s') \leq_{\mathcal{R}} \hat{\pi}_{df}(s)$. By similar arguments, we obtain $\hat{\pi}_{df}(s') \geq_{\mathcal{R}} \hat{\pi}_{df}(s)$. We thus have $\hat{\pi}_{df}(s') \,\mathcal{R}\, \hat{\pi}_{df}(s) = s \in R$. A symmetric argument shows that $\hat{\pi}_{ef}(s') \,\mathcal{R}\, \hat{\pi}_{ef}(s) = s \in R$. Since $\pi_{de} = \pi_{df} \circ \pi_{ef} \circ \pi_{df}$, we conclude that $\hat{\pi}_{de}(s) \in R$. Finally, a similar proof shows that $\hat{\pi}_{de}(s) \in L$. Putting all together, we have that for every $s \in H = R \cap L$ and every renaming τ over $D \setminus (\mathrm{mem}_{\mathcal{R}}(H) \cup \mathrm{mem}_{\mathcal{L}}(H))$, $\hat{\tau}(s) \in R \cap L = H$. This shows that H is closed under the action $\hat{}$ of renamings over $D \setminus (\mathrm{mem}_{\mathcal{R}}(H) \cup \mathrm{mem}_{\mathcal{L}}(H))$.

Below, we verify that $\tilde{}$ is a group morphism from the group of renamings $G_{D\setminus(\mathrm{mem}_{\mathcal{R}}(H)\cup\mathrm{mem}_{\mathcal{L}}(H))}$ to the group of automorphisms on $\Gamma(H)$. Clearly, the function $\tilde{}$ maps the identity ι on $G_{D\setminus(\mathrm{mem}_{\mathcal{R}}(H)\cup\mathrm{mem}_{\mathcal{L}}(H))}$ to the trivial automorphism $\tilde{\iota}$ on $\Gamma(H)$ (i.e., $\tilde{\iota}(\gamma_s) = \gamma_{\hat{\iota}(s)} = \gamma_s$). Moreover, $\tilde{}$ is a morphism because

$$\widetilde{\tau\circ\pi}(\gamma_s) \;=\; \gamma_{\widehat{\tau\circ\pi}(s)} \;=\; \gamma_{\hat{\tau}\circ\hat{\pi}(s)} \;=\; (\tilde{\tau}\circ\tilde{\pi})(\gamma_s).$$

Finally, we observe that $\gamma_{s\cdot t} = \gamma_s \circ \gamma_t$ (indeed, for every $h \in H$, we have $\gamma_{s\cdot t}(h) = (s\cdot t)\cdot h = s\cdot(t\cdot h) = \gamma_s(t\cdot h) = \gamma_s\circ\gamma_t(h)$) and hence

$$\hat{\tau}(\gamma_s)\circ\hat{\tau}(\gamma_t) \;=\; \gamma_{\hat{\tau}(s)}\circ\gamma_{\hat{\tau}(t)} \;=\; \gamma_{\hat{\tau}(s)\cdot\hat{\tau}(t)} \;=\; \gamma_{\hat{\tau}(s\cdot t)} \;=\; \hat{\tau}(\gamma_{s\cdot t}) \;=\; \hat{\tau}(\gamma_s\circ\gamma_t).$$

To complete the proof of the lemma, we need to prove that $(\Gamma(H),\circ,\tilde{})$ is orbit finite when $\mathcal{M}$ is orbit finite. Let us consider two elements $s,t \in H$ and suppose that s and t are in the same orbit, namely, that there is $\tau \in G_{D\setminus(\mathrm{mem}_{\mathcal{R}}(H)\cup\mathrm{mem}_{\mathcal{L}}(H))}$ such that $t = \hat{\tau}(s)$. Since $\tilde{}$ is a group action, we know that $\gamma_t = \gamma_{\hat{\tau}(s)} = \tilde{\tau}(\gamma_s)$. This shows that the two elements γ_s and γ_t of $\Gamma(H)$ are on the same orbit. □

It is known that any $\mathcal{H}$-class H of a monoid has the same cardinality of the associated Schützenberger group $\Gamma(H)$. Hence, we obtain the following interesting property:

Corollary 4. *All $\mathcal{H}$-classes of an orbit finite data monoid are finite.*

Proof. Let H be an $\mathcal{H}$-class of an orbit finite data monoid $\mathcal{M}$. By Lemma 5, we can associate with H an orbit finite data group $(\Gamma(H),\circ,\tilde{})$, where $\Gamma(H)$ is the Schützenberger group of H. By Lemma 4, it follows that $\Gamma(H)$ is finite. Moreover, it is known from classical results in algebra (see, for instance, [11]), that the Schützenberger group $\Gamma(H)$ has the same cardinality as the $\mathcal{H}$-class H. We thus conclude that H is finite. □

We are now ready to prove Proposition 4.

Proof (Proof of Proposition 4). We aim at $\mathrm{mem}(s) \subseteq \mathrm{mem}_{\mathcal{R}}(s) \cup \mathrm{mem}_{\mathcal{L}}(s)$. Assume towards a contradiction that there is a value $d \in \mathrm{mem}(s) \setminus (\mathrm{mem}_{\mathcal{R}}(s) \cup \mathrm{mem}_{\mathcal{L}}(s))$. As $d \notin \mathrm{mem}_{\mathcal{R}}(s)$, there is an s' in the $\mathcal{R}$-class of s such that $d \notin \mathrm{mem}(s')$. Then there are $u, u' \in \mathcal{M}$ such that $s\cdot u = s'$ and $s'\cdot u' = s$. Symmetrically, as $d \notin \mathrm{mem}_{\mathcal{L}}(s)$, s has an $\mathcal{L}$-equivalent element s'' such that $d \notin \mathrm{mem}(s'')$, and there are $v, v'' \in \mathcal{M}$ such that $v\cdot s = s''$ and $v''\cdot s'' = s$.

Let $d_1, d_2, \ldots$ be an infinite sequence of pairwise distinct values that are not in the memory of either s, s', or s''. We denote by π_i the transposition of d with d_i. As neither d nor $d_1, d_2, \ldots$ are in the memory of s', $\hat{\tau}_i(s') = s'$ and hence $\hat{\tau}_i(s\cdot u) = \hat{\tau}_i(s') = s'$. Combining this with $s'\cdot u' = s$ we obtain $\hat{\tau}_i(s)\cdot\hat{\tau}_i(u)\cdot u' = s'\cdot u' = s$ and hence $\hat{\tau}_i(s) \geq_{\mathcal{R}} s$. Similarly, one proves that $\hat{\tau}_i(s) \leq_{\mathcal{R}} s$ and hence $\hat{\tau}_i(s)\ \mathcal{R}\ s$. By symmetry $\hat{\tau}_i(s)\ \mathcal{L}\ s$. Hence $\hat{\tau}_i(s)$ belongs to the $\mathcal{H}$-class of s. As d is memorable in s, $\hat{\tau}_i(s)$ is different from s. Thus the $\mathcal{H}$-class of s is infinite, contradicting Corollary 4. □

C.2 The translation in the aperiodic case

Through the rest of this section we assume that $\mathcal{M}$ is an *aperiodic* orbit finite data monoid $\mathcal{M}$ and we prove the inductive statement given in Lemma 3.

We will tacitly assume that all formulas defined hereafter are rigidly guarded *first-order* formulas. Moreover, for the sake of brevity, we will often fill the parameters of a formula $\varphi(x_1, \dots, x_n)$ with $*$ to denote that the corresponding variables are existentially quantified. With this notation, if $\varphi(x, y, z)$ is rigid (according to the general definition given at the beginning of Section C, then so is $\varphi(*, x, y)$, as well as $\varphi(x, *, y)$ and $\varphi(x, y, *)$.

It is also convient to assume that $\mathcal{M}$ is represented by a term-based presentation system $\mathcal{S} = (T, \odot, \check{}\ , \approx)$. This means that the elements of $\mathcal{M}$ are the $\approx$-equivalence classes of terms in T. However, by a slight abuse of notation, we shall often identify the elements of the data monoid $\mathcal{M}$ with the terms in T, writing, for instance, $h(w[x, y]) = o(d_1, \dots, d_k)$.

We start by presenting an special, but important, case of Lemma 3, which shows that the orbits of infixes of length 1 can be computed (this will serve as our base case for the inductive construction):

Lemma 6. *Let $\alpha^1(x, y) =^{def} x = y$. One can compute the orbits under the guard α^1.*

Proof. Remark that the morphism h maps singleton words to orbits that have memory size at most 1. The family $F^1 =^{\text{def}} (\phi^1_o)_{o \in O}$ that computes the orbits under α^1 is simply:

$$\phi^1_o(x, y) =^{\text{def}} \bigvee_{h((d,a)) = o()} a(x) \;\wedge\; x = y \qquad \text{(if } o \text{ has memory size 0)}$$

$$\phi^1_o(x, y, z) =^{\text{def}} \bigvee_{h((d,a)) = o(d)} a(x) \;\wedge\; x = y \;\wedge\; x = z \qquad \text{(if } o \text{ has memory size 1)}$$

$$\phi^1_o(x, y) =^{\text{def}} \texttt{false}. \qquad \text{(otherwise)}$$

□

The following lemma discloses another key argument in the proof of Lemma 3, which is obtained by syntactic transformations of formulas.

Lemma 7 (Sub-definability Lemma). *For all formulas $\varphi(x, y)$ that determine y from x, there exist finitely many formulas $\beta_i(z, y)$ that determine y from z, and such that for all $x \le z \le y$,*

$$w \vDash \varphi(x, y) \quad \textit{implies} \quad w \vDash \beta_i(z, y) \textit{ for some } i.$$

Proof. We remark that the following proof can be read either with formulas meaning “rigidly guarded FO formulas”, or “rigidly guarded MSO formulas”; both results hold, using the same proof.

We start by recalling the following Composition Lemma originating from the Feferman-Vaught/Shelah composition method.

Composition Lemma. *Given a standard MSO/FO sentence φ (that uses only the order $<$, and some unary predicates, but no data comparisons), there exists a finite set of pairs of formulas $(\alpha_i, \beta_i)_{i=1\dots k}$ such that for all words u and v,*

$$uv \vDash \varphi \quad \textit{iff} \quad u \vDash \alpha_i \textit{ and } v \vDash \beta_i \textit{ for some } i.$$

It is routine to check that the statement of the Sub-definability Lemma follows from the above result in the case of standard MSO/FO formulas: the variable z cuts the word into uv, with v starting at letter z (it is sufficient to prove the statement for $x \leq z \leq y$). Let us concentrate ourselves on the data word case.

Preliminary remark: Any rigidly guarded data test $\alpha(x,y) \wedge x \sim y$, since it determines y from x is equivalent to the formula:

$$\alpha(x,y) \wedge \underbrace{(\exists\, z.\, \alpha(x,z) \wedge x \sim z)}_{\text{call it } \alpha^{\sim}(x)}.$$

(this formalizes the fact that rigidly guarded data tests behave almost like unary predicates).

Formal elimination of data tests. Consider now a rigidly guarded MSO/FO formula φ (possibly with free variables). One transforms it into the standard MSO/FO formula φ^* (with the same free variables) inductively as follows:

$$\begin{array}{rlrl} (\exists\, x.\, \psi)^* & =^{\text{def}} \exists\, x.\, \psi^* & (\exists\, X.\, \psi)^* & =^{\text{def}} \exists\, X.\, \psi^* \\ (\neg\psi)^* & =^{\text{def}} \neg\psi^* & (\psi_2 \vee \psi_2)^* & =^{\text{def}} \psi_1^* \vee \psi_2^* \\ (x \in X)^* & =^{\text{def}} x \in X & (x < y)^* & =^{\text{def}} x < y \\ (a(x))^* & =^{\text{def}} a(x) & & \end{array}$$

and most importantly

$$(\alpha(x,y) \wedge x \sim y)^* =^{\text{def}} \alpha^*(x,y) \wedge [\alpha^{\sim}](x)$$
$$\text{(where } [\alpha^{\sim}] \text{ is a new unary predicate)}$$

Given a data word w, let the (classical) word w^* be obtained from w by removing all data values and by adding the predicates $[\alpha^{\sim}]$ such that $w^* \vDash [\alpha^{\sim}](x)$ iff $w \vDash \alpha^{\sim}(x)$. The above construction – thanks to the preliminary remark – is such that $w \vDash \varphi(\bar{X})$ iff $w^* \vDash \varphi^*(\bar{X})$ for all choices of w and $\bar{X}$. This means in particular that any formula φ is equivalent to the formula φ^* where each unary predicate $[\alpha^{\sim}](x)$ is syntactically replaced by $\alpha^{\sim}(x)$.

Main part of the proof. Consider a formula $\varphi(x,y)$ that determines y from x. It can happen that x does not determine y in $\varphi^*(x,y)$ (since the unary predicates $[\alpha^{\sim}]$ could be chosen in a way inconsistent with any choice of data values). This can be corrected easily. Consider:

$$\psi^*(x,y) =^{\text{def}} \varphi^*(x,y) \wedge \forall\, y'.\, \varphi^*(x,y') \to y' = y.$$

This formula ψ^* is equivalent to φ^* as long as φ^* determines y from x for a given choice of w, x. Otherwise, it simply does not hold. Hence x determines y

in $\psi^*(x, y)$ by construction. This means that $\psi^*(x, y)$ is subject to the application of the standard MSO/FO case. Thus let $\beta_1^*(z, y), \ldots, \beta_k^*(z, y)$ be the corresponding formulas.

From each $\beta_i^*(z, y)$ we construct $\beta_i(z, y)$ by syntactically replacing each unary predicate $[\alpha^{\sim}](x')$ by $\alpha^{\sim}(x')$. It is clear that, since $\beta_i^*(z, y)$ determines y from z, so does $\beta_i(z, y)$. Furthermore, for all w and all $x \leq z \leq y$, $w \vDash \varphi(x, y)$ iff $w^* \vDash \varphi^*(x, y)$ iff $w^* \vDash \psi^*(x, y)$ (this is because there is no other y such that $w \vDash \varphi(z, y)$, and hence no other choice of y such that $w^* \vDash \varphi^*(x, y)$). Hence, there exists i such that $w^* \vDash \beta_i^*(z, y)$ and $w \vDash \beta_i(z, y)$ follows. □

An immediate consequence of the above lemma is the following:

Corollary 5. *Every witnessing formula is equivalent to a finite disjunction of rigid formulas.*

Proof. Consider a witnessing formula $\varphi(x, y, z_1, \ldots, z_n)$. Since $\varphi(x, y, *, \ldots, *)$ determines y from x, one can apply Lemma 7 and get some formulas $\alpha_1, \ldots, \alpha_k$.

The desired rigid formulas are

$$\varphi_{i_1,\ldots,i_n}(x, y, z_1, \ldots, z_n) =^{\text{def}} \varphi(x, y, z_1, \ldots, z_n) \wedge \alpha_{i_1}(z_1, y) \wedge \ldots \wedge \alpha_{i_n}(z_n, y).$$

where $i_1, \ldots, i_n$ range over $\{1, \ldots, k\}$. One easily checks that the formulas $\varphi_{i_1,\ldots,i_n}$ are rigid. Indeed, x determines z_k, which itself, by α_{i_k} determines y, which determines x. Since this holds for every $k = 1, \ldots, n$, we get that $\varphi_{i_1,\ldots,i_n}$ is rigid.

Of course, $\varphi_{i_1,\ldots,i_n}$ implies φ by definition. Conversely, given some $x, y, z_1, \ldots, z_n$ such that $w \vDash \varphi(x, y, z_1, \ldots, z_n)$. For each $k = 1, \ldots, n$, by Lemma 7, there exists i_k such that $s \vDash \alpha_{i_k}(z_k, y)$. It follows that $w \vDash \varphi_{i_1,\ldots,i_n}(x, y, z_1, \ldots, z_n)$. Overall, φ is equivalent to a disjunction of rigid formulas. □

Corollary 5 can be used for composing families of formula that compute orbits, as shown by the following lemma:

Lemma 8. *If F and F' compute the orbits under the guards $\alpha(x, y)$ and $\alpha'(x, y)$, respectively, there exists effectively a family $F \cdot F'$ which computes the orbit under the guard*

$$(\alpha \cdot \alpha')(x, y) =^{def} \exists\, z.\, \alpha(x, z) \wedge \alpha'(z + 1, y).$$

Proof. Let F be $(\varphi_o)_{o \in O}$, and F' be $(\varphi'_o)_{o \in O}$. We aim at constructing $F \cdot F' = (\psi_o)_{o \in O}$ which computes the orbits under $\alpha \cdot \alpha'$.

The orbit resulting from the product of an element in orbit o with an an element in orbit o' depends on the relative data values. For each possible relationship, we will produce a formula. This relationship will be represented by using explicit terms, which contain data values. Consider two terms of the form $t = o(c_1, \ldots, c_k)$ and $t' = o'(c'_1, \ldots, c'_{k'})$ (up to renaming, there are only finitely many possibilities for the pair (t, t')). Their product is $o''(d_1, \ldots, d_n) =^{\text{def}} t \cdot t'$. Call $(\varphi_p)_{p=1\ldots m}$ (similarly $(\varphi'_{p'})_{p'=1\ldots m'}$) the rigid formulas obtained from φ_o

(resp. from $\varphi'_{o'}$) by Corollary 5. Consider now the formula:

$$\begin{aligned}
\psi_{t\cdot t'}(x, y, z''_1 \dots, z''_n) =^{\text{def}} \; & \exists\, z_1 \dots z_k, \xi, z'_1 \dots z'_{k'}. && \text{(a)}\\
& \bigvee_{p,p'} \varphi_p(x, \xi, \bar{z}) \;\wedge\; \varphi_{p'}(\xi + 1, y, \bar{z}') && \text{(b)}\\
\wedge\; & \bigwedge_{c_i = c'_j} \alpha_{p,p',i,j}(z_i, z'_j) \;\wedge\; z_i \sim z'_j && \text{(c)}\\
\wedge\; & \bigwedge_{c_i \neq c'_j} \alpha_{p,p',i,j}(z_i, z'_j) \;\wedge\; z_i \nsim z'_j && \text{(d)}\\
\wedge\; & \bigwedge_{d_i = c_j} z''_i = z_j \;\wedge \bigwedge_{d_i = c'_j, d_i \notin \{c_1, \dots, c_m\}} z''_i = z'_j && \text{(e)}
\end{aligned}$$

where

$$\alpha_{p,p',i,j}(z_i, z'_j) \;=^{\text{def}}\; \exists\, y.\, \varphi_p(*, y, \bar{*}, z_i, \bar{*}) \;\wedge\; \varphi_{p'}(y + 1, *, \bar{*}, z'_j, \bar{*}).$$

Given x and y, the formula first guesses the intermediate position ξ and the variables $\bar{z}$ and $\bar{z}'$ witnessing the data values of $h(w[x, \xi])$ and of $h(w[\xi + 1, y])$ respectively (a). It then guesses p, p' which are used to make the formulas rigid, and checks the consistency with the variables (b). Line (c) checks that whenever a data value in t and a data value of t' are equal, then the corresponding witness positions in the word share the same data value. This equality comparison is done using the guard $\alpha_{p,p',i,j}(z_i, z'_j)$ (if this guard would be removed, the formula would still compute the same result, but the formula would not be a rigidly guarded FO formula). This guard of course holds between z_i and z'_j when (b) holds. It is also obviously rigid since the φ_p and $\varphi'_{p'}$ formulas are rigid. The same argument is used for the inequalities in (d). Finally (e) uniquely defines the witnesses for the data values.

Overall, the formula $\psi_{t\cdot t'}$ witnesses o', and furthermore, given any x, ξ, y such that $w \models \alpha(x, \xi)$, $w \models \alpha'(\xi + 1, y)$, $\tau(t) = h(w[x, \xi])$ and $\tau(t') = h(w[\xi + 1, y]$ for some renaming τ, then $w \models \psi_{t,t'}(x, y, \bar{z})$ for some $\bar{z}$.

Finally, the formula ψ_o is simply defined as:

$$\psi_o(x, y, \bar{z}) \;=^{\text{def}} \bigvee_{t\cdot t' \in o} \psi_{t\cdot t'}(x, y, \bar{z}).$$

(it tries every possibilities of a product yielding to orbit o). □

Using Lemma 6 and Lemma 8, one can compute the orbits of infixes of fixed length:

Corollary 6. *Let $\alpha^k(x, y) =^{def} x + k - 1 = y$, there exists effectively a family of formulas which computes orbits under α^k.*

We now recall a direct consequence of Theorem V.1.9 from [11]:

Lemma 9. *For every pair of elements s, t of $\mathcal{M}$, if $s \mathrel{\mathcal{J}} t$ and $s \leq_{\mathcal{R}} t$, then $s \mathrel{\mathcal{R}} t$ (and symmetrically for $\mathcal{L}$).*

The above lemma immediately implies the following:

Lemma 10. *Let $[x,y]$ be a minimal interval such that $h(w[x,y]) \ngeq_{\dot{J}} \dot{J}$, then either*

1. *$x = y$ or $x+1 = y$, or;*
2. *$[x+1, y-1]$ is a maximal interval such that $h(w[x,y]) >_{\dot{\mathcal{J}}} \dot{J}$.*

From now on, we assume that Claims C1 and C2 of Lemma 3 hold for every $\dot{\mathcal{J}}$-class above $\dot{J}$ (which is the induction hypothesis for classes above $\dot{J}$).

Lemma 11. *There exists a formula $\alpha^{\min}_{\ngeq_{\dot{\mathcal{J}}}\dot{J}}(x,y)$ such that $w \vDash \alpha^{\min}_{\ngeq_{\dot{\mathcal{J}}}\dot{J}}(x,y)$ iff $[x,y]$ is a minimal interval such that $h(w[x,y]) \ngeq_{\dot{\mathcal{J}}} \dot{J}$. Furthermore, it is rigid, and one can compute the orbits under $\alpha^{\min}_{\ngeq_{\dot{\mathcal{J}}}\dot{J}}$.*

Proof. Lemma 10 describes what can be such intervals $[x,y]$. Naturally, the last case is the most interesting. Let $\alpha(x,y)$ be a formula stating that $[x,y]$ is a maximal interval such that $h(w[x,y]) >_{\dot{\mathcal{J}}} \dot{J}$ and $x < y$. It is possible to write such a formula using the induction hypothesis C1. This formula is rigid, by definition. Hence, using this time Claim C2, one has a family of formulas F which computes the orbit under the guard α.

The formula $\alpha^{\min}_{\ngeq_{\dot{\mathcal{J}}}\dot{J}}(x,y)$ is then simply:

$$\begin{aligned}
&\alpha^1(x,y) \;\wedge\; F^1(x,y) \ngeq_{\dot{\mathcal{J}}} \dot{J}\\
\vee\;& \alpha^2(x,y) \;\wedge\; F^2(x,y) \ngeq_{\dot{\mathcal{J}}} \dot{J}\\
\vee\;& ((\alpha^1 \cdot \alpha) \cdot \alpha^1)(x,y) \;\wedge\; ((F^1 \cdot F) \cdot F^1)(x,y) \ngeq_{\dot{\mathcal{J}}} \dot{J}
\end{aligned}$$

where we use families of formulas $F, \ldots$ as if they were functions computing orbits. This shorthand of notation should be clear to understand, and can be transformed into regular formulas by explicitly unfolding all cases. This is correct by Lemma 10.

Finally, this formula $\alpha^{\min}_{\ngeq_{\dot{\mathcal{J}}}\dot{J}}(x,y)$ is rigid by definition, and a family of formulas computing the orbits under this guard is also easy to obtain using the same kind of constructions. □

We are now ready to prove the induction steps for both Claim C1 and Claim C2 of Lemma 3 with respect to the $\dot{\mathcal{J}}$-class $\dot{J}$ (we remark that only the proof of C2 relies on the fact that the orbit finite data monoid is aperiodic).

Lemma 12 (Induction step for C1). *There exists a formula $\varphi_{\dot{J}}(x,y)$ such that $w \vDash \varphi_{\dot{J}}(x,y)$ iff $h(w[x,y]) \in \dot{J}$.*

Proof. The formula disproves the existence of a minimal interval $[x',y']$ included in $[x,y]$ such that $\alpha^{\min}_{\ngeq_{\dot{\mathcal{J}}}\dot{J}}(x',y')$ holds, using Lemma 11. This implies $h(w[x,y]) \geq_{\dot{\mathcal{J}}} \dot{J}$. The formula then excludes the case $h(w[x,y]) >_{\dot{\mathcal{J}}} \dot{J}$ using the induction hypothesis C1 for all $\dot{K} >_{\dot{\mathcal{J}}} \dot{J}$. □

Lemma 13 (Induction step for C2). *For every guard $\alpha(x,y)$ such that $w \vDash \alpha(x,y)$ implies $h(w[x,y]) \geq_{\dot{\mathcal{J}}} \dot{J}$, one can compute the orbits under α.*

Proof. Of course, it is sufficient to prove the lemma for the case when $w \vDash \alpha(x,y)$ implies $h(w[x,y]) \in \dot{J}$, since using Claim C2 it is possible to compute the orbits in the other cases. Let $\alpha^{\min}_{\dot{J}}(x,y)$ be the formula expressing that $[x,y]$ is minimal such that $h(w[x,y]) \in \dot{J}$ (doable thanks to Lemma 12). From its definition, $\alpha^{\min}_{\dot{J}}$ is rigid.

Consider the formula $\beta(x,y,z_1,z_2,z_3,z_4)$ which holds if (i) $\alpha(x,y)$, (ii) $z_1 \geq x$ is minimal such that $\alpha^{\min}_{\dot{J}}(z_1,z_2)$, and (iii) $z_4 \leq y$ is maximal such that $\alpha^{\min}_{\dot{J}}(z_3,z_4)$. Remark first that since (i) implies $h(w[x,y]) \in \dot{J}$, all points z_1, z_2, z_3, and z_4 belong to $[x,y]$. Hence, β is a witnessing formula, and by Corollary 5, it is equivalent to a disjunction of rigid formulas, say $\beta_1, \dots, \beta_n$.

We describe below the steps performed by the formulas computing the orbits under α.

1. It detects for which i, $w \vDash \beta_i(x,y,z_1,z_2,z_3,z_4)$ holds, and guesses the corresponding variables z_1, z_2, z_3, z_4 (this implies in particular $w \vDash \alpha(x,y)$, hence $h(w[x,y]) \in \dot{J}$).
2. It computes the orbit of $h(w[x,z_2])$ (and $h(w[z_3,x])$ in a similar way). This is doable since:
   By definition of β, if $w \vDash \beta_i(x,*,y-1,*,*,*)$ then $h(w[x,y-1]) >_{\dot{\mathcal{J}}} \dot{J}$. Hence, using the induction hypothesis C2, one can compute the orbits under the guard $\beta_i(x,*,y-1,*,*,*)$. Furthermore by Lemma 11, one can compute the orbits under $\alpha^{\min}_{\dot{J}}$. Overall, one can compute the orbits under $\beta_i(x,*,y-1,*,*,*) \cdot \alpha_{\dot{J}}(y,z)$, i.e., under $\beta_i(x,*,*,y,*,*)$. This means that one can compute the orbit of $h(w[x,z_2])$.
3. Recall that in any aperiodic orbit finite data monoid all $\mathcal{H}$-classes are singletons. Since the considered data monoid is aperiodic, we have that $t = h(w[x,y])$ is the only element (up to $\approx$) such that $t \in \mathcal{R}(h(w[x,z_2])) \cap \mathcal{L}(h(w[z_3,y]))$. It follows that we know the orbit of $h(w[x,y])$. It remains to provide witnesses for the data values.
   Since $h(w[x,z_2]) \mathrel{\mathcal{J}} h(w[x,y])$ and $h(w[x,z_2]) \geq_{\mathcal{R}} h(w[x,y])$, we have that $h(w[x,z_2]) \mathrel{\mathcal{R}} h(w[x,y])$. In the symmetric way, $h(w[z_3,y]) \mathrel{\mathcal{L}} h(w[x,y])$. Hence, by Proposition 4, each memorable value from t occurs either in $h(w[x,z_2])$ or in $h(w[z_3,y])$. This means that every data value in t is already witnessed at step 2.

□

The above arguments prove Lemma 3 under the assumption that the orbit finite data monoid $\mathcal{M}$ is aperiodic. We observe that the lemma directly implies the part of Theorem 2 that deals with the aperiodic case:

Corollary 7. *Every data language recognized by a finite orbit aperiodic data monoid is definable by a rigidly guarded FO sentence.*

Proof. One should provide, given an orbit o, a formula which holds over a word w iff $h(w) \in o$. Other situations are obtained by disjunction of this single orbit case. Consider the guard $\alpha(x,y) = (\neg\exists\, z.\ z < x) \wedge (\neg\exists\, z.\ z > y)$. It holds iff x is the first position and y the last position of the word. Since by Claim C2 of Lemma 3 one can compute the orbits under α, the language $h^{-1}(o)$ is definable. □

C.3 The translation in the non-aperiodic case

We have seen in the previous section how to establish Theorem 2 in the aperiodic case. The other part, stating that every data language recognized by an orbit finite data monoid is definable in rigidly guarded MSO logic, is proved by following the same structure, i.e., by relying on the same induction on $\dot{\mathcal{J}}$-classes and on similar constructions. Only two things need to be changed. The first one is that Lemma 7 needs to be reproven for rigidly guarded MSO (doable with the exact same proof). The second issue lies in the proof of Lemma 13, when the hypothesis of aperiodicity is used for the first time, namely at step 3).

We fix for the rest of this section a (possibly non-aperiodic) orbit finite data monoid $\mathcal{M} = (M, \cdot, \hat{\ })$ over a set D of data values and a morphism h from the free data monoid $(D \times A)^*$ to $\mathcal{M}$. We tacitly assume that every 'formula' is a rigidly-guarded MSO formula.

The objective is to reprove Lemma 3, but this time without assuming that the underlying monoid is aperiodic. We note that the proof of Claim C1 given in Section C.2 does not exploit the assumption that the monoid is aperiodic (as far as the induction hypothesis is admitted). Hence we can reuse this part of the proof for the data monoid $\mathcal{M}$. It thus remains to prove Claim C2 for a given $\dot{\mathcal{J}}$-class $\dot{J}$.

To compute the orbits under a rigid guard $\alpha(x,y)$, we will split the infix between x and y into many pieces. That is, given an infix $w[x,y]$ of a word w, our formula will first guess a special partition of $w[x,y]$ into smaller factors $w_1, \dots, w_n$ that can be handled by the induction hypothesis, then it will perform sub-computations for the orbits of the factors, and finally check the orbit of the partial products $h(w_1) \cdot \ldots \cdot h(w_i)$, up to the orbit of the entire product $h(w[x,y])$. In what follows, $\dot{J}$ is a fixed $\dot{\mathcal{J}}$-class, and we assume that $h(w[x,y]) \in \dot{J}$.

The partitions we are interested in are the following ones (here, for simplicity, we assume that the whole word is concerned – this means that we do not have to bother with the extremities of the factor we want to check):

Definition 9. *A* partition *of a word w is a decomposition of w into $u_0, \dots, u_{n+1}$ where each u_i with $1 \le i \le n$ is non-empty. We say that a partition $u_0, \dots, u_{n+1}$ of a word w is an* $\dot{J}$-partition *if*

1. *$h(u_1 \cdot \ldots \cdot u_n) \in \dot{J}$, and*
2. *$h(u_i) \in \dot{J}$ for all $1 \le i \le n-1$.*

We say that a partition $u_0, \dots, u_{n+1}$ of a word w is a almost $\dot{J}$-partition *if*

1. *$h(u_1 \cdot \ldots \cdot u_n) \in \dot{J}$, and*
2. *either $h(u_i) \in \dot{J}$ or $h(u_{i+1}) \in \dot{J}$ for all $1 \le i \le n-1$.*

We want to use the induction hypothesis on the factors of the partition. Hence we consider "rigidly guarded partitions":

Definition 10. *A partition $u_0, \dots, u_{n+1}$ of w is* rigidly guarded *under a guard $\alpha(x,y)$ if*

1. *$w \vDash \alpha(x,y)$ where x is the first position of u_1 and y is the last position of u_n,*
2. *there exist finitely many rigid formulas $\alpha_1(x', y'), \dots, \alpha_m(x', y')$ such that for all $1 \leq i \leq n$, if x' is the position of the beginning of u_i, and y' its the position of the end of u_i, then $w \vDash \alpha_j(x', y')$ for some $1 \leq j \leq m$.*

It is orbit-computable under α *if, in addition, one can compute the orbits under each α_i.*

The following definition makes precise what it means for a partition to be represented in a rigidly guarded MSO logic.

Definition 11. *A* presentation *for a partition $u_0, \dots, u_{n+1}$ of w is a pair (X, Y) of monadic second-order variables such that*

1. *$x \in X$ iff there is some $1 \leq i \leq n$ such that x is the position of the beginning of u_i,*
2. *$y \in Y$ iff there is some $1 \leq i \leq n$ such that y is the position of the end of u_i.*

Lemma 14. *For every rigid guard $\alpha(x,y)$, there is a formula $\alpha_{almost}(X,Y)$ such that for all words w, $w \vDash \alpha_{almost}(X,Y)$ iff (X,Y) is a presentation for an almost-$\dot{J}$-partition of w. In addition, such a partition is orbit-computable under the guard α. Furthermore, for all words w, if $w \vDash \alpha(x,y)$, then there is a pair (X,Y) such that $w \vDash \alpha_{almost}(X,Y)$.*

Proof. The formula $\alpha_{\text{almost}}(X,Y)$ checks that the partition $u_0, \dots, u_{n+1}$ of w induced by (X,Y) is a special almost-$\dot{J}$-partition: If an interval u_i is in $\dot{J}$, then it must be a minimal interval in $\dot{J}$. More precisely $\alpha_{\text{almost}}(X,Y)$ checks that:

- $h(u_1 \cdot \ldots \cdot u_n) \in \dot{J}$. This can be done using the formula $\varphi_{\dot{J}}$ from Claim C1 of Lemma 3.
- For all $1 \leq i \leq n-1$, either $h(u_i) \in \dot{J}$ or $h(u_{i+1}) \in \dot{J}$. This can be checked by a combination of $\alpha^{\min}_{\not\geq_{\dot{\mathcal{J}}} \dot{J}}$ from Lemma 11 and of $\varphi_{\dot{J}}$ from Claim C1.
- For all $1 \leq i \leq n-1$, either u_i is a minimal interval in $\dot{J}$ or $h(u_i)$ is in a $\dot{\mathcal{J}}$-class above $\dot{J}$. The first case can be checked by $\alpha^{\min}_{\not\geq_{\dot{\mathcal{J}}} \dot{J}}$. The second case can be checked using the formulas $\varphi_{\dot{J}}$.

It follows that $w \vDash \alpha_{\text{almost}}(X,Y)$ iff (X,Y) is a presentation for an almost-$\dot{J}$-partition of w.

We now show how α_{almost} can check that the partition $u_0, \dots, u_{n+1}$ is rigidly guarded under α. Clearly it can check that $w \vDash \alpha(x,y)$, where x' is the first position of u_1 and y is the last position of u_n. To check the second condition of Definition 10 we define the following finite set $G^{\rightarrow}$ of formulas:

1. The formula $\alpha^{\min}_{\not\geq_{\dot{\mathcal{J}}} \dot{J}}(x', y')$ from Lemma 11 is in $G^{\rightarrow}$.

2. Recall that $\alpha(x, y)$ is a rigid guard. By the Sub-definability Lemma 7 there exist finitely many formulas $\beta_i(x', y')$ for α that determine y' from x'. We let all these formulas β_i be contained in $G^{\rightarrow}$.
3. Let $\beta_1^{\leftarrow}(y', z'), \dots, \beta_{k'}^{\leftarrow}(y', z')$ be the formulas obtained from the rigid formula $\alpha^{\min}_{\ngeq_{\dot{\mathcal{J}}} J}(x', z')$ using again Lemma 7 (we consider here the version of the lemma for formulas that determine y' from z'). $G^{\rightarrow}$ contains all formulas of the form
$$\beta_j^{\rightarrow}(x', y') \;=^{\text{def}}\; \exists\, z'.\, \alpha^{\min}_{\ngeq_{\dot{\mathcal{J}}} J}(x', z') \,\wedge\, \beta_j^{\leftarrow}(z', y').$$

It is easy to check that each $\gamma^{\rightarrow}(x', y') \in G^{\rightarrow}$ determines y' from x'. We claim that for all $1 \leq i \leq n$, if x' is the position of the beginning of u_i and y' its the position of the end of u_i, then there is a formula $\gamma^{\rightarrow}(x', y') \in G^{\rightarrow}$ such that $w \models \gamma^{\rightarrow}(x', y')$. We distinguish three different cases:

i) Case: u_i is a minimal interval in $\dot{J}$. In this case the formula $\alpha^{\min}_{\ngeq_{\dot{\mathcal{J}}} J}(x', y')$ has the desired properties.

ii) Case $i = n$. Then there is a formula among the $\beta_i(x', y')$ with the desired properties.

iii) Otherwise u_i is an interval in a $\dot{J}$-class above $\dot{J}$, and $1 \leq i \leq n - 1$. In this case we claim that there is an index j such that $\beta_j^{\rightarrow} \in G^{\rightarrow}$ is the suitable formula: Assume that x' is the position at the beginning of u_i and that y' is the position at the end of u_i. Then $\alpha^{\min}_{\ngeq_{\dot{\mathcal{J}}} J}(x', z')$ determines some z' from x' such that $w[x', y']$ is a minimal interval in $\dot{J}$. As u_i is in a higher $\dot{J}$-class than $\dot{J}$ and $1 \leq i \leq n - 1$, it follows that z' must be in u_{i+1}. As $u_1, \dots, u_n$ is an almost-$\dot{J}$ partition, u_{i+1} must be a minimal interval in $\dot{J}$. We observed before that these intervals are rigidly guarded by $\alpha^{\min}_{\ngeq_{\dot{\mathcal{J}}} J}(x', y')$. By the Sub-definability Lemma 7, there must be a formula $\beta_j^{\rightarrow}(x', y')$ that determines y' from x'.

Symmetrically, one can define a set $G^{\leftarrow}$ with the respective properties for going from right to left. We then define

$$G \;=^{\text{def}}\; \{\gamma^{\rightarrow}(x', y') \,\wedge\, \gamma^{\leftarrow}(x', y') \;:\; \gamma^{\rightarrow} \in G^{\rightarrow},\ \gamma^{\leftarrow} \in G^{\leftarrow}\}.$$

Clearly each formula in G is rigid (possibly unsatisfiable). It follows from the above observation that G satisfies the second condition of Definition 10.

Let us sum up what we have proven so far: The partition $u_0, \dots, u_{n+1}$ of w induced by (X, Y) is a special almost-$\dot{J}$-partition where each interval is either in a $\dot{\mathcal{J}}$-class above $\dot{J}$, or it is a minimal interval in $\dot{J}$. In addition, the partition is guarded by G. Hence we can compute the orbit of each of the u_i under the respective guard by using Lemma 11 and the induction hypothesis. This means that $u_0, \dots, u_{n+1}$ is orbit-computable.

What remains to be done is to show that such a partition exists. Consider an almost-$\dot{J}$-partition $u_0, \dots, u_{n+1}$ such that for every $1 \leq i \leq n - 1$, either u_i or u_{i+1} is minimal in $h^{-1}(\dot{J})$. If there exists still some u_i such that $h(u_i) \in \dot{J}$, but which is not minimal, then such u_i can be decomposed into $vv'v''$, where v' is minimal such that $h(v') \in \dot{J}$ and either v or v'' is non-empty. We substitute in the partition $vv'v''$ for u_i, yielding a finer partition which still satisfies the property. This refinement process is iterated, starting from the trivial partition of w into a single word, and it stops when there are no more non-minimal u_i's such that $h(u_i) \in \dot{J}$. If no extra refinement steps are possible, this means that the expected form for the partition has been obtained. □

Lemma 15. *For every rigid guard $\alpha(x,y)$, there is a formula $\alpha_{partition}(X,Y)$ that defines rigidly guarded $\dot{J}$-partitions orbit-computable under $\alpha(x,y)$. Furthermore, for all words w, if $w \vDash \alpha(x,y)$, then there is at least one pair (X,Y) such that $w \vDash \alpha_{partition}(X,Y)$.*

Proof. The formula $\alpha_{\text{partition}}(X,Y)$ starts by guessing two sets X' and Y' such that $\alpha_{\text{almost}}(X',Y')$. Let also $(\alpha_i(x',y'))_{1\leq i\leq k}$ be the rigid guards for the partition. This corresponds to an almost-$\dot{J}$-partition of w into $u_0, u_1, \ldots, u_n, u_{n+1}$.

For the sake of simplicity we assume that n is even (otherwise, the $(n+1)$-th factor has to be treated separately). In this case, one considers a new partition of w into $u_0, v_1 \ldots v_m, u_{n+1}$, where $m = n/2$, $v_1 = u_1u_2$, $v_2 = u_3u_4$, …, $v_m = u_{n-1}u_n$. It is not difficult to define the new partition on the basis of the former one using MSO formulas. Let (X,Y) be the presentation of the new partition $u_0, v_1, \ldots, v_m, u_{n+1}$ of w.

It is also easy to see that, since $v_i = u_{2i-1}u_{2i}$ for all $1 \leq i \leq m$ and $u_1, \ldots, u_n$ is an almost-$\dot{J}$-partition, either $h(u_{2i-1}) \in \dot{J}$ or $h(u_{2i}) \in \dot{J}$. In both cases we get $h(v_i) \in \dot{J}$. It thus follows that $v_1, \ldots, v_n$ is a $\dot{J}$-partition.

What remains to be done is to prove that this partition is rigidly guarded and that one can compute the orbits under α. However, this is very simple. Indeed, since each the u_i's are rigidly guarded by one of the α_i's, each of the v_i's is rigidly guarded by one of the $\alpha_i \cdot \alpha_j$ for some $1 \leq i, j \leq k$. One can also compute the orbits using Lemma 8. □

At this point we know how the formula will be looking for a witness of the value of $h(w[x,y])$ by guessing a partition $u_0, u_1, \ldots, u_n, u_{n+1}$ such that $h(w[x,y] = h(u_1 \ldots u_n)$ using Lemma 15. It remains to develop a tool for being able to evaluate $h(w[x,y]) = h(u_1 \ldots u_n)$, knowing $h(u_1), \ldots, h(u_n)$. Of course, this cannot be done by comparing all the data values involved in $h(u_1), \ldots, h(u_n)$. There are too many of them, and it is impossible to do it using rigidly guarded tests. We show below that it is sufficient to perform some sort of an approximation of this computation. The necessary machinery is defined below.

Definition 12. *Two sequences of terms $t_1, \ldots, t_n$ and $t'_1, \ldots, t'_n$ are* locally consistent *if*

- *for all $1 \leq i \leq n$, both terms t_i and t'_i are in $\dot{J}$, and both products $t_1 \cdot \ldots \cdot t_n$ and $t'_1 \cdot \ldots \cdot t'_n$ are in $\dot{J}$ as well,*
- *for all $1 \leq i < n$, there is a renaming π_i such that $\hat{\pi}_i(t_i) = t'_i$ and $\hat{\pi}_i(t_{i+1}) = t'_{i+1}$,*
- *$t_1 = t'_1$ and $t_n = t'_n$. The sequences are called* almost locally consistent *if, instead of $t_1 = t'_1$ and $t_n = t'_n$, there is a renaming τ such that $\hat{\tau}(t_1) = t'_1$, and $\hat{\tau}(t_n) = t'_n$.*

The following lemma shows why we are interested in locally consistent sequences:

Lemma 16. *Let $t_1, \ldots, t_n$ and $t'_1, \ldots, t'_n$ be two sequences of terms.*

1. *If $t_1, \ldots, t_n$ and $t'_1, \ldots, t'_n$ are locally consistent, then $t_1 \cdot \ldots \cdot t_n = t'_1 \cdot \ldots \cdot t'_n$.*
2. *If $t_1, \ldots, t_n$ and $t'_1, \ldots, t'_n$ are almost locally consistent, then $\hat{\tau}(t_1 \cdot \ldots \cdot t_n) = \hat{\tau}(t'_1 \cdot \ldots \cdot t'_n)$ for some renaming τ.*

Proof. To prove the lemma, we need to introduce further definitions. We denote the arity of a term t by $\text{arity}(t)$. The *domain* $\mathcal{D}$ of a sequence of terms $t_1, \ldots, t_n$ is the set of pairs (i, k), where $i \le n$ specifies the term t_i, and $k \le \text{arity}(t_i)$ specifies a placeholder for a data value in t_i. We equip the domain $\mathcal{D}$ of a sequence $\bar{t}$ of terms with an equivalence relation E, that is the finest equivalence such that $(i, k) \in \mathcal{D}$ is equivalent to $(i+1, k') \in \mathcal{D}$ whenever the k-th memorable value of t_i and the k'-th memorable value of t_{i+1} share the same data value. Of course, all elements in an equivalence class have the same associated data value. A *coloring* of $(\mathcal{D}, E)$ is a labeling of the equivalence classes of E by data values. The sequence $\bar{t}$ naturally defines a coloring of $(\mathcal{D}, E)$ by associating to each equivalence class the data value of its elements. An element (i, k) in $\mathcal{D}$ is a *border position* if $i = 1$ or $i = n$. Two colorings are *border-equal* if they agree on all border positions. A *border class* is an equivalence class that contains a border position. We then establish the following claims:

> ***Claim 1.*** *If two sequences of terms are locally consistent, then they define the same domain $\mathcal{D}$ and the same equivalence E, and, furthermore, the corresponding colorings are border-equal.*
>
> *Proof of Claim 1.* Let $\bar{t}$ and $\bar{t}'$ be locally consistent sequences of terms. It is obvious that $\bar{t}$ and $\bar{t}'$ have the same domain and it is equally trivial that their colorings are border-equal. In addition, the equivalence between two positions (i, k) and $(i+1, k')$ only depends on the equalities between the data values in t_i and t_{i+1}. Those equalities are the same in t_i and t_{i+1} as in t'_i and t'_{i+1} as there is a data renaming π such that $\hat{\pi}(t_i) = t'_i$ and $\hat{\pi}(t_{i+1}) = t'_{i+1}$. □

Two colorings have a *small difference* if their domains and equivalences are the same and they disagree on the color of at most one equivalence class. Two locally consistent sequences have a small difference if their colorings have small difference.

> ***Claim 2.*** *Two locally consistent sequences of terms that have a small difference have the same value.*
>
> *Proof of Claim 2.* Let $t_1, \ldots, t_n$ and $t'_1, \ldots, t'_n$ be locally consistent sequences of terms that have a small difference. Let C be the equivalence class for which the two colorings differ. Let i be the smallest number such that there is $(i, k) \in C$ and let j be the biggest number

such that $(j, k') \in C$ for some $k, k' \in \mathbb{N}$. As $t_1, \ldots, t_n$ and $t'_1, \ldots, t'_n$ are locally consistent, it follows that $1 < i$ and $j < n$. Let d be the color of C in $t_1, \ldots, t_n$, and d' be the color of C in $t'_1, \ldots, t'_n$. We define τ to be the permutation that swaps the data values d and d' and is the identity elsewhere. We observe that:

i) $\tau \circ \tau$ is the identity.

ii) $\hat{\tau}(t_i) = t'_i$ for all $i \in \{i, \ldots, j\}$.

iii) $t_i = t'_i$ for all $i \in \{1, \ldots, i-1\} \cup \{j+1, \ldots, n\}$.

iv) $\hat{\tau}(s') = s'$ for $s' = t'_{i-1} \cdot \ldots \cdot t'_{j+1}$. This is because neither d nor d' are memorable in s': by Proposition 4 every value is either an $\mathcal{L}$-memorable value or a $\mathcal{R}$-memorable value. As $t'_{i-1}, \ldots, t'_{j+1}$ are all in the same $\dot{\mathcal{J}}$-class, it follows that all $\mathcal{R}$-memorable values in s' must occur in t'_{i-1} and all $\mathcal{L}$-memorable values in s' must occur in t'_{j+1}. As neither d nor d' occur in either t'_{i-1} or t'_{j+1} it follows that d and d' are not memorable in s'.

The above remarks allow us to conclude the proof of Claim 2:

$$\begin{aligned}
t_1 \cdot \ldots \cdot t_n &= t_1 \cdot \ldots \cdot t_{i-2} \cdot \hat{\tau}(\hat{\tau}(t_{i-1}) \cdot \ldots \cdot \hat{\tau}(t_{j+1})) \cdot t_{j+2} \cdot \ldots \cdot t_n \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(by (1))} \\
&= t_1 \cdot \ldots \cdot t_{i-2} \cdot \hat{\tau}(t'_{i-1} \cdot \ldots \cdot t'_{j+1}) \cdot t_{j+2} \cdot \ldots \cdot t_n \qquad \text{(by (2))} \\
&= t'_1 \cdot \ldots \cdot t'_{i-2} \cdot \hat{\tau}(t'_{i-1} \cdot \ldots \cdot t'_{j+1}) \cdot t'_{j+2} \cdot \ldots \cdot t'_n \qquad \text{(by (3))} \\
&= t'_1 \cdot \ldots \cdot t'_{i-2} \cdot t'_{i-1} \cdot \ldots \cdot t'_{j+1} \cdot t'_{j+2} \cdot \ldots \cdot t'_n \qquad \text{(by (4))}
\end{aligned}$$

□

We can finally turn to the main proof. Consider two locally consistent sequences $\bar{t}$ and $\bar{t}'$. By Claim 1, $\bar{t}$ and $\bar{t}'$ define the same domain $\mathcal{D}$, and the same equivalence E, and the corresponding colorings are border-equal. We show that one can transform $\bar{t}$ into $\bar{t}'$ by steps of small-difference. Of course, using Claim 2, this would conclude the proof.

It is sufficient to describe how the coloring evolves at each step. Let m be the number of non-border equivalence classes, and let $d_1, \ldots, d_m$ be fresh values (appearing neither in $\bar{t}$ nor in $\bar{t}'$). Transforming $\bar{t}$ to $\bar{t}'$ is done in two phases. One first performs the following m-steps: at step i, for $i = 1, \ldots, m$, one recolors the i-th equivalence class by d_i. One then performs other m-steps during which one recolors the i-th equivalence class, for each $i = 1, \ldots, m$, by its color in $\bar{t}'$. This concludes the proof of the first part of the lemma.

The second part of the lemma, dealing with an almost locally consistent sequence of terms, just follows from the fact that if $t_1, \ldots, t_n$ is almost locally consistent with $t'_1, \ldots, t'_n$, then there exist a renaming τ and a sequence $t''_1, \ldots, t''_n$ of terms such that $t_1, \ldots, t_n$ is locally consistent with $t''_1, \ldots, t''_n$ and $t_1 \cdot \ldots \cdot t_n = \hat{\tau}(t''_1 \cdot \ldots \cdot t''_n)$. □

The general idea for proving the induction step of Claim C2 of Lemma 3 is that the formula that computes the orbits under a rigid guard α will guess a suitable $\dot{J}$-partition $u_0, u_1, \dots u_n, u_{n+1}$ of the underlying data word w. For evaluating the product $h(u_1) \cdot h(u_2) \cdot h(u_n)$, the formula will guess a sequence of terms $t_1 \dots t_n$ which is locally consistent with $h(u_1) \dots h(u_n)$. This may look difficult a priori, because there are infinitely many possible terms (as there are infinitely many data values) and an MSO formula would not be able to guess such a sequence. However, our last preparatory lemma shows that it suffices to considers sequences of terms built over a set of data values of bounded cardinality.

Lemma 17. *For each sequence of terms $t_1 \dots t_n$, there is a sequence of terms $t'_1 \dots t'_n$ that is locally consistent with $t_1 \dots t_n$ and uses at most $4\|\mathcal{M}\|$ distinct data values.*

Proof. We can reus the objects in the proof of Lemma 16, namely, we can view a sequence of terms as a domain, an equivalence relation, and a coloring. We can assume without loss of generality that:

- the set D of all data values is the set of positive integers,
- the data values in t_1 and in t_n belong to the set $[1, 2\|\mathcal{M}\|]$ (call this range of values the set of *border colors*),
- all other data values (i.e. those that to not appear either in t_1 or t_n) have color greater than $4\|\mathcal{M}\|$.

By using induction on $i = 2, \dots, n-1$, we transform the coloring associated with the sequence $t_1, \dots, t_n$ into a coloring where all positions up to position i in the domain have data values smaller than or equal to $4\|\mathcal{M}\|$. Such a transformation is perfmormed as follows. If there is a color at position $i+1$ that is greater than $4\|\mathcal{M}\|$, we recolor all the data values appearing at position $i+1$ that do not belong to $[1, 4\|\mathcal{M}\|]$ by arbitrarily chosen new colors among $[2\|\mathcal{M}\|+1, 4\|\mathcal{M}\|]$. Note that we can do that, since there are at most $\|\mathcal{M}\|$ data values involved in the term t_i and hence there are always at least $\|\mathcal{M}\|$ fresh data values from $[2\|\mathcal{M}\|+1, 4\|\mathcal{M}\|]$ that we can choose.

At the end of the transformation, every position uses only data values among $[1, 4\|\mathcal{M}\|]$, and the coloring is still valid. This means that the resulting sequence $t'_1, \dots, t'_n$ of terms fulfills the concludsions of the lemma. □

We can prove the induction step for Claim C2 of Lemma 3.

Lemma 18 (Induction step for C2). *For every guard $\alpha(x,y)$ such that $w \vDash \alpha(x,y)$ implies $h(w[x,y]) \geq_{\dot{\mathcal{J}}} \dot{J}$, one can compute the orbits under α.*

Proof. The construction starts as in the proof of Lemma 13, but things change at point 3 since the construction there exploits aperiodicity. We continue the construction, without the assumption of aperiodicity, as follows.

3. One guesses a $\dot{J}$-partition which is rigidly guarded and orbit-computable using the formula $\alpha_{\text{partition}}(X, Y)$ of Lemma 15. We denote by $u_0, u_1, \dots, u_n, u_{n+1}$ this partition.

4. One guesses some terms $t_1, \ldots, t_n$ over a finite set data set C of size $4\|\mathcal{M}\|$. There are finitely many such terms, so this can be done using monadic quantifications. This must be done in such a way that the information concerning t_i is located on the factor u_i, say on the first letter. One then computes $t = t_1 \cdot \ldots \cdot t_n$. This can be done inductively using MSO since the partial products of the terms $t_1, \ldots, t_n$ range over a finite set.

5. One checks the almost local consistency between $t_1 \ldots t_n$ and $h(u_1) \ldots h(u_n)$. This is done as follows.

   Consider any index $1 \leq i \leq n-1$ (this amounts at using a universal first-order quantification). Let $\beta(x, y)$ and $\beta'(x', y')$ be the rigid guards for u_i and for u_{i+1} and let $t_i = o(d_1, \ldots, d_k)$ and $t_{i+1} = o'(d'_1, \ldots, d'_h)$ be two terms. One first guesses (using disjunctions) the rigid formula $\varphi(x, y, \bar{z})$ that witnesses the orbit of t_i over u_i, as well as the rigid formula $\varphi'(x', y', \bar{z}')$ that witnesses the orbit of t_{i+1} over u_{i+1}. One then checks that those formula indeed holds for some $\bar{z}$ and some $\bar{z}'$.

   Now, in order to check the almost local consistency it is sufficient that all the equality relations between the data values of t_i and the data values of t_{i+1} are satisfied. For this, one checks that for all values d_l in t_i and $d'_{l'}$ in t_{i+1}:

   - if $d_l = d'_{l'}$, then $(\exists\, y.\ \varphi(*, y, \bar{*}, z_l, \bar{*}) \ \wedge\ \varphi'(y+1, *, \bar{*}, z'_{l'}, \bar{*})) \ \wedge\ z_l \sim z'_{l'}$ holds (the guard is rigid since φ and φ' are rigid),
   - if $d_l \neq d'_{l'}$, then $(\exists\, y.\ \varphi(*, y, \bar{*}, z_l, \bar{*}) \ \wedge\ \varphi'(y+1, *, \bar{*}, z'_{l'}, \bar{*})) \ \wedge\ z_l \nsim z'_{l'}$ holds (same argument for the rigidity).

   Finally, one needs to check equalities between the data values of the first term t_1 and the last term t'_n. For this we reuse the same notation, namely, we assume that $t_1 = o(d_1, \ldots, d_k)$ and $t_n = o'(d'_1, \ldots, d'_h)$ and we check that:

   - if $d_l = d'_{l'}$, then $(\exists\, x, y.\ \varphi(x, *, \bar{*}, z_l, \bar{*}) \wedge \alpha(x, y) \wedge \varphi(*, y, \bar{*}, z'_{l'}, \bar{*})) \wedge z_l \sim z'_{l'}$ holds (rigidity holds as above, since α is rigid),
   - if $d_l \neq d'_{l'}$, then $(\exists\, x, y.\ \varphi(x, *, \bar{*}, z_l, \bar{*}) \wedge \alpha(x, y) \wedge \varphi(*, y, \bar{*}, z'_{l'}, \bar{*})) \wedge z_l \nsim z'_{l'}$ holds (same argument for the rigidity).

6. At this point, $h(w)$ has to be in the same orbit as t. What remains to be done is witness the data values. For this, once more one uses Proposition 4. Since $h(w)\ \dot{\mathcal{J}}\ h(u_1)$, and $h(w) \leq_{\dot{\mathcal{J}}} h(u_1)$, we have $h(w)\ \dot{\mathcal{R}}\ h(u_1)$. In the same way, we have $h(w)\ \dot{\mathcal{L}}\ h(u_n)$. This means that every memorable value of $h(w)$ is already present either in $h(u_1)$ or in $h(u_n)$. Hence, the formula can locate in a rigid way the positions of the memorable values of $h(u_1)$ and $h(u_n)$. If $t_1 \cdot \ldots t_n$ contains a data value d and d occurs in t_1, then the corresponding position in u_1 that witnesses the memorable value d is used, and symmetrically for u_n.

□

D Proofs for Section 6 (Logics for finite memory automata)

Before turning to the proofs of the results about finite memory automata, we describe an equiavalent variant of these automata from [2]. Hereafter, given a sequence of data values $u = d_1 \dots d_k$ from an infinite set D and a set $I \subseteq \mathcal{D}om(u)$ of positions in u, we denote by $u|_I$ the sub-sequence obtained from u by selecting the positions in I. Moreover, we denote by $[u]$ the *isomorphism type* of u, namely, the class of all data words u' for which there is a renaming $\tau \in G_D$ such that $\tau(u') = u$.

Definition 13. *A* (non-deterministic) finite-memory automaton *(FMA) is a tuple of the form $\mathcal{A} = (D, A, k, Q_0, \dots, Q_k, T, I, F)$, where*

- *D is the infinite set of data values,*
- *A is the finite alphabet,*
- *k is the maximum number of stored values,*
- *$Q_0, \dots, Q_k$ are pairwise disjoint finite sets of control states,*
- *T is a finite set of transition rules of the form (p, a, θ, E, q), where $p \in Q_i$ for some $0 \le i \le k$, $a \in A$, θ is the isomorphism type of a data word of length $i+1$, $E \subseteq \{1, \dots, i+1\}$, and $q \in Q_j$, with $j = i + 1 - |E|$,*
- *$I \subseteq Q_0$ is a set of initial control states;*
- *$F \subseteq Q_0 \cup \dots \cup Q_k$ is a set of final control states.*

A *configuration* of $\mathcal{A}$ is defined as a pair of the form (q, r) consisting of a control state $q \in Q_i$, with $0 \le i \le k$, and a memory content $r \in D^i$. The meaning of a transition rule of the form (p, a, θ, E, q) is that the automaton can move from a configuration (p, r) to a configuration (q, s) by consuming a pair $(d, a) \in D \times A$ iff the word rd has isomorphism type θ and s is obtained from rd by removing all positions in E.

We enforce two sanity conditions to every transition rule (p, a, θ, E, q):

1. To guarantee that the length of the target memory content s never exceeds k, we assume that the set E is non-empty whenever $q \in Q_k$.
2. The memory is updated like a stack: if the isomorphism type θ is of the form $[rd]$, with $r(j) = a$ for some $1 \le j \le |r|$, then E must contain the index j.

Note in particular that the second sanity condition has two advantages: the content s of the target memory always contains pairwise distinct data values and, moreover, the order of the data values in the memory is the order of their last occurrences in the consumed input word.

The notion of (successful) *run* of an FMA $\mathcal{A}$ and the notion of *recognized language* $\mathscr{L}(\mathcal{A})$ are defined in the usual way. Finally, we say that an FMA $\mathcal{A} = (D, A, k, Q_0, \dots, Q_k, T, I, F)$ is *deterministic* if (i) the set of initial states I is a singleton $\{q_0\}$ and (ii) there is no pair of transitions $(p, \theta, E, q), (p, \theta, E', q') \in T$ with $q \neq q'$ or $E \neq E'$. Similarly, $\mathcal{A}$ is said to be *complete* if for every control state

$q \in Q_i$ and every isomrphism type θ with $i+1$ positions, T contains a transition rule of the form (q, θ, E, q').

We divide the proof of Proposition 2 into two lemmas, one dealing with the translation of backward-rigidly guarded MSO formulas into equivalent deterministic FMA, and the other one dealing with the counterexample for the converse translation.

Lemma 19. *Every language definable in backward-rigidly guarded MSO is recognizable by deterministic FMA.*

Proof. This proof is based on a technique similar to the proof of Theorem 1. Specifically, we want to exploit closure properties of (suitable forms of) deterministic FMA under complementation (corresponding to negations of formulas), intersection (corresponding to conjunctions of formulas), and projection (corresponding to existential quantifications).

Let us first discuss the operation of projection on deterministic FMA. In general, this operation maps a deterministic FMA to a non-deterministic FMA. Moreover, unlike classical finite automata, arbitrary non-deterministic FMA cannot be determinized: the standard subset construction might turn a non-deterministic FMA into one with an infinite state space (note that the same issue occurs when projecting from an orbit finite data monoid). We solve this problem by restricting to a special form of deterministic FMA, which is similar to the notion of projectability for data monoid morphisms given (see Section 4).

As usual, A denotes a finite set of symbols, B denotes the binary alphabet $\{0,1\}$ (which is used to encode valuations of first-order and monadic second-order variables), and D denotes an infinite set of data values. In the following definition, we denote by $\mathcal{A}(w)$ the configuration reached by the automaton $\mathcal{A}$ after consuming w.

Definition 14. *An FMA $\mathcal{A}$ over the alphabet $D \times A \times B^m$ is* projectable *if (i) it is deterministic and complete and for all data words $w \in (D \times A)^*$ and (ii) for all tuples of predicates $\bar{U} = (U_1, \ldots, U_m)$ and $\bar{V} = (V_1, \ldots, V_m)$, if the configurations $\mathcal{A}(\langle w, \bar{U} \rangle)$ and $\mathcal{A}(\langle w, \bar{V} \rangle)$ have the same control state, then they coincide.*

We now prove the relevant closure properties of projectable FMA.

Claim 1. *Projectable FMA are closed under projection.*

Proof of Claim 1. Let us consider a projectable FMA $\mathcal{A} = (D, A \times B^{m+1}, k, Q_0, \ldots, Q_k, T, I, F)$. We need to construct a projectable FMA that recognizes the language

$$L' \overset{\text{def}}{=} \left\{ \langle w, \bar{U} \rangle \;:\; \exists\, U_{m+1} \subseteq \mathcal{D}om(w).\ \langle w, \bar{U}, U_{m+1} \rangle \in \mathscr{L}(\mathcal{A}) \right\}$$

where $\bar{U} = U_1, \ldots U_m$. We now define $\mathcal{A}'$ to be the FMA $(D, A \times B^m, k', Q'_0, \ldots, Q'_h, T', I', F')$, where

- $k' = k \cdot |Q_0 \cup \ldots \cup Q_k|$, namely, $\mathcal{A}'$ uses up to k registers for each state of $\mathcal{A}$;

- for all $0 \leq i \leq k'$, the set Q'_i contains all partial functions f from $Q_0 \cup \ldots \cup Q_k$ into the powerset of $\{1, \ldots, i\}$ such that

  1. if $q \in Q_j$ and $f(q)$ is defined then $|f(q)| = j$;
  2. $\bigcup_{q \in \mathcal{D}om(f)} f(q) = \{1, \ldots, i\}$.

  Intuitively, a configuration (f, r) of $\mathcal{A}'$, with $f \in Q'_i$ and $r \in D^i$, represents the set of all configurations of $\mathcal{A}$ of the form $(q, r|_{f(q)})$ with $q \in \mathcal{D}om(f)$.

- The transition relation T' is defined in such a way that whenever $\mathcal{A}'$ is in a configuration (f, r) and it reads $(d, a, b_1, \ldots, b_m)$, then it can move to any configuration (g, s) such that

  1. if a state q of $\mathcal{A}$ is in the domain of g, then there is a symbol $b_{m+1} \in \{0, 1\}$ and a control state $p \in \mathcal{D}om(f)$ such that $\mathcal{A}$ moves from configuration $(p, r|_{f(p)})$ to configuration $(q, s|_{g(q)})$ when reading $(d, a, b_1, \ldots, b_m, b_{m+1})$;
  2. if a state q of $\mathcal{A}$ is not in the domain of g, then there is no symbol $b_{m+1} \in \{0, 1\}$, no control state $p \in \mathcal{D}om(f)$, and no set $I \subseteq \{1, \ldots, k'\}$ such that $\mathcal{A}$ moves from $(p, r|_{f(p)})$ to $(q, s|_I)$ when reading $(d, a, b_1, \ldots, b_m, b_{m+1})$.

  We observe that, according to the above explanation, whether $\mathcal{A}'$ moves from a configuration (f, r) to a configuration (g, s) by reading $(d, a, b_1, \ldots, b_m)$ depends only on the control states f and g and on the *equality relationships* between the value d and the values in r and s. In particular, it depends neither on the concrete data value d, nor on the memory contents r and s. For this reason, one can turn the above description into a formal definition for the transition relation T' that uses isomorphism types and that satisfies the sanity conditions introduced at the beginning of this section (we omit the tedious details of such a definition). Below, we verify that these transition rules are deterministic when restricted to reachable control states.

- I' is the partial function that maps every initial state in I of $\mathcal{A}$ to the empty set and that is undefined on all other states;

- F' is the set of all partial functions $f \in Q'_0 \cup \ldots \cup Q'_{k'}$ such that $\mathcal{D}om(f) \cap F \neq \varnothing$.

It is routine to verify that the (non-deterministic) FMA $\mathcal{A}'$ recognizes the language L' above.

We now argue that $\mathcal{A}'$ can be turned into a projectable FMA by simply pruning the unreachable control states (this can be done by a simple reachability analysis, see for instance [2]). In the following we assume that $\mathcal{A}'$ contains no unreachable states.

We start by showing that $\mathcal{A}'$ is deterministic. Assume towards a contradiction that $\mathcal{A}'$ contains transitions $(f, (a, b_1, \ldots, b_m), \theta, E, g)$ and $(f, (a, b_1, \ldots, b_m), \theta, E', g')$ such that $g = g'$ or $E \neq E'$. We first show that in fact g must be equal to g'. As we assumed that $\mathcal{A}'$ contains only reachable states, there must be a data word $\langle w, U_1, \dots, U_m \rangle \in (D \times A \times B^m)^*$ such that

$$(f, r) \ \in \ \mathcal{A}'\big(\langle w, U_1, \dots, U_m \rangle\big).$$

Let d be a data value such that $(d, a, b_1, \dots, b_m)$ activates the two transitions above. Then there are register assignments s and s' such that

$$(g, s), (g', s') \ \in \ \mathcal{A}'\big(\langle w, U_1, \dots, U_m \rangle \cdot (d, a, b_1, \dots, b_m)\big).$$

Let q be any state in the domain of g. This means that there is a symbol $b_{m+1} \in \{0, 1\}$ and a control state p in the domain of f such that $\mathcal{A}$ moves from $(p, r|_{f(p)})$ to $(q, s|_{g(q)})$ when reading $(d, a, b_1, \dots, b_m, b_{m+1})$. Of course, we can say the same for g' and hence $g'(q)$ must be defined as well.

We know from the definition of $\mathcal{A}'$ that for every state in the domain of g, and in particular for the state q, there is a unary predicate $U_{m+1} \subseteq \mathcal{D}om(w)$ and a symbol $b_{m+1} \in \{0, 1\}$ such that $\mathcal{A}$ reaches configuration $(q, s|_{g(q)})$ after first parsing $\langle w, U_1, \dots, U_m, U_{m+1} \rangle$ and then parsing $(d, a, b_1, \dots, b_m, b_{m+1})$, namely,

$$\mathcal{A}\big(\langle w, U_1, \dots, U_m, U_{m+1} \rangle \cdot (d, a, b_1, \dots, b_m, b_{m+1})\big) \ = \ (q, s|_{g(q)}).$$

The same arguments apply to the configuration (g', s'), which prove the existence of a unary predicate $U'_{m+1} \subseteq \mathcal{D}om(w)$ and a symbol $b'_{m+1} \in \{0, 1\}$, such that

$$\mathcal{A}\big(\langle w, U_1, \dots, U_m, U'_{m+1} \rangle \cdot (d, a, b_1, \dots, b_m, b'_{m+1})\big) \ = \ (q, s'|_{g'(q)}).$$

Since $\mathcal{A}$ is projectable, we have $s|_{g(q)} = s'|_{g'(q)}$. Moreover, we know from the sanity conditions that both s and s' are obtained from the same memory content r by first appending the input value d and then selecting some sub-sequences. Since r has no repetitions of the same data value and since $s|_{g(q)} = s'|_{g'(q)}$, we must have $g(q) = g'(q)$. We have just shown that $g = g'$.

Let us finally consider the memory contents s and s'. From the definition of $\mathcal{A}'$ we know that both s and s' contain no repeated data values and, moreover, $\bigcup_{q \in \mathcal{D}om(g)} g(q) = \{1, \dots, |s|\}$. As $g = g'$ and $s|_{g(q)} = s'|_{g(q)}$ for all $q \in \mathcal{D}om(g)$ it follows that $s = s'$. Therefore E must be equal to E'. □

Summing up, we proved that the restriction of $\mathcal{A}''$ to the set of reachable control states is a deterministic FMA. Similar arguments can be used to prove that this automaton is also complete and, moreover, projectable, which concludes the proof of closure under projections.

Claim 2. *Projectable FMA are closed under intersection and complementation.*

Proof (Sketch) of Claim 2. Closure under intersection and complementation of projectable FMA is proved, as in the classical case, by computing "products" of projectable FMA and by complementing the set of final control states, respectively. □

Intersection with non-projectable FMA. There is still one missing property that we need to verify, which is related to the translation of backward-rigidly guarded data tests of the form $\varphi(x,y) \wedge x \sim y$. For this, we inductively translate the backward-rigid guard $\varphi(x,y)$ into a projectable FMA $\mathcal{A}$ that recognizes the data language $[\![\varphi]\!]$ over the alphabet $D \times A \times B^2$. We also construct a deterministic (but not projectable) FMA $\mathcal{B}$ recognizing the language $[\![x \sim y]\!]$ over the same alphabet. To prove that the intersection language $[\![\varphi(x,y) \wedge x \sim y]\!]$ is recognized by a projectable FMA, we need to process $\mathcal{A}$ in order to enforce a stronger notion of projectability (called, as usual, 0-reduced projectability), which is compatible with intersections of non-projectable languages. Below, we say that a configuration of $\mathcal{A}$ is a 0-*configuration* if it has empty memory and all transitions on this configuration are self-loops.

Definition 15. *An FMA $\mathcal{A}$ over the alphabet $D \times A \times B^2$ is* 0-reduced projectable *if (i) it is projectable and (ii) for all data words $w \in (D \times A)^*$ and all positions $x, x', y \in \mathcal{D}om(w)$, if the two configurations $\mathcal{A}(\langle w, \{x\}, \varnothing\rangle)$ and $\mathcal{A}(\langle w, \{x'\}, \varnothing\rangle)$ coincide, then $x = x'$ or both configurations are a* 0*-configuration.*

We now prove the following:

Claim 3. *Let $\varphi(x,y)$ be a backward-rigid formula. Given a projectable FMA $\mathcal{A}$ that recognizes $[\![\varphi]\!]$, one can compute a* 0*-reduced projectable FMA $\mathcal{A}'$ that recognizes $[\![\varphi]\!]$.*

Proof of Claim 3. The 0-reduced projectable FMA $\mathcal{A}'$ is obtained from $\mathcal{A}$ by simply grouping together those control states from which the automaton cannot accept, no matter what it reads. We call these states *trap states*. Computing trap states can be done again by a simple reachability analysis. Formally, one defines $\mathcal{A}'$ as the FMA obtained from $\mathcal{A}$ by introducing a new control state q_{sink}, with memory size 0, by redirecting to q_{sink} all the transitions that reach some trap state (erasing at same time the entire memory), and finally pruning the trap states. Clearly, the thus defined automaton $\mathcal{A}'$ is equivalent to $\mathcal{A}$. Moreover, it is easy to see that is deterministic, complete, and projectable.

We need to prove that $\mathcal{A}'$ is 0-reduced projectable. For this we use the fact that the automaton recognizes the language $[\![\varphi]\!]$, which is defined by the *backward-rigid* formula $\varphi(x,y)$. Let $w \in (D \times A)^*$ be a data word and let $x, x' \in \mathcal{D}om(w)$ be two positions in it. By way of contradiction, we assume that (i) $x \neq x'$ and (ii) the two configurations $\mathcal{A}'(\langle w, \{x\}, \varnothing\rangle)$ and $\mathcal{A}'(\langle w, \{x'\}, \varnothing\rangle)$ concide but they are not the 0-configuration $(q_{\text{sink}}, \varepsilon)$. From this we argue that $\varphi(x,y)$

is not backward-rigid. Since the configuration $\mathcal{A}'(\langle w, \{x\}, \varnothing\rangle)$ is not a 0-configuration, it can reach a final configuration, namely, there is an expanded data word $\langle v, U_1, U_2\rangle \in (D \times A \times B^2)$ such that

$$\langle w, \{x\}, \varnothing\rangle \cdot \langle v, U_1, U_2\rangle \in \mathscr{L}(\mathcal{A}').$$

Similarly, since $\mathcal{A}'(\langle w, \{x\}, \varnothing\rangle) = \mathcal{A}'(\langle w, \{x'\}, \varnothing\rangle)$, we have that

$$\langle w, \{x'\}, \varnothing\rangle \cdot \langle v, U_1, U_2\rangle \in \mathscr{L}(\mathcal{A}').$$

Since $x \neq x'$, this is against the backward-rigidity of the formula $\varphi(x, y)$ defining $\mathscr{L}(\mathcal{A}')$. This completes the proof of Claim 3. □

We now argue that the data language $[\![\varphi(x, y) \wedge x \sim y]\!]$ is recognized by a projectable FMA. We denote by $\mathcal{A}'$ a 0-reduced projectable FMA recognizing $[\![\varphi]\!]$. Moreover, we denote by $\mathcal{B}$ the "minimal" deterministic FMA that recognizes $[\![x \sim y]\!]$. The automaton $\mathcal{B}$ has four control states:

1. a control state q_{wait}, with empty memory, for parsing the prefix up to $\min(x, y)$,
2. a control state q_{check}, with memory size 1, for parsin the infix from $\min(x, y) + 1$ to $\max(x, y)$,
3. a final control state q_{accept}, with empty memory, for parsing the suffix from $\max(x, y) + 1$ to the end, in the case where the data test $x \sim y$ was successful,
4. a sink control state q_{reject}, with empty memory, for parsing the same suffix above, but in the case where the data test $x \sim y$ failed.

One can construct an FMA $\mathcal{C}$ recognizing $[\![\varphi(x, y) \wedge x \sim y]\!]$ using a suitable product between the 0-reduced projectable FMA $\mathcal{A}'$ and the deterministic (but not projectable) FMA $\mathcal{B}$ described above. This product is very similar to the 0-collapse product of orbit finite data monoids (see the proof of Lemma 2 in Section B). Intuitively, $\mathcal{C}$ mimics the computations of $\mathcal{A}'$ and $\mathcal{B}$ on the input word, but as soon as $\mathcal{A}'$ reaches the (unique) 0-configuration, the component that simulates the computation of $\mathcal{B}$ is reset to a special void configuration with empty memory. Of course the defined FMA $\mathcal{C}$ is deterministic and complete, as $\mathcal{A}'$ and $\mathcal{B}$ are so.

We verify that $\mathcal{C}$ is projectable. Consider a data word $w \in (D \times A)^*$ and some predicates $U_1, U_2, V_1, V_2 \subseteq \mathcal{D}om(u)$. Suppose that the configurations $\mathcal{C}(\langle u, U_1, U_2\rangle)$ and $\mathcal{C}(\langle u, V_1, V_2\rangle)$ contain the same control state. We consider also the configurations reached by $\mathcal{A}'$ after parsing $\langle u, U_1, U_2\rangle$ and $\langle u, V_1, V_2\rangle$. In particular, we distinguish between the case where $\mathcal{A}'(\langle u, U_1, U_2\rangle)$ is the 0-configuration (and hence $\mathcal{A}'(\langle u, V_1, V_2\rangle)$ as well) and the case where $\mathcal{A}'(\langle u, U_1, U_2\rangle)$ is not the 0-configuration (and hence $\mathcal{A}'(\langle u, V_1, V_2\rangle)$ as well). In the former case, we immediate obtain that the two configurations reached by $\mathcal{C}$ after parsing $\langle u, U_1, U_2\rangle$ and $\langle u, V_1, V_2\rangle$ must coincide (this is because the components of these two configurations that simulate $\mathcal{B}$ have been reset as $\mathcal{A}'$ has reached the 0-configuration). In the latter case, since $\mathcal{C}(\langle u, U_1, U_2\rangle)$ and $\mathcal{C}(\langle u, V_1, V_2\rangle)$ contain the same control state, we have the same for the configurations $\mathcal{A}'(\langle u, U_1, U_2\rangle)$ and $\mathcal{A}'(\langle u, V_1, V_2\rangle)$. Moreover, since $\mathcal{A}'$ is projectable, we get $\mathcal{A}'(\langle u, U_1, U_2\rangle) = \mathcal{A}'(\langle u, V_1, V_2\rangle)$. So the only way the configurations $\mathcal{C}(\langle u, U_1, U_2\rangle)$ and $\mathcal{C}(\langle u, V_1, V_2\rangle)$ can differ is in the components that simulate $\mathcal{B}$. Below, we analyse these components and we argue that they coincide:

1. If $U_1 = U_2 = \varnothing$, then $\mathcal{B}(\langle u, U_1, U_2\rangle) = (q_{\text{wait}}, \varepsilon)$. Moreover, since $\mathcal{C}(\langle u, U_1, U_2\rangle)$ and $\mathcal{C}(\langle u, V_1, V_2\rangle)$ share the same control state, the two components of $\mathcal{C}(\langle u, U_1, U_2\rangle)$ and $\mathcal{C}(\langle u, V_1, V_2\rangle)$ for simulating $\mathcal{B}$ encode the same control state of $\mathcal{B}$. In particular, it follows that $\mathcal{B}(\langle v, V_1, V_2\rangle)$ has control state q_{wait} and hence $\mathcal{B}(\langle u, U_1, U_2\rangle) = \mathcal{B}(\langle v, V_1, V_2\rangle)$.
2. If both U_1 and U_2 are non-empty, then $\mathcal{B}(\langle u, U_1, U_2\rangle)$ is either $(q_{\text{accept}}, \varepsilon)$ or $(q_{\text{reject}}, \varepsilon)$. Using arguments similar to the previous case, one obtains $\mathcal{B}(\langle u, U_1, U_2\rangle) = \mathcal{B}(\langle v, V_1, V_2\rangle)$.
3. If $U_1 \neq \varnothing$ and $U_2 = \varnothing$, then clearly U_1 is a singleton of the form $\{x\}$ and the configuration $\mathcal{B}(\langle u, U_1, U_2\rangle)$ contains the control state q_{check} and the data value at the position x of u. Similarly, since the configurations $\mathcal{B}(\langle u, U_1, U_2\rangle)$ and $\mathcal{B}(\langle u, V_1, V_2\rangle)$ share the same control state, we have that V_1 is a singleton of the form $\{x'\}$ and $V_2 = \varnothing$. Moreover, since the two configurations $\mathcal{A}'(\langle u, U_1, U_2\rangle)$ and $\mathcal{A}'(\langle u, U_1, U_2\rangle)$ coincide and they are different from the 0-configuration, we have $x = x'$. This implies $\mathcal{B}(\langle u, U_1, U_2\rangle) = \mathcal{B}(\langle v, V_1, V_2\rangle)$.
4. The last case $U_1 = \varnothing$ and $U_2 \neq \varnothing$ is similar to the previous one.

It follows that $\mathcal{C}(\langle u, U_1, U_2\rangle) = \mathcal{C}(\langle u, V_1, V_2\rangle)$. This shows that $\mathcal{C}$ is a projectable FMA.

Proof of the main lemma. As previously mentioned, the proof that any backward-rigidly guarded MSO formula φ can be translated into an equivalent deterministic (and projectable) FMA goes by structural induction on φ, using the closures of deterministic projectable FMA under projection, complement, intersection. For the backward-rigidly guarded data tests one uses the construction just mentioned above. □

Lemma 20. *There is a language recognized by a deterministic FMA which cannot be defined in backward-rigidly guarded MSO.*

Proof. Consider the language L'' of data words of the form

$$d_1 \; d_2 \; \ldots \; d_n$$

for which there is a sequence of indices $1 = i_1 < i_2 < \ldots < i_k = n$ satisfying $d_{i_j+1} = d_{i_{j+1}}$ for all $1 \leq j < k$ and $d_{i_j+1} \neq d_h$ for all $i_j + 1 < h < i_{j+1}$ [4].

It is easy to see that L'' is recognized by a deterministic FMA that uses only one register: at each phase, the automaton stores the value under the current position and then moves to the right searching for the another occurrence of the stored value – if it does not find such an occurrence, then it rejects, otherwise,

[4] This language is a variant of an example given in [?] to study data languages recognized by pebble automata.

as soon as the occurrence is found, the automaton moves to the next position to the right (if any, otherwise it accepts) and starts a new phase.

Below, we fix a generic formula φ of backward-rigidly guarded MSO and we show that it cannot define the language L''. For this, we let k be the number of occurrences of data comparisons of the form $\alpha(y,z) \wedge y \sim z$ in φ. We then consider a family of data words of the form

$$w_n \;=\; 0\; u_n^{(0)}\; 0 \;\; 1\; u_n^{(1)}\; 1 \;\; \ldots \;\; n\; u_n^{(n)}\; n$$

where n ranges over the positive natural numbers, each word $u_n^{(i)}$ has length exactly n, and the juxtaposition $u_n^{(0)} \cdot u_n^{(1)} \cdot \ldots \cdot u_n^{(n)}$ contains pairwise distinct values from the set $\mathbb{N} \smallsetminus \{0,\ldots,n\}$ (therefore, the only equalities that hold in w_n are those between the occurrences of the data values $0,\ldots,n$). Without loss of generality, we also assume that for every backward-rigidly guarded data test $\alpha(x,y) \wedge x \sim y$ in φ, the guard $\alpha(x,y)$ holds *only if* x is interpreted by a position to the left of the position that interprets y.

Using an encoding similar to the proof of Corollary 5, one can turn each data word w_n into a classical word w_n^* over the finite alphabet $\{0,1\}^k$ where every position y of w_n^* is labeled by a k-tuple of boolean values, each of them representing the "observable" equality between the value at the position y of w_n and the value at the unique position x determined from y via the backward-rigid guard of a corresponding data test $\alpha(x,y) \wedge x \sim y$ of φ. Accordingly, we can rewrite every sub-formula β of φ into a formula β^* of standard MSO by replacing each backward-rigid guard $\alpha(x,y)$ of a data test in β by a fresh unary predicate

$$[\alpha^{\sim}](y).$$

Clearly, we have that for every n, every backward-rigid guard $\alpha(x,y)$ of φ, and every position $x < |w_n|$ in w_n,

$$w_n \vDash \alpha(x,|w_n|) \qquad \text{iff} \qquad w_n^* \vDash \alpha^*(x,|w_n|).$$

Moreover, one can prove that there is a number n_α, which only depends on α, such that for all $n \in \mathbb{N}$ and all pairs of positions $1 \le x < y \le |w_n|$,

$$w_n^* \vDash \alpha^*(x,y) \qquad \text{implies} \qquad 1 \le x \le n_\alpha \;\vee\; y - n_\alpha \le x < y. \qquad\qquad (\star)$$

The proof of $\star$ is by structural induction on the number of nested rigid guards. The statement holds because the word w_n is over the alphabet D, that is, there are no labels from a finite alphabet that can be used to "rigidly jump" far from the original position y and from the endpoints of the word. We omit the details.

If we let $y = |w_n| = (n+1)^2$ in the above implication and we consider a sufficiently large n (i.e. $n > n_\alpha$ for all backward-rigid guards α of φ), we obtain that there is no backward-rigid guard $\alpha(x,y)$ in φ that, given the last position y in w_n, which carries the value n, determines the position x of the other occurrence of the value n in w_n. This shows that

$$w_n \vDash \varphi \qquad \text{iff} \qquad w_n^* \vDash \varphi^* \qquad \text{iff} \qquad w_n[|w_n| \mapsto n+1] \vDash \varphi$$

where $w_n[|w_n| \mapsto n+1]$ is the data word obtained from w_n by relabeling the last position with the fresh value $n+1$. Since $w_n \in L''$ and $w_n[|w_n| \mapsto n+1] \notin L''$, this shows that φ does not define the language L''. □

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

Proof. The first claim follows from Lemma 19. The second claim follows from Lemma 20. □

We now turn to the proof of the main theorem of Section 6.

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

Proof. Having established Lemma 19, the translation from an $\exists$backward-rigidly formula $\exists\, \bar{Z}.\, \varphi$ to an equivalent non-deterministic FMA that recognizes the language $L = [\![\exists\, \bar{Z}.\, \varphi]\!]$ is straightforward. For this, we first translate φ into a *deterministic* FMA $\mathcal{A}$ that recognizes the language $L' = [\![\varphi]\!]$ (note that this language is over the alphabet $D \times A \times B^m$, where $m = |\bar{Z}|$). After that, we project $\mathcal{A}$ onto the alphabet $D \times A$: the resulting automaton is a non-deterministic FMA that guesses the correct interpretation for the second-order variables $\bar{Z}$ on any input data word that belongs to the language $L = [\![\exists\, \bar{Z}.\, \varphi]\!]$.

We now prove the converse direction, that is, we translate a non-deterministic FMA $\mathcal{A} = (D, A, k, Q_0, \dots, Q_k, T, I, F)$ into an equivalent $\exists$backward-rigidly guarded MSO sentence. As in the classical theory, the proof is based on an encoding a successful run of $\mathcal{A}$ on the input word. For this we need a second-order variable Z_t for each transition rule $t \in T$ of $\mathcal{A}$. The formula that defines the recognized language $\mathscr{L}(\mathcal{A})$ begins with a block of existential monadic quantifiers of the form $\exists\, \bar{Z}.$, where $\bar{Z} = (Z_t)_{t \in T}$. The matrix of the desired formula (i.e. the part after the block of existential monadic quantifiers) is a backward-rigidly guarded MSO formula $\varphi(\bar{Z})$ that checks that the interpretations of the variables $\bar{Z}$) encode a valid run of $\mathcal{A}$.

Formally, given an input data word $w = (d_1, a_1) \dots (d_n, a_n)$, the formula $\varphi(\bar{Z})$ checks that:

1. For all positions $1 \le x \le |w|$, exactly one variable Z_t with $t \in T$ holds at x (i.e., $\forall\, x.\ \bigvee_{t \in T} x \in Z_t \wedge \bigvee_{t \neq t' \in T} x \notin Z_t \vee x \notin Z_{t'}$). We will say that a transition rule t is *encoded* at a position x if $x \in Z_t$ holds.
2. For all positions $1 \le x < |w|$, the target state of the transition rule encoded at position x coincides with the source state of the transition rule encoded at position $x+1$ (i.e., $\forall\, x.\ \bigvee_{t=(p,\theta,E,q), t'=(q,a,\theta',E',q') \in T} x \in Z_t \ \wedge\ x+1 \in Z_{t'}$).
3. The transition rule encoded at the first position of the word starts with an intial control state and the transition rule encoded at the last position of the word ends with a final control state.

4. For all positions $1 \leq x \leq |w|$, if $t = (p, a, \theta, E, q)$ is the transition rule encoded at position x, then $a = a_x$, where a_x is the symbol that appears at position x of the input word.
5. For all positions $1 \leq x \leq |w|$, if $t = (p, a, \theta, E, q)$ is the transition rule encoded at position x, then θ is the isomorphism type of $r_{x-1} \cdot d_x$, where r_x is the memory content at position $x-1$ in the encoded run of $\mathcal{A}$ and d_x is the data value at position x of the input word .
   To see how this check is done, consider an example of a partial run of $\mathcal{A}$:
   $$(q_0, r_0) \xrightarrow{(d_1,a_1)} (q_1, r_1) \xrightarrow{(d_2,a_2)} \dots \xrightarrow{(d_{x-1},a_{x-1})} (q_{x-1}, r_{x-1}).$$
   For each register i of r_{x-1}, we need to determine the "provenance" of the value $r_{x-1}[i]$ in the prefix $(d_1, a_1)(d_2, a_2)\dots(d_{x-1}, a_{x-1})$ of the input word, namely, we need to locate some position $y \leq x-1$ such that $d_y = r_{x-1}[i]$. We thus reconstruct, for each $z \leq x-1$, the position $f_{x,i}(z)$ of the occurrence of $r_{x-1}[i]$ (if any) in the memory r_z. This can be done by a suitable formula that first guesses some monadic variables $W_1, \dots, W_k$ encoding the partial function $f_{x,i}(z)$, and then verifies, using the run encoded by the variables $\bar{Z}$, that the guess is correct. One can also do this using only first-order quantifications, since the partial function $f_{x,i}(z)$ can only decrease as z increases; however, we prefer not to discuss this option in detail. We then compute the leftmost position y such that $f_{x,i}(z)$ is defined on all $z = y, \dots, x-1$. Clearly, this implies $d_y = r_{x-1}[i]$.
   The formula that determines y from x using $\bar{Z}$ is of the form
   $$\alpha_i(y, x, \bar{Z}) \;=^{\text{def}}\; \beta_i(y, x, \bar{Z}) \;\wedge\; \forall\, y'.\, y' < y \;\rightarrow\; \neg\beta_i(y', x, \bar{Z})$$
   where
   $$\begin{aligned}\beta_i(y, z, \bar{Z}) \;=^{\text{def}}\; & \exists\, W_1, \dots, W_k.\ \forall\, z. \\ & y \leq z \leq x-1 \;\rightarrow\; \Big(\bigvee_{1 \leq j \leq k} z \in W_j \;\wedge\; \neg \bigwedge_{1 \leq j < j' \leq k} z \in W_j \;\wedge\; z \in W_{j'} \\ & \qquad \wedge \bigvee_{\substack{1 \leq j \leq k,\ p \in Q_j \\ t=(p,a,\theta,E,q) \in T}} z \in Z_t \;\wedge\; z \in W_j \;\wedge\; z-1 \in W_{j-|E \cap \{1,\dots,j\}|} \Big)\end{aligned}$$
   Note that $\alpha_i(y, x, \bar{Z})$ is an ∃backward-rigid formula. We can thus use α_i as a guard to test equality between the data value d_x at position x and the data value d_i $(= r_{x-1}[i])$ at position y.
   Using the above constructions we can check that the isorphism type of the transition rule encoded at position x coincides with the isomorphism type of $r_{x-1} \cdot d_x$.

Thus $\varphi(\bar{Z})$ is a formula that verifies that the sequence of transition rules encoded by $\bar{Z}$ is a valid run of $\mathcal{A}$ on the input word. Hence $\exists\, \bar{Z}.\ \varphi(\bar{Z})$ is a ∃backward-rigidly guarded MSO sentence that defines the language $\mathscr{L}(\mathcal{A})$ recognized by $\mathcal{A}$. □

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 this case whether the formula is ∃backward-rigid.*

Proof. By Theorem 3, any formula of ∃backward-rigidly guarded MSO can be effectively transformed into a non-deterministic FMA. Satisfiability of ∃backward-rigidly guarded MSO then corresponds to non-emptiness of languages recognized by non-deterministic FMA. The latter problem is known to be decidable from [7].

The second claim follows from arguments similar to the proof of Corollary 2, that is, one can inductively check that every sub-formula satisfies the syntactical restrictions given by the grammar of ∃backward-rigidly guarded MSO. For the sub-formulas $\alpha(x, y, \bar{Z})$ that are guards of data equality tests, one has also to verify that α is ∃backward-rigid by checking the validity of the formula

$$\alpha^{\exists\text{backward?}} \overset{\text{def}}{=} \forall\, x, x', y, \bar{Z}.\ \alpha(x, y, \bar{Z}) \,\wedge\, \alpha(x', y, \bar{Z}) \,\rightarrow\, x = x'.$$

□