LOGICS WITH RIGIDLY GUARDED DATA TESTS

THOMAS COLCOMBET[a], CLEMENS LEY[b], AND GABRIELE PUPPIS[c]

[a] CNRS / LIAFA
   e-mail address: thomas.colcombet@liafa.univ-paris-diderot.fr

[b] Independent researcher
   e-mail address: ley.clemens@gmail.com

[c] CNRS / LaBRI
   e-mail address: gabriele.puppis@labri.fr

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 a variant of monadic second-order logic with data equality tests that captures precisely the data languages recognizable by orbit finite data monoids. We also establish, following this time the approach of Schützenberger, McNaughton and Papert, that the first-order fragment of this logic defines exactly the data languages recognizable by aperiodic orbit finite data monoids. Finally, we consider another variant of the logic that can be interpreted over generic structures with data. The data languages defined in this variant are also recognized by unambiguous finite memory automata.

1. INTRODUCTION

Data words have been introduced as a generalization of words over finite alphabets, where the term "data" denotes the presence of symbols from an infinite alphabet. Usually, languages of data words, data languages for short, are assumed to be 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:

L_1 :   *the sets of words containing at least three distinct data values*,
L_2 :   *the sets of words where the first and last positions carry the same data value*,
L_3 :   *the sets of words with no consecutive occurrences of the same data value*,
L_4 :   *the sets of words where each data value occurs at most once.*

The intention behind data values in data words (or data trees, ...) is to model, e.g., the keys in a database, or the process or user identifiers in the log of a system. Those numbers are used as identifiers, and we are interested only in comparing them with 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, for instance, Straubing's book [37] or Pin's survey [30] for an overview of the approach.

In [6] Bojańczyk formalized a notion of recognizability for data languages by introducing generalizations of monoids, called *data monoids*. In the journal version of this paper [7], the algebraic framework of data monoids has been further generalized and connected to the theory of nominal sets, which was originally developed by Fraenkel in 1922. Here we are mainly interested in data languages recognized by *orbit-finite data monoids*, which can be seen as the analogue of finite monoids for languages over infinite alphabets. As a matter of fact, all regular languages over a finite alphabet can be seen as data languages recognized by orbit-finite data monoids. Other examples of data languages recognized by orbit-finite data monoids are the languages L_1, L_2, L_3 that we described above.

Concerning the possibility of defining data languages by logical formulas, a natural approach consists of extending classical logics by introducing a new predicate $x \sim y$, which holds at positions x and y whenever the data values under 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, namely, it 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 extended with the data equality predicate. However, the converse does not hold, as witnessed by the formula

$$\forall x, y \quad x \neq y \ \rightarrow \ x \nsim y, \tag{$\dagger$}$$

defining the language L_4 above, which is known not to be recognizable by orbit-finite data monoids. More generally, it has been shown that monadic logic (in fact, even first-order logic) extended with the data equality predicate has an undecidable satisfiability problem and it can express properties not implementable by reasonable automaton models [28].

The general goal of this paper 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 particular, we aim at answering the following question:

> *Is there a variant of monadic second-order logic that defines precisely the data languages recognizable by orbit-finite data monoids?*

We answer this question positively by introducing a variant of monadic second-order logic with rigidly guarded data equality tests, *rigidly guarded MSO~* for short. This logic allows testing equality of two data values only when the two positions are related in a bijective 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, namely, it defines a partial bijection. For example, one can express the existence of two consecutive positions sharing the same data value: $\exists x, y. \ (x = y + 1) \wedge x \sim y$. The guard $(x = y + 1)$ is rigid since x uniquely determines y, and y uniquely determines x. However, it is impossible to describe the language L_4 in this logic. In particular, the above

formula (†) is logically equivalent to $\neg \exists x, y.\ x \neq y\ \wedge\ x \sim y$, 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 (see Section 3), and (ii) rigidity is decidable for formulas in our logic (cf. Corollary 3.5).

To validate the robustness of our approach, we also answer positively to the following question inspired by the seminal works of Schützenberger, McNaughton, and Papert:

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

The idea underlying the use of guards with data tests can be generalized in different ways. In the present paper, we also consider a less constrained version of rigidly guarded MSO~, which allows one to compare the data values at two positions y and z, whenever both y and z are determined from a common position x by means of suitable formulas. The resulting logic, called semi-rigidly guarded MSO~, can be interpreted over more general structures, such as graphs with data on nodes, and still retains the decidability properties of rigidly guarded MSO~. Towards the end of the paper, we study the expressiveness of semi-rigidly guarded MSO~ on data words and we prove that this logic is strictly subsumed by unambiguous finite memory automata [22].

Related work. This work is related to the well known theory of regular languages. By this we specifically refer to two key results, namely, the equivalence between recognizability by finite state automata and definability in monadic logic [14], and the characterization of first-order definability for regular languages [35, 27].

The other branch of related work is concerned with languages of data words. The first related contribution in this direction is due to Kaminski and Francez [22, 24], who introduced finite memory automata (FMA for short). These automata possess a fixed finite set of registers that can be used to store data values. At each step an FMA can compare the current data value with the values stored in the registers and, on the basis of these tests and the current control state, it can determine the target control state of its transition, and whether or not the current value (or a new guessed value) is stored into some register (replacing the previous content). This model of automaton, in its non-deterministic form, has a decidable emptiness problem and an undecidable universality problem; decidability of universality is however recovered in the deterministic variant of FMA. Deterministic FMA also have minimal canonical forms, provided that a suitable policy in the use of registers is enforced [4, 9] (such a policy does not affect the expressive power of the model). Many other automaton models for data languages have been proposed in the literature, such as automata with pebbles [28], automata with hash tables [5], walking automata [26], data and class automata [8, 11]. We refer the interested reader to [33] for a survey on these models.

As concerns the logical approach, several logics for reasoning effectively on data languages have been proposed, most notably: fragments of first-order logics with data equalities/disequalities [8, 34], variants of XPath called Core-Data-XPath [12], modal logics with registers [17, 21]. The differences between all such formalisms are reflected in the fact that it is difficult to obtain algebraic characterizations for robust classes of data languages. In [3, 8, 10, 13] some preliminary results on relating automata to logics are given. However, the algebraic theory for these automaton models is not fully developed yet. As a matter of fact, the question of characterizing the first-order logic definable language among the languages recognized by deterministic FMA remains open.

The idea of guarding tests with rigid formulas was originally presented in [16]. A similar idea was also exploited in [1] in order to design a class of timed automata that could be determinized. More recently, a similar idea has been investigated in [39] with the aim of developing a robust formalism for querying graph databases.

Contributions and structure of the paper. Our main contributions can be summarized as follows:

(1) We show how orbit-finite data monoids can be finitely represented by systems of equations involving terms with variables for data values. We further develop the theory of Green's relations for data monoids, proving, for instance, that all $\mathcal{H}$-classes in an orbit-finite data monoid are finite (or, equally, that all orbit-finite data groups are finite).

(2) We introduce a logic, called rigidly guarded MSO˜, which can be seen as a natural weakening of MSO logic with data equality tests. We then show that rigidly guarded MSO˜ is exactly as expressive as orbit-finite data monoids, and that its first-order fragment corresponds to aperiodic orbit-finite data monoids.

(3) We show that an extension of rigidly guarded MSO˜ is decidable, even on general classes of structures with data (e.g., data trees). We show that the same extension of rigidly guarded MSO˜ defines a proper subclass of data languages recognized by non-deterministic (in fact, unambiguous) finite memory automata.

Section 2 gives some background knowledge on the theory of nominal sets, data languages and data monoids. In particular, it explains how orbit-finite data monoids can be finitely represented and further develops the theory of Green's relations for these monoids. Section 3 introduces variants of rigidly guarded logics and shows how to decide satisfiability of their formulas over generic classes of data words, data trees, and data graphs. 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 of data words. Section 5 describes the converse translation, namely, from (aperiodic) orbit-finite data monoids to rigidly guarded MSO˜ (resp., FO˜) formulas. Section 6 relates data languages defined by variants of rigidly guarded MSO˜ to data languages recognized by finite memory automata. Section 7 provides an assessment of the results and related open problems.

2. Nominal sets and 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 $w = (d_1, a_1) \ldots (d_n, a_n)$ in $(D \times A)^*$. The domain of w, denoted $\mathsf{dom}(w)$, is $\{1, \ldots, n\}$.

We begin by giving a short account of the theory of nominal sets, which can then be used to derive natural notions of recognizability of data languages (we freely use some terminology and concepts from [6, 7, 9]).

A *(data) renaming* on D is a permutation on the set D of data values that is the identity on *all but finitely many* values. We let G_D the set of all renamings on D. One obtains a group $\mathcal{G}_D = (G_D, \circ)$ by equipping G_D with the operation of functional composition; we call this group the *group of renamings on D*. The above definitions are naturally generalized to

any (possibly finite) subset C of D; for example, we can talk about the group of renamings on C.

Renamings act on sets as follows. Given a set S, an *action* of the group $\mathcal{G}_D$ on S is a group morphism $\hat{\ }$ from $\mathcal{G}_D$ to the group of bijections on S, namely, a function $\hat{\ }$ that maps the identity ι of $\mathcal{G}_D$ to the identity $\hat{\iota}$ on S and such that $\widehat{\tau \circ \pi} = \hat{\tau} \circ \hat{\pi}$ for all renamings $\tau, \pi \in \mathcal{G}_D$. We call $\mathcal{G}_D$-*set* any set S equipped with an action $\hat{\ }$ of $\mathcal{G}_D$ on S.

Given an element s of a $\mathcal{G}_D$-set $(S, \hat{\ })$, we define the *orbit of s* as the set of all elements of the form $\hat{\tau}(s)$, for all renamings $\tau \in \mathcal{G}_D$. Note that orbits are either disjoint or equal, so they can be seen as equivalence classes induced by the possible renamings. We say that a $\mathcal{G}_D$-set is *orbit-finite* if it has only finitely many orbits.

A subset S' of a $\mathcal{G}_D$-set $(S, \hat{\ })$ is said to be *equivariant* if it is preserved by the action of renamings, namely, if $\hat{\tau}(S') = S'$ for all renamings $\tau \in \mathcal{G}_D$ (equivalently, one could say that S' is a union of orbits of S). The concept of equivariant subset can be applied specifically to a function $f : S \to T$ between two $\mathcal{G}_D$-sets $(S, \hat{\ })$ and $(T, \check{\ })$; in this case one easily verifies that f commutes with the renamings, namely, $f(\hat{\tau}(s)) = \check{\tau}(f(s))$ for all $f \in \mathcal{G}_D$ and all $s \in S$. Similarly, by considering the standard action of renamings on sets of data words (i.e., $\hat{\tau}((d_1, a_1) \ldots (d_n, a_n)) =^{\mathrm{def}} (\tau(d_1), a_1) \ldots (\tau(d_n), a_n))$, we define a *data language* over $D \times A$ as an equivariant subset of $(D \times A)^*$ (this basically means that membership in the language is invariant under renamings of data values).

2.1. **Data monoids.** Recall that a monoid is an algebraic structure $\mathcal{M} = (M, \cdot)$ where $\cdot$ is an associative product on M admitting an identity $1_{\mathcal{M}}$ such that $1_{\mathcal{M}} \cdot s = s \cdot 1_{\mathcal{M}} = s$ for all $s \in M$. A monoid $\mathcal{M} = (M, \cdot)$ is said to be *aperiodic* if for all elements $s \in M$, there is $n \in \mathbb{N}$ such that $s^n = s^{n+1}$. A (*monoid*) *morphism* is a function h between two monoids $\mathcal{M} = (M, \cdot)$ and $\mathcal{N} = (N, \odot)$ such that $h(1_{\mathcal{M}}) = 1_{\mathcal{N}}$ and $h(s \cdot t) = h(s) \odot h(t)$ for all $s, t \in M$. The concept of data monoid is nothing but that of a monoid with an equivariant product:

Definition 2.1. A *data monoid* (over a set D of data values) is a triple $\mathcal{M} = (M, \cdot, \hat{\ })$, where $(M, \cdot)$ is a monoid, $\hat{\ }$ is an action of $\mathcal{G}_D$ on M, and $\cdot$ is an equivariant function with respect to $\hat{\ }$. In particular, for all renamings $\tau, \pi \in \mathcal{G}_D$ and all elements $s, t \in M$, we have:

- $\widehat{\tau \circ \pi} = \hat{\tau} \circ \hat{\pi}$,
- $\hat{\iota}(s) = s$, where ι is the identity renaming,
- $\hat{\tau}(1_{\mathcal{M}}) = 1_{\mathcal{M}}$, where $1_{\mathcal{M}}$ is the identity of $(M, \cdot)$,
- $\hat{\tau}(s) \cdot \hat{\tau}(t) = \hat{\tau}(s \cdot t)$.

Unless otherwise stated, data monoids will be defined over the set D of all data values. Moreover, to simplify the notation, we will often use an implicit notation for the group action $\hat{\ }$; for example, when $\hat{\ }$ is understood from the context, we can write $\tau(s)$ in place of $\hat{\tau}(s)$.

The *free data monoid* over $D \times A$ is an example of a data monoid, where the elements are the data words over $D \times A$, the product is the juxtaposition of data words, and the action is the standard one, mapping any renaming τ to the automorphism $\hat{\tau}$ 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 now show how to extract the "memory" of a monoid element s, which intuitively is the minimum set of data values that are important for distinguishing s from all other elements of the data monoid. Given a data monoid $\mathcal{M}$ and an element s in it, we say

that a renaming τ is a *stabilizer* of s if $\tau(s) = s$. 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 again a set that supports s [6, 7, 19]. We can thus define *the memory* of s, denoted $\mathsf{mem}(s)$, as the intersection of all sets that support s.

We remark that there exist finite monoids whose elements have infinite memory (see [6] 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 mainly interested in homomorphic images of the free monoid, hereafter we will consider only monoids whose elements have finite memory – this property is called the *finite support axiom*.

Definition 2.2. Let $\mathcal{M}$ be a data monoid. We define *the memory* of an element s in $\mathcal{M}$ as
$$\mathsf{mem}(s) \;=\; \bigcap \big\{ C \subseteq D \;:\; \forall \tau \in G_D.\ (\forall d \in C.\ \tau(d) = d) \;\rightarrow\; \tau(s) = s \big\} \,.$$
and we assume that this set is always finite. A data value is said to be *memorable* in s if it belongs to $\mathsf{mem}(s)$.

A *morphism* between two data monoids $\mathcal{M} = (M, \cdot, \hat{\ })$ and $\mathcal{N} = (N, \odot, \check{\ })$ is a monoid morphism that is equivariant, namely, a function $h : M \to N$ such that

- $\quad h(1_{\mathcal{M}}) = 1_{\mathcal{N}}$,
- $\quad h(s \cdot t) = h(s) \odot h(t)$ for all $s, t \in M$,
- $\quad h(\hat{\tau}(s)) = \check{\tau}(h(s))$ for all $s \in M$ and all renamings $\tau \in \mathcal{G}_D$.

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 $w \in (D \times A)^*$ in L is determined by the element $h(w)$ of $\mathcal{M}$, namely, if $L = h^{-1}(h(L))$.

We conclude the preliminary discussion on data monoids by recalling the definition of *orbit-finite* $\mathcal{G}_D$-set, that is, a $\mathcal{G}_D$-set that admits only finitely many orbits $\{\tau(s) \;:\; \tau \in \mathcal{G}_D\}$. This property can be naturally applied to the domain of a data monoid $\mathcal{M}$, resulting in the concept of *orbit-finite data monoid*. Below, we give an example of a data language that is recognized by an orbit-finite data monoid and an example of a data language that is recognized only by orbit-infinite data monoids.

Example 2.3. Consider the language $L_2 = \{d_1 \dots d_n \in D^* \;:\; n \geq 1,\ d_1 = d_n\}$ introduced at the beginning of Section 1. One can construct the syntactic data monoid recognizing L_2 by considering the classes of the two-sided Myhill-Nerode equivalence on data words. More precisely, the class of a non-empty word $w = d_1 \dots d_n$ can be identified with the pair (d_1, d_n) of data values, while the class of the empty word is a distinguished element behaving as the identity. Accordingly, the product of two elements (d, e) and (f, g), distinct from the identity, is the pair (d, g). This syntactic data monoid admits only three orbits: the singleton orbit containing the identity element, the orbit $\{(d, d) \;:\; d \in D\}$, and the orbit $\{(d, e) \;:\; d \neq e \in D\}$.

Example 2.4. Consider the language $L_4 = \{d_1 \dots d_n \in D^* \;:\; \forall i \neq j \leq n.\ d_i \neq d_j\}$. The element of the syntactic monoid of L_4 that corresponds to a word $w \notin L_4$ behaves as a null element 0: the product of 0 with any other element of the syntactic monoid gives again 0. On the other hand, the element that correspond to a word $w = d_1 \dots d_n \in L_4$ can be identified with the set $\{d_1, \dots, d_n\}$ of data values. Accordingly, the product of the syntactic monoid maps any two disjoint sets of data values to their union, and any two intersecting sets of data values to the null element 0. It is easy to see that this syntactic monoid has infinitely many orbits.

2.2. **Finite presentations of data monoids.** Orbit-finite data monoids are infinite objects that need to be represented in a finite way in order to be used in algorithms. Here we propose to represent these objects by means of systems of equations involving terms. The starting point consists of looking at restrictions of data monoids to finite sets of data values:

Definition 2.5. Given a data monoid $\mathcal{M} = (M, \cdot, \,\hat{}\,)$ and a (finite or infinite) set $C \subseteq D$, we define the *restriction of $\mathcal{M}$ to C* as the data monoid $\mathcal{M}|_C = (M|_C, \cdot|_C, \,\hat{}\,|_C)$, where $M|_C$ consists of all elements $s \in M$ such that $\mathsf{mem}(s) \subseteq C$, $\cdot|_C$ is the restriction of $\cdot$ to $M|_C$, and $\hat{}\,|_C$ is the restriction of $\hat{}\,$ to $\mathcal{G}_C$ and $M|_C$.

Despite the fact that the restriction of a data monoid to a finite set C is still a data monoid, 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 from [6] does not hold for data monoids over finite sets. However, most of the properties that we outline hereafter hold independently of whether data monoids are defined over finite or infinite sets of data values.

We observe that if s and t are elements in the same orbit of a data monoid, then their memories have the same cardinality. This allows us to denote by $\|\mathcal{M}\|$ the maximum cardinality of the memories of the elements of an orbit-finite data monoid $\mathcal{M}$. The following 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 a number of natural operations on orbit-finite data monoids can be performed at the level of the finite restriction. Some noticeable examples of such operations are the disjoint union and the product of two orbit-finite data monoids and the quotient of an orbit-finite data monoid with respect to a congruence. Thus, restrictions of orbit-finite data monoids provide a convenient way to effectively manipulate orbit-finite data monoids.

Proposition 2.6. *Let $\mathcal{M}, \mathcal{N}$ be orbit-finite data monoids such that $\|\mathcal{M}\| = \|\mathcal{N}\|$ and let $C \subseteq D$ be a set of cardinality at least $2\|\mathcal{M}\|$. If $\mathcal{M}|_C$ and $\mathcal{N}|_C$ are isomorphic, then so are $\mathcal{M}$ and $\mathcal{N}$.*

Proof. Let $\mathcal{M} = (M, \cdot, \,\hat{}\,)$ and $\mathcal{N} = (N, \odot, \,\check{}\,)$ and let f_C be a data monoid isomorphism from $\mathcal{M}|_C$ to $\mathcal{N}|_C$. We show how extend f_C to an isomorphism from $\mathcal{M}$ to $\mathcal{N}$. Given $s \in M$, we let τ be any renaming such that $\tau(\mathsf{mem}(s)) \subseteq C$ (such a renaming exists since $|\mathsf{mem}(s)| \leq |C|$); we then observe that the element $\hat{\tau}(s)$ belongs to the data monoid $M|_C$ and we accordingly define
$$f(s) \;=^{\mathrm{def}}\; \hat{\tau}^{-1}(f_C(\hat{\tau}(s))) \;.$$
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 so, we consider two renamings τ and π such that $\tau(\mathsf{mem}(s)) \subseteq C$ and $\pi(\mathsf{mem}(s)) \subseteq C$, we define $t = \hat{\tau}^{-1}(f_C(\hat{\tau}(s)))$ and $t' = \hat{\pi}^{-1}(f_C(\hat{\pi}(s)))$, and we prove that $t = t'$. Let $\theta = \pi \circ \tau^{-1}$. Since $\pi = \theta \circ \tau$, we have
$$t' \;=\; \check{\pi}^{-1}(f_C(\hat{\pi}(s))) \;=\; \check{\pi}^{-1}(f_C(\hat{\theta}(\hat{\tau}(s)))) \;.$$
Since θ is a renaming over C and f_C is a morphism between data monoids over C, we have $f_C \circ \hat{\theta} = \check{\theta} \circ f_C$ and hence
$$\check{\pi}^{-1}(f_C(\hat{\theta}(\hat{\tau}(s)))) \;=\; \check{\pi}^{-1}(\check{\theta}(f_C(\hat{\tau}(s)))) \;.$$
Moreover, since $\tau^{-1} = \pi^{-1} \circ \theta$, we get
$$\check{\pi}^{-1}(\check{\theta}(f_C(\hat{\tau}(s)))) \;=\; \check{\tau}^{-1}(f_C(\hat{\tau}(s))) \;=\; t \;.$$

This proves that the function f is well defined.

Next, we claim that f is a bijection from M to N. Surjectivity is straightforward, since for every element $t \in N$, there exists a renaming τ such that $\tau(\mathsf{mem}(t)) \subseteq C$, and hence, if we let $s = \hat{\tau}^{-1}(f_C^{-1}(\hat{\tau}(t)))$, we have $f(s) = t$. The proof that f is injective is analogous to the proof that f is well defined, and thus omitted. It remains to prove that f is a data monoid isomorphism.

Commutativity with renamings. We claim that f commutes with the action of renamings. Given an element $s \in M$ and a renaming $\pi \in \mathcal{G}_D$, we choose a renaming τ such that $\tau(\mathsf{mem}(s)) \subseteq C$ and $\tau(\mathsf{mem}(\hat{\pi}(s))) \subseteq C$ hold (note that such a renaming exists since $|\mathsf{mem}(s) \cup \mathsf{mem}(\hat{\pi}(s))| \le |C|$). In particular, both elements $\hat{\tau}(s)$ and $\hat{\tau}(\hat{\pi}(s))$ belong to the data monoid $\mathcal{M}|_C$. We also define the renaming $\theta = \tau \circ \pi \circ \tau^{-1}$. Note that, by construction, we have $\hat{\theta}(\hat{\tau}(s)) = \hat{\tau}(\hat{\pi}(s))$. Moreover, by exploiting the definition of f and the fact that f_C is a data monoid morphism from $\mathcal{M}|_C$ to $\mathcal{N}|_C$, we obtain

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

Commutativity with products. We conclude the proof by showing that f preserves identities and commutes with products. Recall that $M|_C$ (resp., $N|_C$) contains the identity $1_{\mathcal{M}}$ of $\mathcal{M}$ (resp., the identity $1_{\mathcal{N}}$ of $\mathcal{N}$). Since f_C is a monoid morphism from $\mathcal{M}|_C$ to $\mathcal{N}|_C$, it follows that $f(1_{\mathcal{M}}) = f_C(1_{\mathcal{M}}) = 1_{\mathcal{N}}$. Let us now consider two elements $s, t \in M$. Let τ be a renaming such that $\tau(\mathsf{mem}(s)) \subseteq C$ and $\tau(\mathsf{mem}(t)) \subseteq C$ (again, such a renaming exists since $|\mathsf{mem}(s) \cup \mathsf{mem}(t)| \le |C|$). In particular, both elements $\hat{\tau}(s)$ and $\hat{\tau}(t)$ belong to $\mathcal{M}|_C$. Since f_C is a monoid morphism, we obtain

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

We have just shown that $\mathcal{M}$ and $\mathcal{N}$ are isomorphic data monoids. $\qquad\square$

Proposition 2.6 shows that, assuming orbit-finiteness, one can represent an infinite data monoid by a finite restriction of it. It is also possible to give more explicit representations of orbit-finite data monoids using what we call *term-based presentation systems*. According to such systems, elements are represented by terms of the form $o(d_1, \ldots, d_k)$, where o is an *orbit name*, with an associated arity k, and $d_1, \ldots, d_k$ are distinct data values. Terms are furthermore considered modulo an equivalence relation $\approx$ and equipped with a binary product operation $\odot$. Before entering the details of term-based presentation systems, we explain the general idea by means of an example.

Example 2.7. Let $L_1 = \{d_1 \ldots d_n \in D^* : \exists i, j, k \le n. \ d_i \ne d_j, \ d_j \ne d_k, \ d_i \ne d_k\}$ be the language of data words with at least three distinct values. The elements of the syntactic data monoid of L_1 can be conveniently represented by terms, as follows: the empty word is represented by the term $o(\varepsilon)$ of arity 0; the equivalence class of a constant data word $d \ldots d$ is represented by the term $p(d)$ or arity 1; the equivalence class of a data word containing exactly two distinct data values d, e is represented by the term $q(d, e)$ or, equivalently, by the term $q(e, d)$; the equivalence class for all remaining words is represented by another term $r(\varepsilon)$ of arity 0. Accordingly, the syntactic data monoid of L_1 is represented by the

following system of equations, where d, e, f, g denote pairwise distinct data values and t denotes a generic term built up from the orbit names o, p, q, r:

$$o(\varepsilon) \odot t \;\approx\; t \odot o(\varepsilon) \;\approx\; t \qquad\qquad q(d,e) \odot p(d) \;\approx\; p(d) \odot q(d,e) \;\approx\; q(d,e)$$

$$r(\varepsilon) \odot t \;\approx\; t \odot r(\varepsilon) \;\approx\; r(\varepsilon) \qquad\qquad q(d,e) \odot q(d,e) \;\approx\; q(d,e)$$

$$p(d) \odot p(d) \;\approx\; p(d)$$
$$q(d,e) \odot p(f) \;\approx\; p(f) \odot q(d,e) \;\approx\; r(\varepsilon)$$
$$p(d) \odot p(e) \;\approx\; q(d,e)$$
$$q(d,e) \odot q(d,f) \;\approx\; r(\varepsilon)$$
$$q(d,e) \;\approx\; q(e,d) \qquad\qquad q(d,e) \odot q(f,g) \;\approx\; r(\varepsilon) \;.$$

Hereafter, we will focus on those term-based presentation systems that correctly represent data monoids, namely, whose binary operation $\odot$ is associative over the equivalence classes. Clearly, if the orbit names of the term-based presentation system range over a finite set, then the represented data monoid is orbit-finite. We will see below that a converse result also holds, showing that every orbit-finite data monoid can be represented by a term-based presentation system that uses only finitely many orbit names. This allows us to represent orbit-finite data monoids by finite systems of equations involving terms and products between them.

We now give a formal definition of our term-based presentation system. We denote by $T_{O,C}$ the set of all *terms* of the form $o(d_1, \ldots, d_k)$, where o is an orbit name from a finite set O, k is the arity of o, and $d_1, \ldots, d_k$ are pairwise distinct data values from a (finite or infinite) subset C of D.

Definition 2.8. Let O be a finite 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 $\check{\tau}(o(d_1, \ldots, d_n)) = o(\tau(d_1), \ldots, \tau(d_n))$, and an equivalence $\approx$ on T satisfying the following properties for all terms $s, t, u, v \in T$ and all renamings $\tau \in \mathcal{G}_C$:

- *(identity)* there is a term 1_T of arity 0 such that $1_T \odot s = s \odot 1_T = s$,
- *(equivariance)* $\check{\tau}(s) \odot \check{\tau}(t) = \check{\tau}(s \odot t)$,
- *(associativity up to $\approx$)* $(s \odot t) \odot u \approx s \odot (t \odot u)$,
- *(congruence for products)* if $s \approx t$ and $u \approx v$, then $s \odot u \approx t \odot v$,
- *(congruence for renamings)* if $s \approx t$ then $\check{\tau}(s) \approx \check{\tau}(t)$.

Let $\mathcal{S} = (T, \odot, \check{\ }, \approx)$ be a term-based presentation system. We remark that $(T, \odot, \check{\ })$ is not necessarily a data monoid because associativity only holds up to congruence $\approx$. We say that $\mathcal{S}$ *represents* the structure $\mathcal{M} = (M, \cdot, \hat{\ })$ if

- M is the set of $\approx$-equivalence classes of terms in T,
- $\cdot$ is the binary operation on M defined by $[s]_\approx \cdot [t]_\approx = [s \odot t]_\approx$,
- $\hat{\ }$ maps any renaming $\tau \in \mathcal{G}_C$ to the function $\hat{\tau}$ defined by $\hat{\tau}([s]_\approx) = [\check{\tau}(s)]_\approx$

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

Proposition 2.9. *Every term-based presentation system represents an orbit-finite 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 a set C of data values and let $\mathcal{M} = (M, \cdot, \hat{\ })$ be the structure represented by $\mathcal{S}$. It is easy to see that $\cdot$ is an associative operation and that $[1_T]_{\approx}$ behaves as an identity for $\cdot$. This means that $(M, \cdot)$ is a monoid. Below, we verify the other properties of orbit-finite data monoids:

(1)   Since the identity term 1_T has arity 0, we have that all renamings $\tau \in \mathcal{G}_C$ are stabilizers of $[1_T]_{\approx}$, that is, $\hat{\tau}([1_T]_{\approx}) = [\check{\tau}(1_T)]_{\approx} = [1_T]_{\approx}$.

(2)   We now check that $\widehat{\tau \circ \pi} = \hat{\tau} \circ \hat{\pi}$ for all data renamings $\tau, \pi \in \mathcal{G}_C$. Let $[o(\bar{d})]$ be an element of M (we will drop the subscript $\approx$ in the rest of the proof). Then we have

$$\widehat{\tau \circ \pi}[o(\bar{d})] \;=\; [o((\tau \circ \pi)(\bar{d}))] \;=\; [o(\tau(\pi(\bar{d})))] \;=\; \hat{\tau}([o(\pi(\bar{d}))]) \;=\; \hat{\tau} \circ \hat{\pi}([o(\bar{d})]) \;.$$

(3)   If ι is the identity renaming on C, then $\hat{\iota}([o(\bar{d})]) = [o(\iota(\bar{d}))] = [o(\bar{d})]$.

(4)   Let $[s], [t] \in M$ and let $\tau \in \mathcal{G}_C$ be a data renaming. We prove that $\hat{\tau}([s] \cdot [t]) = \hat{\tau}([s]) \cdot \hat{\tau}([t])$. Assume that $s = o(\bar{d})$, $t = p(\bar{e})$, and $s \odot t = q(\bar{f})$. Then

$$\hat{\tau}([s] \cdot [t]) \;=\; \hat{\tau}([s \odot t]) \;=\; \hat{\tau}([q(\bar{f})]) \;=\; [q(\tau(\bar{f}))] \;=\; [\check{\tau}(q(\bar{f}))] \;=\; [\check{\tau}(s \odot t)] \;.$$

Moreover, from the equivariance property of Definition 2.8, we know that

$$\check{\tau}(s \odot t) \;\approx\; \check{\tau}(s) \odot \check{\tau}(t).$$

We continue our calculation as follows

$$\begin{aligned}
\big[\check{\tau}(s \odot t)\big] \;&=\; \big[\check{\tau}(s) \odot \check{\tau}(t)\big] \;=\; \big[\check{\tau}(o(\bar{d})) \odot \check{\tau}(p(\bar{e}))\big] \;=\; \big[o(\tau(\bar{d})) \odot p(\tau(\bar{e}))\big] \\
&=\; \big[o(\tau(\bar{d}))\big] \cdot \big[p(\tau(\bar{e}))\big] \;=\; \hat{\tau}([s]) \cdot \hat{\tau}([t]) \;.
\end{aligned}$$

Combining the two equations, we get $\hat{\tau}([s] \cdot [t]) = \hat{\tau}([s]) \cdot \hat{\tau}([t])$.

(5)   We can finally claim that $\mathcal{M} = (M, \cdot, \hat{\ })$ is an orbit-finite data monoids: this follows immediately from the fact that the set O of orbit names is finite.

The proof of the second part of the proposition is more tedious, but not really difficult. 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 the natural numbers. We first define a term-based representation system $\mathcal{S} = (T, \odot, \check{\ }, \approx)$ and later we show that $\mathcal{S}$ represents $\mathcal{M}$.

Definition of $\mathcal{S}$.   Let O be a finite set of orbit names that contains exactly one orbit name o for each orbit of $\mathcal{M}$. The arity of o is the size of the memories of the elements of o (recall that memories of elements from the same orbit have the same cardinality). Define T to be the set of all terms that are build up from orbit symbols in O and data values in C. Recall that, in a term-based representation system, the action $\check{\ }$ of the renamings is naturally defined as follows: $\check{\tau}(o(\bar{d})) = o(\tau(\bar{d}))$ for all data renamings $\tau \in \mathcal{G}_C$.

Below we define the operation $\odot$ on T. Since each element of $\mathcal{M}$ can be represented by several terms in T, we need to commit to a specific mapping of elements in $\mathcal{M}$ to terms in T. The rough idea is as follows. We begin by fixing some representatives of the orbits of $\mathcal{M}$ and an isomorphism between these representatives and some canonical terms in T. To compute the product of two terms $s, t \in T$, we first apply a renaming so as to map them to the canonical terms $\tilde{s}$ and $\tilde{t}$; then, we exploit the isomorphism between the canonical terms and the representatives of the orbits of $\mathcal{M}$ to compute the product of $\tilde{s}$ and $\tilde{t}$ inside $\mathcal{M}$; finally, we apply the inverse isomorphism and renaming to obtain the desired product $s \odot t$.

More precisely, we fix a representative m_o inside each orbit o of $\mathcal{M}$ in such a way that $\mathsf{mem}(m_o)$ is a prefix of the natural numbers, namely, $\mathsf{mem}(m_o) = \{1, \ldots, \mathsf{arity}(o)\}$. 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 $d_1, \ldots, d_k$, and vice versa, and that is the identity on $D \smallsetminus \{1, \ldots, k, d_1, \ldots, d_k\}$. We then define the function f from T to M such that, for every term $o(\bar{d})$,

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

Note that f is not injective in general. This allows us to define the equivalence $\approx$ over terms by $s \approx t$ iff $f(s) = f(t)$.

For each element $m \in M$, we need to choose in a canonical way a term $g(m)$ that belongs to the set $f^{-1}(m)$. This can be accomplished by letting $g(m)$ be the term in $f^{-1}(m)$ with the minimal tuple of data values according to the lexicographical order. In a similar way, we can associate with each pair (s, t) of terms in T a *canonical renaming* $\sigma_{s,t}$ as follows. First, we say that a pair (s', t') of terms is *minimal* if s' is of the form $o(1, \ldots, k)$, t' is of the form $p(d_1, \ldots, d_h)$, and, for all $1 \le i < j \le h$, $d_i, d_j \notin \{1, \ldots, k\}$ implies $d_i < d_j$. Then, we define the canonical renaming $\sigma_{s,t}$ as the unique renaming σ such that $(\sigma(s), \sigma(t))$ is a minimal pair $(\sigma(s), \sigma(t))$.

We can now define the product $\odot$ of two terms $s, t \in T$ as follows:

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

Note that the term $s \odot t$ belongs to the set $f^{-1}(f(s) \cdot f(t))$. Accordingly, we define the identity term 1_T 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 2.8, we establish the following claim.

Claim 2.10. Let $s, t \in T$ and $\tau \in \mathcal{G}_C$. Then

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

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

(C3) $\tau \circ \sigma_{s,t}(d) \ = \ \sigma_{\check{\tau}(s), \check{\tau}(t)}(d)$ for all $d \in \mathsf{mem}(s) \cup \mathsf{mem}(t)$,

(C4) $\sigma_{\check{\tau}(s), \check{\tau}(t)} \circ \tau(d) \ = \ \sigma_{s,t}(d)$ for all $d \in \mathsf{mem}(s) \cup \mathsf{mem}(t)$.

Proof of claim. We first prove Condition C1. Recall that if $\bar{d} = d_1, \ldots, d_k$ is a tuple of data values, then $\sigma_{\bar{d}}$ is the data renaming that maps the numbers $1, \ldots, k$ to the values $d_1, \ldots, d_k$, and vice versa, and that is the identity on $D \smallsetminus \{1, \ldots, k, d_1, \ldots, d_k\}$. For all renamings τ and numbers $i \le k$, we have that

$$\tau \circ \sigma_{\bar{d}}(i) \ = \ \tau(d_i) \ = \ \sigma_{\tau(\bar{d})}(i) \ . \tag{$\star$}$$

For every term $s = o(\bar{d})$, with $\bar{d} = d_1, \ldots, d_k$, we verify that

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

Next, we verify Condition C2:

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

As for Condition C3, suppose that $s = o(d_1, \ldots, d_k)$ and $t = p(e_1, \ldots, e_h)$. We first consider the case of a data value $d \in \mathsf{mem}(s)$, namely, $d = d_i$ for some $1 \le i \le k$. By definition of $\sigma_{s,t}$, we have $\sigma_{s,t}(d) = i$. Hence

$$
\tau \circ \sigma_{s,t}(d) \ = \ \tau(i) \ = \ \sigma_{\check{\tau}(s), \check{\tau}(t)}(d_i) \ = \ \sigma_{\check{\tau}(s), \check{\tau}(t)}(d) \ .
$$

Next, we consider the case of a data value $d \in \mathsf{mem}(t)$, namely, $d = e_i$ for some $1 \le i \le h$. By definition of $\sigma_{s,t}$, we have $\sigma_{s,t}(d) = |\{e_1, \ldots, e_i\} \smallsetminus \{d_1, \ldots, d_k\}|$. From this we derive

$$
\tau \circ \sigma_{s,t}(d) \ = \ \tau\big(|\{e_1, \ldots, e_i\} \smallsetminus \{d_1, \ldots, d_k\}|\big) \ = \ \sigma_{\check{\tau}(s), \check{\tau}(t)}(e_i) \ = \ \sigma_{\check{\tau}(s), \check{\tau}(t)}(d) \ .
$$

The proof of the last condition $\sigma_{\check{\tau}(s), \check{\tau}(t)} \circ \tau\,(d) \ = \ \sigma_{s,t}(d)$ is similar. $\qquad\square$

Turning to the main proof of the proposition, we show that $\mathcal{S}$ is indeed a valid presentation system by verifying that all the conditions of Definition 2.8 are satisfied:

(1)   *Identity.* Recall that we defined the identity term to be $1_T = g(1_{\mathcal{M}})$. As $1_{\mathcal{M}}$ has empty memory, we have $g(1_{\mathcal{M}}) = f^{-1}(1_{\mathcal{M}})$. For a generic $t \in T$, we get

$$1_T \odot t \;=\; \check{\sigma}_{1_{\mathcal{S}},t}^{-1} \circ g\left(f \circ \hat{\sigma}_{1_{\mathcal{S}},t}(1_{\mathcal{S}}) \;\cdot\; f \circ \hat{\sigma}_{1_{\mathcal{S}},t}(t)\right)$$

$$=\; \check{\sigma}_{1_{\mathcal{S}},t}^{-1} \circ g\left(f \circ \hat{\sigma}_{1_{\mathcal{S}},t}(f^{-1}(1_{\mathcal{M}})) \;\cdot\; f \circ \hat{\sigma}_{1_{\mathcal{S}},t}(t)\right)$$

$$=\; \check{\sigma}_{1_{\mathcal{S}},t}^{-1} \circ g\left(1_{\mathcal{M}} \;\cdot\; f \circ \hat{\sigma}_{1_{\mathcal{S}},t}(t)\right)$$

$$=\; \check{\sigma}_{1_{\mathcal{S}},t}^{-1} \circ g\left(f \circ \hat{\sigma}_{1_{\mathcal{S}},t}(t)\right)$$

$$=\; t \;.$$

(2)   *Equivariance.* We verify that $\check{\tau}(s) \odot \check{\tau}(t) = \check{\tau}(s \odot t)$ for all $s, t \in T$ and $\tau \in \mathcal{G}_C$:

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

$$=\; \check{\sigma}_{\check{\tau}(s),\check{\tau}(t)}^{-1} \circ g\left(f \circ \hat{\sigma}_{s,t}(s) \;\cdot\; f \circ \hat{\sigma}_{s,t}(t)\right) \qquad \text{(by Condition C4)}$$

$$=\; \check{\tau} \circ \check{\sigma}_{s,t}^{-1} \circ g\left(f \circ \hat{\sigma}_{s,t}(s) \;\cdot\; f \circ \hat{\sigma}_{s,t}(t)\right) \qquad \text{(by Condition C3)}$$

$$=\; \check{\tau}(s \odot t).$$

(3)   *Associativity up to $\approx$.* Recall that two terms are $\approx$-equivalent iff f maps them to the same monoid element. We consider some terms s, t, u and we prove that $(s \odot t) \odot u \;\approx\; s \odot (t \odot u)$ as follows:

$$f((s \odot t) \odot u) \;=\; (f(s) \cdot f(t)) \cdot f(u) \qquad \text{(by Condition C2)}$$

$$=\; f(s) \cdot (f(t) \cdot f(u)) \qquad \text{(by associativity of $\cdot$)}$$

$$=\; f(s \odot (t \odot u)) \;. \qquad \text{(by Condition C2)}$$

(4)   *Congruence for products.* Assume that $s \approx t$ and $u \approx v$. By exploiting Condition C2 we easily verify that $s \odot u \;\approx\; t \odot v$:

$$f(s \odot u) \;=\; f(s) \cdot f(u) \;=\; f(t) \cdot f(v) \;=\; f(t \odot v) \;.$$

(5)   *Congruence for renamings.* Assume that $s \approx t$ and let σ be a renaming. We need to prove that $\check{\sigma}(s) \approx \check{\sigma}(t)$. We know that $f(s) = f(t)$. Moreover, since $\hat{\sigma}$ is a function on M, we know that $\hat{\sigma}(f(s)) = \hat{\sigma}(f(t))$. Finally, we know from Condition C1 that $f(\check{\sigma}(s)) = f(\check{\sigma}(t))$, whence $\check{\sigma}(s) \approx \check{\sigma}(t)$.

We have just proved that $\mathcal{S}$ is a term-based presentation system.

The term-based system represents $\mathcal{M}$. It remain to verify that $\mathcal{S} = (T, \odot, \check{\ })$ represents the data monoid $\mathcal{M} = (M, \cdot, \hat{\ })$. Let $\widetilde{\mathcal{M}} = (\widetilde{M}, \tilde{\cdot}, \tilde{\hat{\ }})$ be the structure represented by $\mathcal{S}$, 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

$$[s] \;\tilde{\cdot}\; [t] \;=\; [s \odot t] \qquad\qquad \tilde{\hat{\tau}}([s]) \;=\; [\check{\tau}(s)] \qquad\qquad 1_{\widetilde{\mathcal{M}}} \;=\; [1_{\mathcal{S}}] \;.$$

We know form the first part of the proposition that $\widetilde{\mathcal{M}}$ is a data monoid. We need to show that $\widetilde{\mathcal{M}}$ and $\mathcal{M}$ are isomorphic. For this, we consider the function $h : \widetilde{\mathcal{M}} \to \mathcal{M}$ defined by

$$h([s]) \;\overset{\text{def}}{=}\; f(s)$$

and we show that h is a data monoid isomorphism. We first check that h is a data monoid morphism. There are three properties to check:

(1)   We need to check that h commutes with products. Using Condition C2, we can calculate
$$h([s] \tilde{\cdot} [t]) \;=\; h([s \odot t]) \;=\; f(s \odot t) \;=\; f(s) \cdot f(t) \;=\; h([s]) \cdot h([t]) \;.$$

(2)   Next, we verify that h preserves the identity:
$$h(1_{\widetilde{\mathcal{M}}}) \;=\; h([1_{\mathcal{S}}]) \;=\; h(g(1_{\mathcal{M}})) \;=\; 1_{\mathcal{M}} \;.$$

(3)   Finally, we verify that h commutes with the renamings:
$$h(\tilde{\hat{\tau}}([s])) \;=\; h([\check{\tau}(s)]) \;=\; f(\check{\tau}(s)) \;=\; \hat{\tau}(f(s)) \;=\; \hat{\tau}(h([s])) \;.$$

Furthermore, h is injective by construction. It remains to show that h is surjective. Let some $m \in M$ be given. We will show that there is a term $t \in T$ such that $f(t) = m$. This will imply that m is the image via h of the element $[t] \in \tilde{M}$: indeed, we have $h([t]) = f(t) = m$.

   Let o be the orbit of m and assume that it has arity k. Recall that we fixed a representative for each orbit of $\mathcal{M}$, in particular, the representative of the orbit o is m_o. As m and m_o are in the same orbit there must exist a data renaming τ such that $\hat{\tau}(m_o) = m$. Moreover, recall that Condition 1 implies $f \circ \check{\tau} = \hat{\tau} \circ f$. By multiplying with $\check{\tau}^{-1}$ to the right, we get $f = \hat{\tau} \circ f \circ \check{\tau}^{-1}$. Towards a conclusion, define $t = \check{\tau}(o(1, \ldots, k))$ and observe that

$$
\begin{aligned}
f(\check{\tau}(o(1, \ldots, k))) \;&=\; \hat{\tau} \circ f \circ \check{\tau}^{-1}\left(\check{\tau}(o(1, \ldots, k))\right) \\[2mm]
&=\; \hat{\tau} \circ f\left(o(1, \ldots, k)\right) \\[2mm]
&=\; \hat{\tau}(\hat{\sigma}_{1,\ldots,k}(m_o)) && \text{(by the definition of } f) \\[2mm]
&=\; \hat{\tau}(m_o) && \text{(since } \sigma_{1,\ldots,k} \text{ is the identity)} \\[2mm]
&=\; m \;.
\end{aligned}
$$

We have just shown that $f(t) = m$ and hence h is surjective. This completes the proof of the proposition. $\qquad\square$

2.3. **Green's relations and memorable values.** In Section 5 we will show how recognizability by an orbit finite data monoid corresponds to definability by a formula of rigidly guarded MSO logic. Like in the theorem of Schützenberger [35], the translation from a monoid to a formula exploits an induction on certain ideals of the monoid that are induced by the so-called Green's relations [20, 30]. The goal of this section is to recall the basic ingredients of this theory and further develop it in order to ease the inductive constructions on orbit-finite data monoids.

   As already noticed in [6], a relevant part of the theory of Green's relations, which holds for finite monoids, can be lifted to *locally finite* monoids, namely, to monoids such that all finitely generated sub-monoids are finite. In particular, this applies to orbit-finite data monoids. The basic Green's relations $\leq_{\mathcal{R}}$, $\leq_{\mathcal{L}}$, $\leq_{\mathcal{J}}$ associated with an orbit-finite data

monoid $\mathcal{M}$ are the *preorders* defined by:

$$
\begin{array}{ccccccc}
s & \leq_{\mathcal{R}} & t & \qquad \text{iff} & \qquad s \cdot M & \subseteq & t \cdot M \\
s & \leq_{\mathcal{L}} & t & \qquad \text{iff} & \qquad M \cdot s & \subseteq & M \cdot t \\
s & \leq_{\mathcal{J}} & t & \qquad \text{iff} & \qquad M \cdot s \cdot M & \subseteq & M \cdot t \cdot M \; .
\end{array}
$$

We remark the following crucial property: for every orbit-finite data monoid, the preorder $\leq_{\mathcal{J}}$ is well-founded (for a proof of this result, see Lemma 9.3 in [7]). This provides the inductive principle that will be used in our proofs.

We also denote by $=_{\mathcal{R}}$, $=_{\mathcal{L}}$, $=_{\mathcal{J}}$ the corresponding equivalence relations (e.g., $s =_{\mathcal{J}} t$ iff $s \leq_{\mathcal{J}} t$ and $t \leq_{\mathcal{J}} s$) and we introduce an additional fourth equivalence $=_{\mathcal{H}}$ defined by

$$
s =_{\mathcal{H}} t \qquad \text{iff} \qquad s =_{\mathcal{R}} t \text{ and } s =_{\mathcal{L}} t \; .
$$

Given an element s of a data monoid $\mathcal{M}$, we denote by $\mathcal{R}(s)$ (resp., $\mathcal{L}(s)$, $\mathcal{J}(s)$, $\mathcal{H}(s)$) the $=_{\mathcal{R}}$-class (resp., $=_{\mathcal{L}}$-class, $=_{\mathcal{J}}$-class, $=_{\mathcal{H}}$-class) of s. We remark that the equivalence relation $=_{\mathcal{R}}$ (resp., $=_{\mathcal{L}}$) is a congruence with respect to products on the left (resp., right). For example, we have that $s =_{\mathcal{R}} t$ implies $u \cdot s =_{\mathcal{R}} u \cdot t$.

We naturally lift the above relations to orbits. Specifically, for each $\mathcal{K}$ among $\mathcal{R}$, $\mathcal{L}$, $\mathcal{J}$, we denote by $\leq_{\mathcal{K}^{\circ}}$ the preorder relation such that $s \leq_{\mathcal{K}^{\circ}} t$ iff $s \leq_{\mathcal{K}} \tau(t)$ for some renaming $\tau \in \mathcal{G}_D$. We do the same for the equivalence relations $=_{\mathcal{R}}$, $=_{\mathcal{L}}$, $=_{\mathcal{J}}$, $=_{\mathcal{H}}$, thus obtaining the relations $=_{\mathcal{R}^{\circ}}$, $=_{\mathcal{L}^{\circ}}$, $=_{\mathcal{J}^{\circ}}$, $=_{\mathcal{H}^{\circ}}$.

The last part of this section is devoted to an analysis of the types of data values that can occur in the memory of an element of an orbit-finite data monoid. We begin by distinguishing two types of data values.

Definition 2.11. Given an element s of an orbit-finite data monoid $\mathcal{M}$, we define $\mathsf{mem}_{\mathcal{R}}(s)$ (resp., $\mathsf{mem}_{\mathcal{L}}(s)$) to be the intersection of the memories of the elements in the $=_{\mathcal{R}}$-class (resp., $=_{\mathcal{L}}$-class) of s:

$$
\mathsf{mem}_{\mathcal{R}}(s) \;=^{\text{def}} \bigcap_{t \in \mathcal{R}(s)} \mathsf{mem}(t) \qquad\qquad \mathsf{mem}_{\mathcal{L}}(s) \;=^{\text{def}} \bigcap_{t \in \mathcal{L}(s)} \mathsf{mem}(t) \; .
$$

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

Quite surprisingly, it turns out that the memory of every element of an orbit-finite data monoids consists only of $\mathcal{R}$-memorable and $\mathcal{L}$-memorable values:

Proposition 2.12. *For every element s of an orbit-finite data monoid, we have* $\mathsf{mem}(s) = \mathsf{mem}_{\mathcal{R}}(s) \cup \mathsf{mem}_{\mathcal{L}}(s)$.

Before turning to the proof Proposition 2.12, let us show that a similar result fails for data monoids with infinitely many data orbits.

Example 2.13. Consider the data language $L_{\mathsf{even}} \subseteq D^*$ of all words where every value occurs an even number of times. The syntactic data monoid of the language L_{even} consists of one element s_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 s_C are exactly the values in C, which are neither $\mathcal{L}$-memorable nor $\mathcal{R}$-memorable (the syntactic data monoid is indeed a group).

In order to prove Proposition 2.12, 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 a severe restriction for data groups.

Lemma 2.14. *Every orbit-finite data group is finite.*

Proof. We begin by proving the following claim:

Claim 2.15. If s, t, u are elements of a data group (not necessarily orbit-finite), then

$$\mathsf{mem}(s) = \mathsf{mem}(s^{-1}) \qquad \text{and} \qquad \mathsf{mem}(s \cdot t \cdot u) \supseteq \mathsf{mem}(t) \smallsetminus \mathsf{mem}(s) \smallsetminus \mathsf{mem}(u) \ .$$

Proof of claim. We first prove the equality on the left. More precisely, we prove that $\mathsf{mem}(s) \subseteq \mathsf{mem}(s^{-1})$ (by symmetric arguments, one can prove that $\mathsf{mem}(s^{-1}) \subseteq \mathsf{mem}(s)$ holds as well). Recall that, by Definition 2.2, the memory $\mathsf{mem}(t)$ of an element t contains a data value d iff, for all sets $C \subseteq D \smallsetminus \{d\}$, there is a renaming τ that is the identity on C and such that $t \neq \tau(t)$. Let d be a data value in $\mathsf{mem}(s)$. To prove that $d \in \mathsf{mem}(s^{-1})$, we consider a generic set $C \subseteq D \smallsetminus \{d\}$. Since $d \in \mathsf{mem}(s)$, we know that there is a renaming τ that is the identity on C and such that $s \neq \tau(s)$. Moreover, because the identity 1 of the data group has empty memory, we have that $1 = \tau(1)$, and hence

$$s \cdot s^{-1} = 1 = \tau(1) = \tau(s \cdot s^{-1}) = \tau(s) \cdot \tau(s^{-1}) \ .$$

Finally, because $s \neq \tau(s)$ and because each element of the data group has exactly one inverse, we derive $s^{-1} \neq \tau(s^{-1})$. This proves that $d \in \mathsf{mem}(s^{-1})$ and hence $\mathsf{mem}(s) \subseteq \mathsf{mem}(s^{-1})$.

We conclude by proving the containment on the right. For this, it is sufficient to observe that $\mathsf{mem}(t) = \mathsf{mem}(s^{-1} \cdot s \cdot t \cdot u \cdot u^{-1}) \subseteq \mathsf{mem}(s \cdot t \cdot u) \cup \mathsf{mem}(s^{-1}) \cup \mathsf{mem}(u^{-1}) = \mathsf{mem}(s \cdot t \cdot u) \cup \mathsf{mem}(s) \cup \mathsf{mem}(u)$, and hence $\mathsf{mem}(t) \smallsetminus \mathsf{mem}(s) \smallsetminus \mathsf{mem}(u) \subseteq \mathsf{mem}(s \cdot t \cdot u)$. $\square$

To prove the lemma assume, towards a contradiction, that $\mathcal{G}$ is an infinite data group with finitely many orbits. $\mathcal{G}$ must contain an infinite orbit o, and hence we can inductively construct an infinite subset $G = \{g_1, g_2, \ldots\}$ of o such that each element g_i has a distinguished memorable value d_i that is not memorable in any other element of G, namely, for all i, we have $d_i \in \mathsf{mem}(g_i) \smallsetminus \bigcup_{j \neq i} \mathsf{mem}(g_j)$. Observe that, for all $i \leq k$, $\bigcup_{j < i} \mathsf{mem}(g_j) \supseteq \mathsf{mem}(g_1 \cdot \ldots \cdot g_{i-1})$ and $\bigcup_{j > i} \mathsf{mem}(g_j) \supseteq \mathsf{mem}(g_{i+1} \cdot \ldots \cdot g_k)$ (this holds in any data monoid, not necessarily in a data group). In particular, we have that for all $i \leq k$, $d_i \in \mathsf{mem}(g_i) \smallsetminus \mathsf{mem}(g_1 \cdot \ldots \cdot g_{i-1}) \smallsetminus \mathsf{mem}(g_{i+1} \cdot \ldots \cdot g_k)$. Finally, using the previous claim, we derive that, for all k,

$$\{d_1, \ldots, d_k\} \subseteq \mathsf{mem}(g_i \cdot \ldots \cdot g_k) \ .$$

Becasue the values $d_1, \ldots, d_k$ are pairwise distinct, this contradicts the finite memory axiom. $\square$

With each $=_{\mathcal{H}}$-class H of a monoid one can associate a group $\Gamma(H)$, called the Schützenberger group [30] (in fact there exist two such groups, but we will only consider one of them here). To define $\Gamma(H)$, we first introduce the set $T(H)$ of all elements $t \in H$ such that $t \cdot H$ is a subset of H. For each $t \in T(H)$, we then let γ_t be the transformation on H that maps $h \in H$ to $t \cdot h$. Finally, we define the Schützenberger group $\Gamma(H)$ as the set of all transformations γ_t, with $t \in T(H)$, equipped with the functional composition $\circ$ as binary product.

There is also a natural way to extend the action $\hat{\ }$ on 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 $\tau \in \Gamma_{D \smallsetminus (\mathsf{mem}_{\mathcal{R}}(H) \cup \mathsf{mem}_{\mathcal{L}}(H))}$, where $\mathsf{mem}_{\mathcal{R}}(H) = \mathsf{mem}_{\mathcal{R}}(h)$ and $\mathsf{mem}_{\mathcal{L}}(H) = \mathsf{mem}_{\mathcal{L}}(h)$ 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 2.16. *If $\mathcal{M} = (M, \cdot, \hat{\ })$ is a data monoid over D and H is an $=_{\mathcal{H}}$-class of $\mathcal{M}$, then $(\Gamma(H), \circ, \tilde{\ })$ is a data group over $D \smallsetminus (\mathsf{mem}_{\mathcal{R}}(H) \cup \mathsf{mem}_{\mathcal{L}}(H))$. Moreover, if $\mathcal{M}$ is orbit-finite, then so is $(\Gamma(H), \circ, \tilde{\ })$.*

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 $\tilde{\ }$ induced by the renamings over $D \smallsetminus (\mathsf{mem}_{\mathcal{R}}(H) \cup \mathsf{mem}_{\mathcal{L}}(H))$. By definition of $\tilde{\ }$, this is equivalent to verifying that H is closed under the action $\hat{\ }$ of renamings over $D \smallsetminus (\mathsf{mem}_{\mathcal{R}}(H) \cup \mathsf{mem}_{\mathcal{L}}(H))$. The proof is thus similar to proof of the Memory Theorem for $=_{\mathcal{J}}$-classes in [6]; 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. Since a renaming is a permutation that is the identity on all but finite many values, any renaming over $D \smallsetminus (\mathsf{mem}_{\mathcal{R}}(H) \cup \mathsf{mem}_{\mathcal{L}}(H))$ can be decomposed into a sequence of transpositions of pairs of values from $D \smallsetminus (\mathsf{mem}_{\mathcal{R}}(H) \cup \mathsf{mem}_{\mathcal{L}}(H))$. Therefore, in order to prove the closure of $H = R \cap L$ under the action of renamings over $D \smallsetminus (\mathsf{mem}_{\mathcal{R}}(H) \cup \mathsf{mem}_{\mathcal{L}}(H))$, it is sufficient to prove a similar closure property for the transpositions π_{de} of pair of elements $d, e \notin \mathsf{mem}_{\mathcal{R}}(H) \cup \mathsf{mem}_{\mathcal{L}}(H)$. We first show that R is closed under such transpositions. Let d and e be two values outside $\mathsf{mem}_{\mathcal{R}}(H)$ and let π_{de} be their transposition. Since $d, e \notin \mathsf{mem}_{\mathcal{R}}(H)$, we know that there exist two elements $s, t \in R$ such that $d \notin \mathsf{mem}(s)$ and $e \notin \mathsf{mem}(t)$. Let f be a data value outside $\mathsf{mem}(s) \cup \mathsf{mem}(t)$. 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 some elements 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 $\hat{\pi}_{ef}(s') =_{\mathcal{R}} \hat{\pi}_{ef}(s) = s \in R$. Moreover, 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 $D \smallsetminus (\mathsf{mem}_{\mathcal{R}}(H) \cup \mathsf{mem}_{\mathcal{L}}(H))$, $\hat{\tau}(s) \in R \cap L = H$. This shows that H is closed under the action $\hat{\ }$ of renamings over $D \smallsetminus (\mathsf{mem}_{\mathcal{R}}(H) \cup \mathsf{mem}_{\mathcal{L}}(H))$.

Below, we verify that $\tilde{\ }$ is a group morphism from the group of renamings over $D \smallsetminus (\mathsf{mem}_{\mathcal{R}}(H) \cup \mathsf{mem}_{\mathcal{L}}(H))$ to the group of automorphisms on $\Gamma(H)$. Clearly, the function $\tilde{\ }$ maps the identity ι on $\mathcal{G}_{D \smallsetminus (\mathsf{mem}_{\mathcal{R}}(H) \cup \mathsf{mem}_{\mathcal{L}}(H))}$ 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

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

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 show 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 $\tau \in \mathcal{G}_{D \smallsetminus (\mathsf{mem}_{\mathcal{R}}(H) \cup \mathsf{mem}_{\mathcal{L}}(H))}$ 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. $\qquad\square$

It is known that any $=_{\mathcal{H}}$-class H of a monoid has the same cardinality of the associated Schützenberger group $\Gamma(H)$ (see, for instance, [30]). This implies the following crucial property:

Corollary 2.17. *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 2.16, we can associate with H an orbit-finite data group $(\Gamma(H), \circ, \tilde{\ })$, where $\Gamma(H)$ is the Schützenberger group of H. Lemma 2.14 implies that $\Gamma(H)$ is finite. Finally, since the $=_{\mathcal{H}}$-class H has the same cardinality as its Schützenberger group $\Gamma(H)$, we conclude that H is finite. $\qquad\square$

We are now ready to prove that every memorable value is either $\mathcal{R}$-memorable or $\mathcal{L}$-memorable:

Proof of Proposition 2.12. Let s be an element of an orbit-finite data monoids $\mathcal{M}$. We aim at proving that $\mathsf{mem}(s) \subseteq \mathsf{mem}_{\mathcal{R}}(s) \cup \mathsf{mem}_{\mathcal{L}}(s)$. Assume towards a contradiction that there is a value $d \in \mathsf{mem}(s) \smallsetminus (\mathsf{mem}_{\mathcal{R}}(s) \cup \mathsf{mem}_{\mathcal{L}}(s))$. Since $d \notin \mathsf{mem}_{\mathcal{R}}(s)$, there is an s' in the $=_{\mathcal{R}}$-class of s such that $d \notin \mathsf{mem}(s')$. Then there are elements $u, u' \in \mathcal{M}$ such that $s \cdot u = s'$ and $s' \cdot u' = s$. Symmetrically, as $d \notin \mathsf{mem}_{\mathcal{L}}(s)$, s has an $\mathcal{L}$-equivalent element s'' such that $d \notin \mathsf{mem}(s'')$, 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', $\tau_i(s') = s'$ and hence $\tau_i(s \cdot u) = \tau_i(s') = s'$. Combining this with $s' \cdot u' = s$, we obtain $\tau_i(s) \cdot \tau_i(u) \cdot u' = s' \cdot u' = s$ and hence $\tau_i(s) \geq_{\mathcal{R}} s$. Similarly, one proves that $\tau_i(s) \leq_{\mathcal{R}} s$ and hence $\tau_i(s) =_{\mathcal{R}} s$. By symmetry, one gets $\tau_i(s) =_{\mathcal{L}} s$. We have just shown that $\tau_i(s)$ belongs to the $=_{\mathcal{H}}$-class of s.

Towards a conclusion, we recall that $d \in \mathsf{mem}(s)$ and $d \notin \mathsf{mem}(\tau_i(s))$, and hence s is different from $\tau_i(s)$. Similarly, for all $j \neq i$, we have that $d_i \in \mathsf{mem}(\tau_i(s))$ and $d_i \notin \mathsf{mem}(\tau_j(s))$, and hence $\tau_i(s)$ is different from $\tau_j(s)$. We must conclude that the $=_{\mathcal{H}}$-class of s is infinite, contradicting Corollary 2.17. $\qquad\square$

3. Rigidly guarded MSO$^{\sim}$ and its variants

From now on, we abbreviate by *MSO$^{\sim}$* the monadic second-order logic extended with data equality tests. Formally, MSO$^{\sim}$ formulas are built up from atoms of the form $x < y$, $x \in X$, or $x \sim y$, $a(x)$, where a ranges over a fixed finite alphabet, using boolean connectives and existential quantifications over first-order variables (e.g., $x, y, z, \ldots$) and monadic second-order variables (e.g., $X, Y, Z, \ldots$). The meaning of the 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. We write $u \vDash \varphi$ whenever a formula φ holds over the data word u. Moreover, we write $\varphi(x_1, \ldots, y_n, X_1, \ldots, X_m)$ whenever we want to make explicit that the free variables of φ are among $x_1, \ldots, x_n, X_1, \ldots, X_m$.

We recall that the satisfiability problem for MSO$^{\sim}$ interpreted over data words is undecidable. Intuitively, the source of undecidability lies in the fact that the equality relationships between the values in a data word may form a grid-like structure onto which one can

encode computations of Turing machines [29]. Here we pursue the idea of restricting the use of data equality tests in MSO$^\sim$ formulas with the general goal of excluding the possibility of logically defining grids inside the input data words. We will see that this approach can be used to rule out the source of undecidability of MSO even when the logic is interpreted over classes of graphs that contain grid-minors of unbounded size [32].

The general idea is to guard the data equality predicate $x \sim y$ by a formula $\varphi(x, y)$ that is *rigid*, in the sense that it defines a partial bijection on the positions of every input data word. This gives rise to a fragment of MSO$^\sim$ that we call rigidly guarded MSO$^\sim$:

Definition 3.1. The logic *rigidly guarded MSO$^\sim$* consists of formulas generated according to the following grammar:
$$\varphi \; \coloneqq \; \exists x \; \varphi \; \mid \; \exists Y \; \varphi \; \mid \; a(x) \; \mid \; x < y \; \mid \; x \in Y \; \mid \; \neg\varphi \; \mid \; \varphi \wedge \varphi \; \mid \; \varphi_{\mathsf{rigid}}(x, y) \wedge x \sim y$$
where a ranges over a fixed finite alphabet A and $\varphi_{\mathsf{rigid}}(x, y)$ denotes a formula generated by the same grammar that in addition satisfies the rigidity constraint, that is, for all data words $u \in (D \times A)^*$ and all positions x (resp., y) in u, there is *at most one* position y (resp., x) in u such that $u \vDash \varphi_{\mathsf{rigid}}(x, y)$.
We call *rigidly guarded FO$^\sim$* the first-order fragment of rigidly guarded MSO$^\sim$.

The notion of rigidity is a semantic property, and this may seem problematic. However, we can enforce rigidity syntactically as follows. Instead of guarding a data test by a generic rigid formula $\varphi_{\mathsf{rigid}}(x, y)$, one uses the new guard
$$\tilde{\varphi}_{\mathsf{rigid}}(x, y) \; =^{\mathrm{def}} \; \varphi_{\mathsf{rigid}}(x, y) \; \wedge \; \forall x', y' \; \varphi_{\mathsf{rigid}}(x', y') \to (x = x' \leftrightarrow y = y') \; .$$
It is easy to check that $\tilde{\varphi}_{\mathsf{rigid}}$ is always rigid and, furthermore, if the original formula φ_{rigid} is rigid, then it is also equivalent to $\tilde{\varphi}_{\mathsf{rigid}}$. This trick allows us to enforce rigidity syntactically. We will prove later in Corollary 3.5 that one can decide if a given formula respects the semantic assumption of rigidity in all its guards (the problem is of course undecidable when data tests are not guarded).

We also remark that in rigidly guarded MSO$^\sim$, the similar constructions $\varphi_{\mathsf{rigid}}(x, y) \wedge x \nsim y$, $\varphi_{\mathsf{rigid}}(x, y) \to x \sim y$, and $\varphi_{\mathsf{rigid}}(x, y) \to x \nsim y$ can be derived. This is thanks to the Boolean equivalences $\alpha \to \beta$ iff $\alpha \to (\alpha \wedge \beta)$, $\alpha \wedge \neg\beta$ iff $\neg(\alpha \to \beta)$, and $\alpha \to \neg\beta$ iff $\neg(\alpha \wedge \beta)$.

Example 3.2. We show how to define in rigidly guarded FO$^\sim$ 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 that the input data word is not empty, e.g., by the sentence $\exists x$ true. If $k = 2$ it is sufficient to check the existence of two distinct *consecutive* data values, e.g., by the sentence $\exists x, y \; (x{+}1 = y) \wedge x \nsim y$. 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 $L_{\geq k-2} \smallsetminus L_{\geq k-1}$, which is definable in rigidly guarded FO$^\sim$ thanks to the inductive hypothesis. Moreover, the formula $\varphi(x', y')$ that defines the endpoints $x' = x + 1$ and $y' = y - 1$ of a *maximal* factor in $L_{\geq k-2} \smallsetminus L_{\geq k-1}$ is rigid. We can thus define the language $L_{\geq k}$ by means of the rigidly guarded FO$^\sim$ sentence $\exists x, y \; \varphi(x + 1, y - 1) \wedge x \nsim y$.

Rigidly guarded MSO$^\sim$ will be the main object of our study. As we already mentioned, in Sections 4 and 5 we will show that the data languages definable in rigidly guarded MSO$^\sim$ are exactly the languages recognizable by orbit-finite data monoids. This result, which is

interesting in its own right, also implies that rigidly guarded MSO~ has a decidable satisfiability problem over the class of data words: satisfiability indeed reduces to the problem of checking emptiness of data languages recognized by orbit-finite data monoids. However, one can prove decidability of rigidly guarded MSO~ by a more direct (and general) argument. In the following, we outline the argument underlying decidability of a logic that is even more expressive than rigidly-guarded MSO~ and that is interpreted over generic classes of *relational structures with data values*. Formally, for a fixed signature consisting of m relational symbols $R_1, \ldots, R_m$, we consider structures of the form $\mathcal{S} = (U, R_1^{\mathcal{S}}, \ldots, R_m^{\mathcal{S}}, \lambda)$, where U is the universe of the structure, $R_i^{\mathcal{S}}$ is a relation over U of the same arity as R_i, say $R_i^{\mathcal{S}} \subseteq U^{k_i}$, and $\lambda : U \to D$ is a labelling function associating data values with the elements of $\mathcal{S}$. Similar structures have been considered for example in [2, 25].

To define the variant of rigidly-guarded MSO~, we relax the rigidity constraint. Given a class $\mathscr{C}$ of relational structures with data values and given a generic formula $\varphi(x, y)$ interpretable over $\mathscr{C}$, we say that $\varphi(x, y)$ is *semi-rigid* (with respect to $\mathscr{C}$) if for every relational structure $\mathcal{S} = (S, R_1, \ldots, R_m, \lambda)$ in $\mathscr{C}$ and every element $x \in S$, there is *at most one* vertex $y \in S$ such that $\mathcal{S} \vDash \varphi(x, y)$. As we already seen before, semi-rigidity can be enforced syntactically over any class of data graphs, so it is not really important here under which class of structures the guards are assumed to be semi-rigid. We define below the variant of MSO~ in which data tests are guarded by semi-rigid formulas.

Definition 3.3. The logic *semi-rigidly guarded MSO~*, interpreted over a class $\mathscr{C}$ of relational structures with data values, consists of formulas generated by the following grammar:

$$\varphi := \exists x \, \varphi \, \mid \, \exists Y \, \varphi \, \mid \, R_i(x_1, \ldots, x_{k_i}) \, \mid \, x \in Y \, \mid \, \neg\varphi \, \mid \, \varphi \wedge \varphi$$
$$\mid \, \varphi_{\mathsf{semirigid}}(x, y) \, \wedge \, \varphi_{\mathsf{semirigid}}(x, z) \, \wedge \, y \sim z$$

where the occurrences of $\varphi_{\mathsf{semirigid}}(x, y)$ and $\varphi_{\mathsf{semirigid}}(x, z)$ above denote possibly different formulas generated by the same grammar, sharing a common variable x, and being semi-rigid (w.r.t. $\mathscr{C}$).

Clearly, rigidly guarded MSO~ can be seen as a fragment of semi-rigidly guarded MSO~ over the class of data words. The interesting feature of these logics is that their satisfiability problems can be reduced to the satisfiability problem for classical MSO over the same classes of structures, where of course data values become immaterial. For example, one can decide whether a given semi-rigidly guarded MSO~ formula is satisfiable over the class of all (finite or infinite) data words/trees.

Theorem 3.4. *Let $\mathscr{C}$ be a class of relational structures for which membership of a structure $\mathcal{S}$ in $\mathscr{C}$ does not depend on labelling of the elements of $\mathcal{S}$ by data values. The satisfiability problem of semi-rigidly guarded MSO~ over $\mathscr{C}$ is reducible to the satisfiability problem of classical MSO over the same class $\mathscr{C}$.*

Proof. We begin by making the following crucial remark: every semi-rigidly guarded data test $\alpha(x, y) \, \wedge \, \beta(x, z) \, \wedge \, y \sim z$ can be normalized into the formula

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

This formalizes the idea that semi-rigidly guarded data tests behave almost like unary predicates. It is indeed possible to transform a given semi-rigidly guarded MSO~ formula φ

into a classical MSO formula φ^- inductively as follows:

$$(\exists x \ \psi)^- \ =^{\mathrm{def}} \ \exists x \ \psi^- \qquad\qquad (x \in Y)^* \ =^{\mathrm{def}} \ x \in Y$$

$$(\exists Y \ \psi)^- \ =^{\mathrm{def}} \ \exists Y \ \psi^- \qquad\qquad (\neg\psi)^- \ =^{\mathrm{def}} \ \neg\psi^-$$

$$(R_i(x_1,\ldots,x_{k_i}))^- \ =^{\mathrm{def}} \ R_i(x_1,\ldots,x_{k_i}) \qquad\qquad (\psi_2 \wedge \psi_2)^- \ =^{\mathrm{def}} \ \psi_1^- \wedge \psi_2^-$$

and, most importantly,

$$\big(\alpha(x,y) \ \wedge \ \beta(x,z) \ \wedge \ y \sim z\big)^- \ =^{\mathrm{def}} \ \alpha^-(x,y) \ \wedge \ \beta^-(x,z) \ \wedge \ x \in c_{\alpha,\beta}^{\sim}$$

where $c_{\alpha,\beta}^{\sim}$ is a fresh unary predicate.

Let φ be a sentence for which we want to decide satisfiability. For the sake of brevity, let $C = \{c_1,\ldots,c_n\}$ be the set of unary predicates $c_{\alpha,\beta}^{\sim}$ that correspond to the normalized semi-rigidly guarded data tests $\gamma_{\alpha,\beta}^{\sim}(x)$ occurring in φ. Given a relational structure with data values $\mathcal{S}$, we denote by $\mathcal{S}^-$ the relational structure without data values that is obtained from $\mathcal{S}$ by removing the labelling function and by expanding the structure with the predicates $c_{\alpha,\beta}^{\sim} \in C$ in such a way that

$$\mathcal{S}^- \vDash c_{\alpha,\beta}^{\sim}(x) \qquad \text{iff} \qquad \mathcal{S} \vDash \gamma_{\alpha,\beta}^{\sim}(x) \ .$$

Clearly, for every relational structure $\mathcal{S}$ with data values, we have that $\mathcal{S} \vDash \varphi$ iff $\mathcal{S}^- \vDash \varphi^-$.

Now, it is possible to characterize the class of relational structures without data values of the form $\mathcal{S}^-$ without taking into account the data values in $\mathcal{S}$. For this it is sufficient to test whether the universe of the structure can be partitioned into classes in such a way that, for every element x, x satisfies $c_{\alpha,\beta}^{\sim}$ iff there exist (unique) elements y and z in the same class that satisfy $\alpha^-(x,y)$ and $\beta^-(x,z)$, respectively. If such a partition exists, then one can reconstruct the corresponding relational structure with data values $\mathcal{S}$ by assigning different data values to elements of different classes. Conversely, if the relational structure is known to be of the form $\mathcal{S}^-$, then a partition can be found by simply grouping the elements of $\mathcal{S}$ having the same data value. Moreover, one can easily see that the coarsest partitions satisfying the above property contain at most n classes (recall that n is the number of occurrences of semi-rigidly guarded data tests in our sentence φ). This allows us to define the class of relational structures of the form $\mathcal{S}^-$ by means of a simple MSO formula:

$$\varphi_{\mathsf{partition}} \ =^{\mathrm{def}} \ \exists Z_1,\ldots,Z_n \bigwedge_{1 \le i < j \le n} (Z_i \cap Z_j = \varnothing)$$

$$\wedge \ \forall x \ \Big(c_{\alpha,\beta}^{\sim}(x) \ \leftrightarrow \ \exists y,z \ \alpha^-(x,y) \wedge \beta^-(x,z) \wedge \bigvee_{1 \le i \le n} (y \in Z_i \wedge z \in Z_i) \Big) \ .$$

Putting everything together, we have that a relational structure with data values $\mathcal{S} \in \mathscr{C}$ satisfies the semi-rigidly guarded MSO$^{\sim}$ sentence φ iff $\mathcal{S}$ (or any other relational structure that differs from $\mathcal{S}$ only in the data values) satisfies the MSO sentence

$$\varphi^= \ =^{\mathrm{def}} \ \exists c_1,\ldots,c_n \ \varphi^- \ \wedge \ \varphi_{\mathsf{partition}} \ . \qquad\qquad \square$$

Corollary 3.5. *The satisfiability problem for semi-rigidly guarded MSO$^{\sim}$ over the class of data trees is decidable. Moreover, one can decide whether a given formula belongs to semi-rigidly guarded MSO$^{\sim}$, or even belongs to rigidly guarded MSO$^{\sim}$.*

Proof. By Theorem 3.4, the satisfiability problem for semi-rigidly guarded MSO~ over the class of all data trees is reduced to the satisfiability problem for classical MSO over trees, which is known to be decidable [31].

We explain how to decide whether a given formula φ belongs to semi-rigidly guarded MSO~ (a similar argument can be used to test membership in rigidly guarded MSO~). For this it is sufficient to check, in bottom-up manner, that every sub-formula of φ satisfies the syntactic and semantic restrictions enforced by the grammar of semi-rigidly guarded MSO~. In particular, if $y \sim z$ is a data test that occurs as a sub-formula of φ, one needs to check that (i) this test is guarded by a conjunction of two formulas $\alpha(x, y)$ and $\beta(x, z)$ and (ii) both sub-formulas $\alpha(x, y)$ and $\beta(x, z)$ are semi-rigid. Assuming that the sub-formula $\alpha(x, y)$ is already known to belong to semi-rigidly guarded MSO~, one can decide semi-rigidity of $\alpha(x, y)$ by testing the validity of the sentence

$$\alpha_{\mathsf{semirigid?}} \ =^{\mathrm{def}} \ \forall x, y, y' \ \alpha(x, y) \wedge \alpha(x, y') \ \rightarrow \ y = y' \ .$$

A similar test can be performed on the sub-formula $\beta(x, z)$. $\square$

4. From rigidly guarded MSO~ to orbit-finite 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.

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 upper-case 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, 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

$$\llbracket \varphi \rrbracket \ = \ \big\{ \langle w, U_1, \ldots, U_m \rangle \ : \ w \in (D \times A)^*, \ U_1, \ldots, U_m \subseteq \mathsf{dom}(w), \ w \vDash \varphi(U_1, \ldots, U_m) \big\}$$

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

The principle of the proof is to establish that, given a rigidly guarded MSO~ formula $\varphi(\bar{X})$, the language $\llbracket \varphi \rrbracket$ is recognized by an orbit-finite data monoid. Though this statement is true, it is convenient to strengthen it in order to be able to use it as the invariant of an inductive proof based on φ. 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. This is why our induction hypothesis needs to be stronger, namely, need to state that $\llbracket \varphi \rrbracket$ is recognized by an orbit-finite data monoid via a *projectable* morphism, as defined just below.

Definition 4.1. Let h be a morphism from the free data monoid $(D \times A \times B^m)^*$ to a data monoid $\mathcal{M}$. We say that h is *projectable* (*over B^m*) if for all data words $w \in (D \times A)^*$ and

all tuples of predicates $\bar{U} = (U_1, \ldots, U_m)$ and $\bar{V} = (V_1, \ldots, V_m)$,

$$h(\langle w, \bar{U} \rangle) \overset{\circ}{=} h(\langle w, \bar{V} \rangle) \qquad \text{implies} \qquad h(\langle w, \bar{U} \rangle) = h(\langle w, \bar{V} \rangle)$$

where $s \overset{\circ}{=} t$ means that the elements s and t are in the same orbit of $\mathcal{M}$.

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

Theorem 4.2. *For all rigidly guarded MSO~ formulas $\varphi(\bar{X})$, the language $[\![\varphi]\!]$ is effectively recognized by an orbit-finite data monoid via a projectable morphism.*

From the above theorem we obtain, in particular, the following corollary.

Corollary 4.3. *Every data language definable in rigidly guarded MSO~ (resp., rigidly guarded FO~) is effectively recognized by an orbit-finite data monoid (resp., aperiodic orbit-finite data monoid).*

Proof. The case of rigidly guarded MSO~ corresponds just to Theorem 4.2 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 recall a result from [6] which proves that every data language definable in FO~ (non-necessarily rigidly guarded FO~) is recognized by an aperiodic data monoid. In particular, if we consider the syntactic data monoid of a language definable in rigidly guarded FO~, we easily see that it is aperiodic thanks to the result in [6] and, moreover, has finitely many orbits since it is the quotient of an orbit-finite data monoid obtained from Theorem 4.2. $\square$

Before entering the details, we give an overview of the proof of Theorem 4.2, which 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 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. We recall that the standard powerset construction returns new elements that are sets of elements from the original monoid. In general, due to the presence of infinitely many elements in a data monoid, the standard powerset construction may remember sets of unbounded size, possibly resulting in a data monoid that has infinitely many orbits, even if the original data monoid has finitely many of them. In our case, however, because the morphism is projectable, it is sufficient to apply a variant of the powerset construction that remembers at most one element for each orbit of the original monoid, thus producing an orbit-finite data monoid.

The most technical part of the proof 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: 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. In this case, we use the rigidity property for "normalizing" the data monoid, allowing the construction to go through.

In addition, for the translation to be effective, we need to compute the results of algebraic operations on orbit-finite data monoids, such as product, projection, and subset construction. It turns out that all the operations that are needed in the proof are compatible with the operation of *finite restriction* that we introduced in Definition 2.5. As

an example, for every two orbit-finite data monoids $\mathcal{M}$ and $\mathcal{N}$ and for every finite subset C of D, we have that the product $\mathcal{M} \times \mathcal{N}$ is an orbit-finite data monoid and, moreover, $(\mathcal{M} \times \mathcal{N})|_C = \mathcal{M}|_C \times \mathcal{N}|_C$. It follows from Proposition 2.6 that we can compute a (finite) representation of the result of an algebraic construction starting from some given (finite) 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 purely algebraic constructions. In particular, by a slight abuse of terminology, we will say that an orbit-finite data monoid $\mathcal{N}$ is *computed* from other orbit-finite data monoids $\mathcal{M}_1, \ldots, \mathcal{M}_n$ when a representation of $\mathcal{N}$ can be obtained effectively from some representations of $\mathcal{M}_1, \ldots, \mathcal{M}_n$. 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 g can be computed from other morphisms $h_1, \ldots, h_n$ when the images via g of all singleton words can be obtained effectively from the images via $h_1, \ldots, h_n$ of all singleton words (note that there exist only finitely many images of singleton words up to renamings).

We begin by describing the translation of the existential closures of formulas (hereafter, all formulas are meant to be rigidly guarded MSO~ formulas).

Lemma 4.4. *Let $\psi(\bar{X}, X_{m+1})$ be a formula and let $\varphi(\bar{X}) = \exists X_{m+1} \ \psi(\bar{X}, X_{m+1})$. If $[\![\psi]\!]$ is recognized by an orbit-finite data monoid $\mathcal{M}$ via a projectable morphism $h : (D \times A \times B^{m+1})^* \to \mathcal{M}$, then one can compute an orbit-finite data monoid $\mathcal{N}$ and a projectable morphism $g : (D \times A \times B^m)^* \to \mathcal{N}$ recognizing $[\![\varphi]\!]$.*

Proof. For the sake of brevity, we denote by L the language over $D \times A \times B^{m+1}$ that is defined by the formula $\psi(\bar{X}, X_{m+1})$, and by $\exists L$ the language over $D \times A \times B^m$ that is defined by $\varphi(\bar{X}) = \exists X_{m+1} \ \psi(\bar{X}, X_{m+1})$. We assume that L is recognized by an orbit-finite data monoid $\mathcal{M}$ via a morphism h. We will apply a variant of the powerset construction to the orbit-finite data monoid $\mathcal{M}$ to obtain an orbit-finite data monoid $\mathcal{N}$ that recognizes $\exists L$. The same construction can be applied to any finite restriction $\mathcal{M}|_C$ that represents $\mathcal{M}$, so as to compute a restriction $\mathcal{N}|_C$ that represents $\mathcal{N}$. We observe, however, that the cardinality of the set C must be at least twice the maximal size of the memories of the elements of $\mathcal{N}$.

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

- the elements of N are the subsets of M that contain only pairwise orbit-distinct elements, namely, those sets $S \subseteq M$ such that for all $s, s' \in S$, $s \stackrel{\mathrm{o}}{=} s'$ implies $s = s'$;

- the product $\odot$ is defined on pairs of sets $S, T \in N$ by

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

  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, for all $S \in M'$,

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

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 $\mathcal{N}$, and the singleton $\{1_{\mathcal{M}}\}$ is the identity element in $\mathcal{N}$.

Below, we verify that the data monoid $\mathcal{N}$ is orbit-finite. Let n be the number of orbits of $\mathcal{M}$. We observe that every set $S \in N$ has cardinality at most n (indeed, if this were not

the case, then S would contain two distinct elements s and t such that $s \overset{\mathrm{o}}{=} t$, which would contradict the definition of N). From this property it follows that $\mathcal{N}$ is the projection of $\mathcal{M}^{\leq n}$ under some equivariant mapping, where $\mathcal{M}^{\leq n} = \biguplus_{i \leq n} \mathcal{M}^i$ and each $\mathcal{M}^i$ is the i-fold product of $\mathcal{M}$ with itself. Because orbit-finite sets are closed under products, finite disjoint unions, and images under equivariant mappings, we have that $\mathcal{N}$ is orbit-finite.

The morphism. We now define a morphism g from the the free data monoid $(D \times A \times B^m)^*$ to the orbit-finite data monoid $\mathcal{N}$. For every expanded data word $\langle w, U_1, \ldots, U_m \rangle$, we let

$$g(\langle w, U_1, \ldots, U_m \rangle) \ =^{\mathrm{def}} \ \big\{ h(\langle w, U_1, \ldots, U_m, U_{m+1} \rangle) \ : \ U_{m+1} \subseteq \mathsf{dom}(w) \big\}$$

(note that, since h is projectable, then $g(\langle w, U_1, \ldots, U_m \rangle)$ contains only pairwise orbit-distinct elements and hence it is an element of the data monoid $\mathcal{N}$).

We verify that the morphism g is projectable. Consider a data word w and some tuples of predicates $\bar{U} = U_1, \ldots, U_m$ and $\bar{V} = V_1, \ldots, V_m$, and suppose that $g(\langle w, \bar{U} \rangle) \overset{\mathrm{o}}{=} g(\langle w, \bar{V} \rangle)$. This means that there is a renaming τ such that

$$g(\langle w, \bar{V} \rangle) \ = \ \check{\tau}\big(g(\langle w, \bar{U} \rangle) \big).$$

Moreover, by definition of g, we have

$$\big\{ h(\langle w, \bar{V}, V_{m+1} \rangle) \ : \ V_{m+1} \subseteq \mathsf{dom}(w) \big\} \ = \ \big\{ \hat{\tau}(h(\langle w, \bar{U}, U_{m+1} \rangle)) \ : \ U_{m+1} \subseteq \mathsf{dom}(w) \big\}.$$

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

Recognizability. It remains to prove that the language $\exists L$ is recognized by $\mathcal{N}$ via the morphism g. For the sake of brevity, let $F = h(L)$ and $G = g(\exists L)$. We consider an expanded data word $\langle w, \bar{U} \rangle \in (D \times A \times B^m)^*$ and we prove that $\langle w, \bar{U} \rangle \in \exists L$ iff $g(\langle w, \bar{U} \rangle) \in G$. The left-to-right implication is trivial, so we prove the converse implication. Suppose that $g(\langle w, \bar{U} \rangle) \in G$. Since $G = g(\exists L)$, we know that there is an expanded data word $\langle w', \bar{V} \rangle \in \exists L$ such that $g(\langle w, \bar{U} \rangle) = g(\langle w', \bar{V} \rangle)$. From the definition of g we also know that

$$\big\{ h(\langle w, \bar{U}, U_{m+1} \rangle) \ : \ U_{m+1} \subseteq \mathsf{dom}(w) \big\} \ = \ \big\{ h(\langle w', \bar{V}, V_{m+1} \rangle) \ : \ V_{m+1} \subseteq \mathsf{dom}(w') \big\} \ .$$

Moreover, from the definition of $\exists L$ we know that $(\langle w', \bar{V}, V_{m+1} \rangle) \in L$ for some unary predicate $V_{m+1} \subseteq \mathsf{dom}(w')$. Finally, since L is recognized by $\mathcal{M}$ via the morphism h and since $\langle w', \bar{V}, V_{m+1} \rangle$ belongs to L, we have $h(\langle w', \bar{V}, V_{m+1} \rangle) \in F$ and hence $h(\langle w, \bar{U}, U_{m+1} \rangle) \in F$ for some unary predicate $U_{m+1} \subseteq \mathsf{dom}(w)$. This shows that $\langle w, \bar{U}, U_{m+1} \rangle \in L$ and hence $\langle w, \bar{U} \rangle \in \exists L$. $\qquad\square$

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

Lemma 4.5. *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 $\mathcal{M}$ 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 the image via h of the free data monoid $(D \times A \times B^2)^*$, which is a data sub-monoid of $\mathcal{M}$, can be computed from $\mathcal{M}$ and h:

Claim 4.6. *From the orbit-finite data monoid $\mathcal{M}$ and the morphism $h : (D \times A \times B^2)^* \to \mathcal{M}$, one can compute the data sub-monoid $h((D \times A \times B^2)^*)$.*

Proof of claim. Suppose that the orbit-finite data monoid $\mathcal{M}$ is represented by its restriction $\mathcal{M}|_C$, for some finite subset C of D such that $|C| \geq 2\|\mathcal{M}\|$. Let $\mathcal{M}' = h\big((D \times A \times B^2)^*\big)$ be the data sub-monoid induced by h. Clearly, we have $\|\mathcal{M}'\| \leq \|\mathcal{M}\|$ and hence, by Proposition 2.6, 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 $h\big((C \times A \times B^2)^*\big)$, which is computable from $\mathcal{M}|_C$ and $h|_{C \times A \times B^2}$. 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 $h\big((C \times A \times B^2)^*\big)$. This shows that $\mathcal{M}'|_C$ can be computed from $\mathcal{M}|_C$ and $h|_{C \times A \times B^2}$. □

Thanks to above claim, we can assume, without loss of generality, that h is a surjective morphism. Unfortunately, even under this assumption, the property of projectability is not straightforwardly preserved when we translate the morphism h for the rigid guard $\varphi(x,y)$ to a morphism for the rigidly guarded comparison $\varphi(x,y) \wedge x \sim y$. For this, we must derive from the rigidity assumption on $\varphi(x,y)$ a stronger notion of projectability, which is defined below and which is called 0-reduced projectability.

An element s of a data monoid $\mathcal{N}$ is a *null* if $s \cdot t = t \cdot s = s$ for all elements t of $\mathcal{N}$. If a data monoid has a null element, then this element is unique, and in this case it is denoted by $0_{\mathcal{N}}$. Moreover, it is easy to see that if a language L is recognized by an orbit-finite data monoid, then L is also recognized by an orbit-finite data monoid with a null element.

Definition 4.7. Let h be a morphism from the free data monoid $(D \times A \times B^2)^*$ to a data monoid $\mathcal{N}$ with null element $0_{\mathcal{N}}$. We say that h is 0-*reduced* if for all data words $w \in (D \times A)^*$ and positions $x, x', y, y' \in \mathsf{dom}(w)$, the following implications hold:

- if $h(\langle w, \{x\}, \varnothing \rangle) = h(\langle w, \{x'\}, \varnothing \rangle)$, then $x = x'$ or $0_{\mathcal{N}} = h(\langle w, \{x\}, \varnothing \rangle) = h(\langle w, \{x'\}, \varnothing \rangle)$
- if $h(\langle w, \varnothing, \{y\} \rangle) = h(\langle w, \varnothing, \{y'\} \rangle)$, then $y = y'$ or $0_{\mathcal{N}} = h(\langle w, \varnothing, \{y\} \rangle) = h(\langle w, \varnothing, \{y'\} \rangle)$.

Below, we show that the data language L is equally recognized by an orbit-finite data monoid with null element and a morphism that is surjective, projectable, and 0-reduced (for simplicity, we call it a 0-reduced projectable morphism).

Claim 4.8. From the orbit-finite data monoid $\mathcal{M}$ and the projectable surjective morphism $h : (D \times A \times B^2)^* \to \mathcal{M}$ recognizing the language $L = [\![\varphi(x,y)]\!]$ of the rigid formula $\varphi(x,y)$, one can compute an orbit-finite data monoid $\mathcal{N}$ with null element $0_{\mathcal{N}}$ and a 0-reduced projectable morphism $g : (D \times A \times B^2)^* \to \mathcal{N}$ recognizing the same language L.

Proof of claim. The desired orbit-finite data monoid $\mathcal{N}$ is obtained from a suitable quotient of $\mathcal{M}$, precisely, by collapsing those elements of $\mathcal{M}$ that do not represent factors of data words in L. As usual, the construction can be applied effectively to a restriction $\mathcal{M}|_C$ that represents $\mathcal{M}$, thus obtaining a representation $\mathcal{N}|_C$ of $\mathcal{N}$.

Collapsing bad elements. Let $F = h(L)$ and let G be the maximal set of all elements such that $M \cdot G \cdot M \cap F = \varnothing$. Intuitively, G contains those elements of $\mathcal{M}$ that cannot be extended to 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, $\tau(G) \subseteq G$ 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 $s, t \in G$. Note that $\approx_G$ is a congruence with respect to the product of $\mathcal{M}$, namely, if $s \approx_G t$ and $u \approx_G v$, then $s \cdot u \approx_G t \cdot v$. The equivalence $\approx_G$ is also compatible with the action of renamings, namely, if $s \approx_G t$, then $\tau(s) \approx_G \tau(t)$ for all renamings τ. This allows us to define $\mathcal{N}$ as the quotient

of $\mathcal{M}$ with respect to $\approx_G$, where the elements are the $\approx_G$-equivalence classes, the product is defined by

$$[s]_{\approx_G} \odot [t]_{\approx_G} =^{\text{def}} [s \cdot t]_{\approx_G}$$

and the action of renamings is defined by

$$\tau([s]_{\approx_G}) =^{\text{def}} [\tau(s)]_{\approx_G}$$

(note that the above functions are well defined).

Clearly $\mathcal{N}$ is an orbit-finite data monoid. Moreover, for all $s \in M \smallsetminus G$, the $\approx_G$-equivalence class of s is the singleton $\{s\}$. The only other element of $\mathcal{N}$ is the entire set G, which is also the null element, and is thus denoted by $0_\mathcal{N}$.

The morphism. We now define the morphism $g : (D \times A \times B^2)^* \to \mathcal{N}$ 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 $\mathcal{N}$ defined by

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

We recall that h and h_G are surjective morphisms, so h is also surjective. Moreover, since $h_G^{-1} \circ h_G$ is the identity on $F = h(L)$, we have

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

This shows that g is a surjective morphism recognizing the data language L.

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

$$g(\langle w, V_1, V_2 \rangle) = \tau(g(\langle w, U_1, U_2 \rangle)) = \tau(0_\mathcal{N}) = 0_\mathcal{N} = g(\langle w, U_1, U_2 \rangle) .$$

A similar conclusion can be obtained when $g(\langle w, V_1, V_2 \rangle) = 0_\mathcal{N}$.

In the remaining case, we assume that neither $g(\langle w, U_1, U_2 \rangle)$ nor $g(\langle w, V_1, V_2 \rangle)$ are the null element. We know from the definition of $\mathcal{N}$ that neither $h(\langle w, U_1, U_2 \rangle)$ nor $h(\langle w, V_1, V_2 \rangle)$ belong to the ideal G and hence $g(\langle w, U_1, U_2 \rangle) = \{h(\langle w, U_1, U_2 \rangle)\}$ and $g(\langle w, V_1, V_2 \rangle) = \{h(\langle w, V_1, V_2 \rangle)\}$. Moreover, we have

$$g(\langle w, V_1, V_2 \rangle) = \tau(g(\langle w, U_1, U_2 \rangle)) = \{\tau(h(\langle w, U_1, U_2 \rangle))\}$$

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

0-Reduced. It remains to prove that g is 0-reduced. Here, we exploit the fact that the language L is defined by a *rigid* formula $\varphi(x, y)$. Let $w \in (D \times A)^*$ be a data word and let $x, x' \in \mathsf{dom}(w)$ be two positions in it. By way of contradiction, assume that $x \neq x'$ and $g(\langle w, \{x\}, \varnothing \rangle) = g(\langle w, \{x'\}, \varnothing \rangle) \neq 0_\mathcal{N}$. We need to derive that $\varphi(x, y)$ is not rigid (the same conclusion can be obtained from the assumption that there exist two positions $y, y' \in \mathsf{dom}(w)$ such that $y \neq y'$ and $g(\langle w, \varnothing, \{y\} \rangle) = g(\langle w, \varnothing, \{y'\} \rangle) \neq 0_\mathcal{N}$). Since $g(\langle w, \{x\}, \varnothing \rangle) = g(\langle w, \{x'\}, \varnothing \rangle) \neq 0_\mathcal{N}$, we know that $h(\langle w, \{x\}, \varnothing \rangle) = h(\langle w, \{x'\}, \varnothing \rangle) \notin G$ and hence there exist $s, t \in M$ such that $s \cdot h(\langle w, \{x\}, \varnothing \rangle) \cdot t = s \cdot h(\langle w, \{x'\}, \varnothing \rangle) \cdot t \in F$. Moreover, since h is surjective, we know that there exist two expanded data words $\langle w', U_1, U_2 \rangle$

and $\langle w'', V_1, V_2 \rangle$ such that $h(\langle w', U_1, U_2 \rangle) = s$ and $h(\langle w'', V_1, V_2 \rangle) = t$. Since $\mathcal{M}$ and h recognize the language $L = [\![\varphi(x,y)]\!]$ and $F = h(L)$, we have that $\varphi(x,y)$ is satisfied both by the data word $\langle w', U_1, U_2 \rangle \langle w, \{x\}, \varnothing \rangle \langle w'', V_1, V_2 \rangle$ and by the data word $\langle w', U_1, U_2 \rangle \langle w, \{x'\}, \varnothing \rangle \langle w'', V_1, V_2 \rangle$. Finally, since $x \neq x'$, we conclude that $\varphi(x,y)$ is not rigid. This completes the proof of our claim. $\qquad\square$

We can now turn back to the proof of Lemma 4.5. By the above claim, we can assume that the language $L = [\![\varphi(x,y)]\!]$ is recognized by an orbit-finite data monoid $\mathcal{M}_1$ with null element $0_{\mathcal{M}_1}$ via a 0-reduced projectable morphism $h_1 : (D \times A \times B^2)^* \to \mathcal{M}_1$. Moreover, we can construct the syntactic data monoid $\mathcal{M}_2$ of the language defined by $x \sim y$ and the corresponding morphism $h_2 : (D \times A \times B^2)^* \to \mathcal{M}_2$ recognizing $[\![x \sim y]\!]$. We observe that the data monoid $\mathcal{M}_2$ has finitely many orbits and its elements can be identified with terms of one the following forms:

(1) $o(\varepsilon)$, which plays the role of the identity in $\mathcal{M}_2$ and corresponds to the image under h_2 of the empty data word;

(2) $p(d)$, for any $d \in D$, which corresponds to the image under h_2 of data words w expanded with a singleton predicate $U = \{x\}$, where $w(x) = d$, and with the empty predicate $V = \varnothing$;

(3) $q(d)$, for any $d \in D$, which corresponds to the image under h_2 of data words expanded with the empty predicate $U = \varnothing$ and with a singleton predicate $V = \{y\}$, where $w(y) = d$;

(4) $r(\varepsilon)$, with corresponds to the image under h_2 of the expanded data words that satisfy $x \sim y$;

(5) $s(\varepsilon)$, which plays the role of the null element in $\mathcal{M}_2$ and corresponds to the image under h_2 of data words expanded with two non-empty predicates U, V that do not satisfy $x \sim y$.

For example, we have $p(d) \odot q(d) = r(\varepsilon)$ and $p(d) \odot q(e) = s(\varepsilon)$, for all pairs of distinct values $d, e \in D$. We also observe that the morphism h_2 is not projectable, which explains why, in order to recognize the intersection of the data languages $L = [\![\varphi(x,y)]\!]$ and $[\![x \sim y]\!]$, we introduce below a variant of the product of data monoids.

The 0-collapse product. The orbit-finite data monoid $\mathcal{M}'$ for the formula $\varphi(x,y) \wedge x \sim y$ is defined using a suitable variant of the product of data monoids with null elements, which we call 0-*collapse product* (strictly speaking, the 0-collapse product is a special form of semi-direct product). Formally, let $\mathcal{M}_1 = (M_1, \cdot, \hat{\ })$ and $\mathcal{M}_2 = (M_2, \odot, \check{\ })$. We define $\mathcal{M}' = (M', \odot, \tilde{\ })$, where

- M' consists of all pairs $(s_1, s_2) \in M_1 \times M_2$ such that $s_1 = 0_{\mathcal{M}_1}$ implies $s_2 = 0_{\mathcal{M}_2}$;

- for every $(s_1, s_2), (t_1, t_2) \in M'$, the product $(s_1, s_2) \odot (t_1, t_2)$ is either the pair $(s_1 \cdot t_1, s_2 \odot t_2)$ or the pair $(0_{\mathcal{M}_1}, 0_{\mathcal{M}_2})$, depending on whether $s_1 \cdot t_1 \neq 0_{\mathcal{M}_1}$ or $s_1 \cdot t_1 = 0_{\mathcal{M}_1}$;

- $\tilde{\tau}(s_1, s_2) = (\hat{\tau}(s_1), \check{\tau}(s_2))$ for all all renamings τ and all $(s_1, s_2) \in M'$.

Clearly, $\mathcal{M}'$ is an orbit-finite data monoid.

The morphism. Accordingly, we define the morphism h' that maps any expanded data word $w \in (D \times A \times B^2)^*$ either to the pair $(h_1(w), h_2(w))$ or to the pair $(0_{\mathcal{M}_1}, 0_{\mathcal{M}_2})$, depending on whether $h_1(w) \neq 0_{\mathcal{M}_1}$ or $h_1(w) = 0_{\mathcal{M}_1}$. Clearly, h' recognizes the language $[\![\varphi(x,y) \wedge x \sim y]\!]$.

Projectability. It remains to prove that the morphism h' is projectable. Consider a data word $w \in (D \times A)^*$ and some predicates $U_1, U_2, V_1, V_2 \subseteq \mathsf{dom}(w)$, and suppose that the

elements $h'(\langle w, U_1, U_2 \rangle)$ and $h'(\langle w, V_1, V_2 \rangle)$ are in the same orbit. We distinguish between the case where $h_1(\langle w, U_1, U_2 \rangle) = 0_{\mathcal{M}_1}$ (and hence $h_1(\langle w, V_1, V_2 \rangle) = 0_{\mathcal{M}_1}$ as well) and the case where $h_1(\langle w, U_1, U_2 \rangle) \neq 0_{\mathcal{M}_1}$ (and hence $h_1(\langle w, V_1, V_2 \rangle) \neq 0_{\mathcal{M}_1}$ as well). In the former case, we immediately get

$$h'(\langle w, U_1, U_2 \rangle) = (0_{\mathcal{M}_1}, 0_{\mathcal{M}_2}) = h'(\langle w, V_1, V_2 \rangle) \,.$$

In the latter case, we have $h'(\langle w, U_1, U_2 \rangle) = \big(h_1(\langle w, U_1, U_2 \rangle), h_2(\langle w, U_1, U_2 \rangle)\big)$ and $h'(\langle w, V_1, V_2 \rangle) = \big(h_1(\langle w, V_1, V_2 \rangle), h_2(\langle w, V_1, V_2 \rangle)\big)$. From $h'(\langle w, U_1, U_2 \rangle) \overset{\text{o}}{=} h'(\langle w, V_1, V_2 \rangle)$, we obtain $h_1(\langle w, U_1, U_2 \rangle) \overset{\text{o}}{=} h_1(\langle w, V_1, V_2 \rangle)$ and $h_2(\langle w, U_1, U_2 \rangle) \overset{\text{o}}{=} h_2(\langle w, V_1, V_2 \rangle)$. Moreover, since h_1 is projectable, we know that $h_1(\langle w, U_1, U_2 \rangle) = h_1(\langle w, V_1, V_2 \rangle)$. It remains to prove that $h_2(\langle w, U_1, U_2 \rangle) = h_2(\langle w, V_1, V_2 \rangle)$. To do so, we distinguish between the following subcases:

(1) $U_1 = U_2 = \varnothing$. We have $h_2(\langle w, U_1, U_2 \rangle) = 1_{\mathcal{M}_2}$ and hence, since $1_{\mathcal{M}_2}$ has empty memory and $h_2(\langle w, U_1, U_2 \rangle) \overset{\text{o}}{=} h_2(\langle w, V_1, V_2 \rangle)$, we get $h_2(\langle w, V_1, V_2 \rangle) = 1_{\mathcal{M}_2}$.

(2)   Both U_1 and U_2 are non-empty. In this case $h_2(\langle w, U_1, U_2 \rangle)$ must be either the null element $0_{\mathcal{M}_2}$ 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 $h_2(\langle w, U_1, U_2 \rangle) \overset{\text{o}}{=} h_2(\langle w, V_1, V_2 \rangle)$ we get $h_2(\langle w, U_1, U_2 \rangle) = h_2(\langle w, V_1, V_2 \rangle)$.

(3) $U_1 \neq \varnothing$ and $U_2 = \varnothing$. Clearly, U_1 is a singleton of the form $\{x\}$. Similarly, since $h_2(\langle w, U_1, U_2 \rangle) \overset{\text{o}}{=} h_2(\langle w, V_1, V_2 \rangle)$, we have that V_1 is a singleton of the form $\{x'\}$ and $V_2 = \varnothing$. We also recall that $h_1(\langle w, U_1, U_2 \rangle) = h_1(\langle w, V_1, V_2 \rangle) \neq 0_{\mathcal{M}_1}$ and that the morphism h_1 is 0-reduced, which implies $x = x'$. This shows that $h_2(\langle w, U_1, U_2 \rangle) = h_2(\langle w, V_1, V_2 \rangle)$.

(4) $U_1 = \varnothing$ and $U_2 \neq \varnothing$. This case is symmetric to the previous one.

We have just shown that h' is a projectable morphism recognizing $[\![\varphi(x, y) \wedge x \sim y]\!]$.   □

We are now ready to prove the main theorem of this section, that is, that every language $[\![\varphi]\!]$ defined by a rigidly guarded MSO$^\sim$ formula $\varphi(\bar{X})$ is effectively recognized by an orbit-finite data monoid via a projectable morphism.

Proof of Theorem 4.2. As already mentioned, the proof is by induction on the structure of the rigidly guarded MSO$^\sim$ formula $\varphi(\bar{X})$. As for the base cases, the languages defined by the atomic formulas $x < y$, $a(x)$, and $x \in Y$ are clearly recognized by orbit-finite data monoids via projectable morphisms.

As for the inductive step, suppose that a formula φ with m free variables $X_1, \ldots, X_m$ is given and that one can compute an orbit-finite data monoid $\mathcal{M}$ and a projectable morphism $h : (D \times A \times B^m)^* \to \mathcal{M}$ recognizing the language defined by φ. It follows that the complement language defined by $\neg \varphi$ is recognized by the same orbit-finite data monoid $\mathcal{M}$ via the same projectable morphism h.

Similarly, for the disjunction of two formulas, suppose that φ_1 and φ_2 are given. Without loss of generality (namely, by introducing dummy free variables via cylindrification), we can assume that the two formulas φ_1 and φ_2 have the same free variables $X_1, \ldots, X_m$. Furthermore, suppose that one can compute two orbit-finite data monoids $\mathcal{M}_1$ and $\mathcal{M}_2$ and 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_1 and φ_2. As these languages are over the same alphabet, we can construct an orbit-finite data monoid $\mathcal{M}_1 \times \mathcal{M}_2$ and a projectable morphism $h_1 \times h_2$ that recognize the language defined by $\varphi_1 \vee \varphi_2$.

As for the existential closure, Lemma 4.4 implies that the language defined by the formula $\exists X_m \; \varphi$ is recognized by a suitable orbit-finite data monoid $\mathcal{N}$ via a projectable morphism g, 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 4.5 how to compute an orbit-finite data monoid $\mathcal{N}$ and a projectable morphism g that recognizes the language defined by $\varphi(x_1, x_2) \wedge x_1 \sim x_2$. This concludes the proof of the theorem. $\qquad\square$

5. From orbit-finite monoids to rigidly guarded MSO˜

Having shown that every language defined by a rigidly guarded MSO˜ (resp., FO˜) sentence is recognized by an orbit-finite data monoid (resp., by an aperiodic orbit-finite data monoid), we now prove the converse. This is the most technical result of the paper.

We remark that in the classical theory of regular languages, the analogous of this result (at least the part involving only MSO) is straightforward: indeed, a monoid can be used as an automaton, and in this case 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 an approach with data monoids: not only there is no equivalent automaton model, but furthermore, the above approach is intrinsically not compatible with the notion of rigidity. Another consequence is that, as opposed to the classical case, the proof is significantly more involved for rigidly guarded MSO˜ than for rigidly guarded FO˜.

We also recall that data languages are invariant under renamings. This means that every data language that is recognized by an orbit-finite data monoid $\mathcal{M}$ via a morphism h can be described as the union over some orbits o of $\mathcal{M}$ of the inverse images $h^{-1}(o)$. The result we aim to prove is thus the following:

Theorem 5.1. *Given an orbit-finite data monoid $\mathcal{M}$, a morphism h from a free data monoid to $\mathcal{M}$, and an orbit o of $\mathcal{M}$, one can compute a rigidly guarded MSO˜ sentence φ that defines the data language $h^{-1}(o)$. Moreover, if $\mathcal{M}$ is aperiodic, then φ is a rigidly guarded FO˜ sentence.*

The proof of the theorem 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 objective of our proof is to find suitable formulas that, given some positions $x \leq y$ in a data word w, determine the orbit of the image via h of the infix of w between x and y, that is, determine the monoid element $h(w[x, y])$. We will reach this objective by exploiting an induction on a well-founded partial order that is defined on the $=_{\mathcal{J}^\circ}$-classes of $\mathcal{M}$ and that is induced by the preorder $\leq_{\mathcal{J}}$ (refer to Section 2.3 for an account of these orders).

Roughly speaking, we first construct the desired formulas for shorter infixes of the data word and then we move up towards longer infixes, until we determine the orbit of the entire word. To do 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. Moreover, since the product of two elements depends not only on the orbits, but also on the memorable values, we need to be able to compute the latter as well. Here, by "computing the memorable values" of $w[x, y]$ we mean being able to locate some positions in $w[x, y]$ that carry the memorable values of the element $h(w[x, y])$. For this, we use formulas of the form $\varphi(x, y, z_1, \ldots, z_n)$ which determine, not only the orbit of $h(w[x, y])$, but also some positions $z_1, \ldots, z_n$ witnessing

the memorable values. This must be done with care, however, as in our logic positions with memorable values can be compared only if they are guarded by rigid formulas.

We tacitly assume that all formulas defined hereafter are either rigidly guarded FO~ formulas or rigidly guarded MSO~ formulas, depending on whether $\mathcal{M}$ is aperiodic or not.

We begin by generalizing the notion of rigidity to formulas with more than two variables.

Definition 5.2. We say that x_i *determines* x_j in a formula $\varphi(x_1, \ldots, x_n)$ if for all data words w and all positions $x_1, \ldots, x_n, x'_1, \ldots, x'_n \in \mathsf{dom}(w)$,

$$\begin{cases} w \vDash \varphi(x_1, \ldots, x_n) \\ w \vDash \varphi(x'_1, \ldots, x'_n) \qquad \text{implies} \qquad x_j = x'_j \\ x_i = x'_i \end{cases}$$

The formula $\varphi(x_1, \ldots, x_n)$ is *rigid* if x_i determines x_j for all $i, j \in \{1, \ldots, n\}$. Similarly, a formula $\varphi(x, z_1, \ldots, z_k, y)$ is *inward-rigid* if $w \vDash \varphi(x, z_1, \ldots, z_k, y)$ implies $x \le z_1, \ldots, z_k \le y$, and, in addition, x determines $z_1, \ldots, z_k, y$ and y determines $x, z_1, \ldots, z_k$ in it.

We will mainly work with formulas $\varphi(x, z_1, \ldots, z_k, y)$ that are inward-rigid. Under certain conditions, these formulas can be used to determine the orbit and the positions of some memorable values of factors of a data word (we will describe formally what this means in Definition 5.6). However, in order to compare memorable values and simulate products in $\mathcal{M}$, we need to be able to turn an inward-rigid formula into a fully rigid formula, or a finite disjunction of such formulas. The following crucial lemma shows how to do so. We remark that the lemma can be read with formulas meaning either rigidly guarded MSO~ formulas or rigidly guarded FO~ formulas: both results hold, and the proof is in fact the same.

Lemma 5.3 (Sub-definability). *For all formulas $\varphi(x, y)$ where x determines y, there exist finitely many formulas $\beta_i(z, y)$ where z determines y such that for all $x \le z \le y$,*

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

Proof. We begin by recalling a result that originates from the composition methods developed by Feferman-Vaught and Shelah [18, 36]:

Claim 5.4. Given a classical MSO / FO formula $\varphi(x, y)$ that only uses the order $<$ and some unary predicates, but no data tests, and that entails $x \le y$, there exist finitely many pairs of formulas $(\varphi_i^L(x), \varphi_i^R(y))_{i=1 \ldots n}$ such that, for all words $w = u\,v$ and all positions x in u and y in v,

$$u\,v \vDash \varphi(x, |u| + y) \qquad \text{iff} \qquad u \vDash \varphi_i^L(x) \text{ and } v \vDash \varphi_i^R(y) \text{ for some } i \in \{1, \ldots, n\}.$$

By relativising quantifications, we can then obtain formulas in two variables $\alpha_i(x, z)$, $\beta_i(z, y)$ such that

$$w \vDash \alpha_i(x, z) \quad \text{iff} \quad w[1, \ldots, z-1] \vDash \varphi_i^L(x) \qquad \text{and}$$

$$w \vDash \beta_i(z, y) \quad \text{iff} \quad w[z, \ldots, y] \vDash \varphi_i^R(y).$$

Our sub-definability lemma for the classical MSO / FO formula $\varphi(x, y)$ follows easily from the above result, since $\varphi(x, y)$ is equivalent to

$$\bigvee_{i=1 \ldots n} \exists z \; \alpha_i(x, z) \; \wedge \; \beta_i(z, y) \; \wedge \; x \le z \le y.$$

Below, we generalize this argument to formulas that use rigidly guarded data tests.

Let $\varphi(x,y)$ be a formula of rigidly guarded MSO˜/ FO˜. We use a technique similar to that of the proof of Theorem 3.4 to syntactically replace in φ every occurrence of a data test $x' \sim y'$ with a fresh unary predicate $c_\alpha^\sim(x')$, where $\alpha(x',y')$ is the rigid formula that guards the occurrence of $x' \sim y'$ in φ and $c_\alpha^\sim(x')$ encodes the existence of a (unique) position y' satisfying $\alpha(x',y') \wedge x' \sim y'$. We denote by $\varphi^-(x,y)$ the resulting formula of classical MSO / FO and, for every data word w, we denote by w^- the word obtained from w by removing all data values and by adding the predicates $c_\alpha^\sim$ at positions x' in such a way that

$$w^- \vDash c_\alpha^\sim(x') \qquad \text{iff} \qquad w \vDash \exists y'\ \alpha(x',y')\ \wedge\ x' \sim y'\ .$$

Clearly, for all data words w and all positions x,y in it, we have

$$w \vDash \varphi(x,y) \qquad \text{iff} \qquad w^- \vDash \varphi^-(x,y)\ .$$

Now, suppose that x determines y in $\varphi(x,y)$. It can happen that x does not determine y in $\varphi^-(x,y)$, since the unary predicates $c_\alpha^\sim$ could be chosen in a way that is inconsistent with any choice of data values. This can be easily corrected by 'rigidifying' φ^-, namely, by letting

$$\varphi^=(x,y) \ =^{\text{def}}\ \varphi^-(x,y)\ \wedge\ \forall y'\ \varphi^-(x,y')\ \rightarrow\ y' = y\ .$$

Indeed, when interpreted on a generic data word w and a position x in it, the formula $\varphi^=(x,y)$ is equivalent to $\varphi^-(x,y)$ as long as there is at most one position y in w that satisfies φ^-. Otherwise, $\varphi^=(x,y)$ simply does not hold.

Knowing that $\varphi^=(x,y)$ is a classical MSO / FO formula and that, by construction, x determines y in it, we can apply the sub-definability lemma to $\varphi^=(x,y)$, thus obtaining finitely many formulas $\beta_i^-(z,y)$ where z determines y and such that, for all $x \leq z \leq y$,

$$w \vDash \varphi^=(x,y) \qquad \text{implies} \qquad w \vDash \beta_i^-(z,y) \text{ for some } i.$$

From each formula $\beta_i^-(z,y)$, we reconstruct a formula of rigidly guarded MSO˜/ FO˜ formula $\beta_i(z,y)$ by syntactically replacing every occurrence of unary predicate $c_\alpha^\sim(x')$ with $\exists y'\ \alpha(x',y') \wedge x' \sim y'$. It is clear that, since $\beta_i^-(z,y)$ defines a unique y from z, so does $\beta_i(z,y)$. Furthermore, for all data words w and all positions $x \leq z \leq y$, we have

$$w \vDash \varphi(x,y) \qquad \text{iff} \qquad w^- \vDash \varphi^-(x,y) \qquad \text{iff} \qquad w^- \vDash \varphi^=(x,y)$$

and hence, there is i such that $w^- \vDash \beta_i^-(z,y)$ and $w \vDash \beta_i(z,y)$. $\qquad\square$

An immediate consequence of the above lemma is the following:

Corollary 5.5. *Every inward-rigid formula is equivalent to a finite disjunction of rigid formulas.*

Proof. Consider an inward-rigid formula $\varphi(x,z_1,\ldots,z_k,y)$ and define $\phi(x,y) = \exists z_1,\ldots,z_k\ \varphi(x,z_1,\ldots,z_k,y)$. Since x determines y in $\phi(x,y)$, we can apply Lemma 5.3, thus obtaining the formulas $\alpha_1(z,y)$, ..., $\alpha_n(z,y)$. Accordingly, the desired rigid formulas are defined by

$$\phi_{i_1,\ldots,i_k}(x,z_1,\ldots,z_k,y) \ =^{\text{def}}\ \varphi(x,z_1,\ldots,z_k,y)\ \wedge\ \alpha_{i_1}(z_1,y)\ \wedge\ \ldots\ \wedge\ \alpha_{i_k}(z_k,y)$$

where $i_1,\ldots,i_k$ are indices ranging over $\{1,\ldots,n\}$.

One easily checks that the formulas $\phi_{i_1,\ldots,i_k}(x,z_1,\ldots,z_k,y)$ are rigid. Indeed, x determines z_i, which in its turn determines y, and y determines x. Of course, $\phi_{i_1,\ldots,i_k}$ entails φ, by construction. Conversely, given some positions $x,z_1,\ldots,z_k,y$ such that $w \vDash \varphi(x,z_1,\ldots,z_k,y)$, we know that $x \leq z_1,\ldots,z_k \leq y$ and, by Lemma 5.3,

there exist $i_1, \ldots, i_k$ such that $w \vDash \alpha_{i_1}(z_1, y) \wedge \ldots \wedge \alpha_{i_k}(z_k, y)$, and hence $w \vDash \phi_{i_1, \ldots, i_k}(x, z_1, \ldots, z_k, y)$. We have just proved that $\varphi(x, z_1, \ldots, z_k, y)$ is equivalent to the finite disjunction $\bigvee_{1 \leq i_1, \ldots, i_k \leq n} \phi_{i_1, \ldots, i_k}(x, z_1, \ldots, z_k, y)$ of rigid formulas. $\square$

We now formalize the meaning of "computing the type under a guard". For this it is convenient to fix an orbit-finite data monoid $\mathcal{M}$ that is given by a term-based presentation system $\mathcal{S} = (T, \odot, \breve{\ }, \approx)$. This means that the elements of $\mathcal{M}$ are the $\approx$-equivalence classes of the terms in T. However, by a slight abuse of notation, we shall often identify the terms $o(d_1, \ldots, d_k)$ in T with the corresponding elements $[o(d_1, \ldots, d_k)]_\approx$ of $\mathcal{M}$. For example, we can write $h(w[x, y]) = o(d_1, \ldots, d_k)$.

Definition 5.6. Let o be an orbit of the data monoid $\mathcal{M}$ having memory size k. A formula $\varphi(x, z_1, \ldots, z_k, y)$ *witnesses the orbit o* if it is inward-rigid and

$$w \vDash \varphi(x, z_1, \ldots, z_n, y) \qquad \text{implies} \qquad h(w[x, y]) = o(w[z_1], \ldots, w[z_n]) \ .$$

A family of formulas $F = (\varphi_o)_{o \in O}$ *computes the types under the guard $\alpha(x, y)$* if each formula $\varphi_o(x, \bar{z}, y)$ witnesses the orbit o and, moreover, the guard $\alpha(x, y)$ is logically equivalent to $\bigvee_{o \in O} \exists \bar{z} \ \varphi_o(x, \bar{z}, y)$. We say that *one can compute the types under the guard $\alpha(x, y)$* if there exists such a family of formulas.

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 types under $\alpha(x, y)$. As we mentioned, the proof of Theorem 5.1 exploits an induction on the partial order $\leq_{\mathcal{J}^\circ}$ of the $=_{\mathcal{J}^\circ}$-classes of $\mathcal{M}$. The invariant of the induction is given in the following lemma.

Lemma 5.7 (Inductive statement). *For every $=_{\mathcal{J}^\circ}$-class O of $\mathcal{M}$:*

(C1) *there exists a formula $\varphi_O(x, y)$ such that $w \vDash \varphi_O(x, y)$ iff $h(w[x, y]) \in O$;*

(C2) *for all rigid guards $\alpha(x, y)$ such that $w \vDash \alpha(x, y)$ implies $h(w[x, y]) \geq_{\mathcal{J}^\circ} O$, one can compute the types under α.*

We will prove the above lemma first under the assumption that $\mathcal{M}$ is *aperiodic*, constructing formulas of the rigidly guarded FO~ logic. In the aperiodic case, we use the fact that the orbit of an infix is determined by its $\mathcal{L}^\circ$-class and its $\mathcal{R}^\circ$-class and by the equality relationships between the memorable values in these two classes (this follows basically from the fact that the $\mathcal{H}$-classes of an aperiodic monoid are singletons). In the second part, we will reprove the same lemma without the assumption of aperiodicity, constructing formulas of the rigidly guarded MSO~ logic. In this case different objects have to be guessed by quantifying over monadic second-order predicates.

5.1. **The translation in the aperiodic case.** In this section we assume that $\mathcal{M}$ is an *aperiodic* orbit-finite data monoid and we prove the inductive statement given in Lemma 5.7, where formulas are meant to be rigidly-guarded FO~ formulas.

For the sake of brevity, we can fill the parameters of a formula $\varphi(x_1, \ldots, x_n)$ with $\star$ to denote the fact that the corresponding variables are existentially quantified. With this notation, if $\varphi(x, y, z)$ is rigid according to Definition 5.2, then so is $\varphi(\star, y, z)$, as well as $\varphi(x, \star, z)$ and $\varphi(x, y, \star)$.

We begin by presenting a special, but important, case of Lemma 5.7, which shows that the types of infixes of length 1 can be computed (this will serve as our base case for the inductive construction).

Lemma 5.8. *Let $\alpha_1(x,y) =^{def} (x = y)$. One can compute the types under the guard α_1.*

Proof. Note that the morphism h maps singleton words to orbits that have memory size at most 1. A family F_1 that computes the types under α_1 consists of formulas ϕ_o, for all orbits $o \in O$, defined by

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

$$\phi_o(x,z_1,y) \;=^{\mathrm{def}} \bigvee_{h((d,a)) = o(d)} a(x) \;\wedge\; x = y \;\wedge\; x = z_1 \qquad \text{(if o has memory size 1)}$$

$$\phi_o(x,y) \;=^{\mathrm{def}} \;\mathsf{false} \qquad\qquad\qquad\qquad\qquad\qquad \text{(otherwise)}$$

$\square$

The next lemma shows how to compose families of formulas that compute the types under some given guards. This is one of the places where the products of the data monoid $\mathcal{M}$ are simulated by comparing memorable values. In particular, the lemma depends on the fact that any inward-rigid formula used to witness an orbit can be written as a finite disjunction of rigid formulas (Corollary 5.5).

Lemma 5.9. *Given two rigid guards $\alpha(x,y)$ and $\alpha'(x',y')$, let $\alpha \cdot \alpha'$ be the rigid guard defined by $(\alpha \cdot \alpha')(x,y) \;=^{def} \exists z \; \alpha(x,z) \;\wedge\; \alpha'(z+1,y)$. Given two families of formulas F and F' that compute the types under the guards α and α', respectively, one can construct a family $F \cdot F'$ that computes the types under the guard $\alpha \cdot \alpha'$.*

Proof. Let $F = (\varphi_o)_{o \in O}$ and $F' = (\varphi'_o)_{o \in O}$. We aim at constructing $F \cdot F' = (\psi_o)_{o \in O}$ that computes the types under $\alpha \cdot \alpha'$. We begin by recalling that the orbit that results from the product of an element in orbit o with an element in orbit o' depends on the respecive memorable values. The equality relationships between memorable values will be represented by pairs of terms with data values (up to renaming, there are only finitely many pairs) and, for each such pair, we will produce a corresponding formula.

Consider two terms $t = o(d_1, \ldots, d_k)$ and $t' = o'(e_1, \ldots, e_h)$ and let $t \cdot t' = o''(f_1, \ldots, f_\ell)$ be their product according to $\mathcal{M}$. Let $\bigvee_{1 \leq p \leq n} \phi_{o,p}(x, z_1, \ldots, z_k, y)$ and $\bigvee_{1 \leq q \leq m} \phi'_{o',q}(x, z_1, \ldots, z_h, y)$ be the finite disjunctions of rigid formulas, equivalent to φ_o and $\varphi'_{o'}$, respectively, that are obtained from Corollary 5.5. Define

$$\beta^{i,j}_{p,q}(z_i, z'_j) \;=^{\mathrm{def}} \exists y. \; \phi_{o,p}(\star, \bar{\star}, z_i, \bar{\star}, y) \;\wedge\; \phi'_{o',q}(y+1, \star, \bar{\star}, z'_j, \bar{\star}, \star)$$

and

$$\psi_{t \cdot t'}(x, z''_1 \ldots, z''_\ell, y) \;=^{\mathrm{def}} \exists \xi, \; \exists \bar{z} = z_1 \ldots z_k, \; \exists \bar{z}' = z'_1 \ldots z'_h. \tag{a}$$

$$\wedge \bigvee_{p,q} \Big(\; \phi_{o,p}(x, \bar{z}, \xi,) \;\wedge\; \phi'_{o',q}(\xi+1, \bar{z}', y) \tag{b}$$

$$\wedge \bigwedge_{d_i = e_j} \beta^{i,j}_{p,q}(z_i, z'_j) \;\wedge\; z_i \sim z'_j \tag{c}$$

$$\wedge \bigwedge_{d_i \neq e_j} \beta^{i,j}_{p,q}(z_i, z'_j) \;\wedge\; z_i \nsim z'_j \; \Big) \tag{d}$$

$$\wedge \bigwedge_{f_i = d_j} z''_i = z_j \quad\wedge \bigwedge_{\substack{f_i = e_j \\ f_i \notin \{d_1, \ldots, d_k\}}} z''_i = z'_j \;. \tag{e}$$

Given x and y, the formula $\psi_{t \cdot t'}(x, z_1'' \ldots, z_\ell'', y)$ first guesses the intermediate position ξ and the variables $\bar{z}$ and $\bar{z}'$ that contain the memorable values of $h(w[x, \xi])$ and of $h(w[\xi + 1, y])$ (a). It then guesses the indices p, q for the rigid formulas $\phi_{o,p}$ and $\phi'_{o',q}$ that hold over the factors $w[x, \xi]$ and $w[\xi + 1, y]$ (b). Line (c) checks that, whenever a memorable value of t and a memorable value of t' are equal, then the corresponding positions in the factors share the same data value. Note that this comparison is done under the rigid guard $\beta_{p,q}^{i,j}(z_i, z_j')$, which of course holds between z_i and z_j' whenever (b) holds. Similar conditions for disequalities are verified in line (d). Finally, line (e) uniquely determines the positions of the memorable values of $t \cdot t'$ (in case a memorable value is shared between the left and the right term, priority is given to the leftmost position).

Overall, the formula $\psi_{t \cdot t'}(x, z_1'' \ldots, z_\ell'', y)$ is inward-rigid and witnesses the orbit o''. Furthermore, if x, ξ, y are positions such that $w \vDash \alpha(x, \xi)$ and $w \vDash \alpha'(\xi + 1, y)$ and τ is a renaming such that $\tau(t) = h(w[x, \xi])$ and $\tau(t') = h(w[\xi + 1, y])$, then $w \vDash \psi_{t \cdot t'}(x, \bar{z}'', y)$ for some tuples of positions $\bar{z}''$. Therefore, the family $F \cdot F' = (\psi_o)_{o \in O}$ of formulas that computes the types under the guard $\alpha \cdot \alpha'$ can be obtained by associating with each orbit o'' the formula

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

(this tries every possible pair of terms t, t', among the finitely many different possibilities up to renamings, whose product yields the orbit o). $\qquad\square$

Using Lemmas 5.8 and 5.9, one can compute the orbits of infixes of fixed length:

Corollary 5.10. *Let $\alpha_k(x, y) =^{def} (x + k - 1 = y)$. One can compute the types under the guard α_k.*

We now prove a technical lemma that is similar to Theorem V.1.9 from [30]. The difference here is that the hypothesis of the lemma uses the coarser equivalence $=_{\mathcal{J}^\circ}$ in place of $=_{\mathcal{J}}$. We will exploit this result several times in the paper, for instance, in the proof of Theorem 5.15.

Lemma 5.11. *For every pair of elements s, t of $\mathcal{M}$, if $s =_{\mathcal{J}^\circ} s \cdot t$, then $s =_{\mathcal{R}} s \cdot t$. Similarly, if $t =_{\mathcal{J}^\circ} s \cdot t$, then $t =_{\mathcal{L}} s \cdot t$.*

Proof. Suppose that $s =_{\mathcal{J}^\circ} s \cdot t$ (symmetric arguments can be used when $t =_{\mathcal{J}^\circ} s \cdot t$ and with $\mathcal{L}$ in place of $\mathcal{R}$). By definition of $=_{\mathcal{J}^\circ}$, we know that there is a renaming τ such that $s \in M \cdot \tau(s \cdot t) \cdot M$, and hence there exist some elements u, v of $\mathcal{M}$ a such that

$$s \;=\; \tau(u \cdot s \cdot t \cdot v) \;=\; \tau(u) \cdot \tau(s) \cdot \tau(t \cdot v) .$$

By repeatedly applying the mapping τ and substituting s with $\tau(u) \cdot \tau(s) \cdot \tau(t \cdot v)$, we obtain

$$s \;=\; \underbrace{\tau(u) \cdot \ldots \cdot \tau^n(u)}_{u_n} \cdot \underbrace{\tau^n(s)}_{s_n} \cdot \underbrace{\tau^n(t \cdot v) \cdot \ldots \cdot \tau(t \cdot v)}_{z_n} .$$

Since τ is a permutation on D that is the identity on all but finite many data values, we have that τ^{n_0} is the identity for some $n_0 \geq 1$. In particular, for all multiples $m \cdot n_0$ of n_0, we have

$$u_{m \cdot n_0} \;=\; u_{n_0}^m \qquad\qquad s_{m \cdot n_0} \;=\; s \qquad\qquad z_{m \cdot n_0} \;=\; z_{n_0}^m .$$

Moreover, since $\mathcal{M}$ is locally finite, we can fix $m \geq 1$ large enough in such a way that $z_{n_0}^m$ is an idempotent. We thus obtain

$$s \;=\; u_{n_0}^m \cdot s \cdot z_{n_0}^m \;=\; u_{n_0}^m \cdot s \cdot z_{n_0}^m \cdot z_{n_0}^m \;=\; s \cdot z_{n_0}^m$$

whence

$$s \;=\; s \cdot \tau^{n_0}(t \cdot v) \cdot \big(\tau^{n_0}(t \cdot v)\big)^{m-1} \;=\; s \cdot t \cdot v \cdot \big(\tau^{n_0}(t \cdot v)\big)^{m-1}.$$

We have just shown that $s \cdot t \geq_{\mathcal{R}} s$. As the converse relation $s \geq_{\mathcal{R}} s \cdot t$ holds trivially, we conclude that $s =_{\mathcal{R}} s \cdot t$. $\quad\square$

From the above lemma we easily obtain the following result:

Lemma 5.12. *Let O be a $=_{\mathcal{J}^\circ}$-class of $\mathcal{M}$ and let $[x, y]$ be a minimal interval such that $h(w[x, y]) \not\geq_{\mathcal{J}^\circ} O$. We have that*

(1)   either $x = y$,

(2)   or $x + 1 = y$,

(3)   or $[x + 1, y - 1]$ is an interval such that $h(w[x + 1, y - 1]) >_{\mathcal{J}^\circ} O$.

In particular, in the third case, there exists a $=_{\mathcal{J}^\circ}$-class P that is strictly above O (i.e., $P >_{\mathcal{J}^\circ} O$) and such that $[x + 1, y - 1]$ is a maximal interval satisfying $h(w[x + 1, y - 1]) \in P$.

Proof. Let $[x, y]$ be a minimal interval such that $h(w[x, y]) \not\geq_{\mathcal{J}^\circ} O$ and suppose that neither the first case nor the second case holds, namely, $x < x + 1 \leq y - 1 < y$. For the sake of brevity, let $s = h(w(x))$, $t = h(w[x + 1, y - 1])$, and $u = h(w(y))$. We begin by noting that the minimality of $[x, y]$ implies $s \cdot t \geq_{\mathcal{J}^\circ} O$ and $t \cdot u \geq_{\mathcal{J}^\circ} O$.

Below, we aim at proving that $t >_{\mathcal{J}^\circ} s \cdot t$ and $t >_{\mathcal{J}^\circ} t \cdot u$, as this would imply that $t >_{\mathcal{J}^\circ} O$ and that $[x + 1, y - 1]$ is a maximal interval such that $h(w[x + 1, y - 1])$ belongs to the $=_{\mathcal{J}^\circ}$-class of t. Suppose, by contradiction, that $t \not>_{\mathcal{J}^\circ} s \cdot t$. Since $t \geq_{\mathcal{J}} s \cdot t$, we derive $t =_{\mathcal{J}} s \cdot t$. By applying Lemma 5.11 we obtain $t =_{\mathcal{L}} s \cdot t$. Moreover, since $\mathcal{L}$ is a congruence with respect to products on the right, we derive $t \cdot u =_{\mathcal{L}} s \cdot t \cdot u$. Finally, since $=_{\mathcal{L}}$ refines $=_{\mathcal{J}}$, we obtain $t \cdot u =_{\mathcal{J}} s \cdot t \cdot u$, which contradicts the minimality of the interval $[x, y]$. We must conclude that $t >_{\mathcal{J}^\circ} s \cdot t$ and, by symmetric arguments, $t >_{\mathcal{J}^\circ} t \cdot u$. $\quad\square$

From now on, we assume that O is a $=_{\mathcal{J}^\circ}$-class of $\mathcal{M}$ and that both claims C1 and C2 of Lemma 5.7 hold for all $=_{\mathcal{J}^\circ}$-classes P that are strictly above O (we refer to this assumption as our inductive hypothesis).

Lemma 5.13. *There exists a formula $\alpha_{\not\geq O}^{\min}(x, y)$ such that $w \vDash \alpha_{\not\geq O}^{\min}(x, y)$ iff $[x, y]$ is a minimal interval such that $h(w[x, y]) \not\geq_{\mathcal{J}^\circ} O$. Furthermore, the formula $\alpha_{\not\geq O}^{\min}(x, y)$ is rigid and one can compute the types under it.*

Proof. Lemma 5.12 describes three types of intervals $[x, y]$ such that $h(w[x, y]) \not\geq_{\mathcal{J}^\circ} O$. For the first type of intervals, one simply lets $\alpha_1(x, y) =^{\mathrm{def}} (x = y)$ and accordingly constructs the family F_1 that computes the types under α_1 using Lemma 5.8. Similarly, for the second type of intervals, one lets $\alpha_2(x, y) =^{\mathrm{def}} (x + 1 = y)$ and uses Corollary 5.10 to construct a family F_2 computing the types under α_2.

We now focus on the most interesting type of intervals, which are of the form $[x, y]$, where $[x + 1, y - 1]$ is maximal such that $h(w[x + 1, y - 1]) \in P$ and P is a specific $=_{\mathcal{J}^\circ}$-class strictly above O. Let $\alpha_P^{\max}(x, y)$ be a formula stating that $x \leq y$ and $[x, y]$ is a maximal interval such that $h(w[x, y]) \in P$ (this formula exists thanks to the inductive hypothesis C1). Note that the formula $\alpha_P^{\max}(x, y)$ is rigid by construction. Hence, by using this time the inductive hypothesis C2, one can construct a family $F_P^{\max}$ that computes the types under the guard $\alpha_P^{\max}(x, y)$.

The desired formula $\alpha_{\nleq O}^{\min}(x,y)$ can be defined as follows:

$$\alpha_{\nleq O}^{\min}(x,y) \;=^{\mathrm{def}}\; \alpha_1(x,y) \;\wedge\; F_1(x,y) \nleq_{\mathcal{J}^\circ} O$$

$$\vee \;\; \alpha_2(x,y) \;\wedge\; F_2(x,y) \nleq_{\mathcal{J}^\circ} O$$

$$\vee \;\; \bigvee_{P >_{\mathcal{J}^\circ} O} \Big((\alpha_1 \cdot \alpha_P^{\max} \cdot \alpha_1)(x,y) \;\wedge\; (F_1 \cdot F_P^{\max} \cdot F_1)(x,y) \nleq_{\mathcal{J}^\circ} O \Big)$$

where the families $F_1, F_2, F_P^{\max}$ are used as if they were functions computing types and the operations of compositions are those outlined in Lemma 5.9 (this shorthand of notation should be clear to understand and can be transformed into standard formulas by unfolding the finitely many cases). We also observe that, since $\mathcal{M}$ is orbit-finite, the disjunction over all $=_{\mathcal{J}^\circ}$-classed P strictly above O is finite.

It is easy to see that the above formula $\alpha_{\nleq O}^{\min}(x,y)$ correctly defines the minimal intervals $[x,y]$ such that $h(w[x,y]) \nleq_{\mathcal{J}^\circ} O$. Finally, the formula is rigid by construction and a family computing the types under this guard can be easily obtained using the same kind of constructions. $\qquad\square$

We are now ready to prove the induction steps for claims C1 and C2 of Lemma 5.7 with respect to the $\mathcal{J}^\circ$-class O. We remark that only the proof of Claim C2 relies on the assumption that the monoid $\mathcal{M}$ is aperiodic, as well as on the properties of memorable values that we outlined in Section 2.3.

Lemma 5.14 (Induction step for C1)**.** *There exists a formula $\varphi_O(x,y)$ such that $w \vDash \varphi_O(x,y)$ iff $h(w[x,y]) \in O$.*

Proof. One first disproves the existence of an interval $[x',y']$ included in $[x,y]$ and satisfying $\alpha_{\nleq O}^{\min}(x',y')$. This property implies $h(w[x,y]) \geq_{\mathcal{J}^\circ} O$ and can be easily defined by a formula obtained from Lemma 5.13. One then excludes the case $h(w[x,y]) >_{\mathcal{J}^\circ} O$ by verifying the conjunction of the properties $h(w[x,y]) \notin P$ over all $=_{\mathcal{J}^\circ}$-classes P strictly above O. The latter properties can be defined thanks to the inductive hypothesis C1. $\qquad\square$

Lemma 5.15 (Induction step for C2)**.** *For all rigid guards $\alpha(x,y)$ such that $w \vDash \alpha(x,y)$ implies $h(w[x,y]) \geq_{\mathcal{J}^\circ} O$, one can compute the types under α.*

Proof. Thanks to the inductive hypothesis, for each of the finitely many $=_{\mathcal{J}^\circ}$-classes P that are strictly above O, one can construct a formula $\varphi_P(x,y)$ that checks whether $h(w[x,y]) \in P$ and in this case compute the types under the rigid guard $\alpha(x,y) \wedge \varphi_P(x,y)$. Thus, to prove the lemma, it is sufficient to consider the case where $w \vDash \alpha(x,y)$ implies $h(w[x,y]) \in O$.

We begin by introducing the formula $\alpha_O^{\min}(x',y')$ that expresses the property that $[x',y']$ is a minimal interval satisfying $h(w[x',y']) \in O$ – such a formula exists thanks to Lemma 5.14 and, moreover, it is rigid. Next, we assume that $\alpha(x,y)$ holds and we consider the intervals $[x',y']$ that are included in $[x,y]$ and satisfy $\alpha_O^{\min}(x',y')$; we call these intervals *blocks*. We focus in particular on the block $[x_1,y_1]$ whose left endpoint x_1 is as close as possible to x, as well as on the block $[x_2,y_2]$ whose right endpoint y_2 is as close as possible to y. These two special blocks can be defined from x and y by the following formula

$$\beta(x,x_1,y_1,x_2,y_2,y) \;=^{\mathrm{def}}\; \alpha(x,y) \;\wedge\; x \leq x_1 \;\wedge\; y_2 \leq y \;\wedge$$
$$\alpha_O^{\min}(x_1,y_1) \;\wedge\; \forall x' \left(\alpha_O^{\min}(x',\star) \;\rightarrow\; x' < x \;\vee\; x_1 \leq x' \right)$$
$$\alpha_O^{\min}(x_2,y_2) \;\wedge\; \forall y' \left(\alpha_O^{\min}(\star,y') \;\rightarrow\; y < y' \;\vee\; y' \leq y_2 \right)$$

(note that the formula implies $h(w[x,y]) \in O$ and hence $[x_1, y_1], [x_2, y_2] \subseteq [x, y]$). It is easy to see that β is an inward-rigid formula: indeed, x determines x_1, which determines y_1, and y determines y_2, which determines x_2. Thus, by Corollary 5.5, the formula is equivalent to a finite disjunction of rigid formulas, say $\beta_1, \ldots, \beta_n$.

We can now describe the steps for computing the types under $\alpha(x, y)$:

(1)   Guess an index $i \in \{1, \ldots, n\}$ and some positions x_1, y_1, x_2, and y_2 such that $w \vDash \beta_i(x, x_1, y_1, x_2, y_2, y)$.

(2)   Compute the orbits under the rigid guard $\beta_i(x, \star, y_1, \star, \star, \star)$. This is doable since (i) $\beta_i(x, \star, y_1, \star, \star, \star)$ entails $\beta(x, \star, y_1, \star, \star, \star)$, which in its turn entails $\beta(x, x_1 - 1, \star, \star, \star, \star) \cdot \alpha_O^{\min}(x_1, y_1)$, (ii) thanks to the fact that $w \vDash \beta(x, x_1 - 1, \star, \star, \star, \star)$ implies $h(w[x, x_1 - 1]) >_{\mathcal{J}\circ} O$, one can exploit the inductive hypothesis C2 to compute the types under the guard $\beta(x, x_1 - 1, \star, \star, \star, \star)$, (iii) by Lemma 5.13, one can compute the types under the guard $\alpha_O^{\min}(x_1, y_1)$, and (iv) by Lemma 5.9, one can compute the types under the guard $\beta(x, x_1 - 1, \star, \star, \star, \star) \cdot \alpha_O^{\min}(x_1, y_1)$.

We also claim that the element $h(w[x, y_1])$, which is determined by the guard $\beta_i(x, \star, y_1, \star, \star, \star)$, belongs to the same $=_{\mathcal{R}}$-class as the element $h(w[x, y])$. Indeed, we have $h(w[x, y]) = h(w[x, y_1]) \cdot h(w[y_1 + 1, y])$. Moreover, by construction, both elements $h(w[x, y])$ and $h(w[x, y_1])$ belong to the same $=_{\mathcal{J}\circ}$-class O. By Lemma 5.11 it follows that $h(w[x, y]) =_{\mathcal{R}} h(w[x, y_1])$.

(3)   In a similar way, compute the types under the rigid guard $\beta_i(\star, \star, \star, x_2, \star, y)$. By symmetric arguments, we know that the element $h(w[x_2, y])$ is in the same $=_{\mathcal{L}}$-class as the element $h(w[x, y])$.

(4)   Compute the orbits under the rigid guard $\alpha(x, y)$, as follows. First, recall that $h(w[x, y]) =_{\mathcal{R}} h(w[x, y_1])$ and $h(w[x, y]) =_{\mathcal{L}} h(w[x_2, y])$. Moreover, since $\mathcal{M}$ is aperiodic, all its $=_{\mathcal{H}}$-classes are singletons. In particular, the intersection of the $=_{\mathcal{R}}$-class of $h(w[x, y_1])$ and the $=_{\mathcal{L}}$-class of $h(w[x_2, y])$ is the singleton that contains precisely the element $h(w[x, y])$. It follows that the orbit of $h(w[x, y])$ can be determined from the orbits and from the memorable values of the elements $h(w[x, y_1])$ and $h(w[x_2, y])$. This information is available from to the previous constructions. In particular, one can compare the memorable values of $h(w[x, y_1])$ and $h(w[x_2, y])$ using suitable rigid guards, in a way that is similar to the proof of Lemma 5.9 (note that this requires applying Corollary 5.5 to the inward-rigid formulas that witness the orbits of $h(w[x, y_1])$ and $h(w[x_2, y])$). It remains to determine from the endpoints x and y some positions that contain the memorable values of $h(w[x, y])$. For this, one recalls that the $\mathcal{R}$-memorable values of $h(w[x, y])$ are the same as the $\mathcal{R}$-memorable values of $h(w[x, y_1])$, for which some witnessing positions can be determined thanks to the previous constructions. Similarly, one determines some positions for the $\mathcal{L}$-memorable values of $h(w[x_2, y])$, which are known to be the same as the $\mathcal{L}$-memorable values of $h(w[x, y])$. Finally, by Proposition 2.12, one knows that there are no other memorable values for $h(w[x, y])$.

It is routine to translate the above steps into a family of rigidly guarded FO$^\sim$ formulas that compute the types of $h(w[x, y])$ under the guard $\alpha(x, y)$.   □

The above arguments prove Lemma 5.7 under the assumption that the orbit-finite data monoid $\mathcal{M}$ is aperiodic. We conclude this part by proving the claim of Theorem 5.1 that deals with the aperiodic case.

Corollary 5.16. *Every data language recognized by a morphism into an orbit-finite aperiodic data monoid is effectively definable by a rigidly guarded $FO^{\sim}$ sentence.*

Proof. Since the image of the data language via the recognizing morphism h is a finite union of orbits, it is sufficient to construct, for each orbit o, a corresponding sentence φ_o that holds over a data word w iff $h(w) \in o$. For this, we consider the guard $\alpha(x,y) \stackrel{\text{def}}{=} (\neg \exists z \; z < x) \wedge (\neg \exists z \; z > y)$, which holds over w iff x is the first position and y the last position of w. By claim C2 of Lemma 5.7, we can construct a family of formulas $\varphi_o(x, \bar{z}, y)$ that compute the types under α. The language $h^{-1}(o)$ is thus defined by the sentence $\exists x, \bar{z}, y \; \alpha(x,y) \wedge \varphi_o(x, \bar{z}, y)$. $\qquad\square$

5.2. **The translation in the non-aperiodic case.** In the previous section we have seen how to establish Theorem 5.1 in the aperiodic case. The remaining claim, stating that every data language recognized by an orbit-finite data monoid is definable in rigidly guarded MSO$^{\sim}$ logic, is proved by following the same structure, namely, by relying on the same induction on $=_{\mathcal{J}\circ}$-classes and on similar constructions.

We fix for the rest of this section an orbit-finite data monoid $\mathcal{M}$ over a set D of data values, and a morphism h from the free data monoid $(D \times A)^*$ to $\mathcal{M}$. We assume that all formulas defined hereafter are of rigidly-guarded MSO$^{\sim}$.

The goal is to reprove Lemma 5.7, but this time without assuming that the monoid $\mathcal{M}$ is aperiodic. We recall that the proof of claim C1 (Lemma 5.14) does not exploit the assumption of aperiodicity (as far as the induction hypothesis is admitted). Hence we can reuse this part of the proof for the monoid $\mathcal{M}$. The only proof that needs to be changed is that of Lemma 5.15, and more precisely the constructions described in step 4. Below, we focus only on this part of the proof, assuming that O is a $=_{\mathcal{J}\circ}$-class of $\mathcal{M}$ and that the inductive hypothesis holds, namely, the claim of Lemma 5.7 for all $=_{\mathcal{J}\circ}$-classes P strictly above O.

To compute the types under a rigid guard $\alpha(x,y)$, we will divide the infix $w[x,y]$ into several blocks. That is, given a data word w and two positions $x \leq y$ such that $w \vDash \alpha(x,y)$, a formula will first guess a factorization of $w[x,y]$ into some infixes $w_1, \ldots, w_n$ which are small enough that they can be handled by the inductive hypothesis. Then, the formula will perform sub-computations that determine the orbit of each factor, as well as some positions carrying the memorable values in it. Finally, it will recursively compute the types of the partial products $h(w_1) \cdot \ldots \cdot h(w_i)$, for $i = 1, \ldots, n$, eventually determining the orbit the entire product $h(w[x,y]) = h(w_1) \cdot \ldots \cdot h(w_n)$.

We begin by describing the factorizations we are mainly interested in (for the sake of simplicity, the definitions are given with respect to the whole word w, as if the rigid guard $\alpha(x,y)$ held over w with $x = 1$ and $y = |w|$).

Definition 5.17. A *factorization* of a data word w is a sequence $w_1, \ldots, w_n$ of non-empty infixes such that $w = w_1 \cdot \ldots \cdot w_n$. This factorization is called an *O-factorization*, for some $=_{\mathcal{J}\circ}$-class O, if we have:

- $h(w_1 \cdot \ldots \cdot w_n) \in O$,
- $h(w_i) \in O$, for all $1 \leq i \leq n$.

Similarly, the factorization is called an *O-prefactorization* if we have:

- $h(w_1 \cdot \ldots \cdot w_n) \in O$,

- $\quad h(w_i) \in O$ or $h(w_{i+1}) \in O$, for all $1 \le i \le n-1$,
- $\quad h(w_i) \in O$ implies $h(u) \notin O$, for all proper infixes u of w_i and for all $1 \le i \le n$.

Finally, we call *left endpoint* (resp., *right endpoint*) of w_i the position in w where the factor w_i begins (resp., ends).

We remark that any factorization $w_1 = w[x_1, y_1], \ldots, w_n = w[x_n, y_n]$ of w can be *represented* in MSO by the pair (X, Y) of monadic predicates that contain the left endpoints and the right endpoints of the factors, respectively, i.e. $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$.

The second definition concerns special forms of factorizations where the endpoints of every factor can be determined, one from the other, by means of a rigid formula. Assuming that such a factorization exists, one can move from an endpoint to another adjacent endpoint, either to the left or to the right, in a deterministic manner.

Definition 5.18. Let $G = \{\gamma_j(x', y')\}_{j=1,\ldots,k}$ be a finite family of rigid formulas. We say that a factorization $w_1 \ldots w_n$ of w is *rigidly traversable by* G if for every $1 \le i \le n$, there exists $1 \le j \le k$ such that $w \vDash \gamma_j(x_i, y_i)$, where x_i and y_i are, respectively, the left and the right endpoints of the factor w_i.

Lemma 5.19. *Let O be a $=_{\mathcal{J}\circ}$-class and let $\alpha(x, y)$ be a rigid formula such that $w \vDash \alpha(x, y)$ implies $h(w[x, y]) \in O$. One can construct a formula $\varphi^{\mathsf{fact}}_{\mathsf{pre\text{-}}J}(x, y, X, Y)$ and a finite family $G = \{\gamma_j(x', y')\}_{j=1,\ldots,k}$ of rigid formulas such that:*

(1)   if $w \vDash \alpha(x, y)$, then $w \vDash \varphi^{\mathsf{fact}}_{\mathsf{pre\text{-}}J}(x, y, X, Y)$ for some sets $X, Y \subseteq [x, y]$;

(2)   if $w \vDash \varphi^{\mathsf{fact}}_{\mathsf{pre\text{-}}J}(x, y, X, Y)$, then the pair (X, Y) represents an O-prefactorization of $w[x, y]$ that is rigidly traversable by G;

(3)   one can compute the types under each guard $\gamma_j \in G$.

Proof. We begin by describing the formula $\varphi^{\mathsf{fact}}_{\mathsf{pre\text{-}}J}(x, y, X, Y)$. The main idea is to verify that the pair (X, Y) of monadic predicates represents an O-prefactorization $w_1, \ldots, w_n$ of $w[x, y]$, namely, X contains the left endpoints and Y contains the right endpoints of the factors $w_1, \ldots, w_n$. This is easy to do since we can use the formula $\varphi_O(x', y')$ from claim C1 of Lemma 5.7 to verify all the properties that define an O-prefactorization (recall that the proof of claim C1 that we gave in Section 5.1 holds even when the data monoid $\mathcal{M}$ is not aperiodic). However, this is not yet the definition of $\varphi^{\mathsf{fact}}_{\mathsf{pre\text{-}}J}(x, y, X, Y)$. Because in general there exist many O-prefactorizations for the same word $w[x, y]$ and because we need to be able to move across the endpoints of the factors in a deterministic manner, it is convenient to commit to a single O-prefactorization of $w[x, y]$, which can be uniquely determined from x and y. More precisely, if we denote by $\tilde{\varphi}^{\mathsf{fact}}_{\mathsf{pre\text{-}}J}(x, y, X, Y)$ the formula that checks that (X, Y) represents an O-prefactorization of $w[x, y]$, then we can uniquely determine one such pair (X, Y) from x and y by selecting, for instance, the least one according to some fixed MSO-definable total order $\le$. This is what the formula $\varphi^{\mathsf{fact}}_{\mathsf{pre\text{-}}J}(x, y, X, Y)$ does. In addition, the formula verifies that the guard $\alpha(x, y)$ holds between x and y (this will be used later and is not a restriction). Summing up, we let

$$\varphi^{\mathsf{fact}}_{\mathsf{pre\text{-}}J}(x, y, X, Y) \quad =^{\mathrm{def}} \quad \alpha(x, y) \ \wedge \ \tilde{\varphi}^{\mathsf{fact}}_{\mathsf{pre\text{-}}J}(x, y, X, Y)$$
$$\wedge \quad \forall X', Y' \ \tilde{\varphi}^{\mathsf{fact}}_{\mathsf{pre\text{-}}J}(x, y, X', Y') \ \rightarrow \ (X, Y) \le (X', Y') \ .$$

For short, we call *canonical prefactorization of $w[x, y]$* the factorization that is represented by the unique pair (X, Y) that satisfies $\varphi^{\mathsf{fact}}_{\mathsf{pre\text{-}}J}(x, y, X, Y)$.

We now construct the family G of rigid formulas that can be used to move across the endpoints of the canonical prefactorization of $w[x, y]$. The problem can be reduced to determining the positions x and y from a given endpoint of a factor: once we know x and y, we can reconstruct the canonical prefactorization of $w[x, y]$ and then determine the other endpoint of the factor. We use the sub-definability Lemma 5.3 to find, for each endpoint z, a finite number of possible choices for x and y. More precisely, we recall that $\varphi^{\mathsf{fact}}_{\mathsf{pre}\text{-}J}(x, y, X, Y)$ entails the rigid formula $\alpha(x, y)$ and we apply to this formula the sub-definability Lemma 5.3. In this way we obtain a finite set of formulas $\beta_1(z, y), \ldots, \beta_k(z, y)$ such that

- z determines y in $\beta_j(z, y)$, for all $1 \le j \le k$,
- if z is an endpoint of the canonical prefactorization of $w[x, y]$, then there exists $1 \le j \le k$ such that $w \vDash \beta_j(z, y)$.

We can thus define the family G as the set of formulas

$$\gamma_j(x', y') \quad =^{\mathrm{def}} \quad \exists\, x, y, X, Y \quad \beta_j(x', y) \,\wedge\, \varphi^{\mathsf{fact}}_{\mathsf{pre}\text{-}J}(x, y, X, Y)$$
$$\wedge \quad x' \le y' \,\wedge\, x' \in X \,\wedge\, y' \in Y \,\wedge$$
$$\wedge \quad \forall z \left(x' < z < y' \;\rightarrow\; z \notin X \cup Y \right).$$

It is easy to see that, in every formula $\gamma_j(x', y')$ of G, the variable x' determines y'. Indeed, $\beta_j(x', y)$ determines y from x', $\varphi^{\mathsf{fact}}_{\mathsf{pre}\text{-}J}(x, y, X, Y)$ determines x, X, Y from y, and the remaining of conjuncts of $\gamma_j(x', y')$ determine y' from X, Y. By symmetric arguments, y' determines x' in $\gamma_j(x', y')$, and hence G contains only rigid formulas. Moreover, it is clear from the previous constructions that if $w_i = w[x_i, y_i]$ is a factor of the canonical prefactorization of $w[x, y]$, then $w \vDash \gamma_j(x', y')$ for some formula $\gamma_j \in G$.

It remains to show how to compute the types under the guards in G. For this, we recall that every formula $\gamma_j(x', y')$ determines a factor w_i of a possible O-prefactorization. By Definition 5.17 we know that either w_i is a minimal infix such that $h(w_i) \in O$ (hence $w \vDash \alpha^{\mathrm{min}}_{\not\geq O}(x', y')$), or $h(w_i) \in P$ for some $=_{J^\circ}$-class P strictly above O. Furthermore, we can check with the formula $\varphi_O(x', y')$ which of the two cases holds. We can finally exploit Lemma 5.13 and our inductive hypothesis (claim C2) to construct a family F_{γ_j} of formulas that compute the types under the guard $\gamma_j(x', y')$. $\qquad\square$

The analogous lemma for J-factorizations is shown below.

Lemma 5.20. *Let O be a $=_{J^\circ}$-class and let $\alpha(x, y)$ be a rigid formula such that $w \vDash \alpha(x, y)$ implies $h(w[x, y]) \in O$. One can construct a formula $\varphi^{\mathsf{fact}}_O(x, y, X', Y')$ and a finite family G' of rigid formulas such that:*

(1) *if $w \vDash \alpha(x, y)$, then $w \vDash \varphi^{\mathsf{fact}}_O(x, y, X', Y')$ for some sets $X', Y' \subseteq [x, y]$;*

(2) *if $w \vDash \varphi^{\mathsf{fact}}_O(x, y, X', Y')$, then the pair (X', Y') represents a O-factorization of $w[x, y]$ that is rigidly traversable by G';*

(3) *one can compute the types under each guard $\gamma' \in G'$.*

Proof. Let $\varphi^{\mathsf{fact}}_{\mathsf{pre}\text{-}J}$ and G be as in Lemma 5.19 and let $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$ be some monadic predicates satisfying $\varphi^{\mathsf{fact}}_{\mathsf{pre}\text{-}J}(x, y, X', Y')$. The pair (X', Y') represents an O-prefactorization of $w[x, y]$ with factors $w_1 = w[x_1, y_1], \ldots, w_n = w[x_n, y_n]$. For the sake of simplicity, we assume that n is even (otherwise, the last factor has to be treated differently).

We define a new factorization by grouping the factors $w_1, \ldots, w_n$ two by two, namely, we let $v_i =^{\mathrm{def}} w_{2i-1} \cdot w_{2i}$ for all $1 \le i \le \frac{n}{2}$ and, accordingly, $X' =^{\mathrm{def}} \{x_1, x_3, \ldots, x_{n-1}\}$ and

$Y' \stackrel{\text{def}}{=} \{y_2, y_4, \dots, y_n\}$. Clearly, the sequence $v_1, \dots, v_{\frac{n}{2}}$ is a O-factorization of $w[x,y]$ and the two monadic predicates X', Y' can be defined from X, Y by means of an MSO formula. This is exactly how the formula $\varphi_O^{\text{fact}}(x, y, X', Y')$ defines a O-factorization of $w[x,y]$.

It remains to show that the defined O-factorization is rigidly traversable by a family G' and that one can compute the types under each guard $\gamma' \in G'$. For this, we recall that the original factorization $w_1, \dots, w_n$ is rigidly traversable by G, and we let G' be the set of all formulas $\gamma_1 \cdot \gamma_2 \stackrel{\text{def}}{=} \exists z' \; \gamma_1(x', z') \; \wedge \; \gamma_2(z' + 1, y')$, with $\gamma_1, \gamma_2 \in G$. Clearly, for every factor $v_i = w_{2i-1} \cdot w_{2i} = w[x_{2i-1}, y_{2i}]$, there there are formulas $\gamma_1, \gamma_2 \in G$ such that $w \vDash \gamma_1(x_{2i-1}, y_{2i-1})$ and $w \vDash \gamma_2(x_{2i}, y_{2i})$, whence $w \vDash (\gamma_1 \cdot \gamma_2)(x_{2i-1}, y_{2i})$. Finally, by Lemma 5.9 we conclude that one can compute the types under each guard $\gamma_1 \cdot \gamma_2 \in G'$. $\qquad \square$

What remains to be done is to find a suitable mechanism for determining the orbits and the memorable values of $h(w[x,y])$ on the basis of the orbits and the memorable values of $h(w_1), \dots, h(w_n)$, where $w_1, \dots, w_n$ are some factors of $w[x,y]$ as defined in Lemma 5.20. Of course, this cannot be done by comparing all the memorable values of $h(w_1), \dots, h(w_n)$, as there are unboundedly many of them and, furthermore, it is impossible to do so using rigidly guarded tests. Below, we show how to perform a sort of an approximation of this computation. The rough idea is to apply suitable renamings to the factors, so as to decrease the number of distinct memorable values. In doing so, however, one has to take into account the fact that some memorable values are shared between adjacent factors, and hence the renamings must be propagated to them.

Definition 5.21. We say that two sequences $s_1, \dots, s_n$ and $t_1, \dots, t_n$ of elements of $\mathcal{M}$ are *locally consistent* if

- all elements $s_1, \dots, s_n$, $t_1, \dots, t_n$, and the two products $s_1 \cdot \ldots \cdot s_n$ and $t_1 \cdot \ldots \cdot t_n$ belong to the same $=_{\mathcal{J}^\circ}$-class O,
- for every $1 \le i \le n-1$, there is a renaming π_i such that $\pi_i(s_i) = t_i$ and $\pi_i(s_{i+1}) = t_{i+1}$,
- $s_1 = t_1$ and $s_n = t_n$.

Similarly, the sequences are *almost locally consistent* if the first two conditions hold and, instead of $s_1 = t_1$ and $s_n = t_n$, one has $\pi(s_1) = t_1$ and $\pi(s_n) = t_n$ for some renaming π.

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

Lemma 5.22. *Let $s_1, \dots, s_n$ and $t_1, \dots, t_n$ be two sequences of elements of $\mathcal{M}$.*

(1) *If $s_1, \dots, s_n$ and $t_1, \dots, t_n$ are locally consistent, then $s_1 \cdot \ldots \cdot s_n = t_1 \cdot \ldots \cdot t_n$.*

(2) *If $s_1, \dots, s_n$ and $t_1, \dots, t_n$ are almost locally consistent, then $\tau(s_1 \cdot \ldots \cdot s_n) = t_1 \cdot \ldots \cdot t_n$ for some renaming τ.*

Proof. To prove the lemma, we need to introduce further definitions. As usual we identify elements of $\mathcal{M}$ with terms of some suitable term-based presentation system (the equality relation on the elements of $\mathcal{M}$ is thought of as the congruence of the term-based presentation system). We denote the arity of a term $s = o(d_1, \dots, d_k)$ by $\text{arity}(s) = k$. Let $\bar{s} = s_1, \dots, s_n$ be a sequence of terms. The *domain* $V_{\bar{s}}$ is the set of pairs (i, j), where $i \in \{1, \dots, n\}$ identifies a term s_i and $j \in \{1, \dots, \text{arity}(s_i)\}$ identifies a placeholder of a data value of s_i. We equip the domain $V_{\bar{s}}$ with a relation $E_{\bar{s}}$ that is the finest equivalence that groups every two elements (i, j) and $(i+1, j')$ of the domain whenever the j-th value of s_i and the j'-th value of s_{i+1} coincide. Of course, all elements of an equivalence class of $E_{\bar{s}}$ have the same associated data value. A *colouring* of $(V_{\bar{s}}, E_{\bar{s}})$ is a labelling of the equivalence classes of $E_{\bar{s}}$ by data values.

The sequence $\bar{s}$ naturally induces a colouring of $(V_{\bar{s}}, E_{\bar{s}})$ that maps each equivalence class C of $E_{\bar{s}}$ to the data value that is associated with the elements of C. An element (i, j) in $V_{\bar{s}}$ is a *border position* if $i = 1$ or $i = n$. Two colourings are *border-equal* if they agree on all border positions. A *border-class* is an equivalence class that contains a border position. We prove two claims now.

Claim 5.23. If two sequences $\bar{s}$ and $\bar{t}$ are locally consistent, then they define the same domain $V_{\bar{s}} = V_{\bar{t}}$ and the same equivalence $E_{\bar{s}} = E_{\bar{t}}$; furthermore, the corresponding colourings are border-equal.

Proof of claim. Let $\bar{s} = s_1, \ldots, s_n$ and $\bar{t} = t_1, \ldots, t_n$ be locally consistent sequences of terms. It is obvious that $\bar{s}$ and $\bar{t}$ have the same domain $V_{\bar{s}} = V_{\bar{t}}$, and it is equally easy to see that their colourings are border-equal. Moreover, the fact that two positions (i, j) and $(i+1, j')$ are $E_{\bar{s}}$-equivalent only depends on the equalities between the data values in s_i and s_{i+1}. Those equalities are the same as those for t_i and t_{i+1}, because there is a renaming π_i such that $\pi_i(s_i) = t_i$ and $\pi_i(s_{i+1}) = t_{i+1}$. $\qquad\square$

We give the following additional definitions. Two colourings have a *small difference* if their domains and equivalences are the same and they disagree on the colour of *at most one* equivalence class. Two locally consistent sequences have a small difference if their colourings have small difference.

Claim 5.24. If two sequences $\bar{s} = s_1, \ldots, s_n$ and $\bar{t} = t_1, \ldots, t_n$ are locally consistent and have small difference, then they give the same product, i.e. $s_1 \cdot \ldots \cdot s_n = t_1 \cdot \ldots \cdot t_n$.

Proof of claim. Let $\bar{s} = s_1, \ldots, s_n$ and $\bar{t} = t_1, \ldots, t_n$ be locally consistent sequences of terms with small difference. By the previous claim, the two sequences define the same domain, hereafter denoted by D, and the same equivalence, hereafter denoted by E. Let C be the equivalence class of E for which the two colourings induced by $\bar{s}$ and $\bar{t}$ differ. Let i_0 be the smallest number such that there is $(i_0, j) \in C$ and let i_1 be the largest number such that $(u_1, j') \in C$. Since $\bar{s}$ and $\bar{t}$ are locally consistent, we have that $i_0 > 1$ and $i_1 < n$. Let d and e be the colours associated with the class C by $\bar{s}$ and $\bar{t}$, respectively. We define π to be the renaming that swaps d and e and is the identity elsewhere. We observe that:

(1) $\pi \circ \pi$ is the identity,

(2) $\pi(s_i) = t_i$, for all $i \in \{i_0, \ldots, i_1\}$,

(3) $s_i = t_i$, for all $i \in \{1, \ldots, i_0 - 1\} \cup \{i_1 + 1, \ldots, n\}$,

(4) $\pi(u) = u$ where $u = t_{i_0-1} \cdot \ldots \cdot t_{i_1+1}$. This holds because neither d nor e are memorable values of u. Indeed, we observe that the three elements t_{i_0-1}, t_{i_1+1}, and u are in the same $=_{\mathcal{J}\circ}$-class and, furthermore, $t_{i_0-1} \geq_{\mathcal{R}} u$ and $t_{i_1+1} \geq_{\mathcal{L}} u$. From Lemma 5.11 we know that $t_{i_0-1} =_{\mathcal{R}} u$ and $t_{i_1+1} =_{\mathcal{L}} u$, and hence all $\mathcal{R}$-memorable values of u must occur in t_{i_0-1} and all $\mathcal{L}$-memorable values in u must occur in t_{i_1+1}. However, neither d nor e is $\mathcal{R}$-memorable in t_{i_0-1}, and neither d nor e is $\mathcal{L}$-memorable in t_{i_1+1}. By Proposition 2.12 we conclude that neither d nor e are memorable in u.

The above properties can be used to prove our second claim:

$$s_1 \cdot \ldots \cdot s_n \;=\; s_1 \cdot \ldots \cdot s_{i_0-2} \;\cdot\; \pi\big(\pi(s_{i_0-1} \cdot \ldots \cdot s_{i_1+1})\big) \;\cdot\; s_{i_1+2} \cdot \ldots \cdot s_n \qquad\qquad \text{(by (1))}$$

$$=\; s_1 \cdot \ldots \cdot s_{i_0-2} \;\cdot\; \pi\big(t_{i_0-1} \cdot \ldots \cdot t_{i_1+1}\big) \;\cdot\; s_{i_1+2} \cdot \ldots \cdot s_n \qquad\qquad \text{(by (2))}$$

$$=\; t_1 \cdot \ldots \cdot t_{i_0-2} \;\cdot\; \pi\big(t_{i_0-1} \cdot \ldots \cdot t_{i_1+1}\big) \;\cdot\; t_{i_1+2} \cdot \ldots \cdot t_n \qquad\qquad \text{(by (3))}$$

$$=\; t_1 \cdot \ldots \cdot t_{i_0-2} \;\cdot\; t_{i_0-1} \cdot \ldots \cdot t_{i_1+1} \;\cdot\; t_{i_1+2} \cdot \ldots \cdot t_n \;. \qquad\qquad \text{(by (4))}$$

$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad$ □

We can finally turn to the main proof. Consider two locally consistent sequences $\bar{s}$ and $\bar{t}$. By the first claim, $\bar{s}$ and $\bar{t}$ define the same domain D and the same equivalence E, and the corresponding colourings are border-equal. We show that one can transform $\bar{s}$ into $\bar{t}$ by repeatedly changing the colouring of one equivalence class (note that it is sufficient to describe how the colouring evolves during the transformation steps). The second claim will then imply that the resulting sequences give all the same product.

Let $C_1, \ldots, C_m$ be the non-border equivalence classes of E and let $f_1, \ldots, f_m$ be fresh data values, not appearing in $\bar{s}$ or in $\bar{t}$. Transforming $\bar{s}$ to $\bar{t}$ is done in two phases. One first performs the following m steps: at step i, for $i = 1, \ldots, m$, one recolours the i-th equivalence class C_i by f_i. One then performs other m steps during which one recolours the i-th equivalence class C_i, for $i = 1, \ldots, m$, by its colour in $\bar{t}$.

This concludes the proof of the first part of the lemma. The second part of the lemma that deals with two almost locally consistent sequences $s_1, \ldots, s_n$ and $t_1, \ldots, t_N$ follows easily from the existence of a renaming τ such that $s_1, \ldots, s_n$ is locally consistent with $\tau(t_1), \ldots, \tau(t_n)$.
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad$ □

Lemma 5.22 shows that the product of a sequence of elements, such as the one induced by an O-factorization, does not change if one replaces the sequence with another locally consistent one. Our last preparatory lemma shows that there always exists a locally consistent sequence over a set of data values of bounded cardinality. This enables the possibility of guessing a locally consistent sequence in MSO, that is, by means of a tuple of monadic predicates.

Lemma 5.25. *For every sequence $\bar{s}$ of elements of $\mathcal{M}$, there exists a sequence $\bar{t}$ that is locally consistent with $\bar{s}$ and uses at most $4\|\mathcal{M}\|$ distinct data values.*

Proof. We can reuse the objects in the proof of Lemma 5.22, namely, by identifying elements of $\mathcal{M}$ with terms, we construct from the sequence $\bar{s} = s_1, \ldots, s_n$ a domain V, an equivalence E and a corresponding colouring of the equivalence classes of E. Moreover, without loss of generality, we can assume that:

- the set D of all data values is the set of positive integers,
- the data values of s_1 and s_n belong to the set $\{1, \ldots, 2\|\mathcal{M}\|\}$ (call this range of values the set of *border colours*),
- all other data values, which do not appear in s_1 or s_n, are strictly above $4\|\mathcal{M}\|$.

By using an induction on $i = 2, \ldots, n-1$, we transform the colouring induced by $\bar{s}$ into a new colouring that associates with all positions (i', j) in the domain, where $i' \leq i$, a data value smaller than or equal to $4\|\mathcal{M}\|$. Such a transformation is performed at each step $i = 2, \ldots, n-1$ as follows. If some colours greater than $4\|\mathcal{M}\|$ are associated with the classes

of the positions $(i+1,1)$, ..., $(i+1, \mathsf{arity}(s_{i+1}))$, then we recolour these classes with data values ranging over $\{2\|\mathcal{M}\|+1, \ldots, 4\|\mathcal{M}\|\}$ – intuitively, this amounts at renaming the data values of s_{i+1}. Note that we can perform such a recolouring because there are at most $\|\mathcal{M}\|$ data values involved in the term s_i and at most $\|\mathcal{M}\|$ data values involved in the term s_{i+1}, and hence there are always at least $\|\mathcal{M}\|$ fresh data values from $\{2\|\mathcal{M}\|+1, \ldots, 4\|\mathcal{M}\|\}$ that we can choose.

At the end of the transformation, the resulting colouring uses only data values from $\{1, \ldots, 4\|\mathcal{M}\|\}$. This means that the sequence of terms $\bar{t} = t_1, \ldots, t_n$ that corresponds to this colouring fulfils the claim of the lemma. $\qquad\square$

We have now all the ingredients to prove the inductive step for claim C2 of Lemma 5.7.

Lemma 5.26 (Induction step for C2)**.** *For all rigid guards $\alpha(x,y)$ such that $w \vDash \alpha(x,y)$ implies $h(w[x,y]) \geq_{\mathcal{J}^\circ} O$, one can compute the types under α.*

Proof. The construction starts as in the proof of Lemma 5.15, but things change at point 4., since the construction there exploits aperiodicity. We continue the construction, without the assumption of aperiodicity, as follows.

(4)   Using the formula $\varphi_O^{\mathsf{fact}}(x,y,X',Y')$ from Lemma 5.20, one guesses a pair (X',Y') of monadic predicates that represent a rigidly-traversable O-factorization $w_1, \ldots, w_n$ of $w[x,y]$.

(5)   Let G' be the family of formulas with respect to which the defined O-factorization is rigidly traversable. For every factor w_i, one guesses a corresponding formula γ_i in G' that holds over the endpoints of the factor w_i, namely, such that $w \vDash \gamma_i(x_i, y_i)$, where x_i (resp., y_i) is the left (resp., right) endpoint of w_i in w. This information can be encoded by a tuple of monadic predicates and easily verified to be correct.

(6)   One then guesses some terms $t_1, \ldots, t_n$ over a finite set C of data values of cardinality $4\|\mathcal{M}\|$. There are finitely many such terms, so this can be done using a tuple of monadic predicates. Moreover, this is done in such a way that the information concerning the term t_i is located on the factor w_i, say on the first letter.

(7)   One also guesses some terms $u_1, \ldots, u_n$ over the same set C of data values as above, where each term u_i is meant to be the product of the first i terms $t_1, \ldots, t_i$. One can represent the terms $u_1, \ldots, u_n$ with new monadic predicates and then check that the guess is correct by verifying that $u_1 = t_1$ and $u_{i+1} = u_i \cdot t_i$ for all $1 \leq i \leq n-1$.

(8)   One checks that the sequence of terms $t_1, \ldots, t_n$ is almost locally consistent with the sequence of elements that is induced by the O-factorization, that is, the sequence $h(w_1), \ldots, h(w_n)$. Below we explain how to do so.

Consider any index $1 \leq i \leq n-1$ (this amounts at using a universal first-order quantification). Recall from the previous constructions that γ_i and γ_{i+1} are rigid formulas in G' that hold between the endpoints of w_i and w_{i+1}, respectively. Also recall, from the statement of Lemma 5.20, that one can compute the types under the guards γ_i and γ_{i+1}. In particular, there are finite families of inward-rigid formulas $F = (\varphi_o)_{o \in O}$ and $F' = (\varphi'_o)_{o \in O}$ that compute the types under the guards γ_i and γ_{i+1}. By the sub-definability Lemma 5.3, we can rewrite each inward-rigid formula $\varphi_o(x', \bar{z}, y')$ (resp., $\varphi'_o(x', \bar{z}, y')$) as a finite disjunction of rigid formulas $\varphi_{o,1}(x', \bar{z}, y') \vee \ldots \vee \varphi_{o,\ell}(x', \bar{z}, y')$ (resp., $\varphi'_{o,1}(x', \bar{z}, y') \vee \ldots \vee \varphi'_{o,m}(x', \bar{z}, y')$).

We first verify, using disjunctions, that the orbits of $h(w_i)$ and t_i coincide, namely, that some formula $\varphi_{o,j}(x_i, \bar{z}, y_i)$ holds over the endpoints x_i, y_i of w_i and over some positions $\bar{z}$, where o is the orbit of the term t_i. We do the same for $h(w_{i+1})$ and t_{i+1}, namely, we verify that some formula $\varphi'_{o',j'}(x_{i+1}, \bar{z}', y_{i+1})$ holds over the endpoints x_{i+1}, y_{i+1} of w_{i+1} and over some positions $\bar{z}'$, where o' is the orbit of the term t_{i+1}.

Next, we verify that the equalities between the memorable values of $h(w_i)$ and $h(w_{i+1})$ are the same as the equalities between the data values of the terms t_i and t_{i+1}. For this, we suppose that $t_i = o(d_1, \ldots, d_k)$ and $t_{i+1} = o'(d'_1, \ldots, d'_{k'})$ and we check that for all pairs of data values $d_h, d'_{h'}$:

(a) if $d_h = d'_{h'}$, then $(\beta \cdot \beta')(z_h, z'_{h'}) \wedge z_h \sim z'_{h'}$ holds, where $\beta(z_h, y') \stackrel{\text{def}}{=} \varphi_{o,j}(\star, \bar{\star}, z_h, \bar{\star}, y')$, $\beta'(x', z'_{h'}) \stackrel{\text{def}}{=} \varphi'_{o',j'}(x', \bar{\star}, z'_{h'}, \bar{\star}, \star)$, and $\beta \cdot \beta'$ is defined as in Lemma 5.9 (note that $\beta \cdot \beta'$ is rigid since $\varphi_{o,j}$ and $\varphi'_{o',j'}$ are rigid);

(b) if $d_h \neq d'_{h'}$, then $(\beta \cdot \beta')(z_h, z'_{h'}) \wedge z_h \nsim z'_{h'}$ holds, where β, β', and $\beta \cdot \beta'$ are defined as in (a).

Finally, it remains to check that the equalities between the memorable values of $h(w_1)$ and $h(w_{i+1})$ are the same as the equalities between the data values of t_1 and t_n. For this, we assume that $t_1 = o(d_1, \ldots, d_k)$ and $t_n = o(d'_1, \ldots, d'_{k'})$ and we check that for all pairs of data values $d_h, d'_{h'}$:

(c) if $d_h = d'_{h'}$, then $\beta(z_h, z'_{h'}) \wedge z_h \sim z'_{h'}$ holds, where $\beta(z_h, z'_{h'})$ is defined as $\exists x', y' \; \varphi_{o,j}(x', \bar{\star}, z_h, \bar{\star}, \star) \wedge \alpha(x', y') \wedge \varphi'_{o',j'}(\star, \bar{\star}, z'_{h'}, \bar{\star}, y')$ – note that β is rigid since $\varphi_{o,j}$, $\varphi'_{o',j'}$, and α are all rigid;

(d) if $d_h \neq d'_{h'}$, then $\beta(z_h, z'_{h'}) \wedge z_h \nsim z'_{h'}$ holds, where $\beta(z_h, z'_{h'})$ is defined as in (c).

(9) At this point, we know that $h(w[x,y]) = h(w_1) \cdot \ldots \cdot h(w_n)$ is in the same orbit as $u_n = t_1 \cdot \ldots \cdot t_n$. What remains to be done is to determine some positions in $w[x,y]$ that carry the memorable values of $h(w[x,y])$.

For this, we use once more Lemma 5.11 and Proposition 2.12. Since the elements $h(w_1)$ and $h(w[x,y])$ belong to the same $=_{\mathcal{J}^\circ}$-class O and since $h(w_1) \geq_{\mathcal{R}} h(w[x,y])$, Lemma 5.11 implies $h(w_1) =_{\mathcal{R}} h(w[x,y])$. In a similar way, we derive $h(w_n) =_{\mathcal{L}} h(w[x,y])$. Now, Proposition 2.12 implies that every memorable value of $h(w[x,y])$ is also memorable in $h(w_1)$ or in $h(w_n)$. Thus, a formula can easily locate in a rigid way some positions with the memorable values of $h(u_1)$ and $h(u_n)$.

Towards a conclusion, we recall that the two sequences $h(w_1), \ldots, h(w_n)$ and $t_1, \ldots, t_n$ are almost locally consistent. This means that there is a correspondence between the memorable values of $h(w[x,y])$ (resp., $h(w_1)$, $h(w_n)$) and the data values of u_n (resp., t_1, t_n). In particular, for every memorable value e of $h(w[x,y])$, one can identify a corresponding data value d that occurs in u_n, and hence in t_1 or t_n. From this, one determines in a rigid way a corresponding position in w_1 or w_n that carries the memorable value e. □

6. Logics for finite memory automata

In this section, we consider again a variant of MSO$^\sim$ with guards on data tests. More precisely, we consider the logic *semi-rigidly guarded MSO$^\sim$*, as it was introduced in Section

3, but now interpreted it over the class of data words[1]. The goal is to relate definability in semi-rigidly guarded MSO˜ to recognizability of data languages by means of finite memory automata [22, 24].

Recall that, in semi-rigidly guarded MSO˜, every data test $y \sim z$ is conjoint with a formula $\alpha(x, y) \wedge \beta(x, z)$, where $\alpha(x, y)$ (resp., $\beta(x, z)$) is a formula of semi-ridigly guarded MSO˜ that determines at most one position y (resp., z) from each position x. Below, we recall the definition of finite memory automaton, precisely the one from [9] which is closer in spirit to orbit-finite data monoids. For this we reuse some of the notions introduced in Section 2, in particular, that of $\mathcal{G}_D$-set and that of equivariant subset. All elements s in a $\mathcal{G}_D$-set are tacitly assumed to have finite memory, denoted for short by $\mathsf{mem}(s)$. For simplicity, we also avoid denoting explicitly the group actions underlying $\mathcal{G}_D$-sets.

Definition 6.1. A *finite memory automaton* (*FMA*) is a tuple $\mathcal{A} = (\Sigma, Q, I, F, T)$, where:

- Σ is an orbit-finite $\mathcal{G}_D$-set that represents the input alphabet (e.g., $D \times A$);
- Q is an orbit-finite $\mathcal{G}_D$-set that represents the configuration space;
- I and F are equivariant subsets of Q (i.e., unions of orbits of Q) that contain the initial and final configurations, respectively;
- T is an equivariant subset of $Q \times \Sigma \times Q$ that describes the possible transitions.

The notion of *successful run* of $\mathcal{A}$ and that of *recognized language* are the usual ones for automata running on finite words (they are just generalized to possibly infinite alphabets and configuration spaces). An FMA $\mathcal{A} = (\Sigma, Q, I, F, T)$ is said to be

(1)   *unambiguous*, if it admits at most one successful run on each input data word;

(2)   *deterministic*, if I is a singleton and $(p, c, q), (p, c, q') \in T$ implies $q = q'$;

(3)   *co-deterministic*, if F is a singleton and $(p, c, q), (p', c, q) \in T$ implies $p = p'$.

FMA can be easily shown to be closed under union and intersection (the classical constructions for finite state automata can be used). In addition, deterministic FMA and co-deterministic FMA are closed under complementation. However, FMA cannot be determinized and are not closed under complementation. We also know that FMA (even deterministic or co-deterministic ones) are strictly more expressive than orbit-finite data monoids. For example, a separating language is

$$L_{\curvearrowleft} \stackrel{\text{def}}{=} \{d_1 \ldots d_n \ : \ \exists 1 < i \le n \ d_1 = d_i\}$$

which is recognized by a deterministic FMA, but is not recognizable by an orbit-finite data monoid. Moreover, unlike classical finite automata, deterministic and co-deterministic FMA are incomparable, and both are strictly less expressive than unambiguous FMA, as easily witnessed by the language $L_{\curvearrowleft}$ and its reverse.

With the lack of classical tools like the subset construction, the problem of finding logical characterizations of classes of data languages recognized by FMA becomes challenging. In the following, we give partial results towards this goal.

The first result shows that FMA, even unambiguous ones, are at least as expressive as semi-rigidly guarded MSO˜ interpreted on data words.

---

[1]Results similar to those presented in this section can be obtained for semi-rigidly guarded MSO˜ interpreted over ranked data trees. However, we prefer to avoid dealing with the technicalities concerning trees and we present instead the results for data words as a proof of concept.

Theorem 6.2. *Every language of data words defined by a semi-rigidly guarded MSO~ sentence is effectively recognized by an unambiguous FMA.*

In proving the above result, it is quite natural to try an approach similar to that of Theorem 4.2. Unfortunately, it is difficult to adapt the approach to the present setting, because here there are semi-rigid guards that, from a certain position x, determine at most one position y either to the left or to the right of x, and hence one cannot use deterministic or co-deterministic automata. Nonetheless, we can exploit (unambiguous) non-determinism of FMA to annotate the input data words with additional symbols and data values that ease the translation.

We begin by giving a simple lemma that shows that, as for classical languages, it is possible to reason about recognizability by unambiguous FMA modulo annotations computed by length-preserving functions on data words. We fix the following notation. Let Σ and Δ be two orbit-finite alphabets (e.g., $\Sigma = A \times D$ and $\Delta = B \times D$). Given two data words $u \in \Sigma^*$ and $v \in \Delta^*$ of the same length, we denote by $u \otimes v$ the *convolution* of u and v, defined by $(u \otimes v)(i) = \big(u(i), v(i)\big)$ for all $1 \le i \le |u| = |v|$.

Lemma 6.3. *Let $f : \Sigma^* \to \Delta^*$ be a function that maps any data word u to a data word $f(u)$ of the same length, let $L \subseteq \Sigma^*$ be a data language, and let L_f be the language of data words of the form $u \otimes v$, where $u \in L$ and $v = f(u)$. If L_f is recognized by an unambiguous FMA, the so is L.*

Proof. Let $\mathcal{A}_f = (\Sigma \times \Delta, Q, I, F, T)$ be an unambiguous FMA that recognizes L_f. Define $\mathcal{A} = (\Sigma, Q, I, F, T')$, where T' contains all and only the transitions of the form (p, a, q) such that $\big(p, (a, b), q\big) \in T$ for some $b \in \Delta$. Clearly $\mathcal{A}$ recognizes L. Moreover, $\mathcal{A}$ is unambiguous: if $u \in L$, then there is a unique $v \in \Delta^*$ such that $u \otimes v \in L_f$, and there is a unique successful run of $\mathcal{A}$ on $u \otimes v$. $\square$

Below, we fix a sentence ψ of semi-rigidly guarded MSO~ and we prove the data language defined by ψ is recognized by an unambiguous FMA. To ease the translation, we will use Lemma 6.3 to freely annotate data words with the outputs computed functional finite memory transducers. Formally, we define a *functional finite memory transducer* (*FMT* for short) just as an unambiguous FMA over the alphabet $\Sigma \times \Delta$ that never accepts two data words of the form $u \otimes v$ and $u \otimes v'$, with $v \ne v'$.

The first step consists of rewriting ψ into a sentence of classical MSO (hence containing no data tests), but interpreted over data words with appropriate annotations. This will reduce the problem to constructing some FMTs computing the appropriate annotations. We use a technique similar to the proof of Theorem 3.4, namely, we substitute in ψ, starting from the innermost subformulas, every occurrence of a semi-rigidly guarded data test $\varphi(x, y, z) = \alpha(x, y) \wedge \beta(x, z) \wedge y \sim z$, with the formula $\varphi^*(x, y, z) = \alpha^*(x, y) \wedge \beta^*(x, z) \wedge x \in c_{\alpha,\beta}^{\sim}$, where $c_{\alpha,\beta}^{\sim}$ is a fresh unary predicate associated with that occurrence. Let ψ^* be the sentence of classical MSO that is obtained from ψ by applying the above transformation and let C be the alphabet with the new predicates $c_{\alpha,\beta}^{\sim}$. We have that for every data word $u \in (A \times D)^*$

$$u \vDash \psi \qquad \text{iff} \qquad u \otimes v \vDash \psi^*$$

where v is the unique annotation for u that satisfies

$$\theta_{\alpha,\beta} \; =^{\text{def}} \; \forall x \; \big(x \in c_{\alpha,\beta}^{\sim} \; \leftrightarrow \; \exists y, z \; \alpha^*(x, y) \wedge \beta^*(x, z) \wedge y \sim z \big)$$

for each $c^{\sim}_{\alpha,\beta} \in C$. The MSO sentence ψ^* can be translated to a deterministic finite state automaton $\mathcal{A}^*$, and the latter can be intersected with suitable FMTs recognizing the languages defined by $\theta_{\alpha,\beta}$. Thus, we have reduced the problem of constructing an unambiguous FMA for ψ to the problem of constructing FMTs for the sentences $\theta_{\alpha,\beta}$.

Now, we focus our attention on the sentences $\theta_{\alpha,\beta}$. First of all, note that a sentence $\theta_{\alpha,\beta}$ contains no nested data tests; in particular, semi-rigid guards are classical MSO formulas. In order to further ease the translation, we apply a second normalization step. This time the goal is to rewrite $\theta_{\alpha,\beta}$ so as to avoid the presence of semi-rigid guards with *crossing patterns*, namely, guards that on the same annotated word $w = u \otimes v$, determine distinct positions y and y' from x and x', respectively, such that

$$\big[\mathsf{min}(x,y), \mathsf{max}(x,y)\big] \cap \big[\mathsf{min}(x',y'), \mathsf{max}(x',y')\big] \neq \varnothing.$$

We formalize below the key argument that enables such a transformation:

Lemma 6.4. *Every semi-rigid MSO formula $\varphi(x,y)$ which determines y from x can be rewritten as a finite disjunction of semi-rigid MSO formulas $\varphi_1(x,y), \dots, \varphi_n(x,y)$ that are* cross-free, *namely, such that for all words w, all positions x, x', y, y' in it, and all indices $i = 1, \dots, n$:*

$$\begin{cases} w \vDash \varphi_i(x,y) \\ w \vDash \varphi_i(x',y') \\ \big[\mathsf{min}(x,y), \mathsf{max}(x,y)\big] \cap \big[\mathsf{min}(x',y'), \mathsf{max}(x',y')\big] \neq \varnothing \end{cases} \qquad implies \qquad y = y'.$$

Proof. Without loss of generality, suppose that $\varphi(x,y)$ always entails $x \leq y$ (if this is not the case, rewrite φ as a disjunction of two formulas that entail respectively $x \leq y$, $y \leq x$, and then prove the lemma separately for each disjunct). The lemma follows essentially from compositionality of MSO [36]. Indeed, we can reuse the same argument as in the beginning of the proof of the sub-definability Lemma 5.3 (cf. Claim 5.4) to show that the formula $\varphi(x,y)$ is equivalent to a finite disjunction of the form

$$\bigvee_{i=1\dots n} \exists z \ \alpha_i(x,z) \ \wedge \ \beta_i(z,y) \ \wedge \ x \leq z \leq y.$$

for some MSO formulas $\alpha_i(x,z)$, $\beta_i(z,y)$.

Thus, the desired formulas $\varphi_i(x,y)$ are nothing but the disjuncts above, namely, the semi-rigid formulas $\exists z \ \alpha_i(x,z) \ \wedge \ \beta_i(z,y) \ \wedge \ x \leq z \leq y$, for $i = 1 \dots n$. Indeed, suppose that there are positions x, x', y, y' in w such that $w \vDash \varphi_i(x,y)$, $w \vDash \varphi_i(x',y')$, and $\big[\mathsf{min}(x,y), \mathsf{max}(x,y)\big] \cap \big[\mathsf{min}(x',y'), \mathsf{max}(x',y')\big] \neq \varnothing$. The latter condition implies the existence of a position z satisfying $x \leq z \leq y$, $x' \leq z \leq y'$, $\alpha_i(x,z) \wedge \beta_i(z,y)$, and $\alpha_i(x',z) \wedge \beta_i(z,y')$. It follows that z satisfies also $\alpha_i(x,z) \wedge \beta_i(z,y')$, and hence $w \vDash \varphi_i(x,y')$. Finally, since φ_i is semi-rigid, we deduce that $y = y'$. $\qquad\square$

We can apply Lemma 6.4 to all semi-rigid guards $\alpha^*(x,y)$ and $\beta^*(x,z)$ in a sentence $\theta_{\alpha,\beta}$ and then commute disjunctions and existential quantifications. In this way we obtain a sentence equivalent to $\theta_{\alpha,\beta}$:

$$\theta'_{\alpha,\beta} \ \overset{\mathrm{def}}{=} \ \forall x \ \Big(x \in c^{\sim}_{\alpha,\beta} \ \leftrightarrow \ \bigvee_{\substack{i=1\dots n \\ j=1\dots m}} \exists y,z \ \alpha^*_i(x,y) \ \wedge \ \beta^*_j(x,z) \ \wedge \ y \sim z \Big)$$

where $\alpha^*_i(x,y)$ and $\beta^*_j(x,z)$ are semi-rigid cross-free formulas. The following definition and lemma give the additional ingredients to complete the translation of $\theta'_{\alpha,\beta}$ into an equivalent FMT.

Definition 6.5. Let $\varphi(x, y)$ be a semi-rigid cross-free formula and let w be a data word. The *trace of* φ *on* w is the annotated data word $w \otimes t_\varphi$, where t is the word over the orbit-finite alphabet $\{\bot\} \uplus D$ such that, for all $1 \le z \le |w| = |t|$,

$$t_\varphi(z) \; =^{\text{def}} \; \begin{cases} d & \text{if there are } x, y \text{ such that } z \in \big[\min(x,y), \max(x,y)\big], \\ & \quad w \vDash \varphi(x,y), \text{ and } \mathsf{mem}(w(y)) = \{d\} \\ \bot & \text{otherwise.} \end{cases}$$

(note that d above is well defined because φ is semi-rigid and cross-free).

Lemma 6.6. *For every semi-rigid cross-free formula* $\varphi(x, y)$*, one can construct an FMT* $\mathcal{T}$ *that defines the traces of* φ *on the input data words.*

Proof. For the sake of brevity, let us call *memorable position of z in w* the unique position y (if there is any) for which exists x such that $z \in \big[\min(x,y), \max(x,y)\big]$ and $w \vDash \varphi(x,y)$. The memorable position of z is clearly MSO-definable from z, and so are the following properties parametrized by z: "z has some memorable position", "z is the memorable position of itself", and "z and $z + 1$ have the same memorable position". As usual, we can assume without loss of generality that the latter properties are explicitly encoded on the positions z of an input data word w by means of additional bits of information (we recall that unambiguity is preserved when we project out these bits).

It easy to construct a deterministic FMA that parses a word $w \otimes t$ and performs the following actions on the basis of the bits of information at position z:

- it checks that $t(z) \ne \bot$ iff z has some memorable position,

- if z is the memorable position of itself, then it also verifies that the data value in $t(z)$ is equal to the data value in $w(z)$,

- finally, if z has the same memorable position as $z + 1$, then it stores the data value of $t(z)$ and on the next position verifies that the value is the same as the one in $t(z + 1)$.

We omit the formal specification of the deterministic FMA and we only remark that it accepts precisely the words of the form $w \otimes t_\varphi$, when the bits of information on z are accessible. Moreover, the latter bits can be produced by a functional transducer, so using Lemma 6.3 we can easily obtain an unambiguous FMA recognizing the desired language. $\square$

We can now conclude the proof of Theorem 6.2. We recall that the sentences $\theta'_{\alpha,\beta}$ are of the form

$$\forall x \; \Big(x \in c^{\sim}_{\alpha,\beta} \; \leftrightarrow \; \bigvee_{\substack{i=1\ldots n \\ j=1\ldots m}} \exists y, z \; \alpha_i^*(x,y) \wedge \beta_j^*(x,z) \wedge y \sim z \Big)$$

Assuming that the above sentences are evaluated on data words annotated with the traces t_{α^*} and t_{β^*}, we can rewrite them as

$$\forall x \; \Big(x \in c^{\sim}_{\alpha,\beta} \; \leftrightarrow \; \bigvee_{\substack{i=1\ldots n \\ j=1\ldots m}} t_{\alpha^*}(x) = t_{\beta^*}(x) \ne \bot \Big)$$

which can be easily verified by a deterministic (hence unambiguous) FMA. Putting everything together, we exploit closure of unambiguous FMA under unions, intersections, and projections of FMT-definable annotations to obtain an unambiguous FMA that recognizes the language defined by ψ.

The second result shows that semi-rigidly guarded MSO~ does not capture the entire class of data languages recognizable by unambiguous FMA. In fact, the separating language is even recognized by a deterministic FMA:

Proposition 6.7. *There is a data language recognized by a deterministic FMA that cannot be defined in semi-rigidly guarded MSO~.*

Proof. For the sake of simplicity, we will assume that data values range over the natural numbers. Consider the language $L_{\rightsquigarrow^*}$ that contains all and only the data words over the alphabet $\mathbb{N}$ of the form [2]

$$d_{i_0+1} \ \ldots \ d_{i_1} \ d_{i_1+1} \ \ldots \ d_{i_2} \ \ldots\ldots \ d_{i_{\ell-1}+1} \ \ldots \ d_{i_\ell}$$

where $\ell \in \mathbb{N}$, $0 = i_0 < i_1 < i_2 < \ldots < i_{\ell-1} < i_\ell = n$, $d_{i_j+1} = d_{i_{j+1}}$ for all $0 \leq j < \ell$, and $d_k \neq d_{i_j}$ for all $i_j + 1 < k < i_{j+1}$ (for the sake of readability, we added arcs linking those data values that are required to be equal). It is easy to see that $L_{\rightsquigarrow^*}$ is recognized by a deterministic FMA: at each phase, the automaton stores the value under the current position and then moves to the right looking for another occurrence of the stored value; if it does not find such an occurrence, then it rejects, otherwise, as soon as the occurrence is found, the automaton moves to the next position (if there is any, otherwise it accepts) and starts a new phase.

Below, we fix a generic sentence ψ of semi-rigidly guarded MSO~ and we show that it cannot define the language $L_{\rightsquigarrow^*}$. For this, we consider data words in $L_{\rightsquigarrow^*}$ of the form

$$w_n \ = \ 1 \ \widehat{u_n^{(1)}} \ 1 \ 2 \ \widehat{u_n^{(2)}} \ 2 \ \ldots\ldots \ n \ \widehat{u_n^{(n)}} \ n$$

where each $u_n^{(i)}$ has length exactly n symbols and the only equalities between data values are those represented by the arcs, namely, the juxtaposition $u_n^{(1)} u_n^{(2)} \ldots u_n^{(n)}$ contains pairwise distinct values from the set $\mathbb{N} \setminus \{1, \ldots, n\}$. We aim at proving that, for n sufficiently large, almost all semi-rigidly guarded data tests performed by ψ fail, thus making it impossible to distinguish w_n from other data words outside $L_{\rightsquigarrow^*}$.

Our argument will make extensive use of the encodings of data tests described in Theorem 6.2. We begin by introducing the alphabet C consisting of one symbol $c_{\alpha,\beta}^{\sim}$ for each data test in ψ of the form $\alpha(x,y) \wedge \beta(x,z) \wedge y \sim z$. We then transform the data word w_n into a classical word w_n^* over C by labelling every position x of w_n^* with the set of symbols $c_{\alpha,\beta}^{\sim}$ such that $w_n \vDash \exists y,z \ \alpha(x,y) \wedge \beta(x,z) \wedge y \sim z$. Accordingly, we transform every sub-formula φ of ψ to a classical MSO formula φ^* that is equivalent in the following sense:

$$w_n \vDash \varphi \qquad \text{iff} \qquad w_n^* \vDash \varphi^*$$

(in particular, $\alpha(x,y) \wedge \beta(x,z) \wedge y \sim z$ is transformed into $\alpha^*(x,y) \wedge \beta^*(x,z) \wedge x \in c_{\alpha,\beta}^{\sim}$).

Now, we prove that every semi-rigid guard $\alpha(x,y)$ in ψ defines a position y from x that is either close to x or close to one of the endpoints of w_n. This is formally stated in the following claim:

---

[2]This language is a variant of an example given in [38] to study data languages recognized by pebble automata.

Claim 6.8. For every semi-rigid guard $\alpha(x,y)$, there is a number k_α that depends only on α (and not on n) and satisfies

$$\forall n > 1,\ 1 \le x \le |w_n| \quad w_n \vDash \alpha(x,y) \quad \text{implies} \quad \begin{array}{ll} 1) & y \le (n+2) \cdot k_\alpha, \quad \text{or} \\ 2) & y \ge (n+2) \cdot (n - k_\alpha), \quad \text{or} \\ 3) & |y - x| \le k_\alpha. \end{array}$$

The proof of the above claim is by induction on the number of data tests in $\alpha(x,y)$.

For the base case, we suppose that $\alpha(x,y)$ is a semi-rigid guard without data tests, namely, a formula of classical MSO defining a partial function. The claim follows essentially from the fact that there are no distinguished symbols in w_n that can be used to determine from x a position y that is far both from x and from the endpoints of w_n. Let $\mathcal{A}$ be a finite state automaton recognizing the language over $\{0,1\} \times \{0,1\}$ defined by $\alpha(x,y)$. Suppose that the claim above does not hold, namely, that for all $k \in \mathbb{N}$, there are $n \in \mathbb{N}$ and $1 \le x, y \le |w_n|$ such that $k < y < |w_n| - k$, $|y - x| > k$, and $w_n \vDash \alpha(x,y)$. Note that the latter condition $w_n \vDash \alpha(x,y)$ can be equally stated as $\langle w_n^-, \{x\}, \{y\} \rangle \in \mathscr{L}(\mathcal{A})$, where w_n^- is the word of length $|w_n|$ over a singleton alphabet. A simple pumping argument shows that there exist $n \in \mathbb{N}$ and some non-empty factors u_n, v_n of $\langle w_n^-, \{x\}, \{y\} \rangle$ such that (i) u_n occurs strictly between the positions x and y, v_n occurs to the right of both x and y, and erasing or repeating u and v results in new words that are also accepted by $\mathcal{A}$. We distinguish two cases depending on whether $|u| = |v|$ or not. In the former case, by removing the factor u and by repeating twice the factor v we obtain a new word of the same length as $\langle w_n^-, \{x\}, \{y\} \rangle$ that is also accepted by $\mathcal{A}$, but that identifies a new pair of positions (x, y'), with $y' \ne y$. This contradicts the fact that $\alpha(x,y)$ functionally defines y from x. In the latter case, we can reach the same contradiction by swapping the order of the two factors u and v. The above arguments prove that the claim holds for the considered semi-rigid guard $\alpha(x,y)$ without data tests.

As for the inductive case, suppose that $\alpha(x,y)$ uses some data tests based on semi-rigid guards $\beta_1, \ldots, \beta_h$. Assume that the inductive hypothesis holds for the guards $\beta_1, \ldots, \beta_h$, let $k_{\beta_1}, \ldots, k_{\beta_h}$ be the corresponding constants, and define k_β to be the maximum of these constants. Thanks to the inductive hypothesis, we know that for n sufficiently large, say $n \ge 2k_\beta$, every data test performed by $\alpha(x,y)$ between pairs of positions determined from $x' \in \{(n+2) \cdot k_\beta + 1, \ldots, (n+2) \cdot (n - k_\beta)\}$ are bound to fail – indeed, the only way a data test could succeed is by comparing positions of w_n that are connected by an arc, but the arcs span large distances by construction. Now, recall that $w_n \vDash \alpha(x,y)$ iff $w_n^- \vDash \alpha^-(x,y)$, where w_n^- is the word annotated with the outcomes of the data tests performed by $\alpha(x,y)$. In particular, we have that w_n^- is almost constant, that is, for all positions $x' \in \{(n+2) \cdot k_\beta + 1, \ldots, (n+2) \cdot (n - k_\beta)\}$, $c_{\beta_i, \beta_j}^{\sim} \notin w_n^-(x')$. Finally, by arguing like in the proof of the base case, we can conclude that there is a sufficiently large number k_α that satisfies the statement for $\alpha(x,y)$ and for all $n \in \mathbb{N}$.

We have just proved that the semi-rigid guards $\alpha(x,y)$ in ψ define positions that are either close to x or close to one of the endpoints of w_n. This means that, for n large enough, almost all data tests performed by ψ fail. In particular, there exist $n \in \mathbb{N}$ and two positions $y = (n+2) \cdot i + 1$ and $z = (n+2) \cdot (i+1)$ that carry the same data value i, but are never tested in ψ. This means that the word w_n' obtained from w_n by replacing the data value in z with a fresh value, also satisfies ψ. However, w_n' is not in the language $L_{\curvearrowright^*}$, and hence ψ cannot define $L_{\curvearrowright^*}$. $\qquad\square$

7. Conclusion and future work

We have shown that the algebraic notion of orbit-finite data monoid corresponds to a variant of monadic second-order logic 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.

Of course, it is not the first time the principle of guarding non-monadic predicates with suitable formulas is used as a mean of taming the expressiveness of a logic and recover decidability. What is more original and interesting 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 expressive power 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, which 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 [6], and it had to be developed even further in the present work.

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?

> Also recall that, in the classical setting of languages over finite alphabets, there exist effective characterizations of fragments of first-order logic within the class of regular languages. To what extent can we generalise these results in the presence of data over infinite alphabets?

Other questions are of purely algebraic nature:

> We used in our proofs the notion of term-based presentation of an orbit-finite data monoid. (cf. Definition 2.8). Such a presentation can be in a simple form, where the congruence $\approx$ is trivial, namely, where any two terms $o(d_1, \ldots, d_n)$ and $o'(d'_1, \ldots, d'_n)$ represent the same monoid element only if they are syntactically equal. Is it the case that every orbit-finite data monoid is the quotient of some orbit-finite data monoid having a simple term-based presentation?

We can answer positively the above question. Indeed, by Theorem 5.1 the elements s of an orbit-finite data monoid $\mathcal{M}$ can be translated to rigidly guarded MSO~ sentences ψ_s defining the set of data words whose products evaluate to s. We also know from Corollary 4.3 that the languages defined by ψ_s are recognized by an orbit-finite data monoid $\mathcal{M}'$. Moreover, a close inspection to the proof of Corollary 4.3 reveals that the inductive construction of $\mathcal{M}'$ can be performed at the level of term-based presentations of simple form.

We finally considered more powerful notions of recognizability, such as those obtained by extending finite state automata with registers [22, 24, 23, 9]:

> Does there exist a larger class of data languages that is as robust as that of orbit-finite data monoid, and gives, in particular, closures under all Boolean

operations and restricted forms of projections? Can we match new notions of recognizability with suitable logical formalisms?

In particular, we left open the problem of finding a logic that captures precisely the class of data languages recognized by unambiguous FMA, which is a candidate model for a robust class of data languages. As a matter of fact, in [16] we described a logic similar to semi-rigidly guarded MSO~ that captures data languages recognized by non-deterministic FMA. However, that logic was not natural, in the sense that it was not closed under negation, and, moreover, did not ease characterizations of sub-classes of data languages such as those definable in first-order logic. Regarding the latter problem, we also recall a result from [3] that shows that the problem of determining whether a language recognized by a non-deterministic FMA is definable in FO~ is undecidable. Thus, the following question is also worth to be investigated:

Can we characterize, among the languages recognized by unambiguous (or even deterministic) FMA, those recognizable by orbit-finite data monoids, or those definable in FO~?

Finally, the problem of finding a natural logic with the same expressiveness of unambiguous FMA is clearly related to the possibility of proving effective closure of unambiguous FMA under complementation. The latter problem is also open and challenging – we remark that a similar closure property holds trivially for *strongly unambiguous FMA* [15], namely, FMA that admit exactly one (accepting or rejecting) run on each data word.

References

[1] R. Alur, L. Fix, and T.A. Henzinger. Event-clock automata: a determinizable class of timed automata. *Theoretical Computer Science*, 211(1-2):253–273, 1999.

[2] P. Barcelo, C. Hurtado, L. Libkin, and P. Wood. Expressive languages for path queries over graph-structured data. In *Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 3–14. Association for Computing Machinery, 2010.

[3] 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.

[4] 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.

[5] H. Björklund and T. Schwentick. On notions of regularity for data languages. *Theoretical Computer Science*, 411(4-5):702–715, 2010.

[6] 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.

[7] M. Bojańczyk. Nominal monoids. *Theory of Computing Systems*, 53(2):194–222, 2013.

[8] M. Bojanczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data words. *ACM Transactions on Computational Logic*, 12(4):27, 2011.

[9] M. Bojanczyk, B. Klin, and S. Lasota. Automata with group actions. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 355–364. IEEE Computer Society, 2011.

[10] M. Bojańczyk and S. Lasota. An extension of data automata that captures XPath. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 243–252. IEEE Computer Society, 2010.

[11] M. Bojańczyk and S. Lasota. An extension of data automata that captures XPath. *Logical Methods in Computer Science*, 8(1), 2012.

[12] M. Bojańczyk, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data trees and XML reasoning. *Journal of the Association for Computing Machinery*, 56(3), 2009.

[13] B. Bollig, P. Habermehl, M. Leucker, and B. Monmege. A fresh approach to learning register automata. In *Proceedings of the 17th International Conference on Developments in Language Theory (DLT)*, volume 7907 of *Lecture Notes in Computer Science*, pages 118–130. Springer, 2013.

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

[15] T. Colcombet. Forms of determinism for automata (invited talk). In *Proceedings of the 29th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 14 of *LIPIcs*, pages 1–23. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[16] T. Colcombet, C. Ley, and G. Puppis. On the use of guards for logics with data. In *Proceedings of the 36th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 6907 of *Lecture Notes in Computer Science*, pages 243–255. Springer, 2011.

[17] S. Demri and R. Lazic. LTL with the freeze quantifier and register automata. *ACM Transactions on Computational Logic*, 10(3), 2009.

[18] S. Feferman and R. Vaught. The first-order properties of products of algebraic systems. *Fundamenta Mathematicae*, 47:57–103, 1959.

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

[20] J. A. Green. On the structure of semigroups. *Annals of Mathematics*, 54:163–172, 1951.

[21] M. Jurdzinski and R. Lazic. Alternation-free modal mu-calculus for data trees. In *Proceedings of the 22nd IEEE Symposium on Logic in Computer Science (LICS)*, pages 131–140. IEEE Computer Society, 2007.

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

[23] M. Kaminski and T. Tan. Tree automata over infinite alphabets. In *Pillars of Computer Science*, volume 4800 of *Lecture Notes in Computer Science*, pages 386–423. Springer, 2008.

[24] M. Kaminski and D. Zeitlin. Finite-memory automata with non-deterministic reassignment. *International Journal of Foundations of Computer Science*, 21(5):741–760, 2010.

[25] L. Libkin, W. Martens, and D. Vrgoč. Querying graph databases with XPath. In *Proceedings of the 16th International Conference on Database Theory (ICDT)*, pages 129–140. Association for Computing Machinery, 2013.

[26] A. Manuel, A. Muscholl, and G. Puppis. Walking on data words. In *Proceedings of the 8th International Computer Science Symposium in Russia (CSR)*, volume 7913 of *Lecture Notes in Computer Science*, pages 64–75. Springer, 2013.

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

[28] 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.

[29] 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.

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

[31] M.O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

[32] N. Robertson and P.D. Seymour. Graph minors ii, algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986.

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

[34] T. Schwentick and T. Zeume. Two-variable logic with two order relations. In *Proceedings of the 19th EACSL Annual Conference on Computer Science Logic (CSL)*, volume 6247 of *Lecture Notes in Computer Science*, pages 499–513. Springer, 2010.

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

[36] S. Shelah. The monadic theory of order. *Annals of Mathematics*, 102:379–419, 1975.

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

[38] T. Tan. Graph reachability and pebble automata over infinite alphabets. In *Proceedings of the 24th IEEE Symposium on Logic in Computer Science (LICS)*, pages 157–166. IEEE Computer Society, 2009.

[39] Z. Wu. Regular path queries on graphs with data: a rigid approach. *CoRR*, abs/1402.6067, 2014.