Forward Looking Logics and Automata

Clemens Ley
Mansfield College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Trinity 2011

Dedicated to Qian.

Acknowledgements

I would like to thank, first and foremost, my supervisor Michael Benedikt. His supervision has been extraordinary, and the meetings with him have always been most enjoyable. Also, I would like to thank Thomas Colcombet and Gabriele Puppis for the wonderful collaboration we had.

Abstract

This thesis is concerned with extending properties of regular word languages to richer structures. We consider intricate properties like the relationship between one-way and two-way temporal logics, minimization of automata, and the ability to effectively characterize logics. We investigate whether these properties can be extended to tree languages or word languages over an infinite alphabet.

It is known that linear temporal logic (LTL) is as expressive as first-order logic over finite words [Kam68, GPSS80]. LTL is a unidirectional logic, that can only navigate forwards in a word, hence it is quite surprising that it can capture all of first-order logic. In fact, one of the main ideas of the proof of [GPSS80] is to show that the expressiveness of LTL is not increased if modalities for navigating backwards are added. It is also known that an extension of bidirectional LTL to ordered trees, called Conditional XPath, is first-order complete [Mar04]. We investigate whether the unidirectional fragment of Conditional XPath is also first-order complete. We show that this is not the case. In fact we show that there is a strict hierarchy of expressiveness consisting of languages that are all weaker than first-order logic. Unidirectional Conditional XPath is contained in the lowest level of this hierarchy.

In the second part of the thesis we consider data word languages. That is, word languages over an infinite alphabet. We extend the theorem of Myhill and Nerode to a class of automata for data word languages, called deterministic finite memory automata (DMA). We give a characterization of the languages that are accepted by DMA, and also provide an algorithm for minimizing DMA.

Finally we extend theorems of Büchi, Schützenberger, McNaughton, and Papert to data word languages. A theorem of Büchi states that a language is regular iff it can be defined in monadic second-order logic. Schützenberger, McNaughton, and Papert have provided an effective

characterization of first-order logic, that is, an algorithm for deciding whether a regular language can be defined in first-order logic. We provide a counterpart of Büchi's theorem for data languages. More precisely we define a new logic and we show that it has the same expressiveness as non-deterministic finite memory automata. We then turn to a smaller class of data languages, those that are recognized by algebraic objects called orbit finite data monoids. We define a second new logic and show that it can define precisely the languages accepted by orbit finite data monoids. We provide an effective characterization of a first-order variant of this second logic, as well as of restrictions of first-order logic, such as its two variable fragment and local variants.

Contents

Introduction

Motivation

The regular word languages are certainly among the fundamental concepts in computer science. They are not only of theoretical importance, but also at the core of many applications. The reasons for their success are manyfold. On one hand, they are expressive enough to describe the patterns that occur in many settings, while on the other hand, they can be processed very efficiently: membership in a regular language can be tested in linear time and constant space in the size of the input word, and static analysis problems, such as language emptiness and equivalence, are decidable. In addition, the regular languages form a robust class of languages: they can be defined by several different formalisms, such as regular expressions, finite automata, finite monoids, and logics, and these formalisms are often invariant under small changes in their definition.

But, there are also other, more intricate, properties of the regular word languages that make them attractive. One such property is captured in the theorem of Myhill and Nerode. It shows that every regular language is accepted by a *unique* minimal deterministic finite automaton. In addition, an effective minimization procedure for deterministic finite automata can be extracted from the proof. Such a procedure clearly has practical applications, but in addition, it can easily be turned into an effective procedure for deciding emptiness and equivalence of regular languages.

Other intricate properties of the regular word languages concern the structure of their subclasses. Consider, for example the subclass of regular languages that can be defined by regular expressions without Kleene star. Schützenberger showed that this class of languages coincides with the class of languages that is accepted by certain algebraic structures, called aperiodic finite monoids [Sch65]. McNaughton, and Papert showed that a language is accepted by an aperiodic finite monoid iff it can be defined in first-order logic [MP71]. These results are remarkable as they connect seemingly unrelated mathematical objects in an unexpected way. Further research has shown that such a correspondence is by no means coincidental. In fact

many subclasses of the regular languages, e.g. those defined by fragments of first-order logics, temporal logics, or parallel complexity classes, can be given algebraic characterizations (see [Pin11, Str94]).

Kamp gave a further characterization of the class of first-order definable languages: he showed that this class is identical with the class of languages that can be defined by linear temporal logic with past modalities (PLTL) [Kam68]. Gabbay et al. showed that the past modalities do not increase the expressiveness of PLTL. More precisely he showed that LTL, the unidirectional fragment of PLTL, is sufficient to capture first-order logic [GPSS80].

The goal of this thesis is to investigate whether intricate properties of regular word languages can be extended to languages of richer structures. In particular, we consider the following questions:

1. Does the ability to get first-order complete unidirectional logics carry-over from the word setting to ordered trees?

2. Can the theorem of Myhill and Nerode be extended to word languages over an infinite alphabet?

3. Can the theorems of Schützenberger, McNaughton, and Papert be extended to word languages over an infinite alphabet?

In a nutshell, we answer the first question negatively, and the two other questions positively. Each question is answered in one chapter of the thesis.

Organization. This thesis consists of two parts. The first part, consisting of Chapter 1, studies the expressiveness of temporal logics on trees. The second part, consisting of Chapters 2 and 3, considers data word languages – word languages over an infinite alphabet. Chapter 2 studies minimization of automata for data languages and Chapter 3 is concerned with characterizations of logics for data languages.

Contribution

In the following we give an overview over the contributions of this thesis.

Chapter 1. How Big Must Complete Temporal Logics Be? Over words it is known that LTL has the same expressiveness as first-order logic [Kam68, GPSS80]. The formulas of LTL are built up from boolean operators, atomic formulas for node labels, two unary operators X (next) and F (eventually), and one binary operator U (until). Formulas of are evaluated with respect to a word and a position in this word, and the operators make it possible to navigate from one position in a word to a new position to the right of the original position.

Temporal logics are closely related to query languages for XML [Wor00]. XPath [Wor99], a W3C recommendation for querying XML, can be thought of as a version of unary LTL for trees: it has modalities for navigating in all four directions up, down, left, and right of a tree. For each of these directions there is one operator corresponding to the next operator and one operator corresponding to the eventually operator. It is known that unidirectional fragments of XPath, those that can only navigate downwards and in one horizontal direction, can be evaluated more efficiently over XML streams [Olt07, BJLW08].

Compared with first-order logic however, XPath is rather weak. It has been shown in [MdR05], that XPath cannot express all first-order definable properties. A first-order complete extension of XPath was proposed in [Mar04]: Conditional XPath is basically an extension of XPath that adds an until operator for each of the directions left, right, up, and down. Hence Conditional XPath can be considered an extension of (multidirectional) linear temporal logic for ordered trees. Clearly the question arises whether the unidirectional fragment of Conditional XPath is as expressive as the unrestricted language. *We show that this is not the case, that is, we show that unidirectional Conditional XPath is not first-order complete.*

In fact, we show a much stronger result. We consider a logic $\mathrm{CTL}^*_{\triangleleft\triangleright}$ which is known to be first-order complete [HT87, BL05]. Formulas of $\mathrm{CTL}^*_{\triangleleft\triangleright}$ are built up from boolean combinations of path quantification for downward, leftward and rightward paths, and LTL formulas can be evaluated on the quantified paths. The atomic formulas of these embedded LTL formulas are either node labels or lower level $\mathrm{CTL}^*_{\triangleleft\triangleright}$ formulas. We consider fragments of $\mathrm{CTL}^*_{\triangleleft\triangleright}$ obtained by restricting the "until-depth" of embedded LTL formulas. The until-depth of an LTL formula is the maximal nesting depth of the until operator in that formula. The nesting depth of the unary

operators is unaffected: that is, a formula with until depth at most k can still have an arbitrary number of nested X and F operators. In addition, we only restrict the until depth within one path quantifier. Hence, when evaluated on words, the logic $\mathrm{CTL}^*_{\triangleleft\triangleright}$, restricted to formulas of bounded until depth is as expressive as LTL (and hence as first-order logic), because arbitrary nesting of the until operator can be mimicked by using several path quantifiers. *We show that when evaluated on finite ordered trees, $CTL^*_{\triangleleft\triangleright}$ with bounded until-depth on downward path quantifiers becomes first-order incomplete.* This result gives rise to a strict infinite hierarchy of expressiveness. All languages in this hierarchy are weaker than first-order logic, and unidirectional Conditional XPath is contained in the lowest level. In addition, we show a similar result for $\mathrm{CTL}^*_{\triangleleft\triangleright}$ formulas with restricted until-depth on horizontal path quantifiers.

Chapter 2. How Small can Automata for Data Languages Get? The theorem of Myhill and Nerode not only presents a characterization of the regular word languages, it also provides an algorithm for minimizing deterministic finite automata. Myhill and Nerode associate every language L with an equivalence relation $\equiv_L$ on finite words. The theorem of Myhill and Nerode states that a language L is regular iff $\equiv_L$ has finitely many equivalence classes. In one direction of the proof a deterministic finite automaton $\mathcal{A}_L$ is constructed from the equivalence classes of $\equiv_L$. One can show that the automaton $\mathcal{A}_L$ is the state-minimal automaton accepting L.

In the second chapter of the thesis we consider extensions of the Myhill-Nerode procedure to data word languages. These are languages of words in which each position has a label from a finite alphabet, and a data value from an infinite alphabet. The study of data languages is motivated by applications such divers areas as XML processing [NSV04, Sch07], verification [AD94, HNSY92], semantics [Tze11], and knot theory [LPS11].

Several generalizations of finite automata for data languages have been studied previously (see [Seg06] for a survey). One such model is called finite memory automata [KF94]. These automata have, in addition to a finite state set, a finite set of registers which can store values from the input word. On each transition, a finite memory automaton can compare the current input symbol with the values stored in its registers. Based on this comparison, it can update its registers with the current input symbol and transition to a new control state.

We show that the theorem of Myhill and Nerode can be extended to deterministic finite memory automata. We also show that there is an effective minimization

procedure for deterministic finite memory automata. A priori, it is not clear what a *minimal* finite memory automaton is. One could try to minimize the number of control states, or the number of registers. We show that deterministic finite memory automata can be minimized with respect to both aspects: We show that for each deterministic finite memory automaton, there is a unique equivalent finite memory automaton that has the minimal number of states and the minimal number of registers. We also investigate the computational complexity of the minimization procedure.

Chapter 3. Which Data Languages Can be Defined by Logics? In this chapter, we extend other fundamental theorems of the regular word languages to data word languages. We focus on the theorems of Büchi and Schützenberger, McNaughton, and Papert.

Büchi showed that a language over a finite alphabet is regular iff it can be defined in monadic second-order logic [Büc60]. Hence the question arises which class of regular languages corresponds to the languages defined by first-order logic. The answer was given in two steps: Schützenberger showed that a language can be defined by a regular expression without Kleene star iff it is accepted by an algebraic structure called "aperiodic finite monoid" [Sch65]. A few years later McNaughton and Papert showed that a language is accepted by an aperiodic finite monoid iff it can be defined in first-order logic [MP71]. These results reveal surprising connections between seemingly unrelated mathematical objects such as logics, regular expressions, and monoids. In addition they also provide an effective characterization for first-order logic – that is, an algorithm for deciding whether a given a regular language can be defined in first-order logic. The results of Schützenberger, McNaughton, and Papert were the starting point for a research program that aimed at understanding the structure of the subclasses of regular word languages. In the process it has been shown that many of these subclasses can be characterized through several distinct formalisms (see [Pin11, Str94]).

In Chapter 3, we study effective characterizations of logics for data languages. First, we show that no effective characterizations can exist with respect to several automata models, even for very weak logics. These models are non-deterministic finite memory automata, two-way deterministic finite memory automata, and a very weak version of pebble automata. The proofs exploit that universality is undecidable for these models.

We then show that there are effective characterizations for one fragment and one extension of first-order logic with respect to certain subclasses of deterministic finite memory automata. Formulas of first-order logic for data languages are built up from Boolean operations, first-order quantification over word positions, a binary predicate $x < y$ for accessing the order on word positions, and a binary predicate $x \sim y$ for checking whether two positions have the same value. One of the main tools in our proofs is the minimization procedure for deterministic finite memory automata, discussed in the second chapter.

The results of Schützenberger make use of algebraic techniques. Until recently, algebraic techniques have not been applied to the study of data languages. The reason is that no class of algebraic structures was known that accepts an interesting class of data languages, while being well behaved enough to allow for effective characterizations. In [Boj11], Bojańczyk identified such a class of structures, called orbit finite data monoids. He showed that all languages that are accepted by aperiodic orbit finite data monoids can be defined in first-order logic. This theorem is similar to results of Schützenberger, McNaughton, and Papert, but the target logic is very strong: first-order logic can define data languages that cannot be accepted by orbit-finite data monoids.

Hence we introduce a natural restriction of first-order logic, called "rigidly-guarded first-order logic". In this logic the data equality predicate $x \sim y$ is only allowed to occur in conjunction with a formula $\phi(x, y)$ in which x determines y and y determines x in the following sense: for every word w and every position x (y respectively) in w, there is at most one position y (x respectively) such that w is a model of $\phi(x, y)$. *We show that rigidly guarded first-order logic defines precisely the languages that are accepted by aperiodic orbit finite data monoids.*

We then consider "rigidly-guarded monadic second-order logic" – the extension of rigidly-guarded first-order logic with monadic second-order quantification. *We show that this logic can define exactly the languages that are accepted by (possibly periodic) orbit finite data monoids. Finally, we show that a natural extension of rigidly-guarded monadic second-order logic can define exactly the languages that are accepted by non-deterministic finite memory automata.* It follows from our characterization results that each of these logics is decidable.

Part I

Tree Languages

Chapter 1

How Big Must Complete XML Query Languages Be?

1.1 Introduction

On finite words, the relationship between first-order logic and modal languages – ones with operators for navigating forward and backward in a word – is well-understood. A starting point is linear temporal logics (LTL). Formulas can be built up from atomic propositions (i.e. node labels) via Boolean operators and the forward modalities for navigating to the right. Indeed, it suffices to have only one modality, the "strong" variant of until (which we use in this work) [GPSS80]: $\varphi\, U\, \psi$ is true at a position x of a word w if there is a position $y > x$ on which ψ is true, and φ is true on all positions between x and y. An LTL formula is said to hold of a word if it holds at the initial position of the word. A refinement of Kamp's theorem [Kam68] shows that LTL is *first-order complete* over words – it can express every property of words that is definable in first-order logic, with unary predicates for the word labels and binary predicates for the ordering relation.

Is the full power of the until operator necessary? Etessami and Wilke [EW00] showed that it is. In particular they define a notion of "until-depth" by restricting the number of nestings of the U operator. The main result of [EW00] is that the subsets LTL(ud $\leq k$) formed by restricting the number of nested U operators to k form a strict hierarchy in expressiveness on words. More precisely, they use a variant of LTL that contains both past and future operators – their until hierarchy thus looks at the nesting of both the U operator and its backward analog S (for "Since"). The result for U only follows from their proof.

Let us turn to the situation for ordered trees. XPath [Wor99] is the W3C standard for querying XML documents; the navigational core of XPath is a query language on finite labeled ordered trees. XPath is a modal language, analogous to

temporal logic with only X (next) and F (eventually). Marx and de Rijke [MdR05] showed that this language is incomplete in a fundamental sense – there are properties expressible in first-order logic over the navigational predicates that XPath cannot express. This incompleteness manifests itself in other shortcomings of XPath. For instance, XPath is not closed under complement of path relations; indeed, Marx has shown [Mar05b] that such closure for an extension of XPath is equivalent to first-order completeness. The incompleteness of XPath can be seen as the analog of the result that an U operator is needed on trees.

How can one extend XPath to get a first-order complete language? One answer is given by Marx in [Mar05a], who defines a first-order complete query language *Conditional XPath* (CXPath). Roughly speaking, CXPath extends XPath by an "until operator" $\phi\, U\, \psi$: this operator holds at a node in an ordered tree iff there is a finite path from the node that satisfies ϕ up to the end of the path, at which point it must satisfy ψ. Marx considers four kinds of paths: leftward within the siblings of a given node, rightward within the siblings, upward through the ancestors, or downward through some chain of descendants.

CXPath may seem analogous to LTL in adding an U operator. But the analogy is misleading, since the U operator of CXPath is combined with selection of a path. Thus the first-order completeness is a surprising result. Still CXPath has a disadvantage that it involves both "forward navigation"(downward and to the right) and "backward navigation": this makes it less amenable to one-pass evaluation. The forward-only fragment of CXPath, which we denote by ForCXPath, is the variant of CXPath where the paths allowed in the until operator are downward and rightward, with an additional filter added to detect whether a child is the first among its siblings. Is ForCXPath complete for arbitrary ordered trees? It follows from results of [BJ07] that this language is first-order complete over finite ordered trees of any *fixed* depth. *In this paper, we determine that* ForCXPath *is* not *first-order complete over arbitrary ordered trees.*

Another way of extending results from words to trees is by separating out path quantification and node quantification into separate operators. A natural way to do this it to consider a variant of the temporal logic CTL* [Eme90]. CTL* contains *path formulas*, which hold with respect to a path within a tree, and also *state formulas*, which hold with respect to a node within a tree. Path formulas are built up from state formulas via the LTL operators, while state formulas are built up from atomic propositions via Boolean operators and path quantification. CTL* is not complete over trees, since it has no means of determining the number of children of a node with a certain property. But this ability to distinguish among children is known

to be the only obstacle to completeness. For example, if we look at infinite binary trees, where a child is labelled as either left or right child, Hafer and Thomas have shown that CTL* extended with predicates for left and right sibling is first-order complete [HT87]. Related results on completeness up to bisimulation equivalence can be found in [MR99].

To extend this to arbitrary ordered trees, we can distinguish between vertical and horizontal paths, and add a path quantifier for each. A simple variation of the argument of Hafer and Thomas (given in Section 1.5.2) shows that this language is first-order complete over ordered trees – this is noted in Barcelo and Libkin's work (Theorem 3.4 of [BL05]). Indeed, we will consider a variant $\mathrm{CTL}^*_{\triangleleft\triangleright}$ with downward paths, rightward paths, and leftward paths, where on the horizontal paths we allow only simple until operators, with no nesting– this language will turn out to be first-order complete. $\mathrm{CTL}^*_{\triangleleft\triangleright}$ has two disadvantages: the first is that it requires paths in both horizontal directions, the second is that it requires all of LTL as a sublanguage in the vertical dimension, unlike CXPath. Can we make due without one of these two restrictions?

We give negative answers to these questions. In our first result, we show that one cannot weaken the horizontal navigation to look only in one direction. Indeed one cannot make due with a language that has the U operator in one horizontal direction (in addition to F and X), but only the F and X operators in the other direction.

In our second result, we consider restricting the power in the vertical direction. For any fragment $\mathcal{F}$ of LTL, we let $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathcal{F})$ be the ordered tree language that restricts $\mathrm{CTL}^*_{\triangleleft\triangleright}$ as follows: node formulas are built up as before from label predicates and a last child test, while being closed under quantification of downward and rightward paths; path formulas are built up by substituting node formulas as propositions within formulas of $\mathcal{F}$. For example,the language ForCXPath is contained in $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathcal{F})$ where $\mathcal{F}$ contains only the formula $p\,U\,q$. Our question can now be formalized as: for which fragments $\mathcal{F}$ is $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathcal{F})$ first-order complete? We show that if $\mathcal{F}$ is any fragment of LTL with bounded 'until-depth', then $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathcal{F})$ is not first-order complete. In fact, our results show that languages $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathcal{F}_n)$ where $\mathcal{F}_n$ is the subset of LTL formulas of until-depth at most n, form a strict hierarchy. In particular it follows from our results that ForCXPath is not first-order complete. Furthermore, it shows that any forward-only first-order complete language must be fairly large.

All results in this chapter have been achieved in collaboration with Michael Benedikt and have been published in [LB09].

Organization The rest of this chapter is organized as follows. Section 1.2 formally defines the logics we consider. Section 1.3 formally states our incompleteness results and Section 1.4 compares our results to previous work. The rest of this chapter is concerned with the proof of our results. Section 1.5 proves our first main result, about incompleteness of languages that restrict the use of horizontal navigation. Section 1.6 gives the second result, concerning incompleteness of languages restricting vertical navigation. Section 1.7 gives conclusions.

1.2 Preliminaries: Logics for Words and Trees

We now provide some definitions that will be used in this chapter. We start by defining linear temporal logic (LTL), a logic for defining languages of finite or infinite words. Then we introduce logics for defining tree languages and briefly defines first-order logic. Finally, we compare the expressiveness of the logics from the preceding sections with first-order logic.

1.2.1 Temporal Logics for Words.

Linear Temporal Logic (LTL) over a set of propositions Σ has formulas built up from the grammar:

$$\varphi = a \mid \neg\varphi \mid \varphi \wedge \varphi \mid X\,\varphi \mid F\,\varphi \mid \varphi\,U\,\varphi$$

where $a \in \Sigma$.

The semantics of LTL is generally defined with respect to infinite words. We will give a variant for finite labelled words: that is, finite sequences $a_1 \dots a_n$ over a finite alphabet Σ. For a word $w = a_1 \dots a_n$, we set $|w| = n$. For $i \leq j \leq |w|$ we let $w[i] = a_i$ and we denote by $w[i,j]$ the infix $a_i \dots a_j$. The suffix $a_i \dots a_n$ of w is denoted by w^i. We note that a word $w = a_1 \dots a_n$ can also be considered a structure $(\mathrm{Dom}, \leq, \mathrm{lab}())$ where $\mathrm{Dom} = \{1, \dots, n\}$, $\leq$ is the usual linear order on Dom, and lab() is a labeling function defined by $\mathrm{lab}(i) = a_i$ for all $i \leq n$.

We now define the semantics of LTL:

$w \vDash a$ iff a is the label of $w[1]$.
$w \vDash X\,\varphi$ iff $|w| > 1$ and $w^2 \vDash \varphi$.
$w \vDash F\,\varphi$ iff there is a j with $1 < j \leq |w|$ and $w^j \vDash \varphi$.
$w \vDash \varphi\,U\,\psi$ iff there is a j with $1 < j \leq |w|$ and $w^j \vDash \psi$
and for all i, if $1 < i < j$ then $w^i \vDash \varphi$.

The semantics of LTL over infinite words is analogous.

Here we use the "strong variant" of U, in which $\varphi\, U\, \psi$ asserts the existence of a node satisfying ψ. It is known that the expressiveness of LTL would be unaffected if we had replaced this by the usual notion, which asserts only that if such a node exists, all the nodes to the right of the first satisfy φ [Eme90]. Using this variant will make our negative results stronger, but will not impact our positive results.

We define the *until-depth* $\mathrm{ud}(\varphi)$ of an LTL formula φ to be the nesting depth of the until-operator:

$$\begin{aligned}
&\mathrm{ud}(a) = 0\\
&\mathrm{ud}(\varphi \wedge \psi) = max\{\mathrm{ud}(\varphi), \mathrm{ud}(\psi)\}\\
&\mathrm{ud}(\neg\varphi) = \mathrm{ud}(F\,\varphi) = \mathrm{ud}(X\,\varphi) = \mathrm{ud}(\varphi)\\
&\mathrm{ud}(\varphi\, U\, \psi) = max\{\mathrm{ud}(\varphi), \mathrm{ud}(\psi)\} + 1
\end{aligned}$$

This is precisely the definition of [EW00], restricted to LTL formulas with only future operators. Note that there are infinitely many semantically different formulas of any fixed until-depth.

We define the next/eventually-depth ned of an LTL formula similarly:

$$\begin{aligned}
&\mathrm{ned}(a) = 0\\
&\mathrm{ned}(\varphi \wedge \psi) = \mathrm{ned}(\varphi\, U\, \psi) = max\{\mathrm{ned}(\varphi), \mathrm{ned}(\psi)\}\\
&\mathrm{ned}(\neg\varphi) = \mathrm{ned}(\varphi)\\
&\mathrm{ned}(F\,\varphi) = \mathrm{ned}(X\,\varphi) = \mathrm{ned}(\varphi) + 1
\end{aligned}$$

1.2.2 Logics for Trees

Let Σ be a finite set of labels. An *ordered tree over* Σ consists of an acyclic parent/child relation in which every node has in-degree at most one, and for which there is a unique node – called *root* – which has in-degree 0; a right-sibling relation which is the successor relation of a linear order on the children of any given node; and a labeling function assigning an element of Σ to each node. We refer to the usual derived notions on ordered trees, such as the ancestor, descendant, following-sibling, and preceding-sibling relations.

A *downward path* in an ordered tree is a set of nodes that is linearly-ordered by the child relation of the tree. A *downward fullpath* is a downward path that contains a leaf node. Similarly, a *rightward path* is a set of nodes linearly-ordered by the right-sibling relation of the tree, and a *rightward fullpath* is a rightward path that contains a node with no right sibling. A *leftward path* and *leftward fullpath* are defined analogously. Clearly a path can be considered a word. We will only consider both finite and infinite trees.

CTL* – to be defined in a minute – is a known tree logic for unordered trees. We define an extension $\mathrm{CTL}^*_{\triangleleft\triangleright}$ of CTL* that can also access the sibling order. For a set of labels Σ, and $k \geq 0$, we inductively define the sets P_k and N_k of $\mathrm{CTL}^*_{\triangleleft\triangleright}$ *path formulas* and *node formulas*.

- The symbols in Σ are the only formulas in N_0.
- Any LTL formula over propositions in N_k is in P_k.
- If $\varphi \in P_k$ then $\exists_\triangleleft \varphi$, $\exists_\triangledown \varphi$, and $\exists_\triangleright \varphi$ are in N_{k+1}.
- Both N_k and P_k are closed under Boolean operations.

$\mathrm{CTL}^*_{\triangleleft\triangleright}$ is the union over k of the formulas in N_k and P_k.

Given a tree T, a node x in T, and a paths π in T, we define the semantics of $\mathrm{CTL}^*_{\triangleleft\triangleright}$ is defined by induction:

- $T, x \vDash a$ iff x is labelled with a in T
- $T, x \vDash \exists_\triangleleft \varphi$ iff there is a leftward fullpath π starting at x such that $T, \pi \vDash \varphi$.
- $T, x \vDash \exists_\triangledown \varphi$ iff there is a downward fullpath π starting at x such that $T, \pi \vDash \varphi$.
- $T, x \vDash \exists_\triangleright \varphi$ iff there is a rightward fullpath π starting at x such that $T, \pi \vDash \varphi$.
- $T, \pi \vDash \varphi$, for $\varphi \in P_k$ iff $N_k(\pi) \vDash \varphi$, where $N_k(\pi)$ is the labelled linear order formed by labeling each node in π with the formulas of N_k that it satisfies in T.

For a node formula φ, we write $T \vDash \varphi$ iff $T, \mathrm{root}(T) \vDash \varphi$.

We can also consider the language $\mathrm{CTL}^*_{\triangleleft\triangle\triangleright}$ formed by allowing a quantifier $\exists_\triangle$; that is, extending the rules for node formulas to say that if $\varphi \in P_k$ then $\exists_\triangle \varphi$ in N_{k+1}. The semantics is analogous to that of $\mathrm{CTL}^*_{\triangleleft\triangleright}$, but the quantifier $\exists_\triangle$ quantifies over *upward fullpaths*.

The *down-closure* $\mathrm{cl}_\triangledown(\varphi)$ of a $\mathrm{CTL}^*_{\triangleleft\triangleright}$ formula is the set of LTL formulas that are directly nested within downward path quantification. That is

$$
\begin{aligned}
&\mathrm{cl}_\triangledown(a) = \varnothing \\
&\mathrm{cl}_\triangledown(\varphi \wedge \psi) = \mathrm{cl}_\triangledown(\varphi\, U\, \psi) = \mathrm{cl}_\triangledown(\varphi) \cup \mathrm{cl}_\triangledown(\psi) \\
&\mathrm{cl}_\triangledown(\neg\varphi) = \mathrm{cl}_\triangledown(F\,\varphi) = \mathrm{cl}_\triangledown(X\,\varphi) = \mathrm{cl}_\triangledown(\varphi) \\
&\mathrm{cl}_\triangledown(\exists_\triangleleft \varphi) = \mathrm{cl}_\triangledown(\exists_\triangleright \varphi) = \mathrm{cl}_\triangledown(\varphi) \\
&\mathrm{cl}_\triangledown(\exists_\triangledown \varphi) = \mathrm{cl}_\triangledown(\varphi) \cup \{\varphi\}
\end{aligned}
$$

The *left-closure* $\mathrm{cl}_\triangleleft(\varphi)$ and *right-closure* $\mathrm{cl}_\triangleright(\varphi)$ of φ are defined similar for leftward and rightward quantification. The *closure* $\mathrm{cl}(\varphi)$ of φ is the union of $\mathrm{cl}_\triangleleft(\varphi)$, $\mathrm{cl}_\triangledown(\varphi)$, and $\mathrm{cl}_\triangleright(\varphi)$.

The *next/eventually-depth* $\mathrm{ned}(\varphi)$ of a $\mathrm{CTL}^*_{\triangleleft\triangleright}$ formula φ is the maximal next/eventually-depth of the elements of the closure of φ. The *down-until-depth* $\mathrm{ud}_\triangledown(\varphi)$ of φ is the maximal until-depth of any element within the down-closure of φ. *Left-until-depth* $\mathrm{ud}_\triangleleft(\varphi)$ and *right-until-depth* $\mathrm{ud}_\triangleright(\varphi)$ are defined alike with respect to the left- and right-closure.

The *down-path-depth* $\mathrm{pd}_\triangledown(\varphi)$ is the nesting depth of $\exists_\triangledown$-quantifiers within φ. Formally

$$\begin{aligned}
&\mathrm{pd}_\triangledown(a) = \varnothing\\
&\mathrm{pd}_\triangledown(\varphi \wedge \psi) = \mathrm{pd}_\triangledown(\varphi\, U\, \psi) = \max\{\mathrm{pd}_\triangledown(\varphi), \mathrm{pd}_\triangledown(\psi)\}\\
&\mathrm{pd}_\triangledown(\neg\varphi) = \mathrm{pd}_\triangledown(F\,\varphi) = \mathrm{pd}_\triangledown(X\,\varphi) = \mathrm{pd}_\triangledown(\varphi)\\
&\mathrm{pd}_\triangledown(\exists_\triangleleft\varphi) = \mathrm{pd}_\triangledown(\exists_\triangleright\varphi) = \mathrm{pd}_\triangledown(\varphi)\\
&\mathrm{pd}_\triangledown(\exists_\triangledown\varphi) = \mathrm{pd}_\triangledown(\varphi) + 1
\end{aligned}$$

The *left-path-depth* $\mathrm{pd}_\triangleleft(\varphi)$ and the *right-path-depth* $\mathrm{pd}_\triangleright(\varphi)$ are defined in the obvious way.

For example the formula

$$\exists_\triangledown\,.\, a\, U\, (\exists_\triangleleft\,.\, b\, U\, c).$$

has next/eventually depth 0, right until depth 0, down until depth 1, until depth 1, and path depth 2.

Let $D = \{\mathrm{pd}_\triangleleft, \mathrm{pd}_\triangledown, \mathrm{pd}_\triangleright, \mathrm{ud}_\triangleleft, \mathrm{ud}_\triangledown, \mathrm{ud}_\triangleright, \mathrm{ned}\}$. For $d_1, \ldots, d_n \in D$ we denote by $\mathrm{CTL}^*_{\triangleleft\triangleright}(d_1 \le x_1, \ldots, d_n < x_n)$ the set of $\mathrm{CTL}^*_{\triangleleft\triangleright}$ formulas φ with $d_1(\varphi) \le x_1, \ldots, d_n(\varphi) \le x_n$. In addition we use the shorthand notation $\mathrm{CTL}^*_{\triangleleft\triangleright}(d_1 \le x_1, \ldots, d_n < x_n, \mathrm{other} \le o)$ to denote the set of $\mathrm{CTL}^*_{\triangleleft\triangleright}$ formulas φ with $d_1(\varphi) \le x_1, \ldots, d_n(\varphi) \le x_n$, and $d(\varphi) \le o$ for all $d \in D \setminus \{d_1, \ldots, d_n\}$. We denote by $\mathrm{CTL}^*_{\triangleleft\triangle\triangleright}(\mathrm{ud} \le 1)$ the set of $\mathrm{CTL}^*_{\triangleleft\triangle\triangleright}$ formulas with until-depth at most 1.

Let Σ be a label alphabet for ordered trees. We consider *first-order logic* over a signature having unary predicates $a(x)$ for every $a \in \Sigma$ as well as binary predicates for the parent/child relation, the immediate right-sibling relation, and the transitive closures of these predicates. The syntax and semantics of first-order logic is as usual [Lib04]. The *quantifier depth* of a first order formula φ is – as usual – the maximal nesting depth of quantifiers in φ.

1.2.3 The Expressive Power of CTL*

It has been shown by Hafer and Thomas that CTL* – CTL$^*_{\triangleleft\triangleright}$ without horizontal path quantification – is equivalent to first order logic over unordered binary trees [HT87]. Moller and Rabinovich show that CTL* is equivalent to the bisimulation invariant fragment of first order logic over unordered trees [MR99]. Theorem 3.4 in [BL05] extends this to ordered trees:

Theorem 1.1 (Barcelo and Libkin)**.** *CTL$^*_{\triangleleft\triangleright}$ is first-order complete. That is, for every first-order sentence φ there is an CTL$^*_{\triangleleft\triangleright}$ sentence ψ such that for all ordered trees T, $T \vDash \varphi$ iff $T \vDash \psi$.*

Theorem 3.4 of [BL05] actually states a much stronger claim, which our Theorem 1.3 contradicts. We thus give a brief sketch of the proof of Theorem 1.1. One approach is via the composition technique of Moller and Rabinovich [MR99]. Indeed, this extension is implicit in the work of Moller and Rabinovich and that of Hafer and Thomas.

Proof. Fix an alphabet Σ. It is known that for each k there are only finitely many distinct first-order formulas over Σ that have quantifier rank at most k. Let Σ_k be the label set consisting of a label for each set of equivalent first-order formulas with one free variable that have quantifier rank at most k. For a node x in a Σ-labeled tree T let subtree-type$_k(x)$ be the set of first-order formulas of quantifier rank at most k true at x. We extend the definition of subtree-type$_k$ to paths in the obvious way.

Given a vertical path π in a tree T and a non-leaf node x on π we define the following: leftchildren(π, x) is the leftward fullpath in T that starts at the unique child of x on π; rightchildren(π, x) is defined analogously for right siblings. For all $k, l \in \mathbb{N}$, type$_{k,l}(x)$ is the pair consisting of the set of first-order sentences of quantifier rank at most l that hold of subtree-type$_k$(leftchildren(π, x)) and the set of first-order sentences of quantifier rank at most l that hold of subtree-type$_k$(rightchildren(π, x)). We let $\Sigma_{k,l}$ be the alphabet with a label for each pair of sets of sentences of quantifier rank at most l in the vocabulary for labelled strings over Σ_k, and an additional label $\perp$. type$_{k,l}(\pi)$ is the $\Sigma_{k,l}$-labelled string expanding π by labeling each interior node x with type$_{k,l}(x)$, and the leaf node of π with a special label $\perp$. Then we have:

> Claim 1. *For every first-order sentence ϕ, there are k and l and a first-order sentence ψ over $\Sigma_{k,l}$-labelled strings such that: $T, x \vDash \phi$ iff* type$_{k,l}(\pi) \vDash \psi$*, where π is the path from the root of T to x.*

This is proved as in Theorem 3.2 of [MR99]. In the presence of an ordering the assumption of "wideness" used there is not needed. Hence the result then follows from:

CLAIM 2. *For every first-order sentence ϕ over $\Sigma_{k,l}$-labelled strings, there is an CTL$^*_{\triangleleft\triangleright}$ formula ψ such that* $\mathrm{type}_{k,l}(\pi) \vDash \phi$ *iff* $T, x \vDash \psi$

This is proved using Kamp's theorem, as in Lemma 4.2 of [MR99]. □

CTL$^*_{\triangleleft\triangleright}$ is a large language, since it contains all of LTL as a sublanguage. Clearly, we can eliminate syntactic features of LTL that do not add expressiveness: for example, the eventually operator $F\,\varphi$ and the next operator $X\,\varphi$ can both be defined using our form of the until operator [Eme90], and so both are unnecessary. What about the use of the until operator?

The following result follows directly from the work of Marx [Mar04, Mar05a]:

Theorem 1.2 (Marx)**.** *CTL$^*_{\triangleleft\triangle\triangleright}$(ud ≤ 1) is first-order complete*

Proof. The Conditional XPath language of [Mar04, Mar05a] has been proved first-order complete in [Mar05a]. In [LS08] it was shown that there is an easy translation from Conditional XPath filters to the language $\mathcal{X}_{\mathrm{until}}$ defined in [Mar04]. $\mathcal{X}_{\mathrm{until}}$ has quantification over partial paths in the downward, upward, leftward, and rightward paths followed immediately by an until operator (this is analogous to the temporal logic CTL, which also restricts the sequencing of path quantifiers and LTL operators). As $\mathcal{X}_{\mathrm{until}}$ is a subset of CTL$^*_{\triangleleft\triangle\triangleright}$(ud = 1), both are first-order complete. □

We thus have two ways of getting first-order completeness: Theorem 1.1 allows arbitrary LTL, downward paths and both horizontal paths, while Theorem 1.2 restricts the use of the Until operator but also allows upward paths. Can one make use of only one horizontal operator? Can one restrict the nesting of Until operators without introducing upward axes? Our main results give negative answers to both these questions.

1.3 Main Results

We show two incompleteness results. First, CTL$^*_{\triangleleft\triangleright}$ becomes incomplete if the number of $\exists_\triangleright$ quantifiers is restricted to some fixed p and the number of nested Us within $\exists_\triangleright$ quantifiers is restricted to some fixed u. Observe that despite this restriction, there still might be arbitrary LTL formulas on the leftward and downward paths, and arbitrarily many X and Fs on rightward paths.

Theorem 1.3 (Horizontal Incompleteness)**.** *For all $p, u \in \mathbb{N}$, $CTL^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleright \leq p, \mathrm{ud}_\triangleright \leq u)$ is not first-order complete on finite ordered trees. In addition, if $up > u'p'$ then $CTL^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleright \leq p, \mathrm{ud}_\triangleright \leq u)$ is more expressive than $CTL^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleright \leq p', \mathrm{ud}_\triangleright \leq u')$. The symmetric statements hold for leftward axes.*

We note that the if we restrict only $\mathrm{ud}_\triangleright$ but not $\mathrm{pd}_\triangleright$ in $\mathrm{CTL}^*_{\triangleleft\triangleright}$, then the resulting logic remains first order complete. This is because an arbitrary nesting of until operators can be mimicked by using several nested path quantifiers. In fact, it follows from our proof that already $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{ud}_\triangleright = 0)$ is not first-order complete. This contradicts a statement made in [BL05]. However, it follows from [BL05] that $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{ud}_\triangleright \leq 1)$ is already first order complete.

Our second incompleteness result is concerned with downward axes. It states $\mathrm{CTL}^*_{\triangleleft\triangleright}$ is not first order complete if the nesting depth of U on downward paths is restricted to u – even with an arbitrary nesting depth of all other modifiers.

Theorem 1.4 (Vertical Incompleteness)**.** *On finite, ordered trees, $CTL^*_{\triangleleft\triangleright}(\mathrm{ud}_\triangledown \leq u)$ is not first-order complete for all $u \in \mathbb{N}$. In addition, the family $CTL^*_{\triangleleft\triangleright}(\mathrm{ud}_\triangledown \leq u)$ forms a strict hierarchy in expressiveness.*

We will show that this incompleteness result holds even over binary trees.

1.4 Related Work

The question of a complete first-order forward-only language was considered in the context of *unordered trees* by Rabinovich and Maoz [RM00]. They consider languages of the form $\mathrm{CTL}^*(\mathcal{F})$ where $\mathcal{F}$ is a fragment of first-order logic on ω-words. The main result of [RM00] is that $\mathrm{CTL}^*(\mathrm{QR}_n)$ is incomplete on finite and infinite trees, where QR_n is the set of first-order formulas of quantifier rank at most n. It follows from our results that none of the languages $\mathrm{CTL}^*(\mathrm{UD}_k)$ are complete on unordered trees, where UD_k is the set of formulas with until-depth at most k. This in turn implies the incompleteness theorem of [RM00], since the sets of formulas QR_n they consider are finite for any fixed vocabulary, and hence each is contained in some UD_k. However, Rabinovich has also shown [Rab08] that formulas of Until-depth k have bounded alternation-depth. Combining this with the result on incompleteness of bounded alternation-depth claimed in [Rab02] implies the restriction of our result to unordered trees. In [Rab02] an extension is announced, stating that $\mathrm{CTL}^*(\mathrm{AD}_n)$ is incomplete, where AD_n is the set of formulas of bounded quantifier-alternation depth.

In [Boj08], Bojańczyk defines a hierarchy of logics using a general notion of a "tree operator" – a tree automaton with "holes" for lower level formula. The result announced in [Boj08] is that no logic based on a finite set of such operators can be first-order complete on unordered trees. When restricted to unordered trees, our results are incomparable to those announced by Bojańczyk in [Boj08], since each of our sets $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{UD}_{\leq i})$ contains formulas of unbounded Operator Depth.

We note that the case of ordered trees does present new difficulties over the unordered case. Briefly put, our separation theorems, as well as the earlier ones, rely on the construction of families of pairs of trees that are "highly indistinguishable" in the syntactically smaller language, but which disagree on a fixed sentence within the syntactically larger language. In the ordered case both the construction of these pairs and the proofs of their equivalence are more difficult, since many properties of the depth of the trees can be distinguished by small formulas. An explanation of the particular difficulties arising in our second result is given in Section 1.6.

There has also been work in the ordered case. Barcelo and Libkin consider several logics on ordered trees; in the first-order context, the main result is that a variant of $CTL^\star$ with quantifiers for vertical and horizontal paths is first-order complete [BL05].

Our notion of "until-depth" is taken from Etessami and Wilke's work [EW00]. They deal with infinite words, and use a variant of LTL that contains both past and future operators. They define a hierarchy within this based on the number of nestings of the operators U and its backward analog S (for "Since"). The main result of [EW00] is that the subsets UD_k formed by restricting the number of nested U or S operators to k, form a strict hierarchy in expressiveness on words. This does not imply the corresponding result for ordered trees, since path quantification coupled with LTL operators could add more expressiveness. For example, when one restricts to trees that consist of exactly one path, $\mathrm{CTL}^*_{\triangleleft\triangleright}$ restricted to formulas with until-depth 1 is first-order complete: arbitrary nesting of untils can be mimicked by putting each until in a path quantifier, since the until-depth only counts nesting within a given path quantification. Consider also the analogous situation when LTL is extended with "past operators" S and P, which are the duals of U and F (see [Eme90] for the precise definition): Etessami and Wilke [EW00] have shown that the subsets of LTL extended with past operators formed by restricting the nesting of both until and its dual S form a strict hierarchy. But the theorem of Marx mentioned above implies that $\mathrm{CTL}^*_{\triangleleft\triangleright}$ based on LTL-with-past but restricted to since/until-depth one is already sufficient for first-order completeness.

1.5 The Horizontal Until Hierarchy

In this section we show our first incompleteness result.

Theorem 1.3 (Horizontal Incompleteness)**.** *For all $p, u \in \mathbb{N}$, $CTL^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleright \leq p, \mathrm{ud}_\triangleright \leq u)$ is not first-order complete on finite ordered trees. In addition, if $up > u'p'$ then $CTL^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleright \leq p, \mathrm{ud}_\triangleright \leq u)$ is more expressive than $CTL^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleright \leq p', \mathrm{ud}_\triangleright \leq u')$. The symmetric statements hold for leftward axes.*

To prove this theorem we will define for each $i \in \mathbb{N}$ a tree language Q_i such that, for all $p_\triangleright, u_\triangleright \in \mathbb{N}$, $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleright \leq p_\triangleright, \mathrm{ud}_\triangleright \leq u_\triangleright)$ can express $Q_{p_\triangleright u_\triangleright}$ but not $Q_{p_\triangleright u_\triangleright+1}$. We can show that $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleright \leq p_\triangleright, \mathrm{ud}_\triangleright \leq u_\triangleright)$ can express $Q_{p_\triangleright u_\triangleright}$ by simply defining a formula that expresses $Q_{p_\triangleright u_\triangleright}$. To show that $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleright \leq p_\triangleright, \mathrm{ud}_\triangleright \leq u_\triangleright)$ cannot express $Q_{p_\triangleright u_\triangleright+1}$ we use the method of Ehrenfeucht and Fraïsé: We define a game that is played on two trees $\tilde{T}$ and $\tilde{S}$ by two players called *Spoiler* and *Duplicator*. This game will corresponds to a fragment $\mathcal{L}$ of $\mathrm{CTL}^*_{\triangleleft\triangleright}$ in the following sense: Duplicator has a winning strategy in this game iff $\tilde{T}$ and $\tilde{S}$ cannot be distinguished by any formula in $\mathcal{L}$. Hence we can show that $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleright \leq p_\triangleright, \mathrm{ud}_\triangleright \leq u_\triangleright)$ cannot express $Q_{p_\triangleright u_\triangleright+1}$, by showing that there are two trees $\tilde{T}$ and $\tilde{S}$ such that: (a) $\tilde{T}$ satisfies $Q_{p_\triangleright u_\triangleright+1}$ but $\tilde{S}$ does not; and (b) Duplicator has a winning strategy for a suitable game on $\tilde{T}, \tilde{S}$.

Before we can prove Theorem 1.3 we need to introduce some prior machinery. In Subsection 1.5.1 we define an Ehrenfeucht-Fraïssé-style game and we show that it corresponds to $\mathrm{CTL}^*_{\triangleleft\triangleright}$. Our proof of Theorem 1.3 (and the one of Theorem 1.4) makes use of a hierarchy theorem on words by Etessami and Wilke [EW00]. As we reuse elements of their proof, we reprove Etessami and Wilke's result in Subsection 1.5.2. Subsection 1.5.3 contains the proof of Theorem 1.3.

1.5.1 Ehrenfeucht-Fraïssé Games

We now describe a game on finite labelled trees, that corresponds to $\mathrm{CTL}^*_{\triangleleft\triangleright}$. In the same way as LTL is embedded into CTL, there is a game on words that is embedded into the game that corresponds to $\mathrm{CTL}^*_{\triangleleft\triangleright}$. This game was introduced by Etessami and Wilke in [EW00] where they showed that it corresponds to LTL. Before we define the game on trees, we define this game on words.

Games on Words

The LTL(ned $\leq e$, ud $\leq u$)-game (or LTL(e,u)-game for short) is played by two players, a male *Spoiler* and a female *Duplicator*, on two words w, v. The goal of Spoiler is to show that the words are different while Duplicator tries to show that they are similar. At each stage of the game a pair w^i, v^j of suffixes of w, v is *selected*. There are different kinds of moves – X-, F-, and U-moves – in which the players alter the selected words.

We describe the play of the LTL(e,u)-game with selected words w, v. If $e = 0$ and $u = 0$, Duplicator wins the game if the roots of the selected words have same label and Spoiler wins otherwise. In this case we call w, v the *final position* of the game.

If $e > 0$ Spoiler can chooses to play either an X-, or an F-move.

> *X-move.* In an *X-move* neither player has any choice: the new selected words are w^2, v^2. Spoiler can only choose this move if one of the words has such a suffix. If one of w, v has such a suffix and the other does not, Duplicator cannot move and Spoiler wins the game. Spoiler also wins the game if the roots of the new selected words have different labels.

> *F-move.* In an *F-move* Spoiler picks one of the two words, say w. He then selects a proper suffix w^i of w with $i > 1$. Again, Spoiler can only choose this move if such a suffix exists. Duplicator selects a suffix v^j of v with $j > 1$. Duplicator loses the game if she cannot respond with a suffix such that the roots of w^i and v^j have the same label.

If Duplicator did not lose the game in the preceding round, the players play the LTL$(e-1,u)$-game on the newly selected words. The winner of this subgame will be declared winner of the LTL(e,u)-game with w, v.

If $u > 0$ then Spoiler can also consider to play an U-move.

> *U-move.* In an *U-move* Spoiler picks one of the words and we assume he picks w. An U-move consists of two *half moves*. In the first half move Spoiler selects a proper suffix w^i of w for $i > 1$. The Duplicator has to reply with a proper suffix v^j of v for some $j > 1$ such that the roots of w^i and v^j have the same label. Again, Spoiler can only choose this move if w^i exists and Duplicator looses the game if the roots of w^i and v^j have different labels.

In the second half move Spoiler selects $v^{j'}$ for some $1 < j' \leq j$. Duplicator has to select $w^{i'}$ such that $1 < i' \leq i$ and such that the roots of $w^{i'}$ and $v^{j'}$ have the same label. In the case that the Spoiler picks $v^{j'}$ such that $j' = j$, Duplicator must choose i' to be equal to i. Again, Duplicator looses the game if she cannot select such a word.

If Duplicator survives both half rounds, the players continue playing the $\mathrm{LTL}(e, u-1)$-game with $w^{i'}, v^{j'}$. The winner of this game is the winner of the $\mathrm{LTL}(e, u)$-game on w, v.

Etessami and Wilke have shown the following [EW00]:

Proposition 1.1 (Etessami, Wilke)**.** *Duplicator has a winning strategy in the $LTL(e, u)$-game on words w, v iff w, v satisfy the same $LTL(\mathrm{ned} \leq e, \mathrm{ud} \leq u)$ formulas.*

Games on Trees

For given $p_\triangleleft, p_\triangledown, p_\triangleright, u_\triangleleft, u_\triangledown, u_\triangleright, e \in \mathbb{N}$ we will abbreviate

$$\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleleft \leq p_\triangleleft, \mathrm{pd}_\triangledown \leq p_\triangledown, \mathrm{pd}_\triangleright \leq p_\triangleright, \mathrm{ud}_\triangleleft \leq u_\triangleleft, \mathrm{ud}_\triangledown \leq u_\triangledown, \mathrm{ud}_\triangleright \leq u_\triangleright, \mathrm{ned} \leq e)$$

by $\mathrm{CTL}^*_{\triangleleft\triangleright}(p_\triangleleft, p_\triangledown, p_\triangleright, u_\triangleleft, u_\triangledown, u_\triangleright, e)$ below. The $\mathrm{CTL}^*_{\triangleleft\triangleright}(p_\triangleleft, p_\triangledown, p_\triangleright, u_\triangleleft, u_\triangledown, u_\triangleright, e)$-game is played by Spoiler and Duplicator on a pair of trees T, S. Again there are different kinds of moves – $\exists_\triangleleft$-, $\exists_\triangleright$-, and $\exists_\triangledown$-moves – in which the players alter a pair of *selected nodes* x, y. Initially, the roots of T, S are selected. We describe the play of the game with x, y selected. Duplicator wins the $\mathrm{CTL}^*_{\triangleleft\triangleright}(p_\triangleleft, p_\triangledown, p_\triangleright, u_\triangleleft, u_\triangledown, u_\triangleright, e)$-game on T, S if $p_\triangleleft + p_\triangledown + p_\triangleright = 0$ and if x and y have the same label. Otherwise Spoiler can choose one of the following moves:

$\exists_\triangleleft$-move. Spoiler can choose an *$\exists_\triangleleft$-move* if $p'_\triangleleft > 0$. He picks one of the two trees T, S, say T. Spoiler then picks the leftward fullpath π that is rooted at x. Duplicator responds by picking the leftward fullpath τ that is rooted at y. Then Spoiler and Duplicator play the $\mathrm{LTL}(e, u_\triangleleft)$-game on π, τ, where these paths are considered as words. If Spoiler wins this word-game then he wins the tree-game. Otherwise Spoiler chooses the roots x', y' of some intermediate position of the LTL-game. Then the players play the $\mathrm{CTL}^*_{\triangleleft\triangleright}(p_\triangleleft - 1, p_\triangledown, p_\triangleright, u_\triangleleft, u_\triangledown, u_\triangleright, e)$-game with on T, S with x', y' selected. The winner of this subgame is the winner of the $\mathrm{CTL}^*_{\triangleleft\triangleright}(p_\triangleleft, p_\triangledown, p_\triangleright, u_\triangleleft, u_\triangledown, u_\triangleright, e)$-game with x, y selected.

$\exists_\triangledown$-*move.* An $\exists_\triangledown$-*move* can be chosen by Spoiler if $p'_\triangledown > 0$. It is played like an $\exists_\triangleleft$-move, just that the players pick some downward fullpaths π, τ instead of the leftward fullpaths and the players play the $\mathrm{LTL}(e, u_\triangledown)$-game on π, τ instead of the $\mathrm{LTL}(e, u_\triangleleft)$-game. Again, Spoiler wins the tree game if he wins the game in π, τ. Otherwise the players proceed to play the $\mathrm{CTL}^*_{\triangleleft\triangleright}(p_\triangleleft, p_\triangledown - 1, p_\triangleright, u_\triangleleft, u_\triangledown, u_\triangleright, e)$-game with the roots of some intermediate position of the path game selected. As before, the winner of this game determines the winner of the $\mathrm{CTL}^*_{\triangleleft\triangleright}(p_\triangleleft, p_\triangledown, p_\triangleright, u_\triangleleft, u_\triangledown, u_\triangleright, e)$-game.

$\exists_\triangleright$-*move.* Spoiler may play an $\exists_\triangleright$-*move* if $p'_\triangleright > 0$. The rules are as above, just with rightward-paths on which the players play the $\mathrm{LTL}(e, u_\triangleright)$-game. Again, Spoiler can win the tree game by winning the game on π, τ. Otherwise the game proceeds with the $\mathrm{CTL}^*_{\triangleleft\triangleright}(p_\triangleleft, p_\triangledown, p_\triangleright - 1, u_\triangleleft, u_\triangledown, u_\triangleright, e)$-game on an intermediate position of the word game.

A *winning strategy* for either player from a given initial position and set of moves is defined as usual. The following lemma shows the correspondence between the tree game and the logic $\mathrm{CTL}^*_{\triangleleft\triangleright}$.

Proposition 1.2. *Let T, S be finite trees. Duplicator has a winning strategy for the $CTL^*_{\triangleleft\triangleright}(p_\triangleleft, p_\triangledown, p_\triangleright, u_\triangleleft, u_\triangledown, u_\triangleright, e)$-game on T, S iff the trees T and S agree on all $CTL^*_{\triangleleft\triangleright}(p_\triangleleft, p_\triangledown, p_\triangleright, u_\triangleleft, u_\triangledown, u_\triangleright, e)$ node formulas.*

Proof. (sketch) We show only the 'if' direction, the other direction is similarly straightforward. Given the hypothesis, we show that Duplicator's winning strategy has the property that if the position after a move is x, y at a stage with $(p'_\triangleleft, p'_\triangledown, p'_\triangleright)$ moves to play, then (T, x) agrees with (S, y) on $\mathrm{CTL}^*_{\triangleleft\triangleright}(p'_\triangleleft, p'_\triangledown, p'_\triangleright, u_\triangleleft, u_\triangledown, u_\triangleright, e)$ node formulas. We show this by induction on $p'_\triangleleft + p'_\triangledown + p'_\triangleright$. The base case of no moves remaining is clear.

One inductive case is when Spoiler plays an $\exists_\triangledown$-move π By induction, (T, x) satisfies the same formulas of $\mathrm{CTL}^*_{\triangleleft\triangleright}(p_\triangleleft, p_\triangledown, p_\triangleright, u_\triangleleft, u_\triangledown, u_\triangleright, e)$ as (S, y), and hence there is a path τ rooted at y such that (T, π) and (S, τ) satisfy the same path formulas of $\mathrm{CTL}^*_{\triangleleft\triangleright}(p_\triangleleft, p_\triangledown - 1, p_\triangleright, u_\triangleleft, u_\triangledown, u_\triangleright, e)$. Duplicator responds with such a path. Consider LTL over the alphabet with propositions form $\mathrm{CTL}^*_{\triangleleft\triangleright}(p_\triangleleft, p_\triangledown - 1, p_\triangleright, u_\triangleleft, u_\triangledown, u_\triangleright, e)$ node formulas, and consider the expansion of π and τ where nodes are decorated by the formulas in the above language that they satisfy. The hypothesis on π and τ and Proposition 1.1 guarantee that there is a winning strategy for Duplicator in the

$LTL(e, u_{\triangledown})$-game over this alphabet. Duplicator uses this strategy in the remainder of the move.

The case of leftward and rightward paths is done similarly. □

1.5.2 The Until Hierarchy on Words

We will now review a winning strategy of Duplicator for a game on words. This strategy has been used by Etessami and Wilke to show that $\mathrm{LTL}(\mathrm{ud} \le u)$ is not first-order complete over finite words for all $u \in \mathbb{N}$. As we will often reuse parts of their proof in our proofs of Theorems 1.3 and 1.4, we reprove the result of Etessami and Wilke here.

Theorem 1.5 ([EW00])**.** *$LTL(\mathrm{ud} \le u)$ is not first-order complete over finite words for every $u \ge 0$. In addition, for each u, $LTL(\mathrm{ud} \le u + 1)$ is more expressive than $LTL(\mathrm{ud} \le u)$.*

The Separating Property

To show Theorem 1.5, we define for each $u \in \mathbb{N}$ a property S_u such that S_{u+1} that can be expressed in $\mathrm{LTL}(\mathrm{ud} \le u + 1)$ but not in $\mathrm{LTL}(\mathrm{ud} \le u)$ for all $u \in \mathbb{N}$.

Definition 1.3. *Let S_u denote the set of words that contain a prefix satisfying the regular expression*

$$a(c^* a)^u.$$

The property S_u can be encoded by the $\mathrm{LTL}(\mathrm{ud} = u)$ formula ψ_u defined recursively as follows:

$$\psi_u = \begin{cases} a & \text{if } u = 0 \\ a \wedge (c \,U\, \psi_{u-1}) & \text{if } u > 0 \end{cases}$$

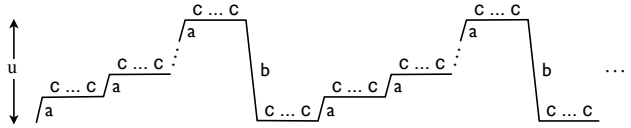
To complete the proof of Theorem 1.5, we now show that S_{u+1} cannot be expressed in $\mathrm{LTL}(\mathrm{ud} \le u)$, even for arbitrary nesting depth of X and F modifiers. This follows from Lemma 1.4 below together with Proposition 1.2. We first show the lemma for infinite words, because it simplifies the presentation. The case for finite words will be a simple consequence (Corollary 1.5).

Lemma 1.4 (Etessami, Wilke)**.** *For all $u, e > 0$ there are two infinite words $\tilde{v}$ and $\tilde{w}$ such that (i) $\tilde{v}$ satisfies S_{u+1} but $\tilde{w}$ does not, and (ii) Duplicator has a winning strategy for the $LTL(\mathrm{ud} \le u, \mathrm{ned} \le e)$-game on $\tilde{v}, \tilde{w}$.*

We show the lemma in the rest of this section. We fix some given $\tilde{u}, \tilde{e} \in \mathbb{N}$ and define the infinite words

$$\tilde{w} = a\big((c^{\tilde{e}}a)^{\tilde{u}}c^{\tilde{e}}b\big)^{\omega}$$
$$\tilde{v} = ac^{\tilde{e}}\,\tilde{w}$$

The word $\tilde{w}$ can be visualized as the following "staircase".



It is obvious that $\tilde{v}$ satisfies S_{u+1} but $\tilde{w}$ does not. Hence part (i) of Lemma 1.4 holds. To prove part (ii) we show that Duplicator has a winning strategy for the LTL(ud $\leq u$, ned $\leq e$)-game on $\tilde{v}$ and $\tilde{w}$. The idea of the proof will be to show that Spoiler can force the selected positions up only one step of the staircase on each U-move. As the number of steps depends on the number of U-moves, Spoiler cannot detect that the selected positions are on different steps of the staircase in the beginning of the game.

Duplicator's Strategy

We first need some notations. Given two positions x, y of the same word we denote by (x, y) the sequence of positions between x and y, excluding x and y. We denote by $\text{right}_a(x)$ the next position to the right of x that is labelled a. $\text{right}_{ab}(x)$ is the next a or b labelled position to the right of x. Intuitively, the *plateau-depth* of a position x is the distance of x to the end of the its "plateau" to the right. Formally, plateau-depth(x) is the number of c-labelled positions in $(x, \text{right}_{ab}(x))$. The *top-depth* of x is the number of steps between x and the top of the staircase, that is top-depth(x) is the number of a positions in $(x, \text{right}_b(x))$. The top-depth of a word is the top-depth of its root and similarly for plateau-depth. For example the top-depth of $\tilde{w}$ is u and its plateau-depth is e. Note also that

$$\text{plateau-depth}(w) = n \qquad \text{if } w \in \Sigma \cdot c^n \cdot \Sigma \setminus \{c\} \cdot \Sigma^{\omega}$$
$$\text{top-depth}(w) = n \qquad \text{if } w \in \Sigma \cdot (c^*a)^n c^* b \cdot \Sigma^{\omega}$$

The following claim implies part (ii) of Lemma 1.4 because $\tilde{w}, \tilde{v}$ satisfy the conditions of Claim 1 if the whole paths are selected.

Claim 1. *Let $e \leq \tilde{e}$ and $u \leq \tilde{u}$. Duplicator can win the LTL(ud $\leq u$, ned $\leq e$)-game on suffixes w, v of the words $\tilde{w}, \tilde{v}$, if*

1. *the roots of w and v have the same label.*
2. *w and v have the same plateau-depth.*
3. $|\text{top-depth}(w) - \text{top-depth}(v)| \leq 1$.
4. *If w and v have different top-depths then*
   (a) $\text{top-depth}(w) \geq u$ *and* $\text{top-depth}(v) \geq u$ *and*
   (b) *if* $\text{top-depth}(w) = u$ *or* $\text{top-depth}(v) = u$
   then $\text{plateau-depth}(w) \geq e$ *(and hence* $\text{plateau-depth}(v) > e$*).*

Proof of Claim 1. The proof is by induction on $u+e$. The base case $u+e = 0$ holds because by Condition 1, the roots of v, w have the same label. In the induction step, we assume that the claim holds for some numbers e, u with $u + e > 0$. We distinguish three cases, according to the kind of move that Spoiler chooses first.

X-moves Spoiler can choose an X-move only if $e > 0$. In this case Duplicator's strategy is determined by the rules of the game: the selected suffixes after the round are v^2, w^2.

We prove that Duplicator can win the $\text{LTL}(\text{ud} \leq u, \text{ned} \leq e-1)$-game on v^2, w^2. This is done by showing that the conditions of Claim 1 hold for u and $e-1$. This implies that the roots of the newly selected suffixes have the same label, and hence the claim follows by induction. The only interesting case is when $|\text{top-depth}(w) - \text{top-depth}(v)| = 1$ and $\text{top-depth}(w) = u$ (and hence $\text{top-depth}(v) = u+1$). In this case $\text{plateau-depth}(w) \geq e$ by Condition 4b. As by definition we have that $\text{plateau-depth}(w^2) \geq \text{plateau-depth}(w)-1$ it follows that the plateau depth of w^2 is greater or equal to $e-1$. A similar argument shows that $\text{plateau-depth}(v^2) \geq e-1$. As $\text{plateau-depth}(w) \geq e > 0$ it also follows that the top-depth of neither word has decreased, that is $\text{top-depth}(w^2) = \text{top-depth}(w) = u$ and $\text{top-depth}(w^2) = \text{top-depth}(w^2) = u$. Hence Condition 4 holds for w^2, v^2. Condition 2 is obvious.

F-moves The strategy on F-moves is very easy for Duplicator. It relies on the observation that as v and w are infinite and satisfy the invariant, they have the same set of proper suffixes. Therefore, given Spoiler's selection, Duplicator can select the same suffix in the other word.

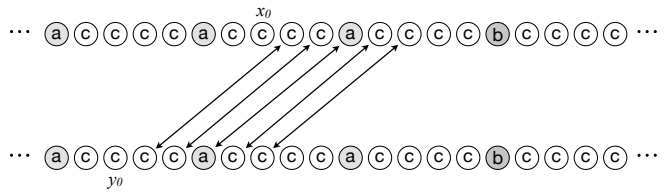


Figure 1.1: Duplicator's strategy if the current position of the game is (x_0, y_0) and Spoiler skips at most $\tilde{e}+1$ positions. If Spoiler picks a position z in either w or v in his first half move, then Duplicator picks the position z' in the other word, such that there is a arrow from z to z'.

U-moves On U-moves, Duplicator cannot use the same strategy as for F-moves in her first half move: if she did, Spoiler might play on the word with the smaller top-depth and just move the selected suffix by one position. Then Duplicator might skip $\tilde{e}+1$ positions to the next isomorphic suffix. In the second half move, Spoiler can pick from $\tilde{e}+2$ positions (the positions that Duplicator skipped and the one she selected), but Duplicator can only choose the position that Spoiler selected. Hence, when Spoiler skips only few positions, Duplicator will skip the same number of positions. Only when Spoiler skips sufficiently many positions will Duplicator try to find an isomorphic suffix.

We now assume that Spoiler chooses to play on w and that he selects a suffix w^n of w in his first half move. If Spoiler skips at most $\tilde{e}+1$ positions (that is $n \leq \tilde{e}+1$) then Duplicator skips the same number of positions as Spoiler did, that is, she picks v^n. Otherwise Duplicator picks the largest proper suffix v^m of v that is isomorphic to w^n (such a suffix exists due to Condition 4). The case where Spoiler plays on v is symmetric. See Figures 1.1 and 1.2 for a visualization of Duplicator's strategy.

We verify that the suffixes w', v' chosen after the first half move satisfy the conditions of Claim 1 for $e, u-1$. This implies that their roots have the same label and thus the claim follows by induction. If Duplicator has chosen a suffix that is isomorphic to Spoiler's choice, then it is clear that the conditions of Claim 1 hold. In the other case both positions have been advanced by $n \leq \tilde{e}+1$ positions. The interesting case is when $|\text{top-depth}(w) - \text{top-depth}(v)| = 1$ and $\text{top-depth}(w) = u$ (and hence $\text{top-depth}(v) = u+1$). Clearly $\text{plateau-depth}(w^n) = \text{plateau-depth}(v^n)$ and hence Condition 2 holds for w^n, v^n. To verify Condition 4 first note that $|\text{top-depth}(w^n) - \text{top-depth}(v^n)| = 1$. There are two cases:

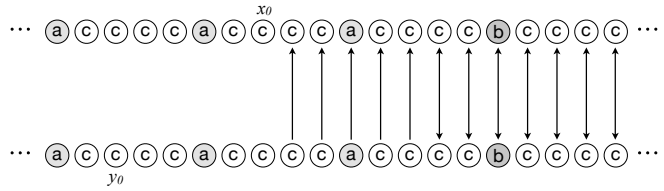


Figure 1.2: Duplicator's strategy if Spoiler plays on the lower word (starting from position y_0) and skips more than $\tilde{e}+1$ positions. The meanings of the arrows are as in Figure 1.1.

- If the positions have not been advanced beyond the next b (formally $n \leq \text{plateau-depth}(w)+1$) then $\text{top-depth}(w^n) = \text{top-depth}(w) = u > u-1$ and similarly $\text{top-depth}(v') > u-1$.
- In the other case $\text{top-depth}(w^n) = u-1$ and $\text{top-depth}(v^n) = u-1$, but as $n \leq e+1$, the plateau-depth of both suffixes cannot have increased. Hence $\text{plateau-depth}(w^n) \geq \text{plateau-depth}(w) > e$ and similarly $\text{plateau-depth}(v^n) > e$.

Assume that the suffixes w^n, v^m have been selected after the first round. In the second half move Spoiler chooses a suffix $v^{m'}$ such that $1 < m' \leq m$. It is easy to check that Duplicator can always choose $w^{n'}$ such that $n-n' = m-m'$. Observe that if $w^{n'}$ and $v^{m'}$ are not isomorphic, then $n', m' \leq e+1$. Hence the conditions of Claim 1 are maintained. □

How must the above construction be altered to show the result for finite words? Note that Duplicator only exploits that the words are infinite on her strategy for F-moves. Hence, if she only jumps to the *next* $(ac^e)^u bc^e$ section on each F-move, then she can win the game provided there are at least $e+1$ such sections. Thus we have:

Corollary 1.5 (Etessami, Wilke)**.** *For all $u, e \in \mathbb{N}$, Duplicator has a winning strategy for the LTL$(\text{ud} \leq u, \text{ned} \leq e)$-game on the finite words*

$$\tilde{w}_{fin} := \left((ac^e)^u bc^e\right)^{e+1}$$
$$\tilde{v}_{fin} := ac^e\, \tilde{w}_{fin}.$$

We will reuse Duplicator's winning strategy that we described in the proof from Claim 1 in Lemma 1.4 in the proofs of Theorems 1.3 and 1.4. There, we will refer to it as the *Etessami-Wilke strategy*. The game on trees will be designed in such a way, that at some point of this game, the players will select paths that are isomorphic to

the words $\tilde{w}_{\text{fin}}$ and $\tilde{v}_{\text{fin}}$. Hence we will be able to use Duplicator's strategy of Claim 1 in Lemma 1.4 to show that Duplicator can win the word-game on these paths.

1.5.3 Proof of Theorem 1.3

We now turn to the proof of Theorem 1.3, that is our horizontal incompleteness result on trees. We start by defining a property Q_i of trees for all $i \in \mathbb{N}$. Later we will show that $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleright \leq p_\triangleright, \mathrm{ud}_\triangleright \leq u_\triangleright)$ can define $Q_{p_\triangleright u_\triangleright}$ but not $Q_{p_\triangleright u_\triangleright+1}$ for all $p_\triangleright, u_\triangleright \in \mathbb{N}$.

The Separating Property

The *right path* of a node x in a tree is the sequence of its right siblings.

Definition 1.6 (Separating Property)**.** *Let Q_i be the set of ordered trees that have a downward fullpath π ending at a leaf labelled d, such that each node on π apart from the root and the leaf has a right path with a prefix satisfying*

$$a(c^*a)^i.$$

If $p_\triangleright \cdot u_\triangleright \geq i$, then Q_i can be expressed in $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleright \leq p_\triangleright, \mathrm{ud}_\triangleright \leq u_\triangleright)$ by the sentence 'there is a downward fullpath ending on d on which every node satisfies μ_i' where μ_i is the formula 'there is a rightward path satisfying $a(c^*a)^i\,\Sigma^*$'. Formally

$$\mu_i = \begin{cases} a & \text{if } i = 0 \\ \exists_\triangleright(\lambda_{u_\triangleright,i}) & \text{if } i > 0 \end{cases}$$

$$\lambda_{j,i} = \begin{cases} \mu_i & \text{if } j = 0 \text{ or } i = 0 \\ a \wedge (c\,U\,\lambda_{j-1,i-1}) & \text{if } j > 0 \text{ and } i > 0 \end{cases}$$

The following lemma shows that $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleright \leq p_\triangleright, \mathrm{ud}_\triangleright \leq u_\triangleright)$ cannot express Q_{i+1} if $p_\triangleright u_\triangleright < i$. Thus it implies Theorem 1.3.

Lemma 1.7. *For all $p_\triangleright, u_\triangleright, o \in \mathbb{N}$, there are finite trees $\tilde{T}$ and $\tilde{S}$ such that (a) $\tilde{T}$ satisfies $Q_{p_\triangleright u_\triangleright+1}$ but $\tilde{S}$ does not and (b) Duplicator has a winning strategy for the $CTL^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleright \leq p_\triangleright, \mathrm{ud}_\triangleright \leq u_\triangleright, \mathrm{other} \leq o)$-game on $\tilde{T}, \tilde{S}$.*

Before we prove Lemma 1.7, we note a consequence of it: Consider the case where $u_\triangleright = 0$. Then Lemma 1.7 states that there are finite trees $\tilde{T}, \tilde{S}$ such that $\tilde{T}$ satisfies Q_1 but $\tilde{S}$ does not and Duplicator wins the $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleright \leq p_\triangleright, \mathrm{ud}_\triangleright = 0; \mathrm{other} \leq o)$-game on $\tilde{T}, \tilde{S}$ for all $p_\triangleright, o \in \mathbb{N}$. It follows that Q_1 cannot be expressed in $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleright \leq p_\triangleright, \mathrm{ud}_\triangleright = 0; \mathrm{other} \leq o)$ for any $p_\triangleright, o \in \mathbb{N}$. Hence we get the following corollary, which implies that ForCXPath is not first-order complete.

Corollary 1.8. *$CTL^*_{\triangleleft\triangleright}(\mathrm{ud}_\triangleright = 0)$ is not first-order complete on finite ordered trees.*

We now start with the proof of Lemma 1.7. Recall that this lemma is all the we need to prove in order to complete the proof of Theorem 1.3.

of Lemma 1.7. The proof is quite long. We will define two trees $\tilde{T}, \tilde{S}$ and then we show that Duplicator has a winning strategy for the $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleright \leq p_\triangleright, \mathrm{ud}_\triangleright \leq u_\triangleright; \mathrm{other} \leq o)$-game on $\tilde{T}, \tilde{S}$. We will first show the lemma for "wide trees" – those in which a node can have infinitely many left and right siblings, but where any two siblings have only finitely many nodes between them (i.e. the sibling order has type $\omega^* + \omega$). Later we explain how the proof must be altered for the finite case.

The Trees

Before we delve into the formal definition of the trees $\tilde{T}$ and $\tilde{S}$ on which the players will play their game, we give some intuition. In the following we will call a node x of a tree *i-positive* if the subtree rooted at x satisfies Q_i. We will define $\tilde{T}$ and $\tilde{S}$ in such a way that both have many inner i-positive nodes. However, the root of $\tilde{T}$ will be i-positive for the appropriate i, whereas the root of $\tilde{S}$ will not be i-positive. In addition, each inner node in $\tilde{T}$ or $\tilde{S}$ that is not positive will be "almost i-positive" in the sense that it is the root of some path that is very similar to a path that is a witness of property Q_i (recall that Q_i is a property of the form "there is a path such that..."). During the game played by Spoiler and Duplicator to will often be the case that one selected node is i-positive, while the other is not. In this case Duplicator must worry that Spoiler plays a path that witnesses that one of the selected nodes is i-positive. Duplicator will exploit that the other node is "almost i-positive" and select the path that is very similar to the path that Spoiler selected.

We now give a formal definition of $\tilde{T}$ and $\tilde{S}$. We need some way to inductively define trees. We will use the concept of templates, defined in the following. A *hedge* is an ordered sequences of wide trees, where the sequence can again be infinite in both directions. A *template* F is a hedge with two sets of distinguished nodes – *positive ports* and *negative ports* – labelled by "+" and "–" respectively. A template F and two hedges $\bar{T}, \bar{S}$ can be combined to form a new hedge $F[\bar{T}, \bar{S}]$ which is obtained from F by replacing each positive port with the hedge $\bar{T}$ and each negative port with the hedge $\bar{S}$. In Figure 1.3 we see an example of a template F and two hedges S and T and the result of applying F to S and T.

We now fix some given $\tilde{p}_\triangleright, \tilde{u}_\triangleright, \tilde{o} \in \mathbb{N}$ for the rest of the proof of Lemma 1.7. We define for each $k \in \mathbb{N}$ two of hedges $\bar{T}^k$ and $\bar{S}^k$. These hedges will be constructed

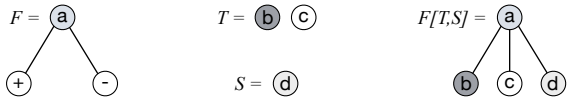


Figure 1.3: Constructing the hedge $F[T, S]$ from the template F and the hedges T and S.

from two templates F and G which we define first. Both F and G are hedges of infinite width. The sequence of roots of both F and G spell out the infinite word

$$((c^n a)^m\, c^n b)^\omega$$

where $n = \tilde{p}_\triangleright \cdot \tilde{o} + \tilde{o}^2$ and $m = \tilde{p}_\triangleright \cdot \tilde{u}_\triangleright + \tilde{o}^2 + 2$. All trees in F or G that have a root labelled c are singleton trees, consisting only of a root. The trees that have a root labelled a or b consist of a root with a single child that is either a positive or a negative port. In both templates all ports but one are negative. What distinguishes F from G is the labeling of the right path that starts at the unique parent of the positive port: In F this path has a label in the language $a(c^*a)^{\tilde{p}_\triangleright \tilde{u}_\triangleright + 1}\, c^* b\, \Sigma^\omega$; in G its label lies in the language $a(c^*a)^{\tilde{p}_\triangleright \tilde{u}_\triangleright} c^* b\, \Sigma^\omega$. Figure 1.4 shows F and G.

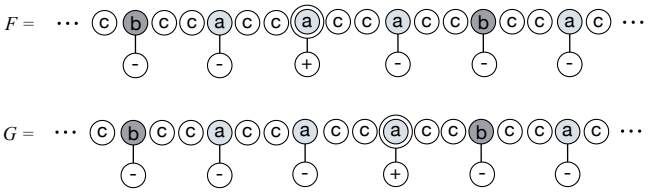


Figure 1.4: The templates F and G for $\tilde{p}_\triangleright = 1$, $\tilde{o} = 1$, and $\tilde{u}_\triangleright = 0$.

We define two sequences $(\bar{T}_k)_{k\in\mathbb{N}}$ and $(\bar{S}_k)_{k\in\mathbb{N}}$ of hedges by induction on k: $\bar{T}_0$ is the single node labelled d and $\bar{S}_0$ is the single node labelled c. For $k > 0$

$$\bar{T}_k := F\left[\bar{T}_{k-1}, \bar{S}_{k-1}\right]$$
$$\bar{S}_k := G\left[\bar{T}_{k-1}, \bar{S}_{k-1}\right]$$

We say that the single node T_0 is *positive* and the single node S_0 is *negative*. We also say that a node x in $\bar{T}_k$ is *positive* (resp. *negative*) if the node in the template that x is obtained from is *positive* (resp. *negative*). The polarity of a subtree is the polarity of its root. Figure 1.5 shows a positive and a negative tree in $\bar{T}_3$.

Now we can define $\tilde{T}$ and $\tilde{S}$, the trees whose existence is asserted in Lemma 1.7. These are also the trees on which Spoiler and Duplicator will play the $\mathrm{CTL}^*_{\triangleleft\triangleright}$-game. Recall that we fixed $\tilde{o} \in \mathbb{N}$ at the beginning of this proof.

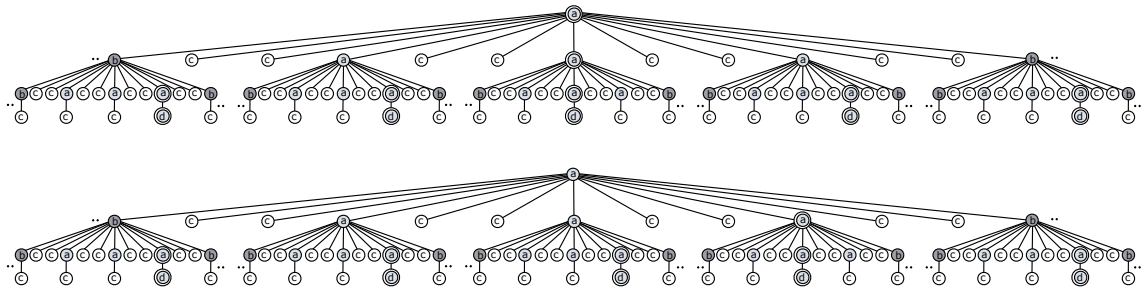


Figure 1.5: A positive tree in (top) and a negative tree (bottom) in $\bar{T}_3$ for $\tilde{p}_\triangleright = 1, \tilde{u}_\triangleright = 0$ and $\tilde{o} = 0$. Roots of positive subtrees are displayed with double circles, roots of negative subtrees with single ones.

Definition 1.9 ($\tilde{T}$, $\tilde{S}$)**.** *Let $\tilde{T}$ be the unique positive tree in the hedge $\bar{T}_{\tilde{o}+1}$. $\tilde{S}$ is some negative tree in $\bar{S}_{\tilde{o}+1}$ whose root is labelled a (note that all trees with this property are isomorphic).*

It is easy to see that $\tilde{T}$ satisfies $Q_{\tilde{p}_\triangleright \tilde{u}_\triangleright + 1}$ while $\tilde{S}$ does not: An inductive argument shows that a subtree of $\bar{T}_{\tilde{o}+1}$ or $\bar{S}_{\tilde{o}+1}$ satisfies $Q_{\tilde{p}_\triangleright \tilde{u}_\triangleright + 1}$ iff it is positive. Hence part (a) of Lemma 1.7 follows.

Challenges for Duplicator

We now provide some intuition about Duplicators strategy.

Duplicator's Strategy

We now show part (b) of Lemma 1.7 for $\tilde{T}$ and $\tilde{S}$ as defined above. In fact, we show the following more general statement on hedges:

> If x is the root of a positive tree in $\bar{T}_{\tilde{o}+1}$ and y is the root of a negative tree in $\bar{S}_{\tilde{o}+1}$, then Duplicator has a winning strategy for the $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleright \leq \tilde{p}_\triangleright, \mathrm{ud}_\triangleright \leq \tilde{u}_\triangleright; \mathrm{other} \leq \tilde{o})$-game on the hedges $\bar{T}_{\tilde{o}+1}$ and $\bar{S}_{\tilde{o}+1}$ starting from position x, y.

We use an extension of $\mathrm{CTL}^*_{\triangleleft\triangleright}$ for hedges: The node formula $\exists_\triangleright \phi$ is true at the root x of a tree in a hedge $\bar{T}$ if ϕ is true on the sequence of roots of $\bar{T}$ to the right of x. In a similar way, the $\mathrm{CTL}^*_{\triangleleft\triangleright}$-game can be extended to hedges, such that the players can choose the sequence of roots right of the current selection in $\exists_\triangleright$ moves. The symmetric definitions hold for leftward paths.

Notation During the game on $\bar{T}_{\tilde{o}+1}$ and $\bar{S}_{\tilde{o}+1}$, Duplicator can maintain an invariant on the string of siblings of the selected nodes x and y. This invariant is similar to the invariant used in the word game from Section 1.5.2, but this time, Duplicator not only has to assure that the strings to the right of the selected nodes are similar, but also those to the left of them. We therefore define the inverse versions of top-depth and plateau-depth. The *inverse-plateau-depth* of a node x is the number of c-labelled nodes between x and the next node labelled a or b to the left of x. The *bottom-depth* of x is the number a-labelled nodes between x and the next b to the left of x. The *plateau-depth* of a node x in $\bar{T}_k$ or $\bar{S}_k$ is the plateau-depth of x with respect to its sequence of right siblings – or on the sequence of roots of $\bar{T}_k$ or $\bar{S}_k$, if x is a root. We lift the notions of inverse-plateau-depth, the top-depth, and the bottom-depth to trees in the same way. In each sequence of siblings in either $\bar{T}_k$ or $\bar{S}_k$ we distinguish the node that is the next b-labelled node to the right of the only positive node. We will say that a node x in $\bar{S}_k$ *corresponds* to a node y in $\bar{T}_k$ (or vice versa) if x and y have the same distance and direction to the distinguished node in their respective sequence of siblings.

The following claim is sufficient for part (b) of Lemma 1.7. As we have already proven part (a) of Lemma 1.7, the proof of the following claim completes the proof of Theorem 1.3.

Claim 1. *Let $p_\triangleright \le \tilde{p}_\triangleright$, and $p_\triangleleft, p_\triangledown \le \tilde{o}$. Duplicator can win the $CTL^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleleft \le p_\triangleleft, \mathrm{pd}_\triangledown \le p_\triangledown, \mathrm{pd}_\triangleright \le p_\triangleright, \mathrm{ud}_\triangleright \le \tilde{u}_\triangleright; \mathrm{other} \le \tilde{o})$-game on $\bar{S}_{\tilde{o}+1}, \bar{T}_{\tilde{o}+1}$ if the selected positions x, y satisfy:*

1. *x and y have the same label.*
2. *x and y have the same plateau-depth.*
3. *$|\text{top-depth}(x) - \text{top-depth}(y)| \le 1$.*
4. *If x and y have different top-depths then*
   (a) *$\text{top-depth}(x) \ge p_\triangleright \tilde{u}_\triangleright$ and $\text{top-depth}(y) \ge p_\triangleright \tilde{u}_\triangleright$*
   (b) *if $\text{top-depth}(x) = p_\triangleright \tilde{u}_\triangleright$ or $\text{top-depth}(y) = p_\triangleright \tilde{u}_\triangleright$ then $\text{plateau-depth}(x) \ge p_\triangleright \tilde{o}$ (and hence $\text{plateau-depth}(y) \ge p_\triangleright \tilde{o}$).*
5. *If x and y have different bottom-depths then*
   (a) *$\text{bottom-depth}(x) \ge p_\triangleleft \tilde{o}$ and $\text{bottom-depth}(y) \ge p_\triangleleft \tilde{o}$*

(b) if bottom-depth$(x) = p_\triangleleft \tilde{o}$ *or* bottom-depth$(y) = p_\triangleleft \tilde{o}$
then inverse-plateau-depth$(x) \geq p_\triangleleft \tilde{o}$
(and hence inverse-plateau-depth$(y) \geq p_\triangleleft \tilde{o})$.

Proof of Claim 1. The claim is proven by induction on $p_\triangleleft + p_\triangledown + p_\triangleright$. The base case holds as by Condition 1 the nodes x and y have the same label. Now assume that $p_\triangleleft + p_\triangledown + p_\triangleright > 0$.

Strategy for $\exists_\triangledown$-moves Assume that Spoiler picks a downward path π rooted at x in either $\bar{T}_{\tilde{o}+1}$ or $\bar{S}_{\tilde{o}+1}$. As x and y have the same label, it follows that there is also an infinite downward-fullpath rooted at y. In fact there are several such paths, and we now describe which of these Duplicator chooses.

Let x' be the child of x on π. We first determine the child y' of y on the path τ that Duplicator chooses. Duplicator's goal is to pick y' such that she can win the $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleleft \leq p_\triangleleft, \mathrm{pd}_\triangledown \leq p_\triangledown - 1, \mathrm{pd}_\triangleright \leq p_\triangleright, \mathrm{ud}_\triangleright \leq \tilde{u}_\triangleright; \mathrm{other} \leq \tilde{o})$-game from x', y' and such that the trees rooted at x' and y' have the same polarity. The easiest case is when the node that corresponds to x' in the other tree has the same polarity as x': in this case Duplicator will choose this node. Otherwise there are two cases: If x' has positive polarity, then Duplicator picks the single child of y that has positive polarity. The remaining case is that x' has negative polarity and its corresponding node in the other tree has positive polarity. In this case Duplicator picks the unique child y' of y who's corresponding node has positive polarity (observe that y' has negative polarity). In any case x' has the same polarity as y'. Thus the trees rooted at x' and y' are isomorphic, and Duplicator can choose a path τ' starting at y' that (a) has the same labeling as the suffix π' of π starting at x' and (b) the node $\pi'(i)$ has the same polarity as the node $\tau'(i)$ for all $i \leq |\pi'|$. As π and τ have the same labeling, Duplicator can play isomorphically on the path game.

Maintaining the Invariant on $\exists_\triangledown$-moves At the end of the path game Spoiler will pick an intermediate position from which the tree game will continue. Hence we verify that all possible intermediate positions of the path game are winning positions for Duplicator for the $\mathrm{CTL}^*_{\triangleleft\triangleright}$-game with $p_\triangledown - 1$ downward path moves. This follows from the induction hypothesis if x

and y – the roots of π and τ – is the final position. For the case that Spoiler chooses to continue to play from position x', y', we verify that Duplicator can win the $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleleft \leq p_\triangleleft, \mathrm{pd}_\triangledown \leq p_\triangledown - 1, \mathrm{pd}_\triangleright \leq p_\triangleright, \mathrm{ud}_\triangleright \leq \tilde{u}_\triangleright; \mathrm{other} \leq \tilde{o})$-game from x' and y'. The interesting case is when x' and y' both have positive polarity. Assume that the sequence of siblings of x' is obtained from F and that the siblings of y' are obtained from G. Then Condition 2 holds because both x' and y' have positive polarity. To check Condition 3 recall that the right fullpath starting at x is in the language

$$a(c^n a)^{\tilde{p}_\triangleright \tilde{u}_\triangleright + 1} c^n b \, \Sigma^\omega$$

with $n = \tilde{p}_\triangleright \cdot \tilde{o} + \tilde{o}^2$, while the right fullpath starting at y' is in the language

$$a(c^n a)^{\tilde{p}_\triangleright \tilde{u}_\triangleright} c^n b \, \Sigma^\omega.$$

Thus Conditions 3 and 4 are maintained. For Condition 5, observe that both x' and y' are contained in a block of siblings labelled

$$b\,(c^n a)^m \; c^n b$$

where $m = \tilde{p}_\triangleright \cdot \tilde{u}_\triangleright + \tilde{o}^2 + 2$. Hence the left path from x' to its next b-labelled left sibling is labelled $a(c^n a)^{\tilde{o}^2} c^n b$ (from right to left) and the string from y' to its next b left sibling is $a(c^n a)^{\tilde{o}^2+1} c^n b$ (from right to left). As $p_\triangleleft \leq \tilde{o}$ this shows that Condition 5 is maintained. If the position that Spoiler chooses is further down in π, τ than x', y', then the selected nodes are isomorphic positions in isomorphic hedges, and therefore winning positions for any $\mathrm{CTL}^*_{\triangleleft\triangleright}$-game.

$\exists_\triangleright$-moves Now assume that Spoiler picks a rightward fullpath π in either hedge. Duplicator has to choose the unique rightward fullpath τ starting at the selected node in the other tree. The players then play the $\mathrm{LTL}(\mathrm{ud} \leq \tilde{u}_\triangleright, \mathrm{ned} \leq \tilde{o})$ game on the selected paths. In this game, Duplicator uses the strategy described in Section 1.5.2. The following claim can be shown using the strategy described in the proof of Claim 1 in Lemma 1.4.

SUBCLAIM 1.1. *Let x, y be positions that satisfy the conditions of Claim 1 and let $\tilde{\pi}, \tilde{\tau}$ be the right fullpaths starting at x, y respectively. For all $u_\triangleright \leq \tilde{u}_\triangleright$ and $o \leq \tilde{o}$, Duplicator has a winning strategy for the $LTL(\mathrm{ud} \leq u_\triangleright, \mathrm{ned} \leq o)$-game on suffixes π, τ of $\tilde{\pi}, \tilde{\tau}$ if*

1. *π and τ have the same plateau-depth.*
2. $|\text{top-depth}(\pi) - \text{top-depth}(\tau)| - 1$.
3. *If π and τ have different top-depths then*
   (a) $\text{top-depth}(\pi) \geq (p_\triangleright - 1)\tilde{u}_\triangleright + u_\triangleright$ *and* $\text{top-depth}(\tau) \geq (p_\triangleright - 1)\tilde{u}_\triangleright + u_\triangleright$
   (b) *if* $\text{top-depth}(\pi) = (p_\triangleright - 1)\tilde{u}_\triangleright + u_\triangleright$ *or* $\text{top-depth}(\tau) = (p_\triangleright - 1)\tilde{u}_\triangleright + u_\triangleright$
   then $\text{plateau-depth}(\pi) \geq (p_\triangleright - 1)\tilde{o} + o$
   (and hence $\text{plateau-depth}(\tau) \geq (p_\triangleright - 1)\tilde{o} + o$*).*

In addition, every intermediate position of this LTL-game is a winning position for Duplicator in the $CTL^*_{\triangleleft\triangleright}(\text{pd}_\triangleleft \leq p_\triangleleft, \text{pd}_\triangledown \leq p_\triangledown, \text{pd}_\triangleright \leq p_\triangleright - 1, \text{ud}_\triangleright \leq \tilde{u}_\triangleright; \text{other} \leq \tilde{o})$-game.

It is easy to check that the conditions of Subclaim 1.1 imply the conditions of Claim 1 for $(p_\triangleright, p_\triangleleft - 1, p_\triangledown)$ moves left to play.

$\exists_\triangleleft$-moves Duplicator's strategy for leftward path moves is symmetric to her strategy on right paths moves. The proof that this strategy maintains the invariant follows the same lines as above. This concludes the proof of Claim 1. □

This concludes the proof of Theorem 1.3 for infinite trees.

The Finite Case We now describe what needs to be changed, in order to prove Theorem 1.3 for finite trees. Recall that roots of both F and G spell out the infinite word

$$((c^n a)^m\ c^n b)^\omega$$

where $n = \tilde{p}_\triangleright \cdot \tilde{o} + \tilde{o}^2$ and $m = \tilde{p}_\triangleright \cdot \tilde{u}_\triangleright + \tilde{o}^2 + 2$. Consider a template F^{fin} that is obtained from F by pruning in such a way that the sequence of its roots spells out the word

$$((c^n a)^m\ c^n b)^{2(\tilde{o}^2 + \tilde{o}\tilde{p}_\triangleright)}$$

and the right-path starting at the unique root with a positive child is labelled $a(c^n a)^{\tilde{p}_\triangleright \tilde{u}_\triangleright + 1}\, c^n b\, w^{2op_\triangleright}$ for $w = (c^n a)^m\, c^n b$. The finite template G^{fin} is obtained from G in a similar way, with the right-path starting at the parent of the unique positive port being in the language $a(c^* a)^{\tilde{p}_\triangleright \tilde{u}_\triangleright} c^* b\, w^{2op_\triangleright}$. Consider the finite hedges $\bar{T}_k^{\text{fin}}, \bar{S}_k^{\text{fin}}$

that are constructed like the hedges $\bar{T}_k$ and $\bar{S}_k$ but from templates F^{fin} and G^{fin} instead of from the templates F and G. We argue that Duplicator can win the $\text{CTL}^*_{\triangleleft\triangleright}(\text{pd}_\triangleleft \leq p_\triangleleft, \text{pd}_\triangledown \leq p_\triangledown, \text{pd}_\triangleright \leq p_\triangleright, \text{ud}_\triangleright \leq \tilde{u}_\triangleright; \text{other} \leq \tilde{o})$-game on $\bar{S}^{\text{fin}}_{\tilde{o}+1}$ and $\bar{T}^{\text{fin}}_{\tilde{o}+1}$. In fact, one can show that if the current position is x and y, then Duplicator can maintain the invariant of Claim 1 together with the additional condition:

6. If the number of b-labelled nodes to the right x is not equal to the number of b-labelled nodes to the right y, then there are at least $2\tilde{o}\tilde{p}_\triangleright$ b-labelled nodes to the right of both x and y and there are at least $2\tilde{o}^2$ b-labelled nodes to the left of both x and y.

On path moves, Duplicator can reuse her strategy from the infinite case. In the path game, Duplicator can use the same strategy as in the infinite case on downward paths, and the Etessami-Wilke strategy from Corollary 1.5 on horizontal paths. It is easy to check that this strategy preserves the invariant. This concludes the proof of Theorem 1.3 for finite trees. □

1.6 The Vertical Until Hierarchy

We now show our second incompleteness result.

Theorem 1.4 (Vertical Incompleteness)**.** *On finite, ordered trees, $CTL^*_{\triangleleft\triangleright}(\text{ud}_\triangledown \leq u)$ is not first-order complete for all $u \in \mathbb{N}$. In addition, the family $CTL^*_{\triangleleft\triangleright}(\text{ud}_\triangledown \leq u)$ forms a strict hierarchy in expressiveness.*

The structure of the proof is similar to the structure of the proof of Theorem 1.3: For each $u_\triangledown \in \mathbb{N}$ we define a property $P_{u_\triangledown}$ that can be expressed in $\text{CTL}^*_{\triangleleft\triangleright}(\text{ud}_\triangledown \leq u_\triangledown)$ but not in $\text{CTL}^*_{\triangleleft\triangleright}(\text{ud}_\triangledown \leq u_\triangledown - 1)$. Again, we first show Theorem 1.3 for infinite trees and then we discuss what needs to be changed for the finite case.

1.6.1 The Separating Property

For $u_\triangledown \in \mathbb{N}$ we define $P_{u_\triangledown}$ be the set of ordered trees which have a fullpath π ending at d such that each suffix of π that starts with b satisfies the regular expression

$$b(c^*a)^{u_\triangledown}\,\Sigma^\omega.$$

$P_{u_\triangledown}$ can be expressed in $\text{CTL}^*_{\triangleleft\triangleright}(\text{ud}_\triangledown \leq u_\triangledown)$ by the formula 'there is a downward fullpath ending at d on which every node satisfies $b \to \mu_{u_\triangledown}$' where $\mu_{u_\triangledown}$ is defined by

$$\mu_i = \begin{cases} \text{true} & \text{if } i = 0 \\ c\,U\,(a \wedge \mu_{i-1}) & \text{if } i > 0 \end{cases}$$

The following lemma shows that, for all $u_\triangledown \in \mathbb{N}$, $P_{u_\triangledown+1}$ cannot be expressed in $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{ud}_\triangledown \leq u_\triangledown)$.

Lemma 1.10. *For all $u_\triangledown, o \in \mathbb{N}$ there are finite trees $\tilde{T}$ and $\tilde{S}$ such that (i) $\tilde{T}$ satisfies $P_{u_\triangledown+1}$ but $\tilde{S}$ does not and (ii) Duplicator has a winning strategy for the $CTL(\mathrm{ud}_\triangledown \leq u_\triangledown; \mathrm{other} \leq o)$ game on $\tilde{T}, \tilde{S}$.*

Before we prove Lemma 1.10 we show how it implies Theorem 1.4.

Proof of Theorem 1.4. Assume towards a contradiction that there is a number $u_\triangledown \in \mathbb{N}$ such that $P_{u_\triangledown+1}$ can be expressed in $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{ud}_\triangledown \leq u_\triangledown)$. Then there is a formula ϕ in $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{ud}_\triangledown \leq u_\triangledown)$ that expresses P_u. In particular, ϕ is in $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{ud}_\triangledown \leq u_\triangledown; \mathrm{other} \leq o)$ for some $o \in \mathbb{N}$.

For the numbers $u_\triangledown$ and o there are, according to Lemma 1.10, two trees $\tilde{T}$ and $\tilde{S}$ on which Duplicator has a winning strategy for the $\mathrm{CTL}(\mathrm{ud}_\triangledown \leq u_\triangledown; \mathrm{other} \leq o)$-game. Thus, by Proposition 1.2, no $\mathrm{CTL}(\mathrm{ud}_\triangledown \leq u_\triangledown; \mathrm{other} \leq o)$ formula can distinguish $\tilde{T}$ and $\tilde{S}$. Therefore ϕ either holds on $\tilde{T}$ and $\tilde{S}$ or on none of them. As $\tilde{T}$ satisfies $P_{u_\triangledown+1}$ but $\tilde{S}$ does not it follows that ϕ does not define $P_{u_\triangledown+1}$. This is a contradiction to the assumption that ϕ defines P_u. □

The proof of Lemma 1.10 is quite involved and will take up the rest of this section. In Section 1.6.2 we construct two trees $\tilde{T}$ and $\tilde{S}$ on which the $\mathrm{CTL}^*_{\triangleleft\triangleright}$-game will be played. Before we define Duplicator's strategy formally, we explain its intuition in Section 1.6.3. Section 1.6.4 describes Duplicator's strategy in detail, and we show that it is a winning strategy. To simplify the presentation, Sections 1.6.2 – 1.6.4 prove the lemma for infinite trees. Section 1.6.5 describes which modifications are necessary for the proof on finite trees.

1.6.2 The Trees

We now fix two numbers $\tilde{u}_\triangledown, \tilde{o} \in \mathbb{N}$ for the rest of the proof of Lemma 1.10. We construct two families of infinite binary trees $(T_k)_{k\in\mathbb{N}}$ and $(S_k)_{k\in\mathbb{N}}$ from the templates F and G shown in Figure 1.6. The only distinction between F and G lies in the node labelled "±": it is a positive port in F and a negative port in G. In each of the two templates, the leftmost branch consists of the root labelled b followed by infinitely many $c^{\tilde{o}}a$ blocks. The final c node in any $c^{\tilde{o}}$ sequence has two children: the left child is labelled a and the right child is a positive or negative port. In both F and G the topmost $\tilde{u}_\triangledown$ ports are negative. In F the next two ports are positive, while in G only the next port is positive. All other ports are negative.

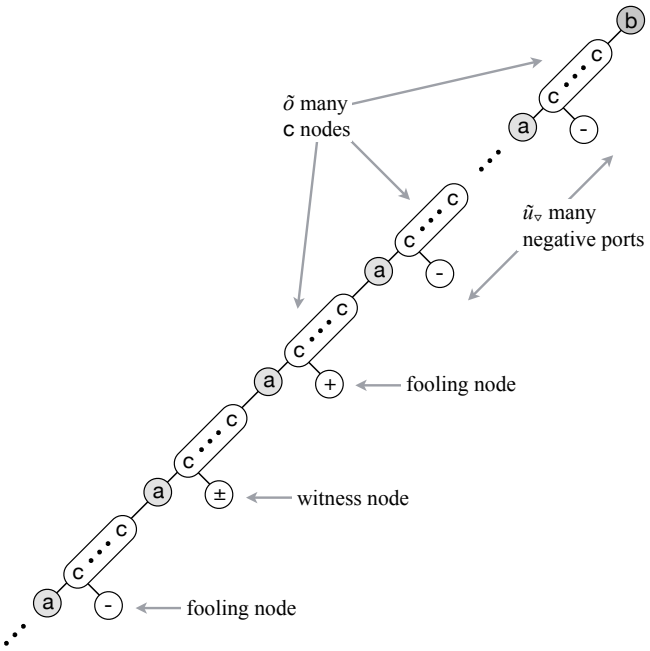


Figure 1.6: The templates F and G

We define the sequences $(T_k)_{k\in\mathbb{N}}$ and $(S_k)_{k\in\mathbb{N}}$ by induction on k. T_0 consists of a single node labelled d, while S_0 consists of a single node labelled c. For $k \geq 1$:

$$T_k := F\left[T_{k-1}, S_{k-1}\right]$$
$$S_k := G\left[T_{k-1}, S_{k-1}\right]$$

For trees T, T' we write $T \equiv T'$ iff T and T' agree in all $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{ud} \leq \tilde{u}_\triangledown; \mathrm{other} \leq \tilde{o})$ formulas. It is easy to see that $\equiv$ is an equivalence relation. The next lemma shows that $\equiv$ has a finite number of equivalence classes.

Lemma 1.11. *The relation $\equiv$ has at most $\mathcal{O}(\Sigma^{\mathcal{O}(2^m)})$ equivalence classes where $m = \max(\tilde{o}^2, \tilde{u}_\triangledown\tilde{o})$.*

Proof. We show that there are at most $\mathcal{O}(\Sigma^{\mathcal{O}(2^m)})$, $m = \max(\tilde{o}^2, \tilde{u}_\triangledown\tilde{o})$ different syntax trees of formulas in $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{ud} \leq \tilde{u}_\triangledown; \mathrm{other} \leq \tilde{o})$. Note that each such syntax tree is a binary tree that has depth at most $m = \max(\tilde{o}^2, \tilde{u}_\triangledown\tilde{o})$. Each node is labelled by either one of the symbols $\{U, X, F, \wedge, \vee, \neg\}$ or by a predicate in Σ. As a binary tree of depth m has at most $2(2^m)$ nodes, it follows that there are at most $(6+|\Sigma|)^{2(2^m)} = \mathcal{O}(\Sigma^{\mathcal{O}(2^m)})$ syntax trees of formulas in $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{ud} \leq \tilde{u}_\triangledown; \mathrm{other} \leq \tilde{o})$. □

Definition 1.12 (Period)**.** *A pair of numbers (μ, λ) is a* period *of the sequences $(T_k)_{k\in\mathbb{N}}$ and $(S_k)_{k\in\mathbb{N}}$ if for all $k \geq \mu$, $T_k \equiv T_{k+\lambda}$ and $S_k \equiv S_{k+\lambda}$.*

Lemma 1.13. *$(T_k)_{k\in\mathbb{N}}$ and $(S_k)_{k\in\mathbb{N}}$ have a period (μ,λ).*

Proof. Let E_k be the pair of $\equiv$-classes of T_k and S_k. Since the number of equivalence classes of $\equiv$ is finite, there must be μ and λ such that $E_{\mu+\lambda} = E_\mu$, and we claim that this μ and λ suffice. Note that the $\equiv$-class of $F[T_k, S_k]$ (and $G[T_k, S_k]$) depends only on the $\equiv$-classes of T_k and S_k for all $k \in \mathbb{N}$. This follows by induction using the usual composition technique for trees (see e.g. [MR99]). Hence from $T_k \equiv T_{k+\lambda}$ and $S_k \equiv S_{k+\lambda}$ we can conclude (applying F to both equivalence classes) that $T_{k+1} \equiv T_{k+\lambda+1}$, and that $S_{k+1} \equiv S_{k+\lambda+1}$ (applying G to both equivalence classes). The result now follows by induction. □

Fix a period (μ,λ) of $(T_k)_{k\in\mathbb{N}}$ and $(S_k)_{k\in\mathbb{N}}$ for the rest of this proof. We can now define the trees $\tilde{T}, \tilde{S}$ on which the CTL-game will be played.

Definition 1.14. *Let $k = 3\tilde{o}^2\lambda + \mu + 1$. Then we define*

$$\tilde{T} := T_k \qquad\qquad \tilde{S} := S_k$$

Notation We call a $b(c^*a)^\omega$ labelled path within T_k or S_k a *stem*. Observe that a stem is isomorphic to F (or G) without ports. A node that is connected to the root of a stem by a path labelled $b(c^*a)^i c^*b$ is called a *witness node* if $i = \tilde{u}_\triangledown + 1$ and *fooling node* if $i = \tilde{u}_\triangledown$ or $i = \tilde{u}_\triangledown + 2$. A path that always departs from the stem on the witness node is a *witness path*. Observe that T_k contains a single witness path that starts at the root, and both T_k and S_k contain several witness paths that start at intermediate nodes. The *enclosing subtree* of a node x in $T \in \{T_k, S_k\}$ is the smallest subtree of T that contains x and that is isomorphic to either T_j or S_j for some $j \le k$. A subtree T of T_k or S_k has *positive polarity* if it is isomorphic to T_j for some $j \le k$ and T has *negative polarity* otherwise. The *polarity* of a node x is the polarity of its enclosing subtree.

We now show part (i) of Lemma 1.10.

Proposition 1.15. *For all $k \in \mathbb{N}$, T_k satisfies $P_{\tilde{u}_\triangledown+1}$ but S_k does not.*

Proof. The proof is by induction on k. The proposition obviously holds for the trees T_0 and S_0. Now let $k > 0$. Note that the witness path departs from the topmost stem on the witness node it starts with a sequence labelled $b(c^*a)^{\tilde{u}_\triangledown+1}c^*b$. Hence the first b is followed by sufficiently many a nodes. In addition, it departs from the topmost stem into a subtree isomorphic to T_{k-1}. Hence it follows by induction that T_k satisfies $P_{\tilde{u}_\triangledown}$.

We now show that S_k does not satisfy $P_{\tilde{u}_\triangledown}$. We know by induction that any strict subtree S of S_k of negative polarity does not contain a path witnessing that $S \in P_{\tilde{u}_\triangledown+1}$. Hence a path witnessing $S_k \in P_{\tilde{u}_\triangledown+1}$ must contain the upper fooling node of the topmost stem. But any path that contains this node starts with a sequence labelled $b(c^*a)^{\tilde{u}_\triangledown}c^*b$. □

1.6.3 The Strategy: Challenges for Duplicator

To prove part (ii) of Lemma 1.10 we will need to show that Duplicator has a winning strategy for the CTL($\mathrm{ud}_\triangledown \leq u_\triangledown; other \leq o$) game on $\tilde{T}, \tilde{S}$. We start with some intuition about the strategy of Duplicator. The idea is that $\tilde{T}$ and $\tilde{S}$ consist of several "similar levels" of subtrees. There is a number λ such that subtrees on levels that are λ apart and of the same polarity are winning positions for Duplicator (in fact this is the number of Lemma 1.13). The game will start on similar levels, but on trees of different polarity. As Spoiler eventually wins on trees of non-similar levels, Duplicator must assure that the selected nodes stay on similar levels throughout the game. Duplicator cannot force the game to a position with both similar levels and of the same polarity, thus her strategy is to maintain the position on similar levels regardless of the polarities. As Duplicator has to keep the position on similar levels, Spoiler can force the game down a fixed number of levels on each move. Thus Duplicator can only win the game on trees with very many similar levels. Thereby she must assure that the selected nodes are high up in the tree if the polarities of their enclosing subtrees differ.

Basically, Duplicator will have to disguise that the witness path in T ends on a d while the witness path in S ends in a c. Clearly, Duplicator must have a remedy when Spoiler plays the witness path in T. In fact, Duplicator's strategy depends on the place where Spoiler's path departs from the witness path. Assume Spoiler picks a path $\pi\pi'$ that departs from the witness path on the root of π'. Then Duplicator's response depends on the length on π.

The case where π is short is "easy" for Duplicator: she picks a path $\tau\tau'$ such that τ is isomorphic to π. To choose τ', observe that there are two possibilities for path $\pi\pi'$ to depart from the witness path. If $\pi\pi'$ departs above the witness node then Duplicator can choose the root of τ' such that the roots π' and τ' have the same polarity. As Duplicator maintained similar levels throughout the game the roots of π' and τ' are on similar levels and hence winning positions for Duplicator. She can use her winning strategy to determine the rest of τ'. If $\pi\pi'$ does not depart from the witness path above the witness node then it departs on the (a-labelled) sibling

of the witness node. We will see that in this case Duplicator has an "easy" strategy to determine τ'. It is easy to see that Duplicator wins the path game if the paths are chosen in this way. There are two cases for its final position: If the final position is in π, τ, the Duplicator has achieved her goal to keep the trees big, and therefore she wins by induction. If the final position is in π', τ' then the selected nodes are on similar levels and of the same polarity – and Duplicator wins by the definition of similar levels.

The case that π is long is more threatening for Duplicator. If Duplicator uses the strategy for "short" moves described above then the final position of the path game might be in small subtrees of different polarity. In this case it is not guarantied that there are sufficiently may levels below the selected nodes for Duplicator to use her strategy. Therefore Duplicator will respond to such a "long" move of Spoiler by picking a path that moves off a stem at a different point some place down the tree – Duplicator has some flexibility as to where to do this "fooling", which we will exploit.

But given that Duplicator has played a fooling path, the first cause for concern is that Spoiler may try to detect a distinction in the paths by moving to the "fooling point" where the two paths are first distinguished – the point in which one path departs from a stem at a different point from the other path. Note that on the witness path, the number of c^*a blocks between the root of a stem and the departure point is $u_\triangledown + 1$. Hence Spoiler will be unable to use only until moves to force the play to this point on the critical path, since his until moves are limited to $u_\triangledown$. But one must still worry that Spoiler can try to push the play down to this point using eventually moves, which he has in some abundance. The response of the Duplicator to these threatening eventually moves will be to jump down to a lower stem. This is analogous to the strategy used by the Duplicator in the linear case of the until-depth hierarchy theorem of Etessami and Wilke ([EW00], Theorem 1.5); there, the Duplicator responds to eventually moves of the Spoiler by jumping to the next $b(c^*a)^*c^*b$ block in the word.

However, this "jumping response" of Duplicator cannot be done so naively in the setting of ordered trees. If Duplicator jumps so that the position is only one level off from the position of Spoiler, then the two nodes are on non-similar levels. In particular the selected nodes are in enclosing subtrees T_i and S_j where i and j have different parities; Spoiler can detect this difference in parity of i and j by playing paths that alternate in the way they jump from stem to stem: e.g. by playing a path that will depart after two a's on even levels and after one a on odd levels. This method of detecting differences in trees of different depths goes back to Potthoff

[Pot95]. The general problem is that two distinct depths of the tree could have cardinalities with different properties, and this difference can be exposed by further path moves.

Duplicator will remedy this problem by making not a small jump down one stem, but an "exaggerated jump" that moves down λ stems to a place that looks locally (on its stem) isomorphic to the place where Duplicator has played. How do we ensure that a locally similar place exists? Duplicator will make sure that in all cases where Spoiler can execute this strategy, the currently-played paths below the fooling point begin with a large segment of the witness path. Duplicator can guarantee this on path moves because if π is not long, there is no need to perform fooling at all. On the other hand, if π is long, Duplicator can play a path that has a long regular structure at the top, which allows him to perform the exaggerated jump.

1.6.4 Duplicator's Strategy in Detail

We now formalize Duplicator's winning strategy in order to prove part (ii) of Lemma 1.10. We start with the notion of 'similar levels'.

Definition 1.16. *Let (μ, λ) be the period of $(T_k)_{k\in\mathbb{N}}$ and $(S_k)_{k\in\mathbb{N}}$. We define $\doteq \subseteq \mathbb{N}\times\mathbb{N}$ by*

$$n \doteq m \quad \text{iff} \quad n, m \geq \mu \text{ and } n = m + \lambda i \text{ for some } i \in \mathbb{Z}.$$

The following is an immediate consequence of the definition of $\doteq$ and Proposition 1.2.

Fact 1.17. *Duplicator has a winning strategy for the $CTL^*_{\triangleleft\triangleright}(\mathrm{ud} \leq \tilde{u}_\triangledown, \mathrm{other} \leq \tilde{o})$-game on T_i, T_j if $i \doteq j$ and the roots are selected. The same holds for S_i, S_j.*

Notation A node x in $\tilde{T}$ or $\tilde{S}$ is on *level* i if its enclosing subtree is isomorphic to either T_i or S_i. Two nodes x, y in $\tilde{T}$ or $\tilde{S}$ are on *similar levels* if x is on level i, y is on level j, and $i \doteq j$. In this case we write $x \doteq y$. We denote by plateau-depth(x) the plateau-depth of x on the stem that contains x. Given a tree T, we denote by $T[x]$ the subtree of T rooted at x.

We noted in the previous section that Duplicator will have to maintain similar positions throughout the game. If she can, in addition, achieve a position in which the selected nodes are locally isomorphic on their stems, and both nodes are not on the witness path then she can win the $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{ud}_\triangledown \leq u_\triangledown; \mathrm{other} \leq o)$-game if it proceeds strictly downwards.

Claim 1. *Let x and y be nodes in $\tilde{T}$ and $\tilde{S}$ respectively, that are on similar levels, have the same plateau depth, the same label, and are both not on witness paths. Then Duplicator can win the $CTL^*_{\triangleleft\triangleright}(\mathrm{ud} \le \tilde{u}_\triangledown, \mathrm{other} \le \tilde{o})$-game on the subtrees $\tilde{T}[x], \tilde{S}[y]$ of $\tilde{T}, \tilde{S}$ rooted at x, y respectively.*

Proof of Claim 1. Without loss of generality we assume that x is positive and y is negative. Hence the enclosing subtree of x is T_i for some i and the enclosing subtree of y is S_j for some j. There is a unique node x' in S_i such that the path from the root of S_i to x' is isomorphic to the path from the root of T_i to x. Note that as x is not on any witness path, the subtree $S_i[x']$ of S_i rooted at x' is isomorphic to the subtree $T_i[x]$ of T_i rooted at x. In addition, Duplicator has a winning strategy for the $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{ud} \le \tilde{u}_\triangledown, \mathrm{other} \le \tilde{o})$ game on $S_i[x']$ and $S_j[y]$ by Fact 1.17. The lemma follows because $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{ud} \le \tilde{u}_\triangledown, \mathrm{other} \le \tilde{o})$ is closed under isomorphism. □

We have seen that Duplicator has a simple winning strategy from certain positions in $\tilde{T}$ and $\tilde{S}$. Unfortunately, if the roots of $\tilde{T}$ and $\tilde{S}$ are selected, neither Fact 1 nor Claim 1 applies. The following lemma is concerned with this situation – that is, it implies that Duplicator has a winning strategy for the $\mathrm{CTL}(\mathrm{ud}_\triangledown \le \tilde{u}_\triangledown; \mathit{other} \le \tilde{o})$ game on $\tilde{T}, \tilde{S}$ if the roots are selected. Hence it proves part (ii) of Lemma 1.10.

Claim 2. *Let $p_\triangleleft, p_\triangledown, p_\triangleright \le \tilde{o}$. Duplicator has a winning strategy for the $CTL^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleleft \le p_\triangleleft, \mathrm{pd}_\triangledown \le p_\triangledown, \mathrm{pd}_\triangleright \le p_\triangleright, \mathrm{ud}_\triangledown \le \tilde{u}_\triangledown; \mathrm{other} \le \tilde{o})$-game on $\tilde{T}, \tilde{S}$ if the selected nodes x, y satisfy*

1. *x, y have the same label,*
2. *x, y are on similar levels,*
3. *x, y have the same plateau-depth,*
4. *$\mathrm{level}(x) > k$ and $\mathrm{level}(y) > k$ where $k = (\lambda(\tilde{u}_\triangledown + \tilde{o}) + 1)(p_\triangleleft + p_\triangledown + p_\triangleright) + \mu$.*

Proof of Claim 2. We prove the claim by induction on $p_\triangleleft + p_\triangledown + p_\triangleright$. The base case $p_\triangleleft + p_\triangledown + p_\triangleright = 0$ holds because x and y have the same label by Condition 1. For the induction step, we fix $p_\triangleleft, p_\triangledown, p_\triangleright$ such that $p_\triangleleft + p_\triangledown + p_\triangleright > 0$. If Spoiler plays a horizontal move, then Duplicator's strategy is trivial. For downward moves we distinguish several cases.

Case 1. Spoiler plays a downward move on $\tilde{T}$ Assume that Spoiler chooses a downward path π that is rooted at x in $\tilde{T}$. As noted above, Duplicator's strategy depends on the length of the prefix of π that is on a witness path. Therefore let $\pi_1, \dots, \pi_n$ be a partition of π such that $\pi_1 \dots \pi_{n-1}$ is a maximal prefix of π on a witness path, and for each $i \leq n-1$, π_i is contained in exactly one stem. We will call Spoiler's move a *short move* if $n \leq \lambda(\tilde{u}_\triangledown + \tilde{o}) + 1$ and a *long move* otherwise. Duplicator's strategy is different for the two kinds of moves.

Case 1.1. Short downward move on $\tilde{T}$ As described in the previous subsection, Duplicator has an easy strategy in this case. Duplicator first chooses a prefix $\tau_1 \cdot \dots \cdot \tau_{n-1}$ of the full path $\tau = \tau_1 \cdot \dots \cdot \tau_n$ that she will choose in the game. She picks this prefix such that τ_i has the same labeling as π_i for $1 \leq i \leq n-1$ (note that any downward path in T and S is determined by its labeling). The τ_i exists by Condition 4.

To determine τ_n, recall that π departs from the witness path on the root of π_n. There are two ways in which a path can depart from a witness path: above the witness node or on the sibling of the witness node. In the first case the root x_n of π_n is labelled b and in the second case x_n is labelled a. In both cases we define the root y_n of τ_n to be the child of the leaf of τ_{n-1} that has the same label x_n.

- If π departs from the witness path on the sibling of the witness node, then both x_n and y_n are not on *any* witness path. Hence it follows from Claim 1 that Duplicator wins the $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{ud}_\triangledown \leq \tilde{u}_\triangledown; \mathrm{other} \leq \tilde{o})$-game on $\tilde{T}[x_n], \tilde{S}[y_n]$.
- If π departs from the witness path above the witness node, then the subtrees rooted at x_n and y_n have the same polarity. As x_n and y_n are on similar levels, Duplicator can win the $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{ud}_\triangledown \leq \tilde{u}_\triangledown; \mathrm{other} \leq \tilde{o})$ on $\tilde{T}[x_n], \tilde{S}[y_n]$ by Fact 1.17.

In both cases, Duplicator can use her winning strategy for the $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{ud}_\triangledown \leq \tilde{u}_\triangledown; \mathrm{other} \leq \tilde{o})$-game on $\tilde{T}[x_n], \tilde{S}[y_n]$ to determine a path τ_n rooted at y_n such that she has a winning strategy for the $\mathrm{LTL}(\mathrm{ud} \leq \tilde{u}_\triangledown, \mathrm{ned} \leq \tilde{o})$-game on π_n, τ_n.

The LTL-game after short downward moves on $\tilde{T}$ We now describe Duplicator's strategy for the LTL game on π, τ. Duplicator can the play isomorphically on $\pi_0 \dots \pi_{n-1}$ and $\tau_0 \dots \tau_{n-1}$ as these paths are isomorphic (that is, she picks the n-th node whenever Spoiler picks the n-th node in the other path). If Spoiler chooses a node in π_n or τ_n then Duplicator can use her winning strategy for the LTL$(\mathrm{ud} \leq \tilde{u}_\triangledown, \mathrm{ned} \leq \tilde{o})$ on π_n, τ_n. These strategies can be composed to derive a winning strategy for π, τ. This composed strategy has the property that if π', τ' is an intermediate position of the path-game, then either the root of π' is in $\pi_0 \cdot \ldots \cdot \pi_{n-1}$ and the root of τ' is in $\tau_0 \cdot \ldots \cdot \tau_{n-1}$ or the root of π' in π_n and the root of τ' is in τ_n.

It remains to show that any intermediate position π', τ' of the path-game on π, τ is a winning position for Duplicator in the $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleleft \leq p_\triangleleft, \mathrm{pd}_\triangledown \leq p_\triangledown - 1, \mathrm{pd}_\triangleright \leq p_\triangleright, \mathrm{ud}_\triangledown \leq \tilde{u}_\triangledown; \mathrm{other} \leq \tilde{o})$-game. For positions where the root of π' is in $\pi_0 \cdot \ldots \cdot \pi_{n-1}$ and the root of τ' is in $\tau_0 \cdot \ldots \cdot \tau_{n-1}$ it is easy to see that the Conditions 2, 3 and 1 of Claim 2 are true. Condition 4 is satisfied as $\pi_0 \cdot \ldots \cdot \pi_{n-1}$ and $\tau_0 \cdot \ldots \cdot \tau_{n-1}$ are 'short' (that is, $n \leq \lambda(u_\triangledown + o) + 1$) and hence the levels of the roots of both π' and τ' are sufficiently high up in the trees. Thus Duplicator wins on these positions by induction. The same argument applies if the roots of π_n and τ_n are selected as the new position. Now consider the case that the root of π' is a descendant of x_n and the root of τ' is a descendant of y_n. In this case the subsequent game can never leave the trees $\tilde{T}[x_n], \tilde{S}[y_n]$ (this is because the game can only move horizontally and downwards – but never upwards). Then the new position is a winning position for Duplicator because by construction, she has a winning strategy for the $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{ud}_\triangledown \leq \tilde{u}_\triangledown; \mathrm{other} \leq \tilde{o})$-game on $\tilde{T}[x_n], \tilde{S}[y_n]$.

Case 1.2. Long downward move on $\tilde{T}$ Recall that in this case Spoiler chooses a path $\pi = \pi_1 \dots \pi_n$ in $\tilde{T}$ such that $n > \lambda(u_\triangledown + o) + 1$ and $\pi_1 \dots \pi_{n-1}$ is a maximal prefix of π on a witness path and each π_i with $i \leq n-1$ is contained in exactly one stem. The idea is that Duplicator picks a path $\tau = \tau_1 \dots \tau_m$ for some large m that is smaller than n such that τ_1 is isomorphic to π_1, τ_2 is similar (but undetectably different) to π_2, the path $\tau_3 \dots \tau_{m-1}$ is a long sequence of segments on the witness path that is isomorphic to $\pi_3 \dots \pi_{m-1}$, and τ_m is chosen by induction in such a way that it is indistinguishable from the path $\pi_m \dots \pi_n$ in a suitable LTL game (see Figure 1.7). Formally

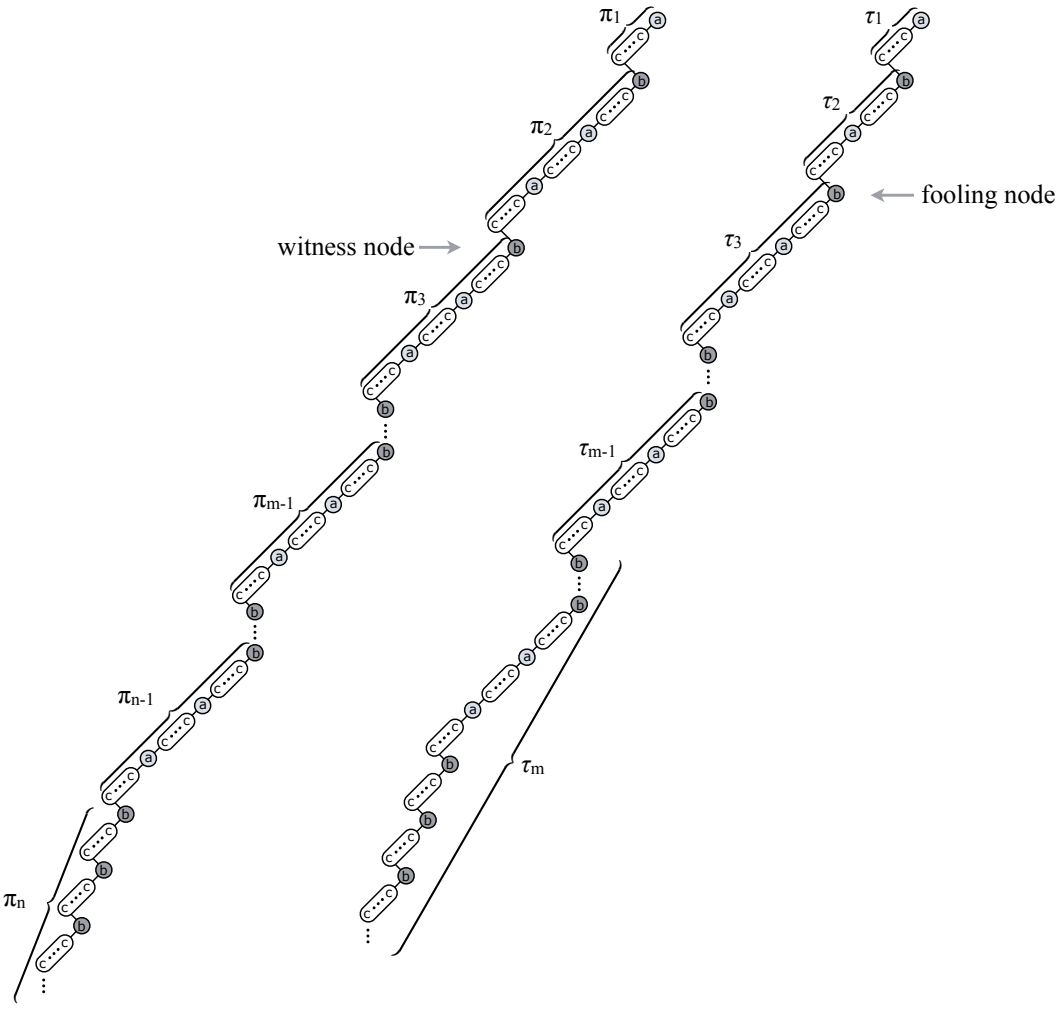


Figure 1.7: The paths chosen by Spoiler and Duplicator in Case 1.2.

- τ_1 is isomorphic to π_1.
- τ_2 is a prefix of a stem that departs from its stem on the upper fooling node. Note that τ_2 is labelled $bc^{\tilde{o}}(ac^{\tilde{o}})^{u_\triangledown - 1}$. This is Duplicator's 'fooling' move.
- $m = \lambda(u_\triangledown + o) + 1$.
- τ_i is isomorphic to π_i for all $3 \leq i \leq m - 1$.

The τ_i with $i \leq m$ exist by Condition 4.

We still need to specify τ_m to complete the description of Duplicator's path move. Let x_m be the root of π_m and let y_m be the child of the leaf of τ_{m-1} that has the same label as x_m. Observe that due to Duplicator's fooling

move, the roots of x_3 and y_3 of π_3 and τ_3 have positive polarity. As $\tau_3 \dots \tau_{m-1}$ is isomorphic to $\pi_3 \dots \pi_{m-1}$ it follows that x_m and y_m have the same polarity. In addition, as the positions x, y at the beginning of the current round were on similar levels (Condition 2) it follows that x_m and y_m are on similar levels. Finally, as both positions are the roots of their respective stems, they have the same plateau depth. Hence it follows from Fact 1.17 that Duplicator has a winning strategy for the $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{ud}_\triangledown \le \tilde{u}_\triangledown; \mathrm{other} \le \tilde{o})$-game on $\tilde{T}[x_m], \tilde{S}[y_m]$. This allows Duplicator to determine a path τ' rooted at y_m such that (a) she wins the $\mathrm{LTL}(\mathrm{ud} \le \tilde{u}_\triangledown, \mathrm{ned} \le \tilde{o})$-game on $\pi_m \dots \pi_n$ and τ' and (b) every intermediate position of the LTL-game is a winning position for the $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{ud}_\triangledown \le \tilde{u}_\triangledown; \mathrm{other} \le \tilde{o})$-game on $\tilde{T}[x_m], \tilde{S}[y_m]$. Duplicator chooses $\tau_m = \tau'$, which completes the description of $\tau = \tau_1 \dots \tau_m$. Figure 1.7 shows the paths π and τ.

The LTL game after long downward moves on $\tilde{T}$ Let π, τ be the paths chosen in the current round as described above and let $p_\triangleleft, p_\triangledown, p_\triangleright$ be the numbers fixed at the beginning of the proof of Claim 2. Then the following subclaim implies Claim 2.

> Subclaim 2.1. *Duplicator can win the LTL$(\mathrm{ud} \le \tilde{u}_\triangledown, \mathrm{ned} \le \tilde{o})$-game on π and τ. In addition she can play in such a way that any intermediate position is a winning position for her in the CTL$^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleleft \le p_\triangleleft, \mathrm{pd}_\triangledown \le p_\triangledown - 1, \mathrm{pd}_\triangleright \le p_\triangleright, \mathrm{ud}_\triangledown \le \tilde{u}_\triangledown; \mathrm{other} \le \tilde{o})$-game on $\tilde{T}, \tilde{S}$.*

We will prove Subclaim 2.1 by defining winning strategies for Duplicator in three $\mathrm{LTL}(\mathrm{ud} \le \tilde{u}_\triangledown, \mathrm{ned} \le \tilde{o})$-games: one on π_1 and τ_1, one on $\pi_2 \dots \pi_{m-1}$ and $\tau_2 \dots \tau_{m-1}$, and one on $\pi_m \dots \pi_n$ and τ_m. All these strategies will have the property that intermediate positions are winning positions for Duplicator in a suitable CTL-game. As in Case 1.1. these strategies can be combined to prove Subclaim 2.1.

The strategy for the game π_1, τ_1 is easy: as the paths are isomorphic, Duplicator will play isomorphically (that is, if Spoiler picks the i-th node in either path, then Duplicator will pick the i-th node in the other path). It follows from the hypothesis of Claim 2 that any intermediate position satisfies all the Conditions of Claim 2 for $p_\triangleleft, p_\triangledown - 1, p_\triangleright, \tilde{u}_\triangledown, \tilde{o}$.

A suitable strategy for Duplicator in the game on $\pi_m \dots \pi_n$ and τ_m exists by definition of τ_m.

It remains to show that Duplicator has a winning strategy for the game on $\pi_2 \dots \pi_{m-1}$ and $\tau_2 \dots \tau_{m-1}$. Observe that the paths $\pi_2 \dots \pi_{m-1}$ and $\tau_2 \dots \tau_{m-1}$ are 'long' versions of the paths used in Corollary 1.5. A stem corresponds to a staircase, and hence the *top-depth* of a node corresponds to its distance to the place where the path leaves the stem. A block of the form $c^{\tilde{o}}a$ within a stem corresponds to a plateau, and hence the *plateau-depth* is the position within such a block. Let the *end-depth* of a path π be the number of b labelled nodes on π – that is, the number of stems remaining in the path.

We show that Duplicator can maintain an invariant similar to the one in Claim 1 in the proof of Claim 1.4. There are two differences. First, the paths $\pi_1 \dots \pi_m$ and $\tau_1 \dots \tau_m$ are finite, and hence Duplicator must make sure that Spoiler cannot exploit that the end of the paths might have different distances from the selected nodes. In addition, the position at the end of the path game must satisfy the conditions of Claim 2, in particular they must be on similar levels. This will have an impact on Duplicator's strategy on F- and U-moves: if Duplicator were to play according to the strategy described in Claim 1 in the proof of Lemma 1.4, then the positions could end up being exactly one stem apart, and hence on *non*-similar levels. Therefore, Duplicator will jump to the next position that is locally isomorphic *and* on a similar level to the position that Spoiler chose (whereas in the proof of Claim 1 of Lemma 1.4 it was sufficient to jump to the next locally isomorphic position). After this 'exaggerated jump', Duplicator will be able to play according to a simpler strategy. Subclaim 2.1.1 deals with this simple strategy. The proof of Subclaim 2.1.2 will describe the slightly more involved strategy before the exaggerated jump.

SUBCLAIM 2.1.1. *[Post-Jump Strategy] Let $u_\triangledown \leq \tilde{u}_\triangledown$ and $o \leq \tilde{o}$. Then Duplicator has a winning strategy for the LTL$(\mathrm{ud} \leq u_\triangledown, \mathrm{ned} \leq o)$-game on $\pi_2 \dots \pi_{m-1}$ and $\tau_2 \dots \tau_{m-1}$, if the selected suffixes $\hat{\pi}$ and $\hat{\tau}$ satisfy:*

1. *the roots of $\hat{\pi}$ and $\hat{\tau}$ have the same label.*
2. *$\hat{\pi}$ and $\hat{\tau}$ are on similar levels.*
3. *$\hat{\pi}$ and $\hat{\tau}$ have the same plateau-depth.*

4. *$\hat{\pi}$ and $\hat{\tau}$ have the same top-depth.*
5. *Either $\hat{\pi}$ and $\hat{\tau}$ have the same end-depth or* $|\text{end-depth}(\hat{\pi}) - \text{end-depth}(\hat{\tau})| = \lambda$, $\text{end-depth}(\hat{\pi}) \geq \lambda(u_\triangledown + o)$, *and* $\text{end-depth}(\hat{\tau}) \geq \lambda(u_\triangledown + o)$.

In addition, Duplicator can play in such a way that each intermediate position is a winning position for her in the $CTL^*_{\triangleleft\triangleright}(\text{pd}_\triangleleft \leq p_\triangleleft, \text{pd}_\triangledown \leq p_\triangledown - 1, \text{pd}_\triangleright \leq p_\triangleright, \text{ud}_\triangledown \leq \tilde{u}_\triangledown; \text{other} \leq \tilde{o})$-game.

Note that it follows from Conditions 3 and 4 that the roots of $\hat{\pi}$ and $\hat{\tau}$ have the same label.

Proof of Subclaim 2.1.1. The proof will be similar to the proof of Claim 1 of Lemma 1.4, but slightly simpler. We use induction on $u_\triangledown + o$. The base case holds as by Condition 1 the roots of $\hat{\pi}$ and $\hat{\tau}$ have the same label. For the inductive step we assume that $u_\triangledown + o > 0$.

X-move If Spoiler plays a X-move, then Duplicator's strategy is determined by the rules of the game. It is easy to check that the conditions of Subclaim 2.1.1 are maintained: This is trivial for Conditions 2, 3, and 4. Condition 5 is maintained because it depends on o. It follows from Claim 2 that the new position is a winning position for Duplicator in the $\text{CTL}^*_{\triangleleft\triangleright}(\text{pd}_\triangleleft \leq p_\triangleleft, \text{pd}_\triangledown \leq p_\triangledown - 1, \text{pd}_\triangleright \leq p_\triangleright, \text{ud}_\triangledown \leq \tilde{u}_\triangledown; \text{other} \leq \tilde{o})$-game.

U-move First consider the case where Spoiler chooses to play on $\hat{\pi}$. In the first half move he selects a suffix $\hat{\pi}^n$ of $\hat{\pi}$. If $\hat{\pi}^n$ is isomorphic to a suffix $\hat{\tau}^m$ of $\hat{\tau}$ then Duplicator chooses $\hat{\tau}^m$ in his first half move. Note that if this is not the case then $\text{end-depth}(\hat{\pi}) - \text{end-depth}(\hat{\tau}) = \lambda$ by Condition 5. In particular, $\hat{\tau}$ must be a suffix of $\hat{\pi}$ and $\hat{\pi}$ has exactly $\lambda(u_\triangledown + o)$ positions more than $\hat{\tau}$. Hence Spoiler has skipped at most $\lambda(u_\triangledown + o)$ positions in his first half move. Duplicator will skip the same number of positions as Spoiler did. It is easy to check that the conditions of Subclaim 2.1.1 are maintained for $u_\triangledown - 1, o$.

In the second half move Spoiler picks a suffix $\hat{\tau}_{m'}$ such that $0 < m' \leq m$. It is easy to check that there is a suffix $\hat{\pi}^{n'}$ such that $m - m' = n - n'$. Again, the conditions of the claim are preserved and

its truth follows from the induction. It follows from Claim 2 that the new position is a winning position for her in the $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleleft \leq p_\triangleleft, \mathrm{pd}_\triangledown \leq p_\triangledown - 1, \mathrm{pd}_\triangleright \leq p_\triangleright, \mathrm{ud}_\triangledown \leq \tilde{u}_\triangledown; \mathrm{other} \leq \tilde{o})$-game.

***F*-move** Duplicator can use exactly the same strategy as in the first half of the U-move. This is possible because Condition 5 depends on o. As above, the second statement of Subclaim 2.1.1 follows from Claim 2. □

Note that the top-depth of the root of $\pi_2 \dots \pi_{m-1}$ is different from the top-depth of $\tau_2 \dots \tau_{m-1}$ (recall that π_2 is labelled $bc^{\tilde{o}}(ac^{\tilde{o}})^u$ while τ_2 is labelled $bc^{\tilde{o}}(ac^{\tilde{o}})^{\tilde{u}_\triangledown - 1}$). Hence in the beginning of the CTL-game on $\pi_2 \dots \pi_{m-1}$ and $\tau_2 \dots \tau_{m-1}$ (basically before Duplicator has executed his exaggerated jump) Duplicator will use a different strategy.

Subclaim 2.1.2. *[Pre-Jump Strategy] Let $u_\triangledown \leq \tilde{u}_\triangledown$ and $o \leq \tilde{o}$. Then Duplicator has a winning strategy for the LTL$(\mathrm{ud} \leq u_\triangledown, \mathrm{ned} \leq o)$-game on $\pi_2 \dots \pi_{m-1}$ and $\tau_2 \dots \tau_{m-1}$, if the selected suffixes $\hat{\pi}, \hat{\tau}$ satisfy:*

1. *the roots of $\hat{\pi}$ and $\hat{\tau}$ have the same label.*
2. *$\hat{\pi}$ and $\hat{\tau}$ have the same plateau-depth.*
3. $|\text{top-depth}(\hat{\pi}) - \text{top-depth}(\hat{\tau})| \leq 1$.
4. *If $\hat{\pi}$ and $\hat{\tau}$ have different top-depths then*
   (a) $\text{top-depth}(\hat{\pi}) \geq u_\triangledown$ *and* $\text{top-depth}(\hat{\tau}) \geq u_\triangledown$ *and*
   (b) *if* $\text{top-depth}(\hat{\pi}) = u_\triangledown$ *or* $\text{top-depth}(\hat{\tau}) = u_\triangledown$ *then* $\text{plateau-depth}(\hat{\pi}) > o$ *(and hence* $\text{plateau-depth}(\hat{\tau}) > o$*).*
5. *$\hat{\pi}$ and $\hat{\tau}$ are on similar levels.*
6. *Either $\hat{\pi}$ and $\hat{\tau}$ have the same end-depth or* $|\text{end-depth}(\hat{\pi}) - \text{end-depth}(\hat{\tau})| = \lambda$, $\text{end-depth}(\hat{\pi}) \geq \lambda(u_\triangledown + o)$, *and* $\text{end-depth}(\hat{\tau}) \geq \lambda(u_\triangledown + o)$.

In addition, Duplicator can play in such a way that each intermediate position is a winning position for her in the $CTL^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleleft \leq p_\triangleleft, \mathrm{pd}_\triangledown \leq p_\triangledown - 1, \mathrm{pd}_\triangleright \leq p_\triangleright, \mathrm{ud}_\triangledown \leq \tilde{u}_\triangledown; \mathrm{other} \leq \tilde{o})$-game.

Proof of Subclaim 2.1.2. The proof is by induction on $u_\triangledown + o$. The base case $u_\triangledown + o = 0$ follows from the conditions 2 and 4. We now assume that $u_\triangledown + o > 0$. Duplicator will use the strategy from the proof of Claim 1 in the proof of Lemma 1.4 on X- and U-moves. It is easy to see that the conditions of Subclaim 2.1.2 hold after Duplicator's move.

Now assume that Spoiler plays an F-move on $\hat{\pi}$. That is, that he selects a suffix $\hat{\pi}^n$ of $\hat{\pi}$. In this case Duplicator cannot use the strategy described in the proof of Claim 1 of Lemma 1.4, because if she did the selected subpaths might not be on similar levels anymore. Hence Duplicator selects the largest suffix $\hat{\tau}^m$ of $\hat{\tau}$ such that $\hat{\pi}$ and $\hat{\tau}^m$ have the same plateau-depth, the same top-depth, and are on similar levels. Clearly such an m can be found in such a way that the end-depth decreases by at most λ. The new position satisfies the conditions of Subclaim 2.1.1 for $u_\triangledown, o-1$, and hence Duplicator wins by induction □

Case 2. **Downward move on** S This case is very similar to Case 1. The only difference is that if Spoiler plays a long move, then Duplicator's 'fooling' move is to play a path that departs from the witness path on the lower fooling node on the second stem. We omit the details. □

1.6.5 The Finite Case

Part (ii) of Lemma 1.10 can also be shown for finite trees. We define trees $\tilde{T}^{\text{fin}}$ and $\tilde{S}^{\text{fin}}$ to be obtained from $\tilde{T}$ and $\tilde{S}$ by pruning the stems at some point. In particular, each stem in $\tilde{T}^{\text{fin}}$ and $\tilde{S}^{\text{fin}}$ spells out the finite word $b(c^*a)^{\tilde{o}(\tilde{u}+\tilde{o})+\tilde{u}}$. The witness node and the fooling nodes are as in $\tilde{T}$ and $\tilde{S}$.

The *stem-depth* of a node x is the number of a labelled nodes on a downward fullpath rooted at x that contains no right siblings. The stem-depth of a path π is the stem-depth of its root.

Lemma 1.10 for finite trees trees follows from the following claim. It is basically the finite tree version of Claim 2 in the proof of Lemma 1.10.

Claim 3. *Let* $p_\triangleleft, p_\triangledown, p_\triangleright \leq \tilde{o}$. *Duplicator has a winning strategy for the* $CTL^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleleft \leq p_\triangleleft, \mathrm{pd}_\triangledown \leq p_\triangledown, \mathrm{pd}_\triangleright \leq p_\triangleright, \mathrm{ud}_\triangledown \leq \tilde{u}_\triangledown; \mathrm{other} \leq \tilde{o})$*-game on* $\tilde{T}^{\text{fin}}, \tilde{S}^{\text{fin}}$ *if the selected nodes* x, y *satisfy*

1. *x, y have the same label,*
2. *x, y have the same plateau-depth,*
3. *x, y are on similar levels,*
4. $\mathrm{level}(x) > k$ *and* $\mathrm{level}(y) > k$ *where* $k = (\lambda(\tilde{u}_\triangledown + \tilde{o}) + 1)(p_\triangleleft + p_\triangledown + p_\triangleright) + \mu$,
5. $\mathrm{sd}(x) \geq p_\triangledown(\tilde{u}_\triangledown + \tilde{o})$ *and* $\mathrm{sd}(y) \geq p_\triangledown(\tilde{u}_\triangledown + \tilde{o})$.

Proof of Claim 3. Duplicator can use a similar strategy as in the infinite case. In particular, whenever Spoiler picks a path that departs from the stem, then Duplicator can use the strategy described in Claim 2 of Lemma 1.10 – using this strategy she can preserve that conditions of Claim 3. But Spoiler might try to expose that the stem depths of x and y are different. To do so, he might select the path π that consists only only of the stem. In this case, Duplicator will also choose the path τ in the other tree that only consists of the stem. The following Subclaim shows that in this case Duplicator can win the $\mathrm{LTL}(\mathrm{ud} \leq \tilde{u}_\triangledown, \mathrm{ned} \leq \tilde{o})$-game on π and τ, while maintaining the conditions of Claim 3. The proof of Subclaim 1 concludes the proof of Claim 3, and hence the proof of Theorem 1.3 for finite trees.

SUBCLAIM 1. *Let $u_\triangledown \leq \tilde{u}_\triangledown$ and $o \leq \tilde{o}$. Duplicator can win the LTL$(\mathrm{ud} \leq u, \mathrm{ned} \leq o)$-game on π and τ if the selected suffixes $\tilde{\pi}$ and $\tilde{\tau}$ of π and τ satisfy:*

1. *the roots of $\tilde{\pi}$ and $\tilde{\tau}$ have the same label.*
2. *$\tilde{\pi}$ and $\tilde{\tau}$ have the same plateau-depth.*
3. $|\text{top-depth}(\tilde{\pi}) - \text{top-depth}(\tilde{\tau})| \leq 1$.
4. *If $\tilde{\pi}$ and $\tilde{\pi}$ have different top-depths then*
   (a) $\text{top-depth}(\tilde{\pi}) \geq (p_\triangledown - 1)\tilde{u}_\triangledown + u_\triangledown$ *and* $\text{top-depth}(\tilde{\tau}) \geq (p_\triangledown - 1)\tilde{u}_\triangledown + u_\triangledown$
   (b) *if* $\text{top-depth}(\tilde{\pi}) = (p_\triangledown - 1)\tilde{u}_\triangledown + u_\triangledown$ *or* $\text{top-depth}(\tilde{\tau}) = (p_\triangledown - 1)\tilde{u}_\triangledown + u_\triangledown$
   then $\text{plateau-depth}(\tilde{\pi}) \geq (p_\triangledown - 1)\tilde{o} + o$
   (and hence $\text{plateau-depth}(\tilde{\tau}) \geq (p_\triangledown - 1)\tilde{o} + o$*).*
5. *If $\tilde{\pi}$ and $\tilde{\pi}$ have different bottom-depths then*
   (a) $\text{bottom-depth}(\tilde{\pi}) \geq (p_\triangledown - 1)\tilde{u}_\triangledown + u_\triangledown$
   and $\text{bottom-depth}(\tilde{\tau}) \geq (p_\triangledown - 1)\tilde{u}_\triangledown + u_\triangledown$

(b) if $\text{bottom-depth}(\tilde{\pi}) = (p_\triangledown - 1)\tilde{u}_\triangledown + u_\triangledown$
or $\text{bottom-depth}(\tilde{\tau}) = (p_\triangledown - 1)\tilde{u}_\triangledown + u_\triangledown$
then $\text{inverse-plateau-depth}(\tilde{\pi}) \geq (p_\triangledown - 1)\tilde{o} + o$
(and hence $\text{inverse-plateau-depth}(\tilde{\tau}) \geq (p_\triangledown - 1)\tilde{o} + o$*).*

6. $\text{sd}(x) \geq p_\triangledown(\tilde{u}_\triangledown + \tilde{o})$ *and* $\text{sd}(y) \geq p_\triangledown(\tilde{u}_\triangledown + \tilde{o})$.

In addition every intermediate position of the LTL$(\text{ud} \leq u, \text{ned} \leq o)$*-game on* π *and* τ *is a winning position for Duplicator for the* $CTL^*_{\triangleleft\triangleright}(\text{pd}_\triangleleft \leq p_\triangleleft, \text{pd}_\triangledown \leq p_\triangledown, \text{pd}_\triangleright \leq p_\triangleright, \text{ud}_\triangledown \leq \tilde{u}_\triangledown; \text{other} \leq \tilde{o})$*-game on* $\tilde{T}^{\text{fin}}, \tilde{S}^{\text{fin}}$.

Proof of Subclaim 1. The proof is similar to the proof of Claim 1 in Lemma 1.4. We use an induction on $u_\triangledown + o$. If $u_\triangledown + o = 0$ then the claim follows as by Condition 1 the roots of $\tilde{\pi}$ and $\tilde{\tau}$ have the same label. For the inductive case we assume that $u_\triangledown + o > 0$. Duplicator's strategy depends on the kind of move that Spoiler plays first.

X-move On X-moves Duplicator's strategy is determined by the rules of the game. We omit the calculations that show that X-moves preserve the invariant.

F-move Assume that Spoiler plays an F-move on $\tilde{\pi}$, in which he selects a suffix $\tilde{\pi}'$ of $\tilde{\pi}$. If $\tilde{\tau}$ contains a suffix $\tilde{\tau}'$ that is isomorphic to $\tilde{\pi}'$ then Duplicator selects $\tilde{\tau}'$. Otherwise $\tilde{\pi}'$ must contain $\tilde{\tau}$ as a suffix. In this case it is Duplicator's goal to jump the fewest number of positions while still preserving the invariant. Hence she selects the largest suffix $\tilde{\tau}'$ of $\tilde{\tau}$ such that $\tilde{\pi}'$ and $\tilde{\tau}'$ have the same plateau-depth and top-depth. Again it is easy to verify that the conditions of Claim 3 are preserved.

U-moves Finally assume that Spoiler plays an U-move. In the first half-move, Duplicator's strategy is similar to her strategy on F-moves. However, as in the proof of Claim 1 in Lemma 1.4, Duplicator must worry that Spoiler plays a very small move on the path with the smaller top-depth. If Duplicator were to use the strategy from the F-move then she might skip more positions than Spoiler did in

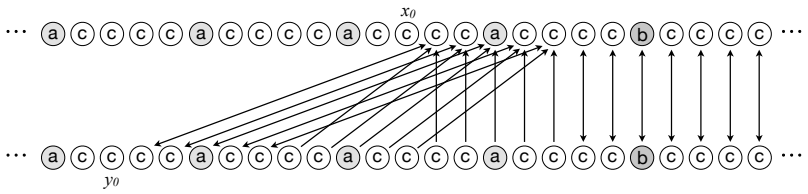


Figure 1.8: Duplicator's strategy if Spoiler plays an U-move. If the current position of the game is (x_0, y_0) and Spoiler picks a position z in either path in his first half move, then Duplicator picks the position z' in the other word, such that there is a arrow from z to z'.

his first half-move, and in the second half-move she might not have a suitable position to jump to (see the description of Duplicators U-move in Claim 1 in Lemma 1.4 for the discussion of this problem). Hence Duplicator will skip as many positions as Spoiler did if Spoiler skips at most $\tilde{o}+1$ positions. Otherwise she will follow her strategy for F-moves in the first half-move. It is easy to see that in the second half-move, Duplicator can always find a position, such that the conditions of Claim 3 are maintained. Duplicators strategy is shown in Figure 1.8. □

This concludes the proof of Claim 3, and hence the proof of Theorem 1.3 for finite trees. □

1.7 Conclusions

In this chapter we have investigated which direction-restricted XML query languages can be first-order complete. We began with the language $\mathrm{CTL}^*_{\triangleleft\triangleright}$ based on the whole of LTL going downwards and sideways, and we have shown that one cannot make due either with no untils in one of the horizontal directions or with a restriction on the number of untils vertically.

Several questions remain open. We need to investigate more thoroughly the relationship of the languages $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{ud}_\triangledown \leq u)$ with the queries of "bounded operator depth" mentioned by Bojańczyk [Boj08]. For the moment we note the following distinction: [Boj08] states that the queries of bounded operator depth cannot capture all languages of the form:

$$Q_n := \exists_\triangledown (a^n b)^*$$

In contrast, all these Q_n are contained at the lowest level of our hierarchy.

It is also unknown to which fragments of first-order logic the languages $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{ud}_\triangledown \leq u)$ correspond. In addition no algebraic characterization of $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{ud}_\triangledown \leq u)$ is known. Thérien and Wilke [TW04] have given an algebraic characterization of the LTL formulas of fixed until-depth on words, and have used this to show how to decide whether a formula is of a given until-depth. We do not know whether one can decide membership in $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{ud}_\triangledown \leq u)$ or in $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{pd}_\triangleright \leq p, \mathrm{ud}_\triangleright \leq u)$.

Part II

Data Word Languages

Chapter 2

How Small can Automata for Data Languages Get?

2.1 Introduction

We now consider an orthogonal generalization of word languages: Instead of tree languages over a finite alphabet, we consider word languages over an infinite alphabet. More precisely we consider *data word languages*, that is, languages of finite words in which the word positions have a value from an infinite alphabet and a label from a finite alphabet. These values can be used to model time stamps [AD94, HNSY92], process identifiers [Tze11, BH10], or attribute values in XML documents [NSV04].

Different models of automata which process data words have been proposed and studied in the literature (see the survey [Seg06]) These automata are generally extensions of finite automata that can store some bounded amount of information about the data values. This information is compared to the input value when the automaton transitions into a new configuration. The basic automata models for data language are pebble automata [NSV04], data automata (or equivalently class memory automata) [BMS+06], and finite memory automata [KF94]. Pebble automata use special markers to annotate positions in a data word. Data automata parse data words in two phases, with one phase applying a finite-state transducer to the input data word and another deciding acceptance on the grounds of a classification of the maximal sub-sequences consisting of the same data values (such a classification is usually specified in terms of membership in a regular language). Finally, finite memory automata, also called register automata, make use of a finite number of registers in order to store and eventually compare values in the processed data word.

One could hope that most of the fundamental results in standard automata theory (i.e., the theory of automata over a finite alphabet) can be transferred to the setting of words over infinite alphabets. However, prior work has shown that

many elementary closure and decision properties of finite automata are absent in the infinite-alphabet case. For example, the equivalence of the non-deterministic and deterministic variants of automata is known to fail for both finite memory automata and class memory automata [BS10]. While in the finite alphabet case the equivalence and universality problems for non-deterministic automata are decidable, for most of the infinite alphabet models they are not [NSV04, KF94].

Among several paradigmatic problems in automata theory, a crucial one, for both theoretical and practical reasons, is the minimization problem. Roughly speaking, it consists of determining the automaton-based representation that uses the "smallest space" for a given language. In the case of standard finite-state automata, minimal space usage is usually translated in terms of the minimum number of states. The well-known Myhill-Nerode theorem [HMU06] gives a canonical automaton for every regular language, which is minimal among deterministic finite automata accepting the same language. When dealing with more general models of automata, however, one may need to take into account different complexity measures at the same time, possibly yielding some tradeoffs between the amount of control state and the number of values/locations being stored.

In this chapter, we consider minimization for a particular model of register automata, which process finite words over an infinite alphabet. On the one hand, the class of memory automata we are dealing with (DMA, for short) is very similar to that of deterministic finite memory automata introduced in [KF94]. Our notion of register automaton is slightly more general in allowing to compare values both with respect to a fixed equality relation on values (as in the standard class of finite memory automata) and with respect to a fixed total ordering relation. For instance, our model of register automaton can recognize the language of all strictly-increasing finite sequences of natural numbers, which cannot be recognized by a finite memory automaton.

The first contribution of the chapter is an isolation of the ideal "minimal storage" for a DMA. This is formalized in terms of the *memorable values* for any word in the language – the set of values that must be stored at any point. Using this we can give a characterization of the class of languages recognized by some DMA, which closely resembles the Myhill-Nerode theorem. Precisely, we associate with each language L a suitable equivalence $\equiv_L$, using the memorable values, and we characterize the class of DMA-recognizable languages as the class of languages L for which $\equiv_L$ has finite index. We remark that a similar result, but restricted to the class of register automata that can only compare values with respect to equality,

has already appeared in [FK03]. In fact, our alternative characterization, besides relating equivalence to space-minimality, holds also for the larger class of DMA that compare values with respect to a fixed arbitrary total ordering relation.

As our second contribution, which stems directly from the previous characterization result, we show that the canonical DMA $\mathcal{A}_L$, which is obtained from a given language L when the corresponding equivalence $\equiv_L$ has finite index, satisfies a strong notion of minimality that takes into account both the number of control states and the number of values stored. Our results hold when the automaton can access either an identity relation on the data values or a linear order. Subsequent to our work, a generalization of our result has been given in [BKL11]. There the authors show that a unique minimal DMA exists also in the presence of a partial order on the data values. Finally, we give an effective means for *minimizing* a DMA, presenting a procedure that begins with an arbitrary DMA and produces the minimal equivalent.

The results of this chapter have been obtained in collaboration with Michael Benedikt and Gabriele Puppis and have been published in [BLP10c, BLP10b].

Organization. This chapter is organized as follows. After providing some background material in Section 2.2, we give an overview over our main results in Section 2.3. We then compare our results with previous work in Section 2.4. Section 2.5 proves the minimization result and Section 2.6 investigates the complexity of the minimization procedure. Section 2.7 concludes.

2.2 Preliminaries

In this section we provide some preliminaries for our work on data languages. We start by formally defining data words and data languages in Subsection 2.2.1. Then, in Subsection 2.2.2, we give a tour of several automata models that accept data word languages. We also review some known decidability and expressiveness results.

2.2.1 Basics on Data Languages

We fix, for the rest of this thesis, an infinite set D of *(data) values* and a finite set Σ of *labels*. Data words come in two different flavors: *plain (data) words* which are finite words over the alphabet D and *adorned (data) words* which are finite words over the alphabet $D \times \Sigma$.

Automata for data languages usually only have access to the values through a binary relation R on D, such as equality $\sim$ or some order $\leq$. Hence, a plain word w can be considered a structure $(\mathrm{Dom}(w), P_R, \leq)$ where $\mathrm{Dom}(w) = \{1, \ldots, |w|\}$, P_R is a predicate on $\mathrm{Dom}(w)$ such that $x\, P_R\, y$ iff $w(x)\, R\, w(y)$ for all positions x and y of w, and $\leq$ is the usual linear order on $\mathrm{Dom}(w)$. By misuse of notation we usually denote R and P_R by the same symbol. Adorned words can also be interpreted as appropriate structures with an additional labeling function lab_w from $\mathrm{Dom}(w)$ to Σ.

If we consider data words as structures in this way we can define isomorphisms between data words as isomorphisims between structures in the usual way. For example, two plain words w, w' over R are *isomorphic* (denoted $w \cong_R w'$) if $|w| = |w'|$ (and hence $\mathrm{Dom}(w) = \mathrm{Dom}(w')$) and $w(i)\ R\ w(j)$ iff $w'(i)\ R\ w'(j)$ for all pairs of positions $i, j \in \mathrm{Dom}(w)$. For adorned words we require in addition that $\mathrm{lab}_w(x) = \mathrm{lab}_{w'}(x)$ for all $x \in \mathrm{Dom}(w)$. Clearly $\cong_R$ is an equivalence relation and we call its classes $\cong_R$-*types*. We will drop the subscript from $\cong_R$ whenever R is clear from the context.

A *plain (adorned) language* over R is a set of plain (adorned) words over R that is closed under isomorphism – that is, if $w \in L$ and $w \cong_R w'$ then $w' \in L$. Given two words w and w', we write $w =_L w'$ if either both w and w' are in L, or both are not. We will consider both plain and adorned languages.

2.2.2 Automata for Data Languages

We now review several models of automata for defining data languages. We start with the model that we will focus on in the rest of this thesis: finite memory automata. We then turn to to other automata models, called pebble automata and data automata. Finally we compare the different models of automata for data words.

Finite Memory Automata

Finite memory automata were introduced by Kaminski and Francez in [KF94]. They extend finite automata by a finite set of registers, storing values from D. When processing a word from left to right, a finite memory automaton can compare the current input value with the values that are stored in the registers. Based on this information, the automaton can store the current input value in one of its registers, update its state and move one symbol to the right.

Our definition of a finite memory automaton differs from the definition given in [KF94]. Still, it is easy to see that both definitions accept the same class of languages. As in [KF94], we define them for plain languages.

Definition 2.1. *An* alternating k-register finite memory automaton *(AMA or FMA)* $\mathcal{A} = (Q, \mathrm{ar}, q_I, \bar{a}_I, F, T)$ *consists of*

- *a finite state set* Q *that is the disjoint union of two sets called existential states* $Q_\exists$ *and universal states* $Q_\forall$*,*
- *an arity function* $\mathrm{ar} : Q \to \mathbb{N}$*,*
- *an initial state* $q_I \in Q$ *with arity* 0 *and a set of finial states* $F \subseteq Q$*,*
- *a finite set of transitions* T *of the form* (p, α, E, q)*, where* $p, q \in Q$*,* α *is the* R*-type of a plain word of length* $\mathrm{ar}(p) + 1$*,* $E \subseteq \{1, \ldots, \mathrm{ar}(p) + 1\}$*, and* $\mathrm{ar}(q) = \mathrm{ar}(p) + 1 - |E|$*.*

A *configuration* of a k-register AMA $\mathcal{A}$ is a pair $(p, \bar{a})$ consisting of a state $p \in Q$ and a register content $\bar{a} \in D^{\mathrm{ar}(p)}$ containing pairwise distinct values. $\mathcal{A}$ can transition from a configuration $(p, \bar{a})$ to a configuration $(q, \bar{b})$ by consuming an input value a, denoted $(p, \bar{a}) \vdash_a (q, \bar{b})$, if there is a transition (p, α, E, q) in T such that $\bar{a}\,a$ (that is, the concatenation of $\bar{a}$ and a) is an element of the type α and $\bar{b}$ is obtained from $\bar{a}\,a$ by removing all symbols with index in E.

We enforce the following sanity conditions on every transition (p, α, E, q) of a k-AMA. First, we assume that E is non-empty whenever the arity of p is k (this is in order to guarantee that the length of the target memory content never exceeds k). Then, we assume that if α is a type of the form $[uava]_\cong$ for plain words u, v and value a, then E contains $|\alpha|$ (this is in order to guarantee that the target memory content contains pairwise distinct elements).

A run of a k-register AMA $\mathcal{A}$ on a plain word $w = a_1 \ldots a_n$ is an unordered tree T of depth $n + 1$ whose root is the initial configuration $(q_I, \bar{a}_I)$ and in which each node $(p, \bar{a})$ at level[1] i, has child $(q, \bar{b})$ iff $(p, \bar{a}) \vdash_{a_{i+1}} (q, \bar{b})$. A leaf node $x = (p, \bar{a})$ in a run T is accepting if p is an accepting state; a non-leaf node $x = (p, \bar{a})$ is accepting if

- p is an existential state and x has a child that is accepting; or
- p is a universal state and all children of x are accepting.

[1]The level of a node x in a tree is the number of edges between x and the root.

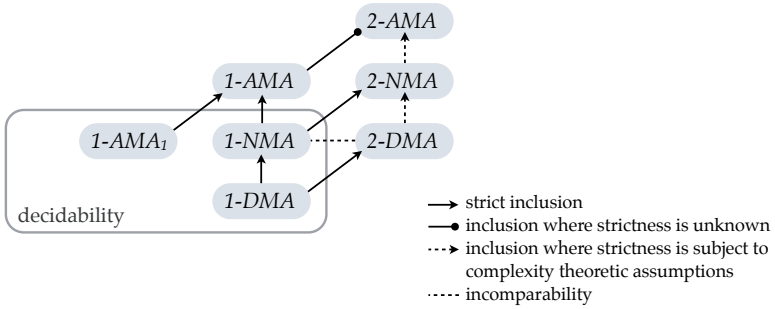


Figure 2.1: An overview over the expressiveness of finite memory automata.

A word w is accepted by $\mathcal{A}$ if $\mathcal{A}$ has a run T on w whose root is accepting. The *language accepted by* $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of words that are accepted by $\mathcal{A}$.

An AMA $\mathcal{A} = (Q_\exists \uplus Q_\forall, q_I, \bar{a}_I, F, T)$ is *non-deterministic* if $Q_\forall$ is empty and *universal* is $Q_\exists$ is empty. We denote the class of non-deterministic AMA by NMA. A NMA is *deterministic* if for each pair of transitions (p, α, E, q), (p', α', E, q') in T, if $p = p'$ and $\alpha = \alpha'$ then $E = E'$ and $q = q'$. We denote the class of deterministic finite memory automata by DMA.

Variants of Finite Memory Automata. Several variants of the basic model of FMA have been studied. Neven et al. compared the expressiveness of one-way, two-way, deterministic, non-deterministic, and alternating variants of FMA [NSV04]. We denote the two way variants of DMA, NMA, and AMA by 2-DMA, 2-NMA, and 2-AMA. If we want to emphasize that we are talking about an one way model we will write 1-DMA, 1-NMA, and 1-AMA to denote the respective variants of finite memory automata.

Most of the variants of finite memory automata differ in expressive power. Consider for example the language

$$L_1 = \{a_1 \dots a_n \in D^* \mid \text{there are } i, j \leq n \text{ such that } a_i = a_j\}.$$

A 1-NMA can easily accept this language by guessing positions i and j. However, one can show that no 1-DMA can accept this language. Intuitively, the reason is that a 1-DMA would have to store every symbol in the input word. It follows that that 1-NMA are strictly more expressive than 1-DMA.

Now consider the complement of L_1, that is the language $\bar{L}_1 = \{a_1 \dots a_n \in D^* \mid a_i \neq a_j \text{ for all } i, j \leq n\}$. Kaminski at. al. show that $\bar{L}_1$ cannot be accepted by a 1-NMA, whereas $\bar{L}_1$ can be accepted by a 2-DMA. This separates 1-NMA from 2-DMA, and shows that 1-NMA are strictly contained in 2-NMA. Neven et al. separated 2-DMA,

	1-DMA	1-NMA	1-AMA
Union	✓	✓	✓
Intersection	✓	✓	✓
Complement	✓	↯	✓
Concatenation	↯	✓	?
Kleene Star	↯	✓	?

Table 2.1: Closure Properties of Finite Memory Automata.

2-NMA, and 2-AMA subject to complexity theoretic assumptions [NSV04]. The containments between different classes of FMA are summarized in Figure 2.1.

Also note that that there is a 1-NMA that accepts L_1, while no 1-NMA can accept $\bar{L}_1$. Hence, 1-NMA are not closed under complement. On the other hand Kaminski et al. showed that 1-NMA are closed under union, intersection, concatenation and Kleene star. 1-DMA on the other hand are easily seen to be closed under complementation (simply by reversing accepting and non-accepting states), but they are not closed under concatenation and Kleene star. More closure properties have been established in [DL09]. Table 2.1 summarizes closure properties of FMA.

FMA can be extended with the ability to non-deterministically guess a value in D [KT08]. Deterministic FMA with this additional capability can accept the language where the last symbol is different from all the others. Fresh-register automaton are an extension of one-way FMA that allow two forms of guessing: as the model described above, they can non-deterministically guess a value; in addition they have the capability to non-deterministically guess a value that does not occur anywhere to the left of the current input symbol [Tze11, MT11]. Deterministic Fresh-Register automata can accept the language that contains all words with pairwise distinct symbols, a language that cannot be accepted by deterministic FMA. The class of languages that is accepted by non-deterministic FRA is still closed under union and intersection, but not under concatenation and Kleene star.

Decidability Results for Finite Memory Automata. It is easy to see that the emptiness problem for 1-NMA is decidable. This is because whenever there is a sequence of transitions in an 1-NMA that connects two states p and q, then there is a word that induces a run along this sequence of transitions [KF94]. Containment of 1-NMA is decidable if the containing automaton has at most two register [KF94].

Kaminski and Francez stated as an open question whether decidability of 1-NMA can be extended to a two way variant of finite memory automata. Neven, Schwentick

	1D-FMA	2D-FMA	1N-FMA	1A-FMA$_1$	1A-FMA
Emptiness	✓	↯	✓	✓	↯
Universality	✓	↯	↯	✓	↯
Containment	✓	↯	↯	✓	↯
Equivalence	✓	↯	↯	✓	↯

Table 2.2: Decidability of Finite Memory Automata.

and Vianu answered this question in a negative way, by showing that emptiness of 2-DMA is undecidable [NSV04]. In addition Neven et al. showed that universality and equivalence of 1-NMA is undecidable [NSV04].

This implies undecidability of the non-emptiness problem for 1-AMA. When restricted to 1-AMA with at most one register, denoted 1-AMA$_1$, decidability of the non-emptiness problem is recovered [DL09]. This model can be further extended with the ability to (a) nondeterministically guess a data value and (b) a synchronization capability called 'spread', without loosing the decidability of the non-emptiness problem [Fig10]. The decidability results for FMA are summarized in Table 2.2.

Pebble Automata

Pebble automata (PA) are an extension of finite automata that use pebbles to mark positions in the input words [NSV04]. These pebbles must be placed according to a stack discipline and the topmost pebble is the only pebble that can be moved. Hence the topmost pebble can be thought of as the head of the automaton. On each transition, a pebble automaton can check (i) how many pebbles are currently placed, (ii) which of the positions on which pebbles are placed have the same value as position of the topmost pebble, and (iii) which of the placed pebbles are at same position as the topmost pebble. Depending on the outcome of this check, a pebble automaton can decide to either lift the topmost pebble, move the topmost pebble to an adjacent position, or place a new pebble on the string.

Two variants of PA have been studied called 'weak' and 'strong' PA. The difference is that weak PA place a new pebble on the position of the current topmost pebble, whereas the strong version places a new pebble on the first position of the word. Clearly, this pebble placement only makes a difference in the case of one-way PAs. We omit a formal definition of a pebble automaton.

Following the notation of [NSV04], we denote variants of PA by dc-PA, where $d \in \{1, 2\}$, $c \in \{D, N, A\}$. Here $d = 1$ denotes the one-way variant while $d = 2$ denotes

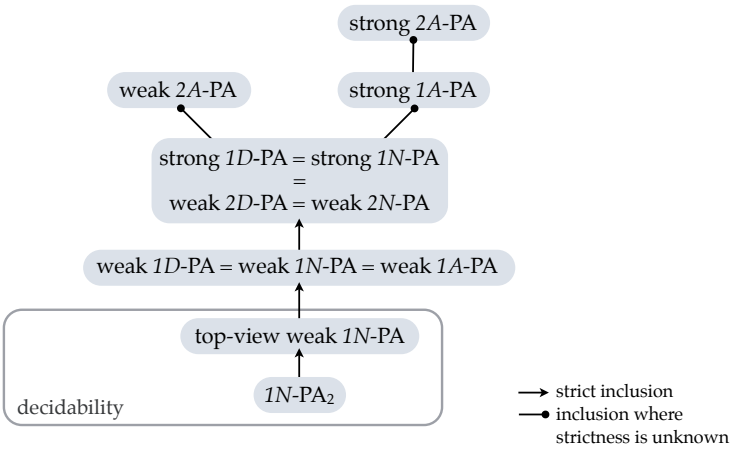


Figure 2.2: The expressiveness of pebble automata.

the two-way variant and D, N, A stand for deterministic, non-deterministic and alternating respectively.

Pebble automata have many good properties. For example, all models of PA are closed under all Boolean operations, concatenation, and Kleene star. Neven, et at. have shown that PA can be determinized and some weak 2-way models have the same expressive power as their strong 1-way variants. It follows that strong 1D-PA, strong 1N-PA, weak 2D-PA, and weak 2N-PA have the same expressive power [NSV04]. It is known that weak 1-NPA are weaker than strong 1-NPA. But weak PA form a robust class as weak 1D-PA, weak 1N-PA and weak 1A-PA all have the same expressive power [Tan10]. The inclusions between different classes of pebble automata are shown in Figure 2.2.

Despite this robustness, already very simple PA have an undecidable emptiness problem. In particular, weak 1-PA with just three pebbles are undecidable [NSV04]. But, with only two pebbles the emptiness problem becomes decidable [Tan10].

Top view weak PA are a restriction of weak PA where the automaton can only compare the value of the last and the second to last placed pebbles. Thus top view weak PA are a generalization of weak PA with two pebbles. Nonetheless the emptiness problem for top view weak PA remains decidable, even if the number of pebbles is unbounded [Tan10].

Data Automata and Class Memory Automata

Whereas pebble automata and finite memory automata can be defined to accept both plain and adorned languages, data automata can only be defined to accept adorned data languages (recall that an adorned data language is a language over an

	Det. CMA	Non-Det. CMA
Union	↯	✓
Intersection	✓	✓
Complement	↯	↯
Concatenation	↯	✓
Kleene Star	↯	↯

Table 2.3: Closure Properties of Class Memory Automata.

alphabet $\Sigma \times D$ where Σ is a finite alphabet and D is an infinite set of values). A data automaton [BMS+06] consists of two devices $(\mathcal{A}, \mathcal{B})$, a letter to letter transducer $\mathcal{A}$ with input alphabet Σ and a finite output alphabet Γ, and a finite automaton $\mathcal{B}$ on Γ. A run of a data automaton processes a word w in two phases: First $\mathcal{A}$ runs on w ignoring the data values producing an output word w' over Γ. In the second phase that automaton checks whether all sequences of letters in w' that have the same data value are accepted by $\mathcal{B}$. If this is the case then the automaton accepts the word.

Björklund and Schwentick showed that data automata are equivalent to a second automata model called *non-deterministic class memory automata* [BS10]. Roughly, a class memory automata $\mathcal{A}$ is a finite automaton that stores, in addition to its global state, a class memory function. The class memory function is used to store for each data value a local state. When processing a symbol (a, d) in the input word, $\mathcal{A}$ can check what local state it was in when it last visited a position with the value d. This allows a class memory automaton to simulate the computation of a data automata in a single run.

The main result for data automata (and hence non-deterministic class memory automata) is that their emptiness is decidable [BMS+06].

Data automata (and hence non-deterministic class memory automata) do not share the nice closure properties of pebble automata. They are closed under union, intersection, product, and concatenation, but neither under complement nor under Kleene star. Deterministic class memory automata are not even closed under union or concatenation, but they can be extended with Presburger arithmetic acceptance conditions to obtain closure under complement [BS10]. The closure properties of class memory automata are summarized in Table 2.3.

Class Automata [BL10] are an extension of data automata. They are originally defined for data trees, we give an adaptation to data words here. The definition is

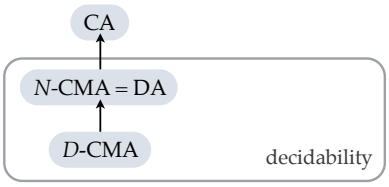


Figure 2.3: The expressiveness of data automata.

of a class automaton $(\mathcal{A}, \mathcal{B})$ is very similar to the definition of a data automaton: $\mathcal{A}$ is a non-deterministic transducer with input alphabet Σ and output alphabet Γ, and $\mathcal{B}$ is a finite automaton. There are two differences though: the input alphabet of $\mathcal{B}$ is $\Gamma \times \{0,1\}$ instead of Γ and a word w is accepted by $(\mathcal{A}, \mathcal{B})$ if there is a word $w' \in \mathcal{A}(w)$ such that for every class X of w, $w' \otimes X \in L(\mathcal{B})$; here $w' \otimes X$ is the word over $\Gamma \times \{0,1\}$ where every position x has the label of w' on the first coordinate and a 1 on the second coordinate iff the position x is in X.

Basically, data automata are class automata where membership $w' \otimes X \in L(\mathcal{B})$ depends only on the labels of positions $x \in X$ and not on labels of positions outside of X. Bojańczyk et al. showed that every unary XPath query can be recognized by (the tree version of) an class automaton [BL10]. As emptiness of XPath is undecidable it follows that class automata are strictly more expressive than data automata. The expressiveness of variants of data automata is summarized in Figure 2.3.

Comparisons Between the Models

Björklund and Schwentick have investigated the relationship between CMA and FMA. They show that non-deterministic class memory automata (and hence data automata) are strictly more expressive than 1-NMA. On the other hand DMA and deterministic CMAs are expressively incomparable [BS10].

PA and FMA are rather incomparable. It is clear that some variants of PA are more expressive than the corresponding variants of FMA. For example the language containing all words with pairwise distinct values can be accepted by a weak 1-DPA with only two pebbles, while it cannot be accepted by a 1-DMA. Generally, no strong pebble model is subsumed by any register model. On the other hand, as pebble automata can only place their pebble according to a stack discipline, FMA can accept languages that are not accepted by the corresponding PA models [NSV04].

The only relationship between CMA and pebble automata that is known is that weak 1-NPA with k pebbles, can be translated into 1-AMA with $k-1$ registers. This

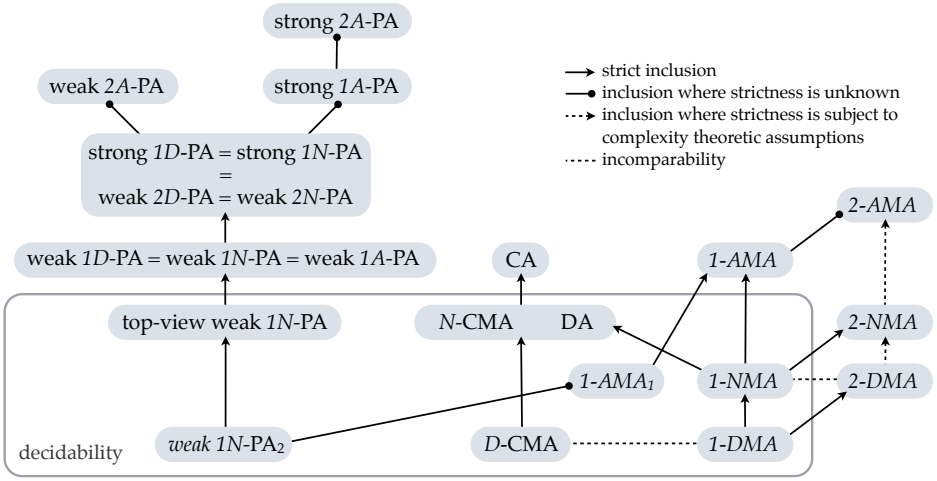


Figure 2.4: The expressiveness of pebble automata, data automata, and finite memory automata compared.

result was shown to obtain decidability for weak pebble automata with 2 pebbles [Tan10].

Which conclusions can we draw from our tour de automata? Much of the previous work on automata for data languages has focused on two aspects of regular languages: decidability and closure properties. We will consider other, more stringent properties, that will imply decidability. One such property is minimization. It is known that for every deterministic finite automaton there is a unique minimal deterministic finite automaton that accepts the same language. We will look for a model of automata for data languages for which the same is true. As it is known that there is no unique non-deterministic finite automaton for a given regular language, we will focus on deterministic models.

Apart from minimization we will also consider the relationship between automata and logics (see Chapter 3). More precisely we will consider decision problems of the form: Given a language L that is accepted by an automaton and a logic $\mathcal{L}$, can L be defined in $\mathcal{L}$? We will show that this problem becomes undecidable for several automata models including two-way finite memory automata and pebble automata. In many of our proofs, we will use the minimization procedure obtained in this section as one of the main technical tools.

For these reasons we will focus on deterministic one-way finite memory automata in the rest of this thesis.

2.3 Main Results

We begin to demonstrate how it is that that one-way deterministic memory automata inherit some of the desirable properties of deterministic finite automata. We begin by showing that the theorem of Myhill and Nerode can be extended to deterministic finite memory automata. That is, we give a characterization of the languages accepted by finite memory automata in terms of an equivalence relation $\hat{\equiv}_L$ on data words. We show:

Theorem 2.1. *Let L be a language over a relation R that is either the identity or a total order on D. Then L is DMA-recognizable iff $\hat{\equiv}_L$ has finite index.*

The structure of the proof of Theorem 2.1 is similar to the proof in the finite alphabet case. In one direction we show that if L is accepted by a finite memory automata $\mathcal{A}$, then $\hat{\equiv}_{L(\mathcal{A})}$ has finite index; this is shown by proving that an equivalence induced by the automaton refines $\hat{\equiv}_{L(\mathcal{A})}$. In the other direction we assume that $\hat{\equiv}_L$ has finite index and construct a deterministic finite memory automaton $\mathcal{A}_L$ from its equivalence classes. We then show that $\mathcal{A}_L$ is in fact the unique minimal deterministic finite memory automaton accepting L.

Theorem 2.2. *The canonical automaton $\mathcal{A}_L$ for a given DMA-recognizable language L is minimal.*

We consider a strong notion of minimality here: $\mathcal{A}_L$ has both the minimal number of states and it stores a minimal number of values. Finally we use Theorem 2.2 to provide an algorithm for minimization of deterministic finite memory automata over $(D, \sim)$.

Theorem 2.3. *Given a DMA $\mathcal{A}$ over $(D, \sim)$ with n states and k registers, a minimal equivalent DMA can be computed in deterministic space $\mathcal{O}(n \cdot (2k+1)^{2k})$*

2.4 Related Work

The first characterization of the languages accepted by deterministic finite memory automata has been given in [FK03]. It is shown that a language can be accepted by a deterministic finite memory automaton iff some equivalence relation has finite index. The authors state as a open question whether the automaton obtained from the characterization is minimal.

We answer this question positively: We show that the canonical deterministic finite memory automaton $\mathcal{A}_L$ that can be constructed from a given language L

satisfies a strong notion of minimality that takes into account both the number of control states and the number of values stored.

Subsequent to our work, a further characterization of deterministic finite memory automata has been given in [BKL11]. There, minimization is studied in the more general setting of "G-automata". A G-automaton can be thought of as infinite graph representing the configurations of an automaton. The "G" is a group that acts on this configuration graph. Intuitively such a group action maps each group element $g \in G$ to a function $\hat{g}$ on the configurations in such a way that the structure of the group is preserved (we will formally define the notion of a group action in the next chapter). For example if $(q, \bar{a})$ is a configuration of an automaton, consisting of a state q and a register assignment $\bar{a} \in D^k$, and if τ is an element of the group of permutations of data values, then the action of τ on $(q, \bar{a})$ can be defined by $(q, \tau(\bar{a}))$. In a similar way, group actions can be defined on the classes of an equivalence relation. The idea of a group action is to model symmetries between different configurations or equivalence classes. A maximal set of configurations or classes that behaves in a symmetric way is called an "orbit". The authors of [BKL11] show that a language can be accepted by an G-automaton with finitely many orbits iff the corresponding Myhill-Nerode equivalence has finitely many orbits. In addition, the authors show that orbit finite G-automata can be represented using a finite amount of storage. The results in [BKL11] show how finite memory automata can be minimized if the underlying alphabet is equipped with either equality, a total order, or a partial order.

A characterization of a class of timed languages – languages over infinite alphabets in which the data values are interpreted as time stamps – has been given in [BPT03]. The authors define a new automata model and show that a language is accepted by this model iff it is accepted by an algebraic structure called monoid. We will consider characterizations of this kind in the next chapter, where we give more details about the construction in [BPT03] (see Section 3.4). The class of languages accepted by the automata in [BPT03] is much larger than those accepted by finite memory automata. For example the language $\{a_1 \dots a_n \mid \text{each } a_i \text{ is a prime number for } i \leq n\}$ is accepted by these automata. Unsurprisingly, the model is undecidable, but the authors present a decidable restriction of the automata model.

Over finite alphabets there are two basic classes of algorithms for minimization: those that refine a partition of states and those that merge states. An example of an algorithm in the first category is Hopcroft's algorithm [Hop71]. It runs in $O(n \log n)$ time and is up to date the most efficient known algorithm for the general

case. Recently, it has been extended to incomplete deterministic finite automata [VL08, BC08]. Another example of a minimization algorithm in the first class is Moore's partition refinement algorithm [Moo56]. It has been shown that, although its worst-case behavior is quadratic, its average running time is $O(n \log n)$ [Dav10]. An example of a minimization algorithm that merges states is Revuz's algorithm for minimizing cycle-free automata [Rev92]. This algorithm runs in linear time, and has been extended to a more general class of automata by Almeida and Zeitoun [AZ08].

For non-deterministic finite automata (NFA) it is known that in general, there is no *unique* minimal NFA for a given regular language. Nonetheless one can still compute some minimal NFA. It has been shown that this problem is PSPACE-complete [SM73]. The minimization problem for automata with varying degrees of non-determinism was studied in [JR93] and it has been shown in [BM08] that already for very restricted classes of NFA, the problem remains NP-complete. An comparison of minimization algorithms for DFA can be found in [BBCF10].

Like for non-deterministic finite automata, minimal bottom-up unranked tree automata are not unique, even in their deterministic variant [MN07]. If non-deterministic finite automata are allowed in the transition of these automata, then minimization becomes PSPACE-hard, because minimization is already PSPACE-hard for non-deterministic finite automata. If the finite automata are restricted to be deterministic, then minimization becomes NP-complete [MN07]. If a slightly different definition is used, then a unique minimal bottom-up unranked tree automaton can be computed in quadratic time [CLT05].

2.5 Myhill-Nerode for Data Languages

We now characterize the class of DMA-recognizable languages. This characterization will allow us to show that for every deterministic finite memory automaton, there is a *unique* minimal deterministic finite memory automaton accepting the same language. In addition this characterization allows us to minimize DMA in a strong sense: We show that for every DMA there is an equivalent DMA that has a minimal number of control states and that stores a minimal number of data values.

2.5.1 The Equivalence

Our characterization is akin to Myhill's and Nerode's characterization of the regular languages over a finite alphabet: Given a language L, the Myhill-Nerode equivalence

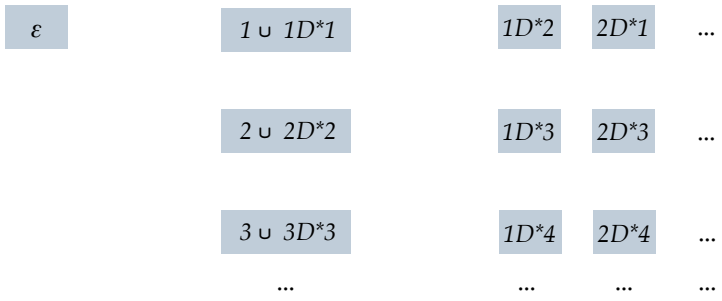


Figure 2.5: The classes of the Myhill-Nerode equivalence for the language $L_{\text{first=last}}$ over the alphabet $D = \mathbb{N}$.

$\equiv_L$ relates two words u and v iff for all words w,

$$uw \in L \quad \text{iff} \quad vw \in L.$$

The theorem of Myhill and Nerode states that a language L is regular iff $\equiv_L$ has finite index.

The definition of $\equiv_L$ also makes sense for data languages. The problem is that even very simple data languages have infinitely many $\equiv_L$-classes. Consider for example the language

$$L_{\text{first-last}} = \{d_1 \dots d_n \in D^* \mid d_1 = d_n\}$$

The equivalence classes of $\equiv_{L_{\text{first=last}}}$ are shown in Figure 2.5. Note that many equivalence classes can be obtained from each another by consistently renaming their elements. These renamings must take the relation R underlying the language into account:

Definition 2.2 (R-Preserving Data Renaming)**.** *Let $R \subseteq D \times D$ be a relation and let $C \subseteq D$. An* R-preserving data renaming over C *is a bijection τ on D that is an automorphism on (C, R): that is, for all $a, b \in C$,*

$$a \, R \, b \quad \textit{iff} \quad \tau(a) \, R \, \tau(b).$$

For example, if R is data equality, then every bijection on D is R-preserving. On the other hand, if R is a total order, then the R-preserving functions are exactly the monotone functions.

2.5.2 Memorable Values

Given a DMA-recognizable language L and a prefix w of an input word, there exist some values in w that need to be stored by *any* DMA that recognizes L. We will

call these values memorable. Before we define this notion formally, consider the following example: Let L be the language $\{x\,y\,z\,y \mid x < y < z\}$ over a dense order $\leq$. Observe that, after parsing the word $w = 1\,2\,3$, any DMA $\mathcal{A}$ that recognizes L must be storing the value 2: otherwise $\mathcal{A}$ could not distinguish the two possible continuations $u = 2$ and $v = \frac{3}{2}$ (this is also the reason that $1\,2\,3 \not\equiv_L 1\,\frac{3}{2}\,3$). Hence, we will define 2 to be memorable in w with respect to the language L.

Definition 2.3. *Let L be a data language over R. A value a is memorable in a word u iff there is a value b that does not appear in u such that the transposition $\tau_{a,b}$ of a and b is R-preserving over the symbols in u and*

$$u \not\equiv_L \tau_{a,b}(u).$$

This definition first appeared in [BLP10b]. Later, a similar definition was used in [Boj11]. The difference is that we use transpositions – that is data renamings that move only two values – whereas [Boj11] uses arbitrary data renamings.

It is convenient to fix a string based representation of the set of memorable values. Hence we denote by $\mathrm{mem}_L(w)$ the sequence that contains exactly the memorable values of w, ordered according to their last appearance in w.

We now show a key lemma about memorable values.

Lemma 2.4. *Let L be a language over a relation R that is either a dense total order or the identity on D. Then, for every word w and every value a, $\mathrm{mem}_L(wa)$ is a sub-sequence of $\mathrm{mem}_L(w)\,a$.*

Proof. We show that every L-memorable value of wa is either an L-memorable value of w or it coincides with a (the convention that the values in $\mathrm{mem}_L(w)$ are ordered according to the positions of their last occurrences in w will then imply the claim of the proposition). Suppose towards a contradiction, that there is a value b, that is not a, that is L-memorable in wa but not in w. As b is L-memorable in wa, we know there exist a word u and an R-preserving transposition $\tau_{b,c}$ that swaps b with a value c not occurring in wau such that

$$wau \not\equiv_L wa\,\tau_{b,c}(u)$$

Similarly, since b is not L-memorable in w, then we know that, for every word v and every R-preserving transposition $\tau_{b,c'}$ that maps b to a value c' not occurring in wv, we have

$$wv \equiv_L w\,\pi_{b,c'}(v).$$

In particular, by letting $v = au$ and $c = c'$, we obtain

$$wau \;=\; wv \;=_L\; w\,\tau_{b,c'}(v) \;=\; w\,\tau_{b,c'}(au) \;=\; w\,\tau_{b,c}(au).$$

As $c = c'$ and as c' does not occur in wav it follows that $w\,\tau_{b,c}(au) = wa\,\tau_{b,c}(u)$. Thus we have a contradiction to $wau \neq_L wa\,\tau_{b,c}(u)$. □

2.5.3 A Characterization

We now define a relation $\hat{\equiv}_L$ that is courser than $\equiv_L$, but finer than $=_L$. Basically, $\hat{\equiv}_L$ relates two words u and v if there is a renaming of one word that is in the $\equiv_L$-class of the other word.

Definition 2.5. *Given a language L over $R \subseteq D \times D$, we define $\hat{\equiv}_L \subseteq D^* \times D^*$ by letting $u \,\hat{\equiv}_L\, v$ iff there is an R-preserving data renaming τ such that $u \equiv_L \tau(v)$. In this case we also say that $u \,\hat{\equiv}_L\, v$ via τ.*

It is easy to see that $\hat{\equiv}_L$ is an equivalence relation. However, unlike $\equiv_L$, $\hat{\equiv}_L$ is not a congruence in general. For example consider the language $L = \{dd \mid d \in D\}$. Then $d \,\hat{\equiv}_L\, d'$ for any pair of values d and d'. However, $dd \not\hat{\equiv}_L\, d'd$ if $d \neq d'$.

We now associate with every DMA $\mathcal{A}$ an equivalence relation $\hat{\equiv}_{\mathcal{A}}$. Later we will show that if $\mathcal{A}$ is minimal, then $\hat{\equiv}_{\mathcal{A}}$ corresponds exactly to $\hat{\equiv}_L$.

Definition 2.6. *Given a DMA $\mathcal{A}$ over $R \subseteq D \times D$, we define $\hat{\equiv}_{\mathcal{A}} \subseteq D^* \times D^*$ by letting $u \,\hat{\equiv}_{\mathcal{A}}\, v$ iff $\mathcal{A}$ reaches the configurations $(p, \bar{a})$ and $(q, \bar{b})$ by reading u and v respectively, such that $p = q$ and $\bar{a} \cong_R \bar{b}$.*

It is easy to see that if $\mathcal{A}$ is a deterministic finite memory automaton over R that has n control states and that stores at most k values, then the corresponding equivalence $\hat{\equiv}_{\mathcal{A}}$ has index at most $n \cdot k!$ (indeed, the $\hat{\equiv}_{\mathcal{A}}$-equivalence class of any word w is uniquely determined by the control state q and by the $\cong$-type of the register assignment $\bar{a}$ of the configuration $(q, \bar{a})$ that is reached by $\mathcal{A}$ after reading w). If R is the identity relation $\sim$, then the upper bound for the number of $\hat{\equiv}_{\mathcal{A}}$-equivalence classes drops down to n.

We are now ready to state the result that characterizes the languages accepted by deterministic finite memory automata. This is the main theorem in this chapter.

Theorem 2.1. *Let L be a language over a relation R that is either the identity or a total order on D. Then L is DMA-recognizable iff $\hat{\equiv}_L$ has finite index.*

Before we prove the theorem, we briefly summarize the key ingredients of the proof. The left-to-right-direction is proved by assuming that L is recognized by a DMA $\mathcal{A}$ and by showing that the corresponding equivalence relation $\hat{\equiv}_{\mathcal{A}}$ refines $\hat{\equiv}_L$ (the theorem follows because $\hat{\equiv}_L$ has finite index). The converse direction is proved by assuming that $\hat{\equiv}_L$ has finite index and by building a finite-memory automaton $\mathcal{A}_L$ from the equivalence classes of $\hat{\equiv}_L$.

Proof. We first show the (easier) direction from left to right. Let $\mathcal{A} = (Q, q_I, F, T, \mathrm{ar})$ be a DMA over $R \subseteq D \times D$ that recognizes the language L. We prove that $\hat{\equiv}_L$ has finite index by showing that $\hat{\equiv}_{\mathcal{A}}$ refines $\hat{\equiv}_L$ (we have already argued that the index of $\hat{\equiv}_{\mathcal{A}}$ has size at most $|Q| \cdot k!$). We fix two words u and v such that $u \,\hat{\equiv}_{\mathcal{A}}\, v$. Let $(p, \bar{a})$ and $(q, \bar{b})$ be the configurations reached by $\mathcal{A}$ after reading u and v, respectively. Since $u \,\hat{\equiv}_{\mathcal{A}}\, v$, we know that $p = q$ and $\bar{a} \cong_R \bar{b}$.

We will show that $u \,\hat{\equiv}_L\, v$, that is, we show that there is an R-preserving data renaming τ such that $u \equiv_L \tau(v)$. As L is closed under R-preserving isomorphisms it is sufficient to show that for all words w

$$uw \in L \quad \text{iff} \quad v\tau(w) \in L.$$

We define τ to be a bijection on D that maps the i-th entry of $\bar{a}$ to the i-th entry of $\bar{b}$ for all $i \leq |\bar{a}|$ and that is the identity on every value that is neither in $\bar{a}$ nor $\bar{b}$. Note that τ is a an R-preserving data renaming because $\bar{a} \cong_R \bar{b}$.

Now let a word $w = a_1 \dots a_n$ be given and assume that

$$(p, \bar{a}) = (p_0, \bar{a}_0) \xrightarrow[r_1, E_1]{a_1} (p_1, \bar{a}_1) \dots (p_{n-1}, \bar{a}_{n-1}) \xrightarrow[r_n, E_n]{a_n} (p_n, \bar{a}_n)$$

is a run of $\mathcal{A}$ on w. It is easy to see that

$$(p_0, \tau(\bar{a}_0)) \xrightarrow[r_1, E_1]{\tau(a_1)} (p_1, \tau(\bar{a}_1)) \dots (p_{n-1}, \tau(\bar{a}_{n-1})) \xrightarrow[r_n, E_n]{\tau(a_n)} (p_n, \tau(\bar{a}_n))$$

is also a run of $\mathcal{A}$. As $(p_0, \tau(\bar{a}_0)) = (q, \bar{b})$ this shows that $uw \in L$ iff $v\tau(w) \in L$.

We now prove that, if $\hat{\equiv}_L$ has finite index, then there is a k-register DMA $\mathcal{A}_L = (Q, q_I, F, T, \mathrm{ar})$ that recognizes L. We first define k. Note that for each $\hat{\equiv}_L$-equivalence class $C = [w]_{\hat{\equiv}_L}$, with $w \in D^*$, there is a number i_C such that $|\mathrm{mem}_L(w)| = i_C$ (by definition of $\hat{\equiv}_L$, this number i_C does not depend on the choice of the representant w). We let k be the maximum number i_C, for all $\hat{\equiv}_L$-equivalence classes C. Then, we define Q as the set of all $\hat{\equiv}_L$-equivalence classes and the arity function is defined by $\mathrm{ar}([w]_{\hat{\equiv}_L}) = |\mathrm{mem}_L(w)|$. The initial state q_I is $[\varepsilon]_{\hat{\equiv}_L}$ and $F = \{[w]_{\hat{\equiv}_L} \mid w \in L\}$ is

the set of final states of $\mathcal{A}_L$. Finally, we let T contain all and only the tuples of the form

$$\left([w]_{\hat{\equiv}_L}, \alpha, E, [w\,a]_{\hat{\equiv}_L}\right)$$

where $w \in D^*$, $a \in D$, α is the $\cong_R$-type of $\mathrm{mem}_L(w) \cdot a$, and E is the set of positions of $\mathrm{mem}_L(w) \cdot a$ such that the removal of all positions $i \in E$ from $\mathrm{mem}_L(w) \cdot a$ yields exactly $\mathrm{mem}_L(w\,a)$. Note that the set E exists, as by Lemma 2.4, $\mathrm{mem}_L(w\,a)$ is a sub-sequence of $\mathrm{mem}_L(w) \cdot a$.

That $\mathcal{A}_L$ accepts L follows from a simple inductive argument. One can show that after reading a word w, $\mathcal{A}_L$ is in state $[w]_{\hat{\equiv}_L}$ with register assignment $\mathrm{mem}_L(w)$. We omit the details.

We still need to prove that $\mathcal{A}_L$ is deterministic, namely, that for every pair of transition rules in T of the form $\left([u]_{\hat{\equiv}_L}, \alpha, E, [u\,a]_{\hat{\equiv}_L}\right)$ and $\left([v]_{\hat{\equiv}_L}, \beta, F, [v\,b]_{\hat{\equiv}_L}\right)$, we have that

$$\text{if } u \,\hat{\equiv}_L\, v \text{ and } \alpha = \beta \text{ then } u\,a \,\hat{\equiv}_L\, v\,b \text{ and } E = F. \qquad (*)$$

We first show three claims. The statement $(*)$ will be a simple consequence of these claims.

Claim 1. *Let u, v be words and let σ be an R-preserving data renaming. If $\sigma(v) \equiv_L u$ then $\sigma(\mathrm{mem}_L(v)) = \mathrm{mem}_L(u)$.*

Proof of Claim 1. Assume that $\sigma(v) \equiv_L u$ and that $a \in \sigma(\mathrm{mem}_L(v))$. As L is closed under R-preserving data renamings it follows that $a \in \mathrm{mem}_L(\sigma(v))$. Then there is a value b that does not occur in $\sigma(v)$ such that $\tau_{a,b}$ is R-preserving and $\sigma(v) \not\equiv_L \tau_{a,b} \circ \sigma(v)$. We need to show that there is a value b' that does not occur in u such that $\tau_{a,b'}$ is R-preserving and $u \not\equiv_L \tau_{a,b'}(u)$. We choose b' such that $\sigma(v)\,b \cong \sigma(v)\,b'$ and such that b' that does not occur in u. A value b' with these properties exists because R is either the identity or a dense order. Then $u \equiv_L \sigma(v) \not\equiv_L \tau_{a,b} \circ \sigma(v) \cong \tau_{a,b'} \circ \sigma(v) \equiv_L \tau_{a,b'}(u)$. Claim 1 follows because L is closed under isomorphisms. The inclusion of $\mathrm{mem}_L(u)$ in $\sigma(\mathrm{mem}_L(v))$ holds by symmetric arguments. □

Claim 2. *Let u be a word and a, b be values. If $a, b \notin \mathrm{mem}_L(u)$ and $\mathrm{mem}_L(u)\,a \cong \mathrm{mem}_L(u)\,b$ then $u \equiv_L \tau_{a,b}(u)$ where $\tau_{a,b}$ is the transposition of a and b.*

Proof of Claim 2. As $\mathrm{mem}_L(u)\,a \cong_R \mathrm{mem}_L(v)\,b$ and as R is either the identity or a dense linear order, there must be a value c that does not appear in either u or $\tau_{a,b}(u)$ and such that $\tau_{a,c}$ and $\tau_{b,c}$ are R-preserving. As $a, b \notin \mathrm{mem}_L(u)$ it holds that $\tau_{a,c}(u) \equiv_L \tau_{b,c}(u) \equiv_L (u)$. As $\tau_{a,b} = \tau_{a,c} \circ \tau_{b,c} \circ \tau_{a,c}$ it follows that $\tau_{a,b}(u) = \tau_{a,c} \circ \tau_{b,c} \circ \tau_{a,c}(u) \equiv_L u$. □

Claim 3. *Let u, v be words, a, b be values, and σ an R-preserving data renaming. If (i) $\sigma(v) \equiv_L u$ and (ii) $\mathrm{mem}_L(u) \cdot a \cong \mathrm{mem}_L(v) \cdot b$, then there is a data renaming τ such that $\tau(v) \equiv_L u$ and $u\,\tau(b) = u\,a$.*

Proof of Claim 3. Assume (i) and (ii). We distinguish two cases:

- **Case** $a \in \mathrm{mem}_L(u)$**.** In this case we chose $\tau = \sigma$. Then $\tau(v) = \sigma(v) \equiv_L u$.

  To prove that $u\tau(b) = ua$ we first note that by (i) and Claim 1 if holds that $\sigma(\mathrm{mem}_L(v)) = \mathrm{mem}_L(u)$. By (ii) and the hypothesis of this case it holds that $b \in \mathrm{mem}_L(v)$ and in particular a is the i-th position of $\mathrm{mem}_L(u)$ iff b is the i-th position of $\mathrm{mem}_L(v)$. Therefore $\sigma(b) = a$ and hence $u\tau(b) = u\sigma(b) = ua$.

- **Case** $a \notin \mathrm{mem}_L(u)$**.** In this case we define τ to be $\tau_{a,\sigma(b)} \circ \sigma$ where $\tau_{a,\sigma(b)}$ is the transposition of a and $\sigma(b)$ and $\circ$ denotes functional composition. Then $u\,\tau(b) = u\,\tau_{a,\sigma(b)}(\sigma(b)) = u\,a$.

  Next we note that by Claim 1 and by (i) we have that $\mathrm{mem}_L(u) = \sigma(\mathrm{mem}_L(v))$. Therefore $\mathrm{mem}_L(u)\,\sigma(b) \cong \sigma(\mathrm{mem}_L(v))\,\sigma(b)$. In addition, as σ is R-preserving, we get that $\sigma(\mathrm{mem}_L(v))\,\sigma(b) \cong \mathrm{mem}_L(v)\,b$. Also, we assume (ii), that is that $\mathrm{mem}_L(v) \cdot b \cong \mathrm{mem}_L(u) \cdot a$. Combining the previous statements we can conclude that $\mathrm{mem}_L(u)\,\sigma(b) \cong \mathrm{mem}_L(u)\,a$. As $a \notin \mathrm{mem}_L(u)$ holds by hypothesis of this case we get that $\sigma(b) \notin \mathrm{mem}_L(u)$. Therefore it follows from Claim 2 that $\tau_{a,\sigma(b)}(u) \equiv_L u$. Thus $\tau(v) = \tau_{a,\sigma(b)} \circ \sigma(v) \equiv_L \tau_{a,\sigma(b)}(u) \equiv_L u$.

This completes the proof of Claim 3. □

We continue with the proof of $(*)$. As we have already argued that after reading w, $\mathcal{A}_L$ stores $\mathrm{mem}_L(w)$, it is sufficient to show that if $u \mathrel{\hat{\equiv}_L} v$ and $\mathrm{mem}_L(u) \cdot a \cong \mathrm{mem}_L(v) \cdot b$ then $u\,a \mathrel{\hat{\equiv}_L} v\,b$. Thus we assume that $u \equiv_L \sigma(v)$ for some R-preserving

data renaming σ and $\mathrm{mem}_L(u)\cdot a \cong \mathrm{mem}_L(v)\cdot b$. By Claim 3 there is a data renaming τ such that $\tau(v) \equiv_L u$ and $u\,\tau(b) = u\,a$. Then for all words w

$$v\,b\,w \in L \quad \text{iff} \quad \tau(v\,b\,w) \in L \quad \text{iff} \quad u\,\tau(b\,w) \in L \quad \text{iff} \quad u\,a\,\tau(w) \in L$$

where the first equality holds because L is closed under R-preserving data renamings, the second equality holds because $u \equiv_L \tau(v)$ and the last equivalence holds because $u\,\tau(b) = u\,a$. It follows that $u\,a \mathrel{\hat{\equiv}}_L v\,b$.

It remains to show that $E = F$. We prove this by showing that the i-th value of the sequence $\mathrm{mem}_L(u)\,a$ is memorable in $u\,a$ iff the i-th value of the sequence $\mathrm{mem}_L(v)\,b$ is memorable in $v\,b$ for all $i \leq |\mathrm{mem}_L(u)| (= |\mathrm{mem}_L(v)|)$. Hence assume that a' is the i-th value in the sequence $\mathrm{mem}_L(u)\,a$ and that a' is memorable in $u\,a$. Then there is a value a'' that does not occur in $u\,a$ such that $\tau_{a',a''}$ is R-preserving and $ua \not\equiv_L \tau_{a',a''}(ua)$. Let b' be the i-th value in the sequence $\mathrm{mem}_L(v)\,b$ and let b'' be a value such that $u\,a\,a'\,a'' \cong v\,b\,b'\,b''$. We have proved in the in the paragraphs above that $v\,b \mathrel{\hat{\equiv}}_L u\,a$, that is, there is a data renaming τ such that $\tau(v\,b) = u\,a$. Then

$$\tau(v\,b) \equiv_L \; u\,a \not\equiv_L \tau_{a',a''}(u\,a) \equiv_L \tau_{a',a''} \circ \tau(v\,b) = \tau \circ \tau_{b',b''}(v\,b)$$

This shows that b' is memorable in $v\,b$. The other direction follows from symmetric arguments. □

Extension to Non-Dense Supports. We now show why the above construction fails over non-dense orders and how the definition of a memorable value can be adapted to overcome this problem. As an example, let us reconsider the language $L = \{xyzy \mid x < y < z\}$ and the word $w = 123$, but now over the support $(\mathbb{N}, <)$. According to Definition 2.3, the value $a = 2$ is not L-memorable in w anymore, since it cannot be substituted in w by any other fresh value b without changing the resulting $\cong_<$-type. In order to overcome this problem, we exploit the fact that, for any fixed support $(D, \prec)$, where $\prec$ is an arbitrary total order over D, and for every word w and value a over the support $(D, \prec)$, there exist a word $\widetilde{w}$ and two values $\widetilde{a}$ and $\widetilde{b}$ such that

$$w \cdot a \;\cong_\prec\; \widetilde{w} \cdot \widetilde{a} \;\cong_\prec\; \widetilde{w} \cdot \widetilde{b}.$$

In particular, this implies that $\widetilde{a}$ can always be substituted with $\widetilde{b}$ in $\widetilde{w}$ without changing the resulting $\cong_\prec$-type. The following definition is the natural generalization of Definition 2.3.

Definition 2.7. *Let L be a language over support $(D, \prec)$, where $\prec$ is a total (possibly not dense) order. A value a is* L-memorable *in a word w if there exists a words w' and two values a', b' such that $w\,a \cong_{\prec} w'\,a'$, $\tau_{a',b'}$ is $\prec$-preserving on the symbols in w' and*

$$w' \notin_L \tau_{a',b'}(w').$$

As an example, given $L = \{xyzy \mid xy \prec z\}$ over the support $(\mathbb{N}, <)$, we now have that $a = 2$ is L-memorable in $w = 123$, since there exist $w' = 246$, $a' = 4$, and $b' = 5$ such that (i) $w\,a \cong_{<} w'\,a'$, (ii) $\tau_{a',b'}$ is $\prec$-preserving on the symbols in w', and (iii) $w' \notin_L \tau_{a',b'}(w')$.

Finally, since languages are assumed to be closed under isomorphisms, Claims 1, 2, and 3 in the proof of Theorem 2.1 can be easily generalized to cope with the new definition of memorable value for supports of the form $(D, \prec)$, where $\prec$ is an arbitrary total order.

2.5.4 Minimality

We now prove that the automaton $\mathcal{A}_L$ constructed in the proof of Theorem 2.1 – henceforth called *canonical automaton* for a given language L – is minimal among all equivalent DMA recognizing L. Here, we adopt a general notion of minimality for DMA that takes into account both the number of control states and the number of stored values on each input word. Precisely, we say that a DMA $\mathcal{A} = (Q, T, q_I, F, \mathrm{ar})$ is *state-minimal* if for every equivalent DMA $\mathcal{A}' = (Q', T', q_I', F', \mathrm{ar}')$ that recognizes the same language, we have that $|Q| \leq |Q'|$. Similarly, we say that $\mathcal{A}$ is *data-minimal* if, for every equivalent DMA $\mathcal{A}' = (Q', T', q_I', F'\,\mathrm{ar}')$ that recognizes the same language and every input word w, we have

$$\text{if } (q_I, \varepsilon) \xrightarrow[\mathcal{A}]{w} (q, \bar{a}) \text{ and } (q_I', \varepsilon) \xrightarrow[\mathcal{A}']{w} (q', \bar{a}') \text{ then } |\bar{a}| \leq |\bar{a}'|.$$

Finally, we say that $\mathcal{A}$ is *minimal* if it is both state-minimal and data-minimal. Below, we show that the canonical automaton is minimal among all equivalent DMA.

Theorem 2.2. *The canonical automaton $\mathcal{A}_L$ for a given DMA-recognizable language L is minimal.*

Proof. We first prove that $\mathcal{A}_L$ is state-minimal. Let us consider a DMA $\mathcal{A}'$ that recognizes L. We introduce a function f that maps control states of $\mathcal{A}_L$ to control states of $\mathcal{A}'$, as follows. First, we associate with each control state q of $\mathcal{A}_L$ a representant w_q of the corresponding $\equiv_L$-equivalence class, namely, we assume $q = [w_q]_{\equiv_L}$ for some word $w_q \in D^*$. Then, for each control state q of $\mathcal{A}_L$, we define $f(q)$ as the

control state reached by $\mathcal{A}'$ after reading w_q. We now prove that f injective. Let us consider two control states q and q' of $\mathcal{A}_L$ such that $f(q) = f(q')$. By construction, we know that $\mathcal{A}'$ reaches two configurations of the form $(f(q), \bar{a})$ and $(f(q'), \bar{a}')$ by reading w_q and $w_{q'}$, respectively. Since $f(q) = f(q')$, we know that w_q and $w_{q'}$ are in the same $\equiv_{\mathcal{A}'}$-equivalence class. Moreover, by recalling the proof of Theorem 2.1, we know that $\equiv_{\mathcal{A}'}$ refines $\equiv_L$. This shows that $w_q \equiv_L w_{q'}$ and hence $q = q'$. It follows that the function f is injective and hence $\mathcal{A}'$ has at least as many control states as $\mathcal{A}_L$.

We now prove that $\mathcal{A}_L$ is data-minimal. Let us fix a DMA $\mathcal{A}'$ that recognizes L and let us consider a generic word w. Suppose that $\mathcal{A}_L$ reaches the configuration $(q, \bar{a})$ by reading w and, similarly, $\mathcal{A}'$ reaches the configuration $(q', \bar{a}')$ by reading w. From the proof of Theorem 2.1, we recall that the stored values $\bar{a}$ coincides with the sequence of L-memorable values of w. Moreover, it is easy to see that $\mathrm{mem}_L(w)$ is a sub-sequence of the stored values $\bar{a}'$. We thus conclude that $|\bar{a}| \leq |\bar{a}'|$. □

We conclude the section by proving that minimal DMA, and in particular canonical automata, are unique up to isomorphisms. Here we think of each DMA $\mathcal{A} = (Q, T, q_I, F\,\mathrm{ar})$ as a finite directed graph, whose vertices are labeled by indices $i \in \{0, ..., k\}$ and represent control states in Q and whose edges are labeled by pairs (α, E) and represent transitions of the form (q, α, E, q').

Corollary 2.8. *Any minimal DMA recognizing a language L is isomorphic to the canonical automaton for L.*

Proof. Let $\mathcal{A}' = (Q', T', q'_I, F', \mathrm{ar}')$ be a DMA recognizing a language L and let $\mathcal{A}_L = (Q, T, q_I, F, \mathrm{ar})$ be the corresponding canonical automaton. The proof follows easily from that of Theorem 2.2. In particular, we first observe that $k = k'$ follows from the minimality of $\mathcal{A}_L$ and $\mathcal{A}'$. The bijection that maps control states of $\mathcal{A}_L$ to control states of $\mathcal{A}'$ is given by the function f introduced in the first part of the proof of Theorem 2.2. Note that this function satisfies $f(q) \in Q'_i$ for all $0 \leq i \leq k = k'$ and all $q \in Q_i$. Moreover, since $\mathcal{A}'$ is state-minimal, f must be surjective. Finally, the correspondence between the transitions of $\mathcal{A}_L$ and those of $\mathcal{A}'$ is uniquely determined by the function f, since both $\mathcal{A}_L$ and $\mathcal{A}'$ are deterministic and complete. □

2.6 Computing Minimal Automata

In this section, we focus on the problem of computing the minimal DMA for a given language over $(D, \sim)$. That is, we show the following theorem.

Algorithm 1: MINIMIZE($\mathcal{A}$)

input: a DMA $\mathcal{A}$ that accepts a language L over $(D, \sim)$
output: the canonical automaton $\mathcal{A}_L$ for L
let k = maximal number of values stored by $\mathcal{A}$
let n = number of control states of $\mathcal{A}$
let Σ = any subset of D of size $k+1$
let $Q \leftarrow \varnothing$
for all $w \in \Sigma^{\leq n}$

$$\textbf{do} \begin{cases} \textbf{if } w \not\equiv_L w' \textbf{ for all } w' \in Q \\ \quad \textbf{then} \begin{cases} Q \leftarrow Q \cup \{w\} \\ \mathrm{ar}(w) \leftarrow |\mathrm{mem}_L(w)| \end{cases} \end{cases}$$

$T \leftarrow \varnothing$
for all $w, w' \in Q$ **and for all** $a \in \Sigma$

$$\textbf{do} \begin{cases} \textbf{if } w \cdot a \mathrel{\hat{\equiv}}_L w' \\ \quad \textbf{then} \begin{cases} \alpha \leftarrow [\mathrm{mem}_L(w) \cdot a]_{\cong} \\ E \leftarrow \{i \mid \forall j \,.\, \mathrm{mem}_L(w \cdot a)(j) \neq (\mathrm{mem}_L(w) \cdot a)(i)\} \\ T \leftarrow T \cup \{(w, \alpha, E, w')\} \end{cases} \end{cases}$$

return $\mathcal{A}_L = (Q, T, \{w \in Q : w \equiv_L \varepsilon\}, \{w \in Q : w \in L\})$

Figure 2.6: Minimization algorithm for DMA over the support $(D, \sim)$.

Theorem 2.3. *Given a DMA $\mathcal{A}$ over $(D, \sim)$ with n states and k registers, a minimal equivalent DMA can be computed in deterministic space $\mathcal{O}(n \cdot (2k+1)^{2k})$.*

It follows from this theorem that the minimal DMA for a given DMA can be computed in deterministic time exponential in the number of states of $\mathcal{A}$ and double exponential in the number of registers of $\mathcal{A}$.

By Theorem 2.2 it is sufficient to compute the canonical automaton $\mathcal{A}_L$ recognizing $L = L(\mathcal{A})$, as it is the minimal DMA recognizing L. Algorithm 1 provides the pseudo-code of a procedure that receives a DMA $\mathcal{A}$ recognizing L as input and computes (a representation of) the corresponding canonical automaton $\mathcal{A}_L$. Such a procedure consists of two main loops: the first one computes a minimal and complete set Q of representatives of $\equiv_L$-equivalence classes, which are then identified with the control states of the canonical automaton $\mathcal{A}_L$; the second loop computes the set of transition rules of $\mathcal{A}_L$.

We now show that Algorithm 1 computes the minimal automaton for the given DMA $\mathcal{A}$. Assume that $\mathcal{A}$ has n states and stores at most k values. We first show that at the end of the computation Q contains exactly one representative for each $\equiv_L$-equivalence class. We claim that $\Sigma^{\leq n}$ contains at least one representative for each

equivalence class of $\equiv_L$: observe that each control state q in $\mathcal{A}$ can be reached by a word in $\Sigma^{\leq n}$ (this is the case because $|\Sigma| > k$ – see the proof of Lemma 2.9 for a similar argument). By the proof of Theorem 2.1 we know that $\hat{\equiv}_{\mathcal{A}}$ refines $\equiv_L$, and hence $\Sigma^{\leq n}$ contains at least one representative of each $\equiv_L$-equivalence class. It is clear that, after the execution of the first loop of the procedure, the set Q contains pairwise non-$\equiv_L$-equivalent representants of all $\equiv_L$-equivalence classes. Hence Q contains exactly one word in each $\equiv_L$-equivalence class. It is clear that the set T contains exactly the transitions of $\mathcal{A}$.

In order to claim that the pseudo-code of Algorithm 1 describes an *effective* minimization procedure, we need to verify that the following two problems are decidable:

- *The memorability problem.* Given a DMA $\mathcal{A}$ recognizing L over $(D, \sim)$, a word w, and a value a, is a is L-memorable in w?
- *The word-equivalence problem.* Given a DMA $\mathcal{A}$ recognizing L over $(D, \sim)$ and two words w and w', is $w \hat{\equiv}_L w'$?

The decidability of both problems is shown using the following lemma.

Lemma 2.9. *Let $\mathcal{A}$ be a DMA over $(D, \sim)$ that stores at most k values and let w, w' be two words. For each word $v \in D^*$ there is an alphabet Δ of size $2k+1$ and a word $u \in \Delta^*$ such that*

$$w\,v \;\equiv_{L(\mathcal{A})}\; w\,u \quad \textit{and} \quad w'\,v \;\equiv_{L(\mathcal{A})}\; w'\,u.$$

Proof. Let $\mathcal{A}$, w, w', and v be as in the claim of the proposition. Let $(\bar{a}_0, \bar{a}'_0)$ be the register content of $\mathcal{A}$ after reading w and w' respectively and let Δ be an alphabet of size $2k+1$ that contains all symbols in $\bar{a}_0$ and $\bar{a}'_0$. We define the (w, w')-*trace* of a word $v = a_1 \dots a_n$ is the sequence $\pi = (\bar{a}_0, \bar{a}'_0) \dots (\bar{a}_n, \bar{a}'_n)$, where, for every index $0 \leq i \leq n$, $\bar{a}_i$ is the memory content reached by $\mathcal{A}$ after reading the word $w \cdot (a_1 \dots a_i)$ and $\bar{a}'_i$ is the memory content reached by $\mathcal{A}$ after reading the word $w' \cdot (a_1 \dots a_i)$. We also introduce a transformation f on (w, w')-traces, which only depends on w, w', and Δ. Given a word $v = a_1 \dots a_n$ and the corresponding (w, w')-trace $\pi = \gamma_0 \dots \gamma_n$, we let m be the position of the first occurrence in v of a value from $D \setminus \Delta$, we let b be a value from Δ that occurs neither in the memory content $\bar{a}_{m-1}$ nor in the memory content $\bar{a}'_{m-1}$ (note that such a value exists since $|\Delta| \geq |\bar{a}_{m-1}| + |\bar{a}'_{m-1}|$), and we accordingly define

$$f(\pi) \;=\; (\gamma_0 \dots \gamma_{m-1}) \cdot \tau_{a_m, b}(\gamma_m \dots \gamma_n)$$

where $\tau_{a_m,b}(\gamma_m \dots \gamma_n)$ is the sequence obtained from $\gamma_m \dots \gamma_n$ by substituting, in every position $i \in \{m, \dots, n\}$ and in both memory contents $\bar{a}_i$ and $\bar{a}'_i$, every occurrence of a_m by b and, vice versa, every occurrence of b by a_m. By a slight abuse of notation, we also denote by $f(v)$ the word $(a_1 \dots a_{m-1}) \cdot \tau_{a_m,b}(a_m \dots a_n)$, where v, m and b are defined as above.

We now prove that $f(\pi)$ is the (w, w')-trace of $f(v)$. For every index $0 \leq i \leq n$, we denote by $(q_i, \bar{a}_i)$ the configuration reached by $\mathcal{A}$ after reading the word $w \cdot (a_1 \dots a_i)$. Similarly, for every index $m-1 \leq i \leq n$, we denote by $(p_i, \bar{b}_i)$ the configuration reached by $\mathcal{A}$ after reading the word $w \cdot (a_1 \dots a_{m-1}) \cdot \tau_{a_m,b}(a_m \dots a_i)$. We can shortly write

$$(q_I, \varepsilon) \xrightarrow[\mathcal{A}]{w} (q_0, \bar{a}_0) \xrightarrow[\mathcal{A}]{a_1 \dots a_{m-1}} (q_{m-1}, \bar{a}_{m-1}) \xrightarrow[\mathcal{A}]{a_m \dots a_n} (q_n, \bar{a}_n)$$
$$(q_I, \varepsilon) \xrightarrow[\mathcal{A}]{w} (q_0, \bar{a}_0) \xrightarrow[\mathcal{A}]{a_1 \dots a_{m-1}} (p_{m-1}, \bar{b}_{m-1}) \xrightarrow[\mathcal{A}]{\tau_{a_m,b}(a_m \dots a_n)} (p_n, \bar{b}_n).$$

We now show by induction on i that that $p_i = q_i$ and $\bar{b}_i = \tau_{a_m,b}(\bar{a}_i)$ for all $i = m-1, \dots, n$. The base case $i = m-1$ is trivial, because both a_m and b do not occur in $\bar{a}_{m-1}$. As for the induction case, suppose that $p_{i-1} = q_{i-1}$ and $\bar{b}_{i-1} = \tau_{a_m,b}(\bar{a}_{i-1})$. If $a_i \neq a_m$ and $a_i \neq b$, then it immediately follows that $p_i = q_i$ and $\bar{b}_i = \tau_{a_m,b}(\bar{a}_i)$. If $a_i = a_m$, then, by definition of m and b, we have

$$\bar{a}_{i-1} \cdot a_i \;=\; \bar{a}_{i-1} \cdot a_m \;\cong\; \tau_{a_m,b}(\bar{a}_{i-1} \cdot a_m) \;=\; \bar{b}_{i-1} \cdot \tau_{a_m,b}(a_i).$$

This shows that the same transition rule of $\mathcal{A}$ is activated in the configuration $(q_{i-1}, \bar{a}_{i-1})$ and in the configuration $(p_{i-1}, \bar{b}_{i-1})$ while consuming, respectively, the value a_i and the value $\tau_{a_m,b}(a_i)$. Therefore, $p_i = q_i$ and $\bar{b}_i = \tau_{a_m,b}(\bar{a}_i)$ follow. The argument for the remaining case $a_i = b$ is symmetric.

This shows that $w \cdot v \mathrel{\hat{\cong}_{\mathcal{A}}} w \cdot f(v)$ according to Definition 2.6. We have shown in the proof of Theorem 2.1 that this implies that $w \cdot v \mathrel{\hat{\cong}_{L(\mathcal{A})}} w \cdot f(v)$. Note that, by definition of $f(v)$, the position of the first occurrence in $f(v)$ of a value from $D \setminus \Delta$ is *strictly greater* than the position of the first occurrence in v of a value from $D \setminus \Delta$. Hence we can iterate this construction to obtain a word $u \in \Delta^*$ such that $w \cdot v \mathrel{\hat{\cong}_{L(\mathcal{A})}} w \cdot u$. The argument that $w' v \mathrel{\cong_{L(\mathcal{A})}} w' u$ is symmetric. □

We will use Lemma 2.9 in many of the proofs below. Hence we have to give a name to the finite alphabet Δ.

Definition 2.10. *The alphabet Δ that exists for each pair of words (w, w') due to Lemma 2.9, is called a* (w, w')-surrogate alphabet*; the word $u \in \Delta^*$ that exists for words w, w', and v is called the* (w, w')-surrogate word *of v.*

The following propositions give the formal statements that the memorability problem and the word-equivalence problem are decidable.

Lemma 2.11. *The memorability problem can be decided in deterministic space* $\mathcal{O}(n \cdot (2k+1)^{2k})$*, where* n *is the number of control states of the input DMA* $\mathcal{A}$ *and* k *is maximum number of values stored by* $\mathcal{A}$*.*

Proof. Let $\mathcal{A}$ be a DMA recognizing a language L and using n control states and at most k stored values. We first prove that:

> A value a is L-memorable in a word w iff there is an alphabet Δ of size $2k+1$ such that a is $L \cap \Delta^*$-memorable in w.

Here, by a slight abuse of terminology, we adapt Definition 2.3 to languages over finite alphabets; precisely, we say that a value a is $L \cap \Delta^*$-memorable in w iff there exist a values $c \in \Delta$ not appearing in w such that $\tau_{a,c}$ is R-preserving over the symbols in w and $w \not\equiv_{L \cap \Delta^*} \tau_{a,c}(w)$.

The direction from right to left is trivial. Now assume that a is memorable in w. Then there is a value b not appearing in w such that $\tau_{a,b}(w) \not\equiv_L w$. This means that there is a word v such that $\tau_{a,b}(w)v \not\equiv_L wv$. We define Δ to be the $(w, \tau_{a,b}(w))$-surrogate alphabet and let u be the $(w, \tau_{a,b}(w))$-surrogate word of v. Then we have that

$$\tau_{a,b}(w)\, u =_L \tau_{a,b}(w)\, v \not\equiv_L w\, u =_L w\, u.$$

This means that $\tau_{a,b}(w) \not\equiv_L w$ and therefore a is $L \cap \Delta^*$-memorable in w.

Hence in order to decide whether the value a is $L \cap \Delta^*$-memorable in w it is sufficient to analyze the finite automaton accepting $L \cap \Delta^*$. The set $Q_{\mathcal{B}}$ of states of $\mathcal{B}$ can be obtained from the set of configurations of $\mathcal{A}$ that store only values in Δ. Thus $|Q_{\mathcal{B}}|$ must be smaller or equal to $n \cdot |\Delta|^k = n \cdot (2k+1)^k$, where n is the number of states of $\mathcal{A}$ and k is the number of $\mathcal{A}$'s registers. In order to decide whether the value a is $L \cap \Delta^*$-memorable in w it is sufficient to provide an upper bound to the length of a shortest word $u \in \Delta^*$ (if there is any) such that $w\, u \not\equiv_L \tau_{a,b}(w)\, u$. Let us assume that a is $L \cap \Delta^*$-memorable in w and let $u \in \Delta^*$ be a word among the shortest ones witnessing $w\, u \not\equiv_L \tau_{a,b}(w)\, u$. Clearly the length of u must be smaller than the number $|Q_{\mathcal{B}}|$. Hence we can check that a value a is L-memorable for a word w in space $\mathcal{O}(n \cdot (2k+1)^{2k})$. □

We can now show that the word equivalence problem is decidable.

Lemma 2.12. *The word-equivalence problem is decidable in deterministic space $\mathcal{O}(n \cdot (2k+1)^{2k})$, n is the number of control states of the input DMA $\mathcal{A}$ and k is maximum number of values stored by $\mathcal{A}$.*

Proof. Let $\mathcal{A}$ be a DMA accepting language L over $(D, \sim)$ that has n states and k registers. Let Δ be the (ϵ, ϵ)-surrogate alphabet and let $\tilde{w}, \tilde{w}' \in \Delta^*$ by the (ϵ, ϵ)-surrogate words of w, w' respectively. By Lemma 2.9 we have that $\tilde{w} \equiv_L w$ and $\tilde{w}' \equiv_L w'$. We now show that

$$w \mathrel{\hat{\equiv}}_L w' \quad \text{iff} \quad \tilde{w} \mathrel{\hat{\equiv}}_{L \cap \Delta^*} \tilde{w}'.$$

Here we assume that $\tilde{w} \mathrel{\hat{\equiv}}_{L \cap \Delta^*} \tilde{w}'$ is defined as follows: there is a bijection $\tilde{\tau}$ on Δ such that for all words $\tilde{u} \in \Delta$, $\tilde{w}\,\tilde{u} =_{L \cap \Delta} \tilde{\tau}(\tilde{w}')\,\tilde{u}$.

The direction from left to right is clear. To show the other direction we assume that $w \not\equiv_L w'$. Thus for all data renamings τ over D there is a word u such that $wu \neq_L \tau(w')u$. We now show that $\tilde{w} \not\equiv_{L \cap \Delta^*} \tilde{w}'$. Let some data renaming $\tilde{\tau}$ on Δ be given. As $w \not\equiv_L w'$ there is a word u such that $wu \neq_L \tilde{\tau}(w')u$. Let Γ be the $(\tilde{w}, \tilde{\tau}(\tilde{w}'))$-surrogate alphabet and let $\tilde{u}$ be the $(\tilde{w}, \tilde{\tau}(\tilde{w}'))$-surrogate word of u. Recall from the definition of the surrogate alphabet that $\Gamma = \Gamma_1 \uplus \Gamma_2$ where Γ_1 is the union the register contents of $\mathcal{A}$ after reading $\tilde{w}$ and $\tilde{w}'$ respectively and Γ_2 is some (arbitrary) set of symbols. As $\Gamma_1 \subseteq \Delta$, Γ_2 is an arbitrary set, and $|\Gamma| = |\Delta|$, we can assume that $\Gamma = \Delta$. Then

$$\tilde{w}\tilde{u} \equiv_L \tilde{w}u =_L wu \neq_L \tilde{\tau}(w')u =_L \tilde{\tau}(\tilde{w}')u \equiv_L \tilde{\tau}(w')\tilde{u}.$$

As $\tilde{w}$, $\tilde{w}'$, and $\tilde{u}$ are all words over the alphabet Δ, we get that $\tilde{w} \not\equiv_{L \cap \Delta^*} \tilde{w}'$.

Thus it is sufficient to check whether $\tilde{w} \mathrel{\hat{\equiv}}_{L \cap \Delta^*} \tilde{w}'$. Again we will analyze the finite automaton accepting $L \cap \Delta^*$. Recall from the proof of Lemma 2.11 that this automaton has size $n \cdot (2k+1)^k$, where n is the number of states of $\mathcal{A}$ and k is the number of $\mathcal{A}$'s registers. The automaton will consist of a main loop that will test for each permutations τ on Δ whether $\tilde{w} \equiv_{L \cap \Delta^*} \tau(\tilde{w}')$. The permutation τ can be stored in space $\Delta^2 = (2k+1)^2$. The test whether $\tilde{w} \equiv_{L \cap \Delta^*} \tau(\tilde{w}')$ can be performed by checking membership of bounded words in $L \cap \Delta^*$. We omit the formal argument here since it is very similar to the one in the proof of Lemma 2.11. It follows that we can solve the memorability problem in deterministic space $\mathcal{O}(n \cdot (2k+1)^{2k})$. □

We can finally prove Theorem 2.3.

Proof of Theorem 2.3. We analyze the space complexity of Algorithm 1. Assume that the input DMA $\mathcal{A}$ has n states and r registers. The set Σ has size 2^{k+1} and hence can be stored in space 2^{k+1}. The set Q is empty initially and will only contain words that are $\hat{\equiv}_L$ distinct. Hence Q is always smaller than n. Thus the space consumption of Algorithm 1 is dominated by the space consumption of the subcomputations for the memorability problem and the word-equivalence problem. We have shown in Lemmas 2.11 and 2.12 that both problems can be solved in space $\mathcal{O}(n{\cdot}(2k{+}1)^{2k})$. □

2.7 Conclusions

We provided a Myhill-Nerode-style theorem that characterizes the class of DMA-recognizable languages in those cases where the underlying alphabet is equipped with either an identity or a (possibly non-dense) total order. As in the classical Myhill-Neorde theorem over finite alphabets, in one direction of the proof we construct a DMA from a given language L in a way that depends only on L. We then show that this automaton – the canonical automaton for L – is minimal in a strong sense: it has the minimal number of states and also the minimal amount of internal storage.

We then show, for the special case where the underlying support contains only the identity, how this minimal automaton can be computed. The basic idea of the minimization algorithm is to compute from a given DMA $\mathcal{A}$ a deterministic finite automaton $\mathcal{B}$ that contains all the information of $\mathcal{A}$. This leads to a minimization algorithm that requires space polynomial in the number of states of $\mathcal{A}$ and exponential in the number of registers of $\mathcal{A}$. One open question is whether this upper bound can be improved. For example it might not be necessary to maintain the entire automaton $\mathcal{B}$, but instead one could try to compute relevant parts of $\mathcal{B}$ whenever required.

Another open question is whether our minimization procedure can be extended to automata over data domains that contain an order or a more general relation. In addition, more general models of automata could be taken into account. One could consider automata with a bounded number of registers, but where each register can store a set of unbounded size instead of a single value. On a transition such an automaton could check which of the stored sets contains the current input value and update its configuration accordingly. These kinds of automata seem to be closely related to class memory automaton [BS10]. It would be interesting to know whether such automata can also be minimized effectively.

Chapter 3

Which Data Languages Can be Defined by Logics?

3.1 Introduction

Having shown that deterministic finite memory automata can be minimized effectively, we continue our exploration of properties of the regular languages that are inherited by automata for data word languages. We now tackle the relationship between logics and automata. We will show that definability in logics is decidable for many languages that are accepted by deterministic finite memory automata. We will also show that this decidability is absent from most automata models that are stronger than deterministic finite memory automata.

One of the first results concerning the relationship of logics and automata is due to Büchi [Büc60]. He showed that on words over a finite alphabet, monadic second-order logic has the same expressiveness as deterministic finite automata. Hence the question arises which class of automata corresponds to first-order logic. The answer was given by Schützenberger, McNaughton, and Papert. It follows from their results that a language L can be defined in first-order logic iff the minimal automaton accepting L does not contain special cycles called counters [Sch65, MP71]. Their results also provide an *effective characterization* of first-order logic: that is, an algorithm that decides whether a given regular language can be defined in first-order logic.

In addition, the theorem of Schützenberger, McNaughton, and Papert reveals a surprising connection between formal languages and algebraic structures. Monoids, basically sets with an associative operation defined on them, can be used as devices that accept formal languages. One can easily show that a language is accepted by a finite monoid iff it is accepted by a finite automaton. The results of Schützenberger,

McNaughton, and Papert show that a language can be defined in first-order logic iff it is accepted by a finite monoid that satisfies an additional property called aperiodicity.

These results have been the starting point for a thorough investigation into the relationship between subclasses of finite monoids and other formalisms for defining regular languages. It has been shown that many such formalisms correspond to natural subclasses of finite monoids. Examples include languages defined by fragments of first-order logic, various temporal logics, and parallel complexity classes.

We consider the possibility to decide membership in logics for data word languages. More precisely, for a logic $\mathcal{L}$ and a class of automata or monoids $\mathcal{M}$ for data word languages, we want to present algorithms that can decide questions of the form:

> *Given a language L that is accepted by an automaton or a monoid in $\mathcal{M}$, can the language L be defined in the logic $\mathcal{L}$?*

If such an algorithm exists, then we say that there is an *effective characterization* of $\mathcal{L}$ with respect to $\mathcal{M}$.

We first show that one cannot hope to characterize effectively logical classes for many models of automata on infinite alphabets – we show this for non-deterministic finite memory automata, for two-way deterministic finite memory automata, and for a very weak version of pebble automata.

We thus focus on deterministic finite memory automata, as well as two restrictions of them. The first restriction is to allow the automaton to only perform "local" data comparisons within a fixed length suffix of the part of the word that has beed processed at any point. These "local" automata behave very much like finite automata, but they are expressively rather weak. For example, languages like "the value that appears on the leftmost a-labelled position must coincide with the value at the rightmost a-labelled position" cannot be accepted by a local automaton. The second restriction we consider are monoids that accept data languages. Finite monoids are even weaker than local automata, for example the language $\{ab \in D^2 \mid a = b\}$ cannot be accepted by a finite monoid [BPT03]. Infinite monoids, on the other hand, can accept any language, and are therefore much too expressive to allow for effective characterizations. A class of infinite, but well behaved monoids, called *orbit finite data monoids*, has been introduced in [Boj11]. Their expressiveness lies strictly between local deterministic finite memory automata and (unrestricted) deterministic finite memory automata.

We will also look at several logics for data languages. First-order logic (FO) for data words is built up using first-order quantification, boolean operators, and binary predicates for comparing positions in a word. These predicates are usually the ordering on word positions $<$ and a predicate $\sim$ for checking data equality. Monadic second-order logic (MSO) is, as usual, the extension of first-order logic by second order quantification. We consider restrictions of these logics in which the data equality predicate $x \sim y$ must be guarded. That is, $x \sim y$ can only appear in conjunction with another formula $\phi(x, y)$ in which x and y are free. We distinguish two kinds of guards: local guards of the form $x + k = y$ for some $k \in \mathbb{N}$, that guarantee that x and y must have a fixed distance k; and "rigid guards" which are formulas that guarantee that the position x determines the position y in every word, and vice versa.

We also distinguish two modes of definability: We say that a language L can be defined "non-uniformly" in a logic $\mathcal{L}$ if for each number n there is an $\mathcal{L}$-formula ϕ_n that defines the restriction $L \cap \Delta_n^*$ of L to an alphabet Δ_n of size n. Uniform definability is the usual notion where the same formula is used for all sizes of the alphabet. Clearly any language that can be defined uniformly can be defined non-uniformly in the same logic, but the converse is not true in general.

In our first contribution, we consider locally guarded logics. We show that locally guarded monadic second-order logic has the same expressiveness as local deterministic finite memory automata. We then show that it is decidable whether the language accepted by a deterministic finite memory automaton can be defined in local FO. Next, we give a method for deciding non-uniform FO definability for two special classes of deterministic finite memory automata – those with at most one register and local ones. These two classes of automata are incomparable in expressiveness.

The main proof techniques used to obtain the results described above are minimization of deterministic finite memory automata and Ehrenfeucht-Fraïssé-games. In the results that we will describe below we use algebraic techniques. We are able to do this, because we assume that the input language is accepted by an orbit finite data monoid. In fact, orbit finite data monoids were invented partially because they allow for a smooth generalization of the algebraic theory of regular languages.

We start with first-order logic with two variables (FO^2). A classic result of Thérien and Wilke states that a regular language can be defined by FO^2 iff it is recognized by a finite monoid in a class of monoids called "DA" [TW98]. We extend this result to plain data word languages. That is, we consider the data words variant of FO^2 and we show that an orbit finite data monoid recognizable language can be

defined in FO^2 iff it is accepted by an orbit finite data monoid in DA. We also show that membership of an orbit finite data monoid in DA is decidable, thereby providing and effective characterization of FO^2 with respect to orbit finite data monoids.

We then consider first-order logic with an arbitrary number of variables. Bojańczyk has shown that every language that is accepted by an aperiodic orbit finite data monoid can be defined in first-order logic [Boj11]. This theorem is similar to the combination of the theorems of Schützenberger, McNaughton, and Papert, but the target language is very strong: There are languages that can be defined in first-order logic that cannot even be accepted by deterministic finite memory automata.

We show that rigid guards reduce the expressivity of FO in such a way that the resulting logic corresponds *exactly* to aperiodic orbit finite data monoids:

> *A language is accepted by an aperiodic orbit finite data monoid*
> *iff it can be defined in rigidly-guarded first-order logic.*

We then consider the question whether, as in the classical (finite alphabet) theory, there is a natural monadic second-order logic that corresponds to orbit finite data monoids. We answer this question positively by proving the following:

> *A language is accepted by an orbit finite data monoid*
> *iff it can be defined in rigidly-guarded monadic second-order logic.*

We then turn to the question whether there is a natural variant of monadic second-order logic which defines precisely the data languages accepted by finite memory automata. We answer this positively as well. ∃backwards rigidly-guarded monadic second-order logic (∃BRMSO) is obtained from rigidly-guarded monadic second-order logic by weakening the guardedness condition. We show

> *A language is accepted by a non-deterministic finite memory automaton*
> *iff it can be defined in ∃BRMSO.*

It follows from our characterization results that these three logics are decidable. The algorithms for checking satisfiability that can be extracted from our proofs have non-elementary complexity. As satisfiability of first-order logic on finite alphabets is already non-elementary [Sto74], these results show that the complexity of first-order satisfiability does not deteriorate when adding rigidly guarded equality.

Most results in this chapter have been obtained in collaboration with Michael Benedikt, Thomas Colcombet, and Gabriele Puppis [BLP10a, CLP11].

Organization. This chapter is organized as follows: Section 3.2 contains preliminaries and prior machinery, more precisely it contains a review of previous work on the relationship between logics, automata and monoids over finite alphabets; it also contains the definitions of the logics, automaton, and monoids for data languages that we will study in this chapter. Section 3.3 summarizes the main results of this chapter and Section 3.4 compares our results with previous work. Section 3.5 contains our results on characterizations of variants of first-order logic. This section contains both our characterizations with respect to automata as well as those with respect to monoids. Section 3.6 studies characterizations of variants of monadic second-order logic in terms of monoids and automata. Section 3.7 concludes.

3.2 Preliminaries and Prior Machinery

We now provide some background material needed in this chapter. We start by reviewing previous work on effective characterizations of languages over finite alphabets (Section 3.2.1). Then we turn back to data word languages and we define some variants of first-order logic (Section 3.2.2). In Section 3.2.3, we consider a restriction of finite memory automata called 'local automata'. Next, we define a class of monoids, called orbit finite data monoids, whose expressiveness lies strictly between local automata and deterministic finite memory automata. Finally, in Section 3.2.4 we compare the expressiveness of automata, logics, and monoids for data languages.

3.2.1 Algebraic Language Theory

As noted in the introduction, monoids can be used as acceptors for formal languages. Formally a monoid $(M, \cdot, 1)$ consists of a set M, a binary operation $\cdot$ on M, and a distinguished element $1 \in M$ called the *identity* of M. One requires that a monoid satisfies the following conditions:

- *Associativity.* $x \cdot (y \cdot z) = (x \cdot y) \cdot z$ for all $x, y, z, \in M$
- *Identity.* $1 \cdot x = x \cdot 1 = x$ for all $x \in M$

An example of a monoid is the *free monoid A^* over a set A.* The elements of A^* are the finite words over A, the operation is concatenation of words, and the empty word plays the role of the identity. A slightly more interesting example is the *syntactic monoid* of a language L. We can define a two-sided version of the Myhill-Nerode

	$[\varepsilon]$	$[a]$
$[\varepsilon]$	$[\varepsilon]$	$[a]$
$[a]$	$[a]$	$[\varepsilon]$

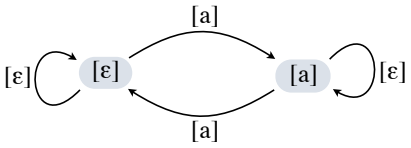


Figure 3.1: The syntactic monoid of the language $(aa)^*$ over the alphabet $\{a\}$. The left hand figure shows its multiplication table and the right hand figure shows its Caley graph. We drop $\equiv_L$ as a subscript.

equivalence $\equiv_L$ of a language L as follows: Two words u, v are equated by $\equiv_L$ iff for all words w, w'

$$wuw' \in L \quad \text{iff} \quad wvw' \in L.$$

The syntactic monoid $\mathcal{M}_L$ of L consists of the equivalence classes of $\equiv_L$, concatenation is defined by $[u]_{\equiv_L} \cdot [v]_{\equiv_L} = [uv]_{\equiv_L}$, and the $\equiv_L$-class of the empty word is the identity element.

Note that we used the same symbol $\equiv_L$ for both the one sided and the two sided Myhill-Nerode equivalence. This will never lead to misunderstandings, because in the rest of this thesis we will *only* consider the two sided version of the Myhill-Nerode equivalence.

There are two basic ways in which a finite monoid $\mathcal{M} = (M, \cdot, 1)$ can be presented. One way is, obviously, to provide its multiplication table. A second possibility to present $\mathcal{M}$ by a directed edge labelled graph, called the *Caley graph*. The nodes of this graph are the elements of M and there is a directed edge from x to y labelled z iff $x \cdot z = y$. Figure 3.1 shows the multiplication table and the Caley graph of the syntactic monoid of the language $(aa)^*$.

To define the notion of acceptance by a monoid, we need to define morphisms between monoids. A *(monoid) morphism* from a monoid $(M, \cdot, 1)$ to a monoid $(N, \odot, e)$ is a mapping $h : M \to N$ such that $h(x \cdot y) = h(x) \odot h(y)$ for all $x, y \in M$ and $h(1) = e$. We say that a language $L \subseteq A^*$ is *accepted* by a monoid $(M, \cdot, 1)$ iff there is a homomorphism from the free monoid A^* to $(M, \cdot, 1)$ and a subset $F \subseteq M$ such that $w \in L$ iff $h(w) \in F$. Basically, the idea is that language membership of a word w in L depends only on the image of w under h.

For example, every language L is accepted by its syntactic monoid: The morphism is the *syntactic morphism* h_L, that is the morphism that maps each word u to its $\equiv_L$-class $[u]_{\equiv_L}$. The set F is defined as the set of $\equiv_L$-classes that contain words in L. It is easy to show that the syntactic monoid of a language L is the smallest monoid accepting L.

A basic theorem states that a language over a finite alphabet is accepted by a finite monoid iff it is regular. This result motivates the following question:

> Which subclasses of regular languages are accepted by which subclasses of finite monoids?

An early and most remarkable result is Schützenberger's Theorem. A regular expression is *star-free* if it is constructed from the letters of the alphabet, the empty set symbol, boolean operators and concatenation, but no Kleene star. Schützenberger showed that a language can be defined by a star-free regular expression iff it is accepted by an aperiodic monoid [Sch65]. Here, a monoid $(M, \cdot)$ is *aperiodic* iff there is an $n \in \mathbb{N}$ such that $x^n = x^{n+1}$ for all $x \in M$. Intuitively, a monoid is aperiodic if it cannot count modulo classes. Also, aperiodic monoids are precisely those whose Caley graphs only have cycles of length 1. Note that the monoid in Figure 3.1 is not aperiodic.

The connection between star free regular expressions and first-order logic was discovered a few years later. McNaughton and Papert showed that a language can be defined by a star-free regular expression iff it can be defined in first-order logic [MP71]. Here, first-order logic has a predicate for every symbol in the (finite) alphabet and a binary predicate $x < y$ to test whether the position corresponding to x is to the left of the position corresponding to y.

McNaughton and Papert have also given a characterization of the class of first-order definable languages in terms of finite automata [MP71]. They have shown that a language can be defined in first-order logic iff it is accepted by a counter-free finite automaton. A *counter* is a sequence $q_1, \ldots, q_n$ of states of a finite automaton such that $n \geq 2$ and there is a word w for which there is a path labelled w leading from q_n to q_1 and from q_i to q_{i+1} for all $i < n$. An automaton is counter free if it does not have a counter. Wilke has given a direct proof of the correspondence between first-order logic and counter free automata that does not use algebraic techniques [Wil99].

The above results were only the starting point for an investigation of the relationship between monoids and subclasses of regular languages. It has turned out that many natural subclasses of regular languages correspond to natural subclasses of finite monoids. Examples include, languages defined in FO(+1), that is first-order logic with successor instead of a linear order [BP89], languages defined by first-order logic with only two variables [TW98, TT02], fragments of temporal logics [Wil99, ASW09, TW04], and parallel complexity classes (see [Pin11, Str94]).

3.2.2 Logics for Data Languages

Logics for data languages fall into two main categories: Fragments of monadic second-order logics and temporal logics. We will review the former in Subsection 3.2.2 and the latter in Subsection 3.2.2. Logics can be defined for both plain and adorned data languages. We generally define logics for the adorned languages, that is over an alphabet of the form $D \times A$ where D is infinite and A is finite. Some of the results in this section are about logics for plain languages, in these cases it will always be clear how the corresponding logics are to be defined for plain languages.

Monadic Second-Order Logic and Its Fragments

Monadic second-order logic (denoted MSO or MSO$(\sim, <, +1)$) formulas are built up according to the following grammar:

$$\begin{aligned}\varphi \;:=\; & \exists x\,.\,\varphi \mid \exists X\,.\,\varphi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \\ & a(x) \mid x < y \mid x + 1 = y \mid x \in Y \mid x \sim y\end{aligned}$$

where a is an element of the finite alphabet A. The meaning of an atom $x \sim y$ is that the positions that correspond to the interpretation of x and y have the same value. The meanings of the other predicates are as usual. *First-order logic* (denoted FO or FO$(\sim, <)$) is the restriction of MSO$(\sim, <)$ that only uses first-order quantification. We use the standard notation to denote logics in with fewer predicates. For example we denote by FO$(\sim, +1)$ the restriction of FO that does not contain the predicate $x < y$.

To simplify the notation, we will sometimes write variables in uppercase letters, without explicitly saying whether these are first-order or second-order variables. In this case their correct types can be inferred from the atoms they appear in.

First-order logic is quite a strong formalism over data words. For example, the language

$$L = \{a_1 \dots a_n \in D^* \mid i \neq j \text{ implies that } a_i \neq a_j \text{ for all } i, j \leq n\}.$$

can be defined in first-order logic. On the other hand, we noted in Section 2.2.2 that this language cannot be accepted by a 1-way NMA (a proof can be found in [KF94]). As NMA can express languages like the set of even strings it follows that NMA and first-order logic are incomparable. This is in stark contrast to the case of regular languages (over a finite alphabet) where the set of languages that can be defined in first-order logic is strictly contained in the set of languages that are accepted by finite automata.

In fact, first-order logic for data words is so expressive that its satisfiability problem is undecidable [BMS+06]. The undecidability of first-order logic has motivated the search for decidable fragments. Most prominently, it has been shown that the two variable fragment of $\mathrm{FO}(\sim, <, +1)$ is decidable [BMS+06]. The decidability of first-order logic with two variables has been further investigated in [SZ11]. It has been shown that on structures with one total preorder and one linear order, first-order logic with two variables remains decidable. In contrast on structures with two total preorders and on structures with one total preorder and two linear orders, first-order logic with two variables becomes undecidable [SZ11].

Local Logics. We will consider fragments of logics that are obtained by restricting the use of the data equality predicate. In particular, we will require that each occurrence of a formula $x \sim y$ is 'guarded', that is the data equality predicate can only occur in subformulas of the form $\phi(x, y) \wedge x \sim y$. We will consider two different kinds of guards. First we consider 'local guards' of the form $x = y + k$ and later we will allow formulas that define injective relations.

Formally, we define *local monadic second-order logic* (denoted local MSO) to consist of all formulas that are built up according to the following grammar:

$$\begin{aligned}\varphi \;:=\; & \exists x\,.\,\varphi \mid \exists X\,.\,\varphi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \\ & a(x) \mid x < y \mid x + 1 = y \mid x \in Y \mid x + l = y \;\wedge\; x \sim y\end{aligned}$$

where a is an element of the finite alphabet A and $l \in \mathbb{N}$. We will often abbreviate the formula $x + l = y \wedge x \sim y$ by $x \sim_l y$. *Local first-order logic* (LFO) is the first-order fragment of local $\mathrm{MSO}(\sim, <)$.

We will consider restrictions of local MSO where only a finite number of predicates $\sim_i$ are available. Accordingly we denote by $\mathrm{MSO}(\sim_{\leq l}, <)$ the logic with the predicates $<$ and $\sim_i$ for all $i \leq l$.

Example 3.1. *An example of a local FO formula is* $\forall x, y\, \exists z\ (x \sim_5 y \to x \not\sim z)$. *It requires that if position x has the same value as its fifth neighbor x to the right, then there is a position z that has a different value than x and y.*

It is easy to see that, for each $l \in \mathbb{N}$, the language $L_l = \{a_1 \dots a_n \in D^* \mid n \geq l \text{ and } a_1 = a_n\}$ can be defined in $\mathrm{FO}(\sim_{\leq l}, <)$, but not in $\mathrm{FO}(\sim_{\leq l-1}, <)$. Hence the family of logics $(\mathrm{FO}(\sim_{\leq l}, <))_{l \in \mathbb{N}}$, forms an infinite strictly increasing hierarchy of expressiveness. Note also that $\mathrm{FO}(\sim, +1)$ can express properties like "the first letter is equal to the last letter", which cannot be expressed in $\mathrm{FO}(\sim_{\leq l}, <)$ for any $l \in \mathbb{N}$. An overview over local logics is shown in Figure 3.2.

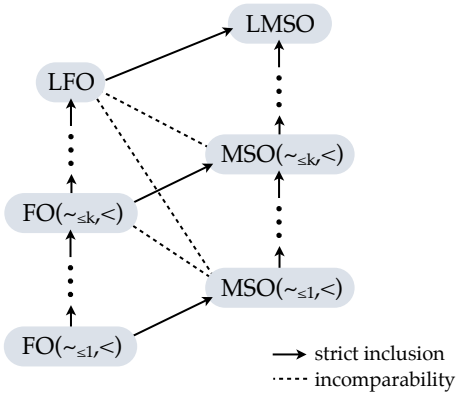


Figure 3.2: Two infinite hierarchies of expressiveness within local logics.

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

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

$$\begin{aligned}\varphi \;:=\; & \exists x\,.\,\varphi \mid \exists X\,.\,\varphi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \\ & a(x) \mid x < y \mid x + 1 = y \mid x \in Y \mid \varphi_{\text{rigid}}(x, y) \wedge x \sim y\end{aligned}$$

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

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

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

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

We will also see in Corollary 3.13 below that one can decide if a formula respects the rigidity assumption in all its guards.

Example 3.2. *Let us consider the language $L_{\geq k}$ of all data words that contain at least k different data values. If $k = 1$ we just need to check the non-emptiness of the word by the sentence $\exists x \,.\, true$. For $k = 2$ it is sufficient to test for the existence of two distinct consecutive data values, using for instance the formula $\exists\, x, y\ (x + 1 = y) \wedge x \not\sim y$. For $k > 2$, one can proceed by induction as follows. One first observes that if a word has at least k distinct data values, then there is a minimal factor witnessing this property, say $[x, y]$. A closer inspection reveals that, in this case, $[x+1, y-1]$ is a maximal factor that uses exactly $k-2$ data values and thus belongs to the language $L_{\geq k-2} \setminus L_{\geq k-1}$. By induction, the fact that $[x + 1, y - 1]$ is a maximal factor that belongs to $L_{\geq k-2} \setminus L_{\geq k-1}$ is expressible in rigidly guarded FO by a formula $\varphi(x, y)$. Furthermore, this formula $\varphi(x, y)$ is rigid according to its semantic definition. We conclude that the language $L_{\geq k}$ is defined by the formula $\exists\, x, y\, \varphi(x, y) \wedge x \not\sim y$.*

Note that the language $L_{\geq k}$ cannot be defined in local first-order logic. A simpler language separating these two logics is $L_{1\sim n} = \{a_1, \ldots, a_n \in D^n \mid a_1 = a_n\}$. In fact $L_{1\sim n}$ also separates local MSO from rigidly guarded MSO. In addition, it follows that local MSO and rigidly guarded FO are incomparable.

We will later show that RMSO is decidable (see Corollary 3.13). Hence RFO is strictly contained in FO, RMSO is strictly contained in MSO, and FO and RMSO are incomparable.

Extensions of Rigidly Guarded MSO. We will show that two extension of rigidly guarded MSO remain decidable. The first extension relaxes the notion of rigidity. In particular, we allow guards of the form $\varphi(x, y)$ where rightmost position $\max(x, y)$ determines the leftmost position $\min(x, y)$ (but possibly not the other way around). Formulas of *backward-rigidly guarded MSO* (BRMSO) are built up using the grammar:

$$\begin{aligned}\varphi \;\coloneqq\; & \exists x \,.\, \varphi \mid \exists X \,.\, \varphi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \\ & a(x) \mid x < y \mid x + 1 = y \mid x \in Y \mid \varphi_{\text{backward}}(x, y) \wedge x \sim y\end{aligned}$$

where a is an element of the finite alphabet A and $\varphi_{\text{backward}}$ is a backward-rigid formula generated from the same grammar (as usual, we can enforce backward-rigidity syntactically).

It is easy to see that BRMSO is more expressive than RMSO. For example the language $\{a_1 \ldots a_n \in D^* \mid a_1 = a_i \text{ for some } i \leq n\}$ can be defined in BRMSO

but not in RMSO. On the other hand, we will show that the languages defined by BRMSO are strictly contained in the languages accepted by deterministic finite memory automata (see Proposition 3.73).

The second extension, called ∃*backward-rigidly guarded MSO* (denoted ∃BRMSO), consists of the formulas $\exists \bar{Z} \,.\, \varphi$, with φ is generated by the grammar

$$\begin{aligned} \varphi \;\coloneqq\; & \exists x \,.\, \varphi \mid \exists X \,.\, \varphi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \\ & a(x) \mid x < y \mid x + 1 = y \mid x \in Y \mid \varphi_{\exists\text{backward}}(x, y, \bar{Z}) \wedge x \sim y \end{aligned}$$

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

One of the main results of this thesis is that ∃BRMSO has exactly the same expressiveness as non-deterministic finite memory automata.

Non-Uniform Logics. For each $n \in \mathbb{N}$, let D_n be a subset of D consisting of exactly n elements. We say that a language $L \subseteq D^*$ is definable in *non-uniform* $FO(\sim, <)$, abbreviated NUFO$(\sim, <)$, if for each $n \in \mathbb{N}$, the language $L_n = L \cap D_n^*$ can be defined in FO$(\sim, <)$ (under the assumption that input words are over the finite alphabet D_n). Non-uniform variants of the other logics considered above are defined similarly.

Clearly every language in FO$(\sim, <)$ can be defined in NUFO$(\sim, <)$ (simply by using the same formula to define each L_n for $n \in \mathbb{N}$). The next example shows that the converse is not true.

Example 3.3. *Consider the language $L_{2\times}$ of all words w whose length is two times the number of distinct values that occur in w. $L_{2\times}$ cannot be defined in $FO(\sim, <)$, but it is definable in NUFO$(\sim, <)$ since $L_{2\times} \cap D_n^*$ is finite for every $n \in \mathbb{N}$.*

On the other hand, LMSO can clearly define languages that cannot be defined in NUFO, for example the language $\{a_1 \dots a_n \in D^* \mid a_i = a_j \text{ for all } i, j \leq n \text{ and } n \text{ is even}\}$. Figure 3.3 gives an overview over the relationship between several logics considered so far.

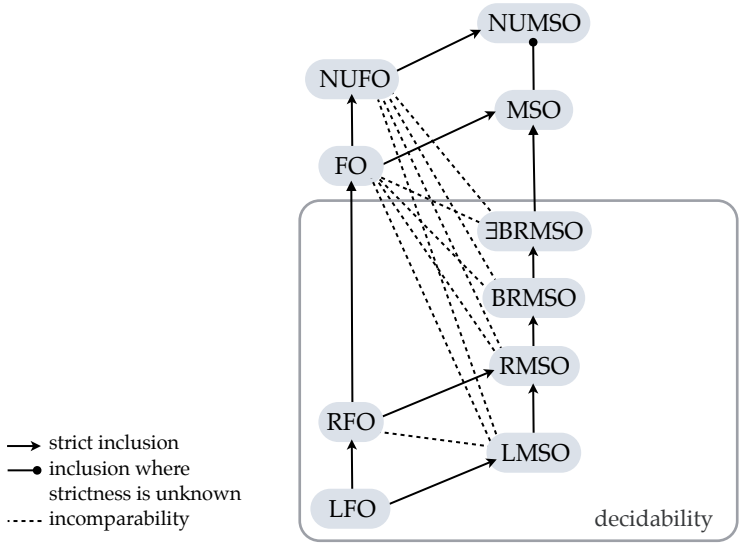


Figure 3.3: An overview over some logics for data words.

LTL with Freeze Quantification

As our focus in this thesis are fragments of MSO, we only briefly mentions the most important results on temporal logics for data languages. Demri and Lazić have investigated linear temporal logic (LTL) over data words [DL09]. More precisely, they consider LTL$^{\downarrow}$, built up from the modalities X (next), F (eventually), and U (until), extended with a freeze quantifier $\downarrow_r \phi$ and a reference to a register $\uparrow_r$. A LTL$^{\downarrow}$ formula is evaluated with respect to a word and a register assignment. Roughly a freeze quantifier $\downarrow_r$ can store a data value in register r. The formula $\uparrow_r$ is true at position i of a word if the value of position i is stored in register r of the register assignment.

Demri et al. show that satisfiability of LTL$^{\downarrow}(X, U)$ with one register is decidable [DL09]. If one more register is added to LTL$^{\downarrow}(X, U)$ then the logic becomes undecidable. In addition, the logic LTL$^{\downarrow}(X, F, F^{-1})$, obtained from LTL$^{\downarrow}(X, U)$ by replacing the until U operator with the weaker eventually operator F and adding the 'eventually in the past' operator F^{-1}, is undecidable [DL09].

Over finite alphabets, an important result is Kamp's theorem [Kam68], an extension of which shows that LTL(X, U) is as expressive as first-order logic sentences. Over words, this is not the case: LTL$^{\downarrow}(X, U)$ with two registers is incomparable with FO$^2(\sim, <, +1)$ over data words [DL09].

3.2.3 Data Monoids, Local Automata, Window Automata

We now introduce yet more automata models for data languages. We will define restrictions of finite memory automata, called 'local automata', and we will later show that they accept languages that correspond to local logics. Then we define a class of infinite monoids that accept all local languages, but also some non-local languages. These monoids are called 'orbit finite data monoids'. One of the main results of this thesis is that the languages accepted by orbit finite data monoids correspond to the languages defined in RMSO.

Automata for Local Languages

We will consider two versions of local automata: A restriction of finite memory automata that have to update *all* its registers whenever a certain fixed number of values has beed consumed; and an equivalent model of automata that have no registers at all.

Local Finite Memory Automata. We introduce a restriction of DMA, called *local DMA*. A DMA $\mathcal{A}$ is *l-local* if for all configurations $(p, \bar{a})$ of $\mathcal{A}$ and all words w of length l, if $w^{\mathcal{A}}(p, \bar{a}) = (q, \bar{b})$, then $\bar{b}$ contains only values that occur in w. A DMA is *local* if it is l-local for some $l \in \mathbb{N}$.

Proposition 3.4. *The following problem is decidable: Given a DMA $\mathcal{A}$, is $\mathcal{A}$ local? If $\mathcal{A}$ is local then we can compute the minimum number l for which $\mathcal{A}$ is l-local.*

Proof. We need some preliminary definitions. A *loop* C in a DMA $\mathcal{A}$ is a cycle of consecutive transitions in $\mathcal{A}$. A loop C is *simple* if C contains exactly one transition departing from each of its states. The *period* of a simple loop is the number of distinct states that occur in it. Let us consider a simple loop C of period m:

$$p_1 \xrightarrow{\alpha_1, E_1} \dots \xrightarrow{\alpha_{m-1}, E_{m-1}} p_m \xrightarrow{\alpha_m, E_m} p_1$$

We say that C is *local* if for all $j < \min(E_1 \cup \ldots \cup E_m)$, there is an index $i \leq m$ such that ϕ_i is the $\cong$-type of a word of the form $\bar{a} \cdot b$, with $\bar{a}(j) = b$. The intuition is that C is local iff, for every infinite word w that cycles through C and every value a that is stored at some point along the corresponding run, a occurs in every subword of w of length m. It is easy to see that $\mathcal{A}$ is local iff every simple loop in $\mathcal{A}$ is local. Moreover, the latter property can be effectively checked by examining all (finitely many) simple loops in the DMA $\mathcal{A}$.

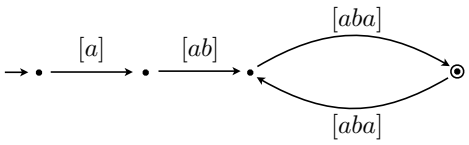


Figure 3.4: A DWA that recognizes the language $L = \bigcup_{a,b \in D}(ab)^*a$.

We now show how to compute, from a given local k-memory DMA $\mathcal{A}$, the minimum number l such that $\mathcal{A}$ is l-local. From the definition of local simple loop, it immediately follows that $\mathcal{A}$ is local only if it is $k|T|$-local, where T is the set of transitions of $\mathcal{A}$. This implies that every value that gets stored at some point along a computation of $\mathcal{A}$ is later, after at most $k|T|$ steps, either removed from the memory or "renewed" (in the sense that this value reoccurs in the input). We can thus examine all sequences of transitions of length $k|T|$ in order to find the longest sequence where a value is being stored without getting renewed. If n is the length of such a sequence, then $l = n + 1$. □

Window Automata. The class of languages recognized by local DMA can be equivalently defined using another automata model, which makes no use of memory:

Definition 3.5. *An* l-window automaton *(l-WA) is a tuple* $\mathcal{A} = (Q, q_I, F, T)$*, where* Q *is a finite set of states,* $q_I \in Q$ *is an initial state,* $F \subseteq Q$ *is a set of final states, and* T *is a finite set of transitions of the form* (p, α, q)*, where* $p, q \in Q$ *and* α *is the* $\cong$*-type of a word of length at most* l*.*

An l-WA $\mathcal{A} = (Q, q_I, F, T)$ processes an input word $w = a_1 \dots a_n$ from left to right, starting form its initial state q_I, as follows. At each step of the computation, if $\mathcal{A}$ has consumed the first i symbols of the input word and has moved to state p, and if T contains a transition of the form (p, α, q), with $q \in Q$ and $\alpha = [a_{i+2-l} \dots a_{i+1}]$, then $\mathcal{A}$ consumes the next symbol a_{i+1} of w and it moves to the target state q. The notions of successful run and recognized language are as usual.

An l-WA is *deterministic* (denoted l-DWA) if for every pair of transitions $(p, \alpha, q), (p', \alpha', q') \in T$, if $p = p'$ and $\alpha = \alpha'$, then $q = q'$. Figure 3.4 shows an example of a 3-DWA.

A *path* is a sequence of consecutive transitions in an automaton. A path π in a DWA is *realizable* if there is a word w that induces a run along π. For example, the path

$$p_0 \xrightarrow{[abc]} p_1 \xrightarrow{[aaa]} p_2$$

is not realizable: Assume that a window automaton uses the first transition to move from position i to $i + 1$ in the input word. This is only possible if the positions

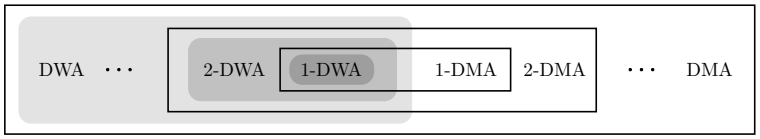


Figure 3.5: The inclusions between DMA and DWA.

$i-1, i, i+1$ have pairwise different values. Then the next transition cannot be used, as it requires that positions $i, i+1, i+2$ have the same value. A DWA $\mathcal{A}$ is *realizable* if all paths in $\mathcal{A}$ is realizable. Observe that an l-DWA is realizable iff for all transitions $(p, [a_1 \dots a_n], q)$, $(q, [b_1 \dots b_m], r)$,

1. if $n \geq m-1$, then $a_{n-m+2} \dots a_n \cong b_1 \dots b_{m-1}$
2. if $n < m-1$, then $a_1 \dots a_n \cong b_{m-n} \dots b_{m-1}$.

Hence, it is decidable whether a DWA is realizable. In addition, for every DWA, there is an equivalent realizable DWA. Note that the DWA shown in Figure 3.4 is realizable.

We now show that local DMA and DWA accept the same languages.

Proposition 3.6. *For every l-local DMA, there is an equivalent l-DWA and, vice versa, for every l-DWA, there is an equivalent l-local DMA.*

Proof. Let $\mathcal{A}$ be an l-local DMA. We sketch the construction of an l-DWA $\mathcal{B}$ equivalent to $\mathcal{A}$. From the locality property of $\mathcal{A}$, we know that every value that is stored by $\mathcal{A}$ must occur within the last l symbols of the consumed input word (hence $\mathcal{A}$ can store at most l values). We define the state space of $\mathcal{B}$ as $Q \times \{\bot, 1, \dots, l\}^l$. $\mathcal{B}$ will maintain the following invariant while processing an input word w: if $\mathcal{B}$ is in state $(q, i_1, \dots, i_l)$, then the last occurrence in w of the value stored by $\mathcal{A}$ into the memory position j (if any) is at the last but i_j-th position. Such an invariant can be guaranteed by a suitable definition of the set T of transitions of cB.

Conversely, any given l-DWA can be simulated by an l-local l-DMA that always stores the last l consumed values. □

In contrast to the above result, there is a non-local 1-DMA that recognizes the language $L = \{a_1 \dots a_n \in D^* \mid a_1 = a_n\}$, which is clearly not WA-recognizable. Figure 3.5 shows the inclusions that hold between DMA and DWA.

Data Monoids

It is known that finite monoids correspond to finite automata over finite alphabets in the sense that both formalisms accept the same class of languages. However finite monoids are very weak when used to accept languages over an infinite alphabet. For example the language $\{ab \in D^* \mid a = b\}$ cannot be accepted by a finite monoid. Hence a class of infinite, but well behaved monoids is needed if the algebraic theory of languages is to be transferred to data languages.

Bojańczyk introduced such a class [Boj11]. The idea is to consider monoids that are infinite, but in which many elements behave in a very similar way. The concept of a group action is used to model these symmetries between monoid elements.

Recall that a *group* is a monoid in which each element x has an inverse x^{-1} satisfying $x \cdot x^{-1} = x^{-1} \cdot x = 1$. A *group action* of a group $(G, \cdot, e)$ on a set X maps each group element $g \in G$ to an function $\hat{g}$ on X such that

- $\widehat{g \cdot h} = \hat{g} \circ \hat{h}$ for all $g, h \in G$, where $\circ$ denotes functional composition,
- $\hat{e}$ is the identity function on X.

Intuitively, a group action is a mapping from group elements to functions that preserves the structure of the group.

In a similar way, we can also define a group action of a group on a *monoid* (instead of a set). This action should not only preserve the structure of the group, but also of the monoid. Formally, *group action* of a group $\mathcal{G} = (G, \odot, e)$ on a monoid $\mathcal{M} = (M, \cdot, 1)$ is a function $\hat{}$ that maps every $g \in G$ to an automorphism $\hat{g}$ on $\mathcal{M}$. That is, for all $g, h \in G$ and all elements $s, t \in M$, we have

1. $\widehat{g \circ h} = \hat{g} \circ \hat{h}$,
2. $\hat{e}$ is the identity function on X,
3. $\hat{g}(s) \cdot \hat{g}(t) = \hat{g}(s \cdot t)$, and
4. $\hat{g}(1) = 1$, where 1 is the identity of $\mathcal{M}$.

Example 3.7. *Consider the free monoid $((D \times A)^*, \cdot)$ consisting of all finite words over $D \times A$ equipped with the operation of juxtaposition (the empty word ε playing the role of the identity). The group G_D of permutations on D acts on the free monoid when the action is defined by $\hat{\tau}((d_1, a_1) \dots (d_n, a_n)) = (\tau(d_1), a_1) \dots (\tau(d_n), a_n)$.*

We will define a data monoid to be a special monoid that is acted upon by a group of data renamings. Given a set $C \subseteq D$ of data values, a *(data) renaming on* C is a permutation on C that is the identity for all but finitely many values of C. We denote by G_C the set of all renamings on C.

We say that a renaming τ is a *stabilizer* of an element s of a monoid $\mathcal{M}$ acted upon by G_C, if $\hat{\tau}(s) = s$. A set $C' \subseteq D$ of data values *supports* an element s if all renamings that are the identity on C' are stabilizers of s. It is known that the intersection of two sets that support s is a set that supports s as well [Boj11, GP02]. Hence we can define *the memory* of s, denoted $\mathrm{mem}(s)$, as the intersection of all sets that support s. Note that there are finite monoids whose elements have infinite memory (see [Boj11] for an example). On the other hand, monoids that are homomorphic images of the free monoid contains only elements with finite memory. As we are interested in homomorphic images of the free monoid, we will consider monoids whose elements have finite memory (this property is called *finite support axiom* and the resulting algebraic objects *data monoids*).

Definition 3.8. *A* data monoid $\mathcal{M} = (M, \cdot, \hat{})$ *over* C *is a monoid* $(M, \cdot)$ *that is acted upon by* G_C*, in which every element has finite memory.*

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

Recall from Section 3.2.1 that the *two-sided Myhill-Nerode equivalence* $\equiv_L \subseteq D^* \times D^*$ of a language $L \subset D^*$ is defined by $u \equiv_L v$ iff for all $w, w' \in D^*$

$$w\,u\,w' \in L \quad \text{iff} \quad w\,v\,w' \in L.$$

We denote by $\hat{\equiv}_L$ the lifting of $\equiv_L$ to orbits. That is, we define

$$u \,\hat{\equiv}_L\, v \quad \text{iff} \quad \text{there is a data renaming } \tau \text{ such that } u \equiv_L \tau(v)$$

The syntactic data monoid can be defined is the obvious way:

Definition 3.9 (Syntactic Data Monoid)**.** *The* syntactic data monoid *of* L *is the data monoid* $(M, \cdot, \hat{})$ *where* M *is the set of* $\equiv_L$*-classes, concatenation is defined by* $[u]_{\equiv_L} \cdot [v]_{\equiv_L} = [uv]_{\equiv_L}$ *and the action is defined by* $\hat{\tau}[u]_{\equiv_L} = [\tau(u)]_{\equiv_L}$ *for all words* u*,* v *and data renamings* τ*.*

Language acceptance by a data monoid can be defined in the usual way: A data language $L \subseteq (D \times A)^*$ is *recognized* by a data monoid $\mathcal{M} = (M, \cdot, \hat{})$ iff there is a homomorphism $h : (D \times A)^* \to \mathcal{M}$ and a subset $F \subseteq M$ such that $w \in L$ iff $h(w) \in F$. Note that every language is accepted by its syntactic data monoid.

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

The *data orbit* of a data value $a \in D$ with respect to a data monoid element $m \in M$ is the set of data values b such that $b = \tau(a)$ for some stabilizer τ of m. We write $a \sim_m b$ if a and b are on the same data orbit with respect to m. The following propositions gives an alternative characterization of the memorable values:

Proposition 3.10. *Let M be a data monoid. A value a is memorable for an element $m \in M$ iff the data orbit of a with respect to m is finite.*

Proof. We first note that a value a is memorable for $m \in M$ iff there is a data renaming τ such that $\tau(a) \neq a$ and $\hat{\tau}(m) \neq m$. To show the proposition we first assume that the orbit of a with respect to m is finite. Then there is a data renaming τ such that $\tau(a) \neq a$. Then, by the definition of a data orbit, it must hold that $\hat{\tau}(m) \neq m$ (otherwise $\tau(a)$ would be on the data orbit of a with respect to m). For the other direction assume that the data orbit of a is infinite. Then the condition 'for all data renamings τ, if $\tau(a) \neq a$ then $\hat{\tau}(m) = m$' is trivially satisfied. It follows that a is not memorable for m. □

A further important lemma about memorable values is the following. Its proof can be found in [Boj11].

Lemma 3.11 (Memory Lemma)**.** *Let M be a data monoid and let $x \in M$. If τ is a data renaming that fixes* $\mathrm{mem}(x)$ *element-wise then $\hat{\tau}$ fixes x.*

As already noticed in [Boj11], a relevant part of Green's theory [Gre51, Pin11], which holds for finite monoids, can be lifted to orbit finite data monoids. The basic Green's relations $\leq_{\mathcal{R}}$, $\leq_{\mathcal{L}}$, $\leq_{\mathcal{J}}$ associated with $\mathcal{M}$ are the preorders defined by:

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

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

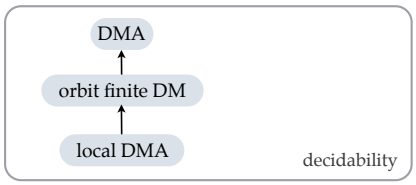


Figure 3.6: Orbit finite data monoids lie between local DMA and DMA.

and $s \mathcal{L} t$. Given an element s of a data monoid $\mathcal{M}$, we denote by $R(s)$ (resp., $L(s)$, $J(s)$, $H(s)$) the $\mathcal{R}$-class (resp., $\mathcal{L}$-class, $\mathcal{J}$-class, $\mathcal{H}$-class) of s.

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

We will later show that orbit finite data monoids accept exactly the languages that can be defined in RMSO (Theorem 3.8). RMSO in turn is a syntactic fragment of BRMSO, and we will later show that the languages defined in BRMSO is a strict subset of the languages that are accepted by DMA. As it is easy to see that every language that is accepted by a local automaton can be accepted by a finite orbit data monoid, it follows that the expressiveness of orbit finite data monoids lies strictly between the expressiveness of local automata and DMA.

3.2.4 Logics vs. Automata vs. Data Monoids

Regular languages over finite alphabets are very robust. For example many different formalisms all define the same class of languages:

$$\text{finite monoids} = \text{DFA} = \text{NFA} = \text{MSO}$$

The languages defined by first-order logic is strictly contained in this class of languages. On data words however, the corresponding classes form a strict hierarchy in expressiveness:

$$\text{orbit-finite data monoids} \subsetneq \text{DMA} \subsetneq \text{NMA} \subsetneq \text{MSO}$$

(The first inclusion is due to Theorem 3.8 and Proposition 3.73, the second is shown in [KF94], the last in [NSV04]). In addition first-order logic is incomparable, even to NMA: as with languages over finite alphabets, the set of all (data) words of even length is clearly recognized by a 0-register DMA, but it cannot be defined in first-order logic. In Section 2.2.2 we gave an example of a language that can be defined in first-order logic but that is not accepted by NMA. Figure 3.7 gives an overview

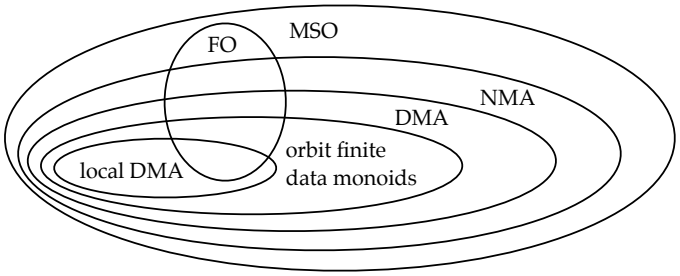


Figure 3.7: Comparing the expressive power of automata with that of logics. All intersections shown are known to be non-empty.

over the relationships between first-order logic, data monoids and variants of finite memory automata.

3.3 Main Results

The goal of this chapter is to find algorithms that decide the following problem:

> *Given a language L that is accepted by an automaton or a monoid, and a logic $\mathcal{L}$, can the language L be defined in the logic $\mathcal{L}$?*

If this problem is decidable, we say that there is an *effective characterization* of $\mathcal{L}$ with respect to the model of automata or class of monoids.

Undecidability. We start with some negative results. In particular, we show that there can be no effective characterizations with respect to many automata models for data languages, even for very weak logics. The automata models for which we prove this are non-deterministic finite memory automata, two-way deterministic finite memory automata, and a very weak version of pebble automata. The formal statements of the theorems are:

Theorem 3.1. *Let $\mathcal{L}$ be a logic that is at most as expressive as $FO(\sim, <)$ and that can define the universal language D^*. The following problem is undecidable: Given an 3-NMA A, can $L(A)$ be defined in $\mathcal{L}$?*

Theorem 3.2. *Let $\mathcal{L}$ be a logic that is at most as expressive as $FO(\sim, <)$ and that can define the universal language D^*. The following problem is undecidable: Given a 2-way DMA A with three registers or a weak 1-way DPA A with 3 pebbles, can $L(A)$ be defined in $\mathcal{L}$?*

The proofs of both theorems are by reduction from Post's correspondence problem.

Local Logics. We next consider local logics, that is, logics that can only compare data values if these have a bounded distance. We show that these logics can be characterized effectively with respect to deterministic finite memory automata.

Theorem 3.3. *The following problem is decidable: Given a DMA $\mathcal{A}$, is there an $l \in \mathbb{N}$ such that $L(\mathcal{A})$ is definable in $FO(\sim_{\leq l}, <)$? If such an l exists, then the minimal l_0 such that $L(\mathcal{A})$ is definable in $FO(\sim_{\leq l_0}, <)$ can be computed. Analogous results hold when $<$ is replaced by $+1$.*

The proof of the above theorem exploits that it is decidable whether a deterministic finite memory automaton $\mathcal{A}$ is local. If not, then we show that $L(\mathcal{A})$ cannot be defined in the local logic in question. Otherwise we reduce the problem to the problem of deciding whether a finite automaton (over a finite alphabet) accepts a language that is definable in first-order logic. The latter problem is known to be decidable [Sch65, MP71].

As a corollary we obtain that local non-uniform first-order logic can be characterized effectively with respect to deterministic finite memory automata.

Corollary 3.12. *The following problem is decidable: Given a DMA $\mathcal{A}$, is there an l such that $L(\mathcal{A})$ is definable in $NUFO(\sim_{\leq l}, <)$? If such an l exists, then we can compute the minimal l_0 such that $L(\mathcal{A})$ is definable in $NUFO(\sim_{\leq l_0}, <)$. Analogous results hold when $<$ is replaced by $+1$.*

Non-Uniform Logics. We then turn to non-uniform logics – that is, logics where the formula depends on the size of the alphabet. We show that there is an effective characterization of non-uniform first-order logic with respect to local deterministic finite automata.

Theorem 3.4. *The following problem is decidable: Given a local DMA $\mathcal{A}$, is $L(\mathcal{A})$ definable in $NUFO(\sim, <)$?*

The idea of the proof is again by reduction to the finite alphabet case: we show that there is a number N such that a language L is definable in $\mathrm{NUFO}(\sim, <)$ iff the restriction of L to a finite alphabet of size N is definable in first-order logic (over finite alphabets). One direction is straightforward: if L is definable in $\mathrm{NUFO}(\sim, <)$, then its restriction to a finite alphabet is clearly definable in $\mathrm{FO}(D_N, <)$. For the other direction we assume that L cannot be defined in $\mathrm{NUFO}(\sim, <)$ and we show that there is a number N such that L restricted to an alphabet of size N cannot be

defined in first-order logic. We complete the proof by showing that we can bound on the size of N.

We then show that the above theorem can also be shown for non-local DMA, if the automaton has at most one register. We note that DMA with one-register are incomparable in expressiveness to local DMA.

Theorem 3.5. *The following problem is decidable: Given a DMA $\mathcal{A}$ with at most one register, is $L(\mathcal{A})$ definable in NUFO$(\sim, <)$?*

First-Order Logics with Two Variables. In the preceding results we have considered the case where the input language is accepted by some kind of automaton. The main proof techniques in these results are minimization and Ehrenfeucht-Fraïssé-style games. We now turn to theorems whose proofs use algebraic techniques. This is possible because we will consider languages that are accepted by finite orbit data monoids. We start by providing an effective characterization the two variable fragment of first-order logic over plain languages.

Theorem 3.6. *Given a plain language L that is accepted by an orbit finite data monoid, it is decidable whether L can be defined in first-order logic with at most two variables.*

The structure of the proof is similar to the proof of the corresponding theorem over finite alphabets [TW98]. We show that an orbit finite data monoid recognizable language can be defined in the two variable fragment of first-order logic iff the accepting monoid is in a special class of data monoids called DA.

Rigid Logics The results on rigid logics are the most technically challenging results of this thesis. The first theorem is inspired by the results of Schützenberger, McNaughton, and Papert.

Theorem 3.7. *A language can be defined in rigidly-guarded first-order logic iff it is accepted by an aperiodic orbit finite data monoid. In addition, the translations between the two formalisms are effective.*

The direction from left to right is proved by structural induction on the rigidly guarded MSO formulas: the translation of the atomic formulas $x < y$, $a(x)$, $x \in Y$ are easy (at least towards non-aperiodic monoids) and the translations of the Boolean connectives are as in the classical case. The translation of the existential closures uses a powerset construction on orbit finite data monoids. Since data monoids are

in general infinite objects, the standard powerset construction would yield infinitely many orbits even if the original data monoid has finitely many of them. We address this issue by showing that the languages defined by rigidly guarded FO are accepted by orbit finite monoids via special morphisms, called 'projectable morphisms'. This allows us to preserve orbit finiteness. The most technical part of this direction 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 for this result: if $\varphi(x, y)$ were not rigid, then the data monoid recognizing $[\![\varphi(x, y) \wedge x \sim y]\!]$ would still be orbit finite, but the morphism would in general not be projectable. The proof that $[\![\varphi(x, y) \wedge x \sim y]\!]$ is recognized via a projectable morphism requires a bit of analysis since rigidity is a semantic assumption and hence one cannot directly deduce from it a property for the data monoid. However, one can use the rigidity property for "normalizing" the data monoid, allowing the construction to go through.

The proof of the other direction follows a structure similar to Schützenberger's proof that languages recognized by aperiodic monoids are definable by star-free expressions (i.e., in first-order). Namely, the proof relies on an induction on the structure of ideals of the data-monoid, the so called *Green's relations*. This requires specific study of this theory for orbit finite data-monoids. Such a study was initiated by Bojańczyk [Boj11], but we had to develop several new tools for our proof to go through.

Our next result can be considered a counterpart of Büchi's Theorem for orbit finite data monoids.

Theorem 3.8. *A language can be defined in rigidly-guarded monadic second-order logic iff it is accepted by an orbit finite data monoid. In addition, the translations between the two formalisms are effective.*

The translation from formulas to monoids follows the same lines as the proof of the corresponding direction of Theorem 3.7. The proof of the other direction is straightforward in the case of classical languages of words. Indeed, a monoid can be used as an automaton, and it is sufficient to write a formula which guesses a run of such an automaton and check that it is valid and accepting. In the context of data monoids, this is impossible. Not only is there no equivalent automaton model, but furthermore, the above approach is intrinsically not compatible with the notion of rigidity. We need a completely different approach.

From Theorem 3.8 we obtain that rigidly guarded MSO is decidable.

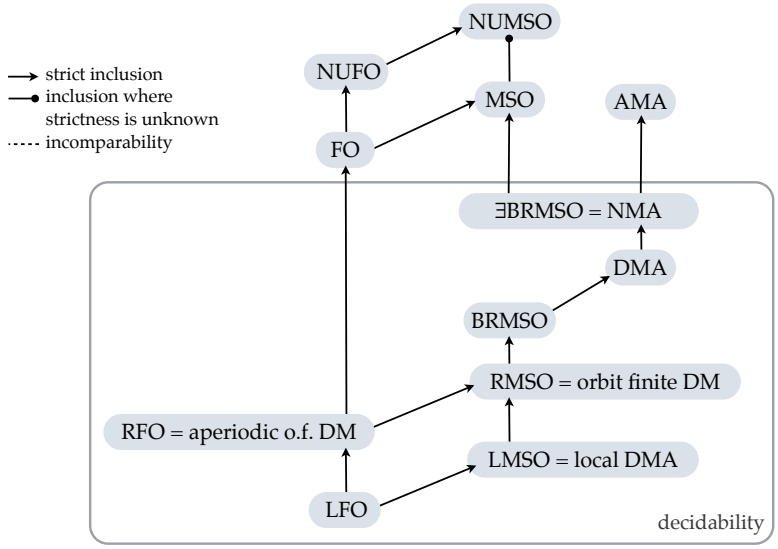


Figure 3.8: The relationship between logics, automata and monoids for data word languages.

Corollary 3.13. *The satisfiability problem for rigidly guarded MSO logic is decidable. Moreover, one can decide whether a formula belongs to the rigidly guarded MSO logic.*

Our last result concerns the relationship between ∃backward-rigidly guarded MSO and non-deterministic finite memory automata

Theorem 3.9. *A language is definable in ∃backward-rigidly guarded MSO iff it is recognizable by non-deterministic FMA.*

The proof of this theorem is similar to the proof of Theorem 3.8. Figure 3.8 shows a visualization of some of our results.

3.4 Related Work

Our work is strongly related to the work in [Boj11]. There, orbit finite data monoids are introduced, and the study of Green's relations for data monoids is initiated. The main result of [Boj11] is that every language that is accepted by an aperiodic orbit finite data monoid can be defined in first-order logic. This result is similar to one direction in the proof of Theorem 3.7. However, first-order logic is considerable stronger than rigidly guarded first-order logic (recall that it can define languages that cannot be defined in non-deterministic finite memory automata). The makes the result of [Boj11] "easier" (in some sense) to prove than Theorem 3.7.

Recently, effective translations between automata for data languages and logics have been provided in [Bol11]. The data model considered there is distinguished from the one considered here in that the data words in [Bol11] can have several data values at each position. In addition, each data word is associated with a finite set of binary predicates $\lhd$. These predicates must satisfy restrictions that are very similar to our notion of rigidity. In particular, for each data word $w \in A \times D^k$ and every relation $\lhd$ there is an associated relation $\lhd^w$ such that for all positions i, i', j, j' of w the following conditions hold:

- $\lhd^w$ complies with $<$, that is $i \lhd^w j$ implies that $i < j$,
- $\lhd^w$ has out-degree at most one, that is, there is at most one k such that $i \lhd^w k$,
- $\lhd^w$ has in-degree at most one, that is, there is at most one k such that $k \lhd^w i$,
- $\lhd^w$ is monotone, that is, if $i \lhd^w j$ and $i' \lhd j'$ and $w(i) = w(i')$ and $w(j) = w(j')$, then $i < i'$ iff $j < j'$.

Two versions of monadic second-order logic are considered there: MSO logic that can compare the data values as usual and a weaker version that can only compare between values that are on the same position. The paper also defines a new model of automata that can be considered as a combination of class memory automata with registers automata. The main result of the paper is that the weak version of existential monadic second-order logic can be translated into the automata model. The paper also provides a translation from automata to MSO but this translation is into the stronger version of MSO. Nonetheless, for the data model where each position can only have a single data value, [Bol11] shows a precise correspondence between a restriction of their version of MSO and fresh register automata of [Tze11]. Hence the version of MSO of [Tze11], when restricted to data words with single values at each position, is strictly more expressive than rigidly guarded MSO.

Monoids that accept data languages have first been studied in [BPT03]. There the authors study a class of data word languages that is accepted by finite monoids. The notion of acceptance is different from the notion that we use here. The authors define a 'register mechanism' that processes a word from left to right and computes for each prefix of the input word a monoid element and a register that can store data values. The monoid element that is output after position i of the input word is computed from the monoid element for position $i-1$, the pair $(a, d) \in A \times D$ that labels position $i-1$ of the input word, and a register that has been output for position $i-1$. The register assignment for position i is computed from the old register

assignment and the current data value d. Acceptance of the monoid depends on the last monoid element that the register mechanism produces. The authors show that a language is accepted by a certain kind of automaton iff it is accepted by a finite monoid via the register mechanism. We note that this class of automata can accept languages that cannot be accepted by finite memory automata.

3.5 Towards Schützenberger for Data

Our goal in this section is to prove a theorem that is similar to a combination of Schützenberger's, McNaughton's and Papert's theorem. More precisely, we want to find an effective characterization of first-order logic.

We start with some undecidability results in Subsection 3.5.1. We then show effective characterizations of first-order logics with local predicates in Subsection 3.5.2 and of non-uniform first-order logic in Subsection 3.5.3. Finally we present an effective characterization of rigidly guarded first-order logic with unrestricted data equality and linear order in Subsection 3.5.5.

3.5.1 Undecidability Results

We start with a negative result. We show that there is no hope for achieving effective characterizations of fragments of FO$(\sim, <)$ within several classes of languages recognized by automata models stronger than DMA. We first consider the class of languages recognized by NMA:

Theorem 3.1. *Let $\mathcal{L}$ be a logic that is at most as expressive as FO$(\sim, <)$ and that can define the universal language D^*. The following problem is undecidable: Given an 3-NMA $\mathcal{A}$, can $L(\mathcal{A})$ be defined in $\mathcal{L}$?*

The proof is by reduction from the Post Correspondence Problem (PCP) and it is along the same lines as the proof of Neven et al. that universality is undecidable for NMA [NSV04].

Proof of Theorem 3.1. The proof is by reduction from the Post Correspondence Problem (PCP). An instance of the PCP is a finite set of pairs of words $I = \{(w_1, v_1), \ldots, (w_k, v_k)\}$ over a binary alphabet $\{a, b\}$. We say that I has a solution if there is an $m \in \mathbb{N}$ and a sequence of indices $i_1, \ldots, i_m \leq k$ such that $w_{i_1} \ldots w_{i_m} = v_{i_1} \ldots v_{i_m}$. It is known that the following problem is undecidable [HMU06]: Given a PCP instance I, does I have a solution?

Let us fix a PCP instance $I = \{(w_1, v_1), \ldots, (w_k, v_k)\}$. In [NSV04] Neven et al. encode a candidate solution of I of the form $S = (w_{i_1}, \ldots, w_{i_m}; v_{j_1}, \ldots, v_{j_n})$, with $n, m \geq 1$ and $i_1, \ldots, i_m,\ j_1, \ldots, j_n \in \{1, \ldots, k\}$, by means of a data word $\mathrm{enc}(S) = \square\, w \,\square\, v$, where $\square \in D$ is used as a delimiter and w and v are two words with no occurrences of $\square$. The words w and v represent the sequence $w_{i_1}, \ldots, w_{i_m}$ and the sequence $v_{j_1}, \ldots, v_{j_m}$ respectively. Neven et al. also prove that the following language, which consists of all words that are *not* encodings of valid solutions of I, is recognized by a 3-memory MA:

$$\tilde{L}_I^{\text{no-sol}} \;=\; \left\{ v \in D^* \;\middle|\; \text{if } v = \mathrm{enc}(S), \text{ then } S \text{ is not a solution of } I \right\}$$

By adopting a slightly modified notion of encoding, we can also allow encodings of candidate solutions that start with arbitrarily long prefixes of the form $\square^i$, with $\square \in D$ and $i \geq 1$. Accordingly, we modify the definition of the language $L_I^{\text{no-sol}}$ in such a way that it contains all words that are not (generalized) encodings of valid solutions of I.

$$L_I^{\text{no-sol}} \;=\; \left\{ \square^i \cdot v \in D^* \;\middle|\; i \geq 1 \text{ and if } v = \mathrm{enc}(S), \text{ then } S \text{ is not a solution of } I \right\}$$

Note that $L_I^{\text{no-sol}}$ is still recognized by a 3-MA. Moreover, the following language is also recognized by an MA:

$$L^{\text{odd}} \;=\; \left\{ \square^{2j+1}\, w \,\square\, v \;\middle|\; \square \in D,\ j \in \mathbb{N},\ w, v \in (D \setminus \{\square\})^* \right\}.$$

As MA are closed under union, there is a 3-MA that recognizes the language $L_I = L_I^{\text{no-sol}} \cup L^{\text{odd}}$.

We now complete the proof of the theorem by showing that L_I can be defined in the logic $\mathcal{L}$ iff I has no solution. If I has a solution S, then we let u_i be an encoding of S of the form $\square^i w \,\square\, v$, for all $i \geq 1$. We then observe that $u_{2^d} \notin L_I$ and $u_{2^d+1} \in L_I$. Moreover, the two words u_{2^d} and u_{2^d+1} cannot be distinguished by any $\mathrm{FO}(\sim, <)$ formula of quantifier depth d (the proof is a straightforward generalization of the proof that a^{2^d} and a^{2^d+1} cannot be distinguished by $\mathrm{FO}(<)$ formulas of quantifier depth d — see, for instance, [Str94]). Therefore, since $\mathcal{L}$ was assumed to be at most as expressive as $\mathrm{FO}(\sim, <)$, no formula in $\mathcal{L}$ can define L_I. For the opposite direction, we assume that I has no solution. Then L_I coincides with the universal language D^*, which can be defined in $\mathcal{L}$ by hypothesis. □

Below, we show that similar negative results hold for two-way deterministic finite-memory automata (2-way DMA) and for the weakest variant of pebble automata,

namely, weak one-way pebble automata (weak 1-way DPA). The proof of the following result is similar to that of Theorem 3.1.

Theorem 3.2. *Let $\mathcal{L}$ be a logic that is at most as expressive as $FO(\sim, <)$ and that can define the universal language D^*. The following problem is undecidable: Given a 2-way DMA $\mathcal{A}$ with three registers or a weak 1-way DPA $\mathcal{A}$ with 3 pebbles, can $L(\mathcal{A})$ be defined in $\mathcal{L}$?*

Proof Sketch. We first prove the claim for 2-way 3-register DMA. We claim that there is a 2-way DMA $\mathcal{A}_{\text{dup}}$ that recognizes the language

$$L_{\text{dup}} \;=\; \big\{ \square\, w \,\square\, w \;\big|\; \square\, w \text{ contains only pairwise distinct symbols} \big\}.$$

Indeed, $\mathcal{A}_{\text{dup}}$ can check that the following properties hold:

1. The input word is of the form $\square w \,\square\, v$.
2. $\square w$ and $\square v$ contain only pairwise distinct symbols.
3. a, b are neighbors in $\square w$ iff a, b are neighbors in $\square v$.

Making use of a construction in [NSV04], it is easy to see that, for any given PCP instance I, there also exist a 2-way 3-DMA that recognizes the language $L_I^{\text{no-sol}}$. The rest of the proof is along the same lines of the proof of Theorem 3.1.

As for 1-way DPA, in [NSV04] it is shown that there is a 1-way DPA $\mathcal{A}_I^{\text{sol}}$ that recognizes the language

$$L_I^{\text{sol}} \;=\; \big\{ v \in D^* \;\big|\; v \text{ encodes a solution of } I \big\}.$$

Since L_I^{sol} is the complement of $L_I^{\text{no-sol}}$ and 1-way DPA are closed under complement, there is a 1-way DPA with 3-pebbles that recognizes $L_I^{\text{no-sol}}$ as well. The rest of the proof is again similar to the proof of Theorem 3.1. □

3.5.2 Characterization of Local First-Order Logic

We start this section with a simple, but very useful proposition. It shows that the languages that can be defined in local-MSO are precisely the languages that can be accepted by local DMA. It follows that local DMA are closed under all boolean operations, projection, concatenation, and Kleene star. The proposition will also allow us to reduce definability problems on local automata to definability problems on finite automata.

Proposition 3.14. *A language L can be defined in l-local MSO iff it can be recognized by a l-local DMA.*

Proof. We first give some preliminary definitions. Given a natural number l, we denote by $T_{\leq l}$ the finite alphabet that consists of all $\cong$-types of words of length at most l. Given a data word $w = a_1 \dots a_n \in D^*$, we then denote by $\mathrm{abs}_{\leq l}(w)$ the word $\alpha_1 \dots \alpha_n$ over the finite alphabet $T_{\leq l}$, where α_i is either the $\cong$-type of the prefix $a_1 \dots a_i$ or the $\cong$-type of the subword $a_{i-l+1} \dots a_i$, depending on whether $i \leq l$ or $i > l$. Similarly, given a data language $L \subseteq D^*$, we denote by $\mathrm{abs}_{\leq l}(L)$ the language over $T_{\leq l}$ that contains all and only the words of the form $\mathrm{abs}_{\leq l}(w)$, with $w \in L$.

We show that $\mathrm{MSO}(\sim_{\leq l}, +1)$ is exactly as expressive as $\mathrm{MSO}(T_{\leq l}, +1)$ up to the encoding of data languages via $\mathrm{abs}_{\leq l}$. More precisely, a data language L is definable in $\mathrm{MSO}(\sim_{\leq l}, <)$ iff $\mathrm{abs}_{\leq l}(L) = \{\mathrm{abs}_{\leq l}(w) \mid w \in L\}$ is definable in $\mathrm{MSO}(T_{\leq l}, <)$: Suppose that L is defined by an $\mathrm{MSO}(\sim_{\leq l}, <)$ sentence ϕ. We define a mapping f from $\mathrm{MSO}(\sim_{\leq l}, <)$ formulas to $\mathrm{MSO}(T_{\leq l}, <)$. f is defined by structural induction: We first consider atoms of the form $x \sim_i y$, with $i \leq l$. Let U_i be the subset of $T_{\leq l}$ that consists of all and only the $\cong$-types of words $a_1 \dots a_n$ such that $i \leq n \leq l$ and $a_{n-i} = a_n$. We then define

$$f(x \sim_i y) \;=\; (y = x + i) \wedge \bigvee_{\alpha \in U_i} \alpha(x)$$

The definitions in the remaining cases are as follows: $f(x < y) = (x < y)$, $f(\neg\phi) = \neg f(\phi)$, $f(\phi \wedge \psi) = f(\phi) \wedge f(\psi)$, $f(\exists x \phi) = \exists x f(\phi)$, and $f(\exists X \phi) = \exists X f(\phi)$. A straightforward proof by induction shows that f satisfies the following property: for every $\mathrm{MSO}(\sim_{\leq l}, <)$ formula ψ with free variables $x_1, \dots, x_n, X_1, \dots, X_m$, every data word $w \in D^*$, every n-tuple of positions $i_1, \dots, i_n \leq |w|$, and all unary predicates $P_1, \dots, P_m \subseteq \{1, \dots, |w|\}$, we have

$$(w, i_1, \dots, i_n, P_1, \dots, P_m) \vDash \psi \quad \text{iff} \quad (\mathrm{abs}_{\leq l}(w), i_1, \dots, i_n, P_1, \dots, P_m) \vDash f(\psi).$$

In particular, this shows that $f(\phi)$ defines exactly the language $\mathrm{abs}_{\leq l}(L)$. The proof for the other direction exploits similar arguments and is omitted.

Next, observe that any l-DWA $\mathcal{A}$ can be thought of as a deterministic finite automaton (DFA) over the finite alphabet $T_{\leq l}$. In particular, any *realizable* l-DWA that recognizes L also recognizes $\mathrm{abs}_{\leq l}(L)$ when it is considered as a DFA. From the equivalence of finite automata and MSO over finite alphabets, it follows that $\mathrm{abs}_{\leq l}(L)$ is recognized by a realizable l-DWA $\mathcal{A}$ when considered as a DFA in $\mathrm{MSO}(T_{\leq l}, <)$. Moreover, the latter property holds iff $\mathrm{abs}_{\leq l}(L)$ is definable L is accepted by an l-local DWA. Finally, by Proposition 3.6, l-DWA are exactly as expressive as l-local DMA. □

In the rest of this subsection we give effective characterizations for first-order logics with local predicates, namely, FO($\sim_l, <$) and FO($\sim_l, +1$). There are actually two variants of the definability problem for each of these logics. The first variant takes as input a DMA $\mathcal{A}$ *and a number* l and asks whether $L(\mathcal{A})$ is definable in FO($\sim_l, <$) (resp., FO($\sim_l, +1$)). The second variant takes as input a DMA $\mathcal{A}$ and asks whether *there is a number* l such that $\mathcal{A}$ is definable in FO($\sim_l, <$) (resp., FO($\sim_l, +1$)). The following theorem shows that both variants of the definability problems for FO($\sim_l, <$) and FO($\sim_l, +1$) are decidable.

Theorem 3.3. *The following problem is decidable: Given a DMA $\mathcal{A}$, is there an $l \in \mathbb{N}$ such that $L(\mathcal{A})$ is definable in l-local FO($\sim, <$)? If such an l exists, then the minimal l_0 such that $L(\mathcal{A})$ is definable in l_0-local FO($\sim, <$) can be computed. Analogous results hold when $<$ is replaced by $+1$.*

Proof of Theorem 3.3. We will only prove the claim for l-local FO($\sim, <$). The proof for the case of the successor relations follows the same lines.

In Chapter 2 we have shown that, given a DMA $\mathcal{A}$, one can compute an equivalent canonical DMA. We thus assume that $\mathcal{A}$ is a DMA already in canonical form. By Proposition 3.4, one can check whether $\mathcal{A}$ is local and, in such a case, compute the minimum number l_0 such that $\mathcal{A}$ is l_0-local. We distinguish two cases.

First assume that $\mathcal{A}$ is not local, that is it is not l-local for any $l \in \mathbb{N}$. Then it follows from Proposition 3.14 that $L(\mathcal{A})$ cannot be defined in l-local MSO for any $l \in \mathbb{N}$. This implies that $L(\mathcal{A})$ cannot be defined in local FO.

Hence we can assume that $\mathcal{A}$ is l-local for some $l \in \mathbb{N}$, and we assume that this l is minimal. Then we can check whether $L(\mathcal{A})$ can be defined in l-local FO by reducing the problem to a problem on finite automata as follows: By Proposition 3.6, one can compute an l-DWA $\mathcal{B}$ that is equivalent to $\mathcal{A}$. We can further assume that $\mathcal{B}$ is realizable. We denote by FO($T_{\leq l}, <$) the first-order logic over the finite vocabulary $T_{\leq l}$. A similar argument as in the proof of Proposition 3.14 shows that $L(\mathcal{B})$ is definable in l-local FO($\sim, <$) iff $\mathrm{abs}_{\leq l}(L(\mathcal{B}))$ is definable in FO($T_{\leq l}, <$) (abs is defined in the proof of 3.14). This reduces the problem of deciding whether $L(\mathcal{A})$ is FO($\sim_{\leq l_0}, <$) decidable to an analogous problem over DFA. It was shown in [Sch65, MP71] that the latter problem is decidable.

We claim that Algorithm 3.9 provides a decision procedure to test membership in local FO($\sim, <$). To show that Algorithm 3.9 is correct, we need to show the following:

1. If $\mathcal{A}$ is l-local and l is minimal, and $L(\mathcal{A})$ can be defined in l-local FO($\sim, <$), then there is no $l' < l$ such that $L(\mathcal{A})$ can be defined in l'-local FO($\sim, <$).

Algorithm 2: Check Membership in Local-FO($\mathcal{A}$)

input: A DMA $\mathcal{A}$ that accepts a language L over $(D, \sim)$
output: A minimal l such that L is definable in l-local FO($\sim, <$) or "no".

Test whether $\mathcal{A}$ is local using Proposition 3.4.
if $\mathcal{A}$ is not local
  then return L cannot be defined in local FO($\sim, <$).
if $\mathcal{A}$ is local
  then
    Compute the minimal l such that $\mathcal{A}$ is l-local using Proposition 3.4.
    Check whether L can be defined in l-local FO($\sim, <$).
    if L can be defined in l-local FO($\sim, <$).
      then return l is minimal s. t. L is definable in l-local FO($\sim, <$).
      else return L cannot be defined in local FO($\sim, <$).

Figure 3.9: An algorithm for deciding whether the language accepted by an DMA can be defined in local-FO($\sim, <$).

2. If $\mathcal{A}$ is l-local and l is minimal, and $L(\mathcal{A})$ cannot be defined in l-local FO($\sim, <$), is there is no $l' > l$ such that $L(\mathcal{A})$ can be defined in l'-local FO($\sim, <$).

We show the two statements in the following two claims respectively. The first statement follows from Claim 1 below.

Claim 1. *If a the language L can be defined in l-local FO($\sim, <$), then the canonical DMA $\mathcal{A}_L$ of L is l-local.*

Proof of Claim 1. We show the contrapositive. Hence suppose that $\mathcal{A}$ is a canonical DMA that is not l-local and let L be the recognized language. By definition, there is a word v of length l and two configurations $(p, \bar{a})$ and $(q, \bar{b})$ such that $v^A(p, \bar{a}) = (q, \bar{b})$ and $\bar{b}$ contains a value a that does not occur in w. Clearly, a occurs in $\bar{a}$. Let w be a word that induces a run of $\mathcal{A}$ from the initial configuration to the configuration $(p, \bar{a})$ (such a run exists since all states of $\mathcal{A}$ are reachable). Now, recall that a canonical DMA stores only memorable values. Since a is stored after reading $w \cdot v$, then a is L-memorable in $w \cdot v$. Thus, by definition of memorable value, there is a data word s and a value b such that $w \cdot v \cdot s \not\equiv_L w \cdot v \cdot \tau_{a,b}(s)$. As a does not occur in v and $|v| = l$, one can show, using a simple game-theoretic argument, that $w \cdot v \cdot s$ and $w \cdot v \cdot \tau_{a,b}(s)$ can not be distinguished in FO($\sim_l, <$). □

The second statement follows from Claim 2 below.

Claim 2. *Let $\mathcal{A}$ be an l-local DMA. If $L(\mathcal{A})$ is not definable in l-local FO$(\sim, <)$, then $L(\mathcal{A})$ is not definable in l'-local FO$(\sim, <)$ for all $l' > l$.*

Proof of Claim 2. We prove the claim by induction on $l' - l$. The base case $l' - l = 0$ is trivial. Let $l' \geq l$ and assume that $L = L(\mathcal{A})$ is not definable in FO$(\sim_{\leq l'}, <)$. We have to prove that L is not definable in $(l'+1)$-local FO$(\sim, <)$ either. Let us fix an arbitrary natural number d for the quantifier depth of formulas. Since there exist no FO$(\sim_{\leq l'}, <)$ formula of quantifier depth d that defines L, there exist two words w and v such that (i) $w \in L$, (ii) $v \notin L$, and (iii) Duplicator has a winning strategy in the d-round l'-local FO$(\sim, <)$-game on w and v. Below, we show that there exist two corresponding words $\tilde{w} \in L$ and $\tilde{v} \notin L$ and a winning strategy for Duplicator in the d-round $(l'+1)$-local FO$(\sim, <)$-game on $\tilde{w}$ and $\tilde{v}$ (this implies that L cannot be defined by any FO$(\sim_{\leq l'+1}, <)$ formula of quantifier depth d).

We first define $\tilde{w}$ starting from w. Suppose that $w = a_1 \dots a_n$ and that there are two positions x and y such that such $y = x + (l'+1)$, $a_x = a_y$, and $a_x \neq a_z$ for all intermediate positions z with $x < z < y$. Let $w' = a_1 \dots a_{y-1} \cdot \tau_{a_y,b}(a_y \dots a_n)$, where b is a fresh value that does not occur in w. Observe that for any two positions x' and y', with $x' \leq y' \leq x' + l'$, we have $w(x') = w(y')$ iff $w'(x') = w'(y')$. In addition, since $\mathrm{abs}_{\leq l}(w) = \mathrm{abs}_{\leq l}(w')$, we have $w' \in L$. By iterating the above construction, one obtain a word $\tilde{w}$ from w such that

1. $\tilde{w} \in L$;
2. if $y \leq x + l'$, then $\tilde{w}(x) = \tilde{w}(y)$ iff $w(x) = w(y)$;
3. if $y = x + (l'+1)$, then $\tilde{w}(x) \neq \tilde{w}(y)$.

A similar construction can be used to obtain a word $\tilde{v} \notin L$ from v that satisfies properties analogous to 2. and 3. above.

We now show that the same strategy used by Duplicator to win the d-round l'-local FO$(\sim, <)$-game on w and v can be used to win the d-round $(l'+1)$-local FO$(\sim, <)$-game on $\tilde{w}$ and $\tilde{v}$. Suppose that the play in the $(l'+1)$-local FO$(\sim, <)$-game on $\tilde{w}$ and $\tilde{v}$ has progressed up to the i-th round and that the pebbles lie at positions $x_1, \dots, x_i$ in $\tilde{w}$ and at positions $y_1, \dots, y_i$ in $\tilde{v}$. Suppose that Spoiler places a new pebble at position x_{i+1} in $\tilde{w}$ (the case of

a new pebble placed at position y_{i+1} in $\tilde{v}$ can be dealt with by symmetric arguments). Duplicator must respond by choosing a suitable position y_{i+1} in $\tilde{v}$ in such a way that the two resulting structures $(w, x_1, \ldots, x_{i+1})$ and $(v, y_1, \ldots, y_{i+1})$ satisfy the same atomic formulas in $(l'+1)$-local FO$(\sim, <)$. Due to Property 2 above, the configuration reached after Spoiler's action can be viewed as a configuration in the l-local FO$(\sim, <)$-game on w and v. Duplicator can thus respond by using the winning strategy in that game. This guarantees that the two resulting structures $(w, x_1, \ldots, x_{i+1})$ and $(v, y_1, \ldots, y_{i+1})$ satisfy the same atomic formulas in l'-local FO$(\sim, <)$. From Property 2, it then follows that $(\tilde{w}, x_1, \ldots, x_{i+1})$ and $(\tilde{v}, y_1, \ldots, y_{i+1})$ satisfy the same atomic formulas in l'-local FO$(\sim, <)$. Finally, from Property 3, the same holds for the atomic formulas in $(l'+1)$-local FO$(\sim, <)$. This shows that $\tilde{w} \in L$ and $\tilde{v} \notin L$ cannot be distinguished by any $(l'+1)$-local FO$(\sim, <)$ formula of (arbitrary) quantifier depth d and hence L cannot be defined in $(l'+1)$-local FO$(\sim, <)$. □

This concludes the proof of Theorem 3.3. □

As an example, consider the 3-local DMA $\mathcal{A}$ equivalent to the 3-DWA depicted in Figure 3.4: if thought of as a DFA, such an automaton contains a counter over the $\cong$-type $[aba]$, where $a \neq b$. It can then be proved that the data language $L(\mathcal{A})$ cannot be defined in l-local FO$(\sim, <)$, for any $l \in \mathbb{N}$. However, it is easy to see that in fact $L(\mathcal{A})$ can be defined by a non-local FO$(\sim, <)$ formula.

The next proposition will allow us to derive decidability of l-local NUFO$(\sim, <)$ within DMA. It shows that definability in the local FO$(\sim, <)$ is equivalent to definability in local NUFO$(\sim, <)$, provided that we restrict to DMA-recognizable languages.

Proposition 3.15. *Let L be a language recognized by a DMA and let $l \in \mathbb{N}$. L can be defined in l-local NUFO$(\sim, <)$ iff it can be defined in l-local FO$(\sim, <)$. An analogous result holds when $<$ is replaced by $+1$.*

We do not know whether Proposition 3.15 also holds for the non-local logics FO$(\sim, <)$ and FO$(\sim, +1)$.

Proof of Proposition 3.15. We fix a k-DMA that recognizes L and $l \in \mathbb{N}$. We prove the proposition in the case of signatures with the order $<$. The interesting direction is from left to right. Our proof is by contraposition, hence we assume that L cannot be defined in l-local FO$(\sim, <)$. Then, for each $d \in \mathbb{N}$, there exist two words w and

v such that $w \in L$, $v \notin L$, but w and v cannot be distinguished by l-local FO$(\sim, <)$ formulas of quantifier depth d.

We will show that there is a pair of words $w' \in L$, $v' \notin L$ that are over a small alphabet (whose size does not depend on d) and w' cannot be distinguished from w by l-local FO$(\sim, <)$ formulas of quantifier depth d (and likewise for v and v'). This will imply that L cannot be defined in l-local NUFO$(\sim, <)$.

Let $w = a_1 \dots a_n$ be a word with more than $k + l$ different values and let

$$(p_0, \bar{a}_0) \xrightarrow{a_1} \dots \xrightarrow{a_n} (p_n, \bar{a}_n)$$

be a run of A on w. Now consider a minimal prefix $a_1 \dots a_m$, $m \leq n$ of w that contains $k+l$ different values. Clearly (a) m must be at least $k+l$, (b) $a_1 \dots a_m$ must contain less than $k+l$ different values, and (c) $a_m \neq a_i$ for all $i < m$. By (c) a_m cannot be in the register assignment $\bar{a}_{m-1}$ of A after step $m-1$. In addition there must be a value a in $a_1 \dots a_{m-1}$ that appears neither in $\bar{a}_{m-1}$ nor in $a_{m-l-1} \dots a_{m-1}$ (this follows from (a) and (b)). Then A cannot distinguish w from $w_1 = a_1 \dots a_{m-1} \cdot \tau_{a_m,a}(a_m \dots a_n)$ and thus $w_1 \in L$. In addition, for all positions x, y in w, the two structures (w, x, y) and (w', x, y) satisfy the same set of atomic l-local FO$(\sim, <)$ formulas. Thus, w and w_1 cannot be distinguished in l-local FO$(\sim, <)$.

By iterating the above construction one can obtain a word $\tilde{w} \in L$ that cannot be distinguished from w by any l-local FO$(\sim, <)$ formula. Moreover, $\tilde{w}$ contains at most $k + l$ distinct symbols. Using the same argument, one can obtain a word $\tilde{v} \notin L$ over an alphabet of size l that cannot be distinguished from v by any l-local FO$(\sim, <)$ formula. Since w and v cannot be distinguished by l-local FO$(\sim, <)$ formulas of quantifier depth d, by transitivity the same holds for $\tilde{w}$ and $\tilde{v}$.

Now, let D be a finite alphabet that contains all symbols in $\tilde{w}$ and in $\tilde{v}$. Since $\tilde{w} \in L \cap D^*$ and $\tilde{w} \notin L \cap D^*$, it follows that no l-local FO$(\sim, <)$ formula can define $L \cap D^*$ and hence L is not l-local NUFO$(\sim, <)$ definable. □

The next corollary follows from Theorem 3.3 and Proposition 3.15.

Corollary 3.12. *The following problem is decidable: Given a DMA $\mathcal{A}$, is there an l such that $L(\mathcal{A})$ is definable in l-local NUFO$(\sim, <)$? If such an l exists, then we can compute the minimal l_0 such that $L(\mathcal{A})$ is definable in l_0-local NUFO$(\sim, <)$. Analogous results hold when $<$ is replaced by $+1$.*

3.5.3 Characterization of Non-Uniform FO

In this section we look for an effective characterizations of NUFO$(\sim, <)$. Precisely, we will show that definability in NUFO$(\sim, <)$ is decidable for languages recognized by local DMA and 1-memory DMA (these two models are incomparable in expressive power).

Theorem 3.4. *The following problem is decidable: Given a local DMA $\mathcal{A}$, is $L(\mathcal{A})$ definable in NUFO$(\sim, <)$?*

The idea of the proof is to show that $L = L(\mathcal{A})$ is definable in NUFO$(\sim, <)$ iff there is some $n \in \mathbb{N}$ that can be computed from $\mathcal{A}$ such that the restriction $L_n = L \cap \Delta_n$ of L to an alphabet Δ_n of size n is definable in FO$(\Delta_n, <)$. The latter statement is decidable because L_n is a regular language over a finite alphabet and an effective characterization of regular language definable in FO$(\Delta_n, <)$ is known from [Sch65, MP71]. One direction of this claim is straightforward: if L is definable in NUFO$(\sim, <)$, then L_n is clearly definable in FO$(\Delta_n, <)$. For the opposite direction, we assume that L is not definable in NUFO$(\sim, <)$. In this case, one can prove that there is a (potentially very big) number N such that $L_N = L \cap \Delta_N$ cannot be defined in FO$(\Delta_N, <)$. It follows from [Sch65, MP71] that the minimal DFA $\mathcal{A}_{L_N}$ recognizing L_n has a counter. We then prove that there is a (potentially) much smaller alphabet Δ_n for which the minimal DFA $\mathcal{A}_n$ recognizing $L_n = L \cap \Delta_n$ has a counter. Thus L_n cannot be defined in FO$(\Delta_n, <)$.

Proof of Theorem 3.4. Before we start with the main part of the proof we will show three claims. The following claim shows that for all $m \in \mathbb{N}$, FO$(\sim, <)$ interpreted by words over a finite alphabet Δ_m of size m has the same expressive power as FO$(\Delta_m, <)$ when restricted to isomorphism closed languages.

> ***Claim 1.*** *Let $L \subseteq D^*$ be an isomorphism closed language and let $m \in \mathbb{N}$. Then $L_m = L \cap \Delta_m^*$ can be defined in FO$(\sim, <)$ over Δ_m iff L_m can be defined in FO$(\Delta_m, <)$.*
>
> *Proof of Claim 1.* First assume that there is an FO$(\sim, <)$ formula ϕ that defines $L_m \subseteq \Delta_m^*$. Let ψ be the FO$(\Delta_m, <)$ obtained from ϕ by replacing every occurrence of $x \sim y$ by $\bigvee_{a \in \Delta_m} a(x) \wedge a(y)$. A simple proof by structural induction shows that ψ defines $L \cap \Delta_m^*$.
>
> For the other direction, let ψ be an FO$(\Delta_m, <)$ formula that defines L_m. We assume that ψ is of the form $Q_1 x_1 \dots Q_r x_r . \theta$ where each Q_i, $i \leq r$ is

a quantifier and θ is a quantifier-free formula. Note that negations can be removed from θ by first pushing them down to the atoms and then replacing every literal $\neg a(x)$ by $\bigvee_{b \in \Delta_m \setminus \{a\}} b(x)$. Hence, we can assume θ to be in positive disjunctive normal form, namely, θ is a disjunction of clauses, where each clause is a conjunctions of positive atoms. Moreover, we can assume that no clause contains two conjuncts of the form $a(x)$ and $b(x)$, with $a \neq b$. Therefore, the symbols of Δ_m induce a partition of the variables that occur in each clause. Assume that θ contains exactly x clauses and, for each $1 \leq i \leq x$, let $X_{i,a}$ be the set of all variables that occur in the i-th clause with the predicate a. We define

$$\tilde{\theta} = \bigvee_{i \leq x} \left(\bigwedge_{\substack{a \in \Delta_m \\ x,y \in X_{i,a}}} (x \sim y) \quad \wedge \bigwedge_{\substack{a,b \in \Delta_m,\, a \neq b \\ x \in X_{i,a},\, y \in X_{i,b}}} (x \not\sim y) \right)$$

By exploiting the fact that L is closed under isomorphism, one can show that $Q_1 x_1 \dots Q_r x_r . \theta$ is equivalent to $Q_1 x_1 \dots Q_r x_r . \tilde{\theta}$. □

Recall that a counter of a deterministic finite automaton $\mathcal{B}$ is a sequence $q_1, \dots, q_m$ of states of $\mathcal{B}$ such that $m > 1$ and there is a word w that moves $\mathcal{B}$ from state q_i to state $q_{i+1 \mod m}$:

$$q_1 \xrightarrow{w} q_2 \dots q_{m-1} \xrightarrow{w} q_m \xrightarrow{w} q_1$$

We now introduce a similar concept for finite memory automata. A *counter* of a DMA $\mathcal{A}$ is a sequence $\mathcal{C} = (p_1, \bar{a}_1), \dots, (p_m, \bar{a}_m)$ of configurations of $\mathcal{A}$ such that $m > 1$ and there is a word w that moves $\mathcal{A}$ from configuration $(p_i, \bar{a}_i)$ to configuration $(p_{i+1 \mod m}, \bar{a}_{i+1 \mod m})$. In this case we also say that $\mathcal{C}$ is a counter for w.

It follows from the next claim that a local DMA has a counter for a word over a small alphabet whenever it as any counter.

Claim 2. *Let $\mathcal{A}$ be a l-local DMA with k registers. If $\mathcal{A}$ has a counter $\mathcal{C} = (p_1, \bar{a}_1), \dots, (p_m, \bar{a}_m)$ for a word w, then $\mathcal{C}$ is a counter for a word $\tilde{w}$ over an alphabet Δ of size $kl + 1$.*

Proof of Claim 2. Let Δ' be a set of data values that appear in $\bar{a}_1, \dots, \bar{a}_m$. The point of the claim is that the size of Δ' is independent from m. This is due to the fact that $\mathcal{A}_L$ is local, hence $\bar{a}_i$ can only contain symbols in the l-suffix of w. For this reason $|\Delta'| \leq l$ and we can pick Δ to be a superset of Δ' that has size $kl + 1$.

The rest of the claim can be proved using the same technique that was used in the proof of Lemma 2.9. The idea is to consider the leftmost position $i \leq j$ in $w = b_1 \dots b_j$ that is labeled by a value $b_i \notin \Delta$. Consider the set $\mathcal{C}'$ of configurations that $\mathcal{A}$ reaches by reading $b_1 \dots b_{i-1}$ starting in a configuration in $\mathcal{C}$. It is clear that $\mathcal{C}'$ can only contain values in Δ. We then consider the word

$$w' = b_1 \dots b_{i-1} \cdot \tau_{b_i,b}(b_i \dots b_j)$$

where b is a value from Δ that does not occur in $\mathcal{C}'$. Such a value always exists because $|\mathcal{C}'| \leq kl$ and $|\Delta| = kl + 1$. Using the same arguments as in Lemma 2.9, we can show that $\mathcal{C}$ is a counter for w'. By iterating this construction we obtain a word $\tilde{w}$ over the alphabet Δ for which $\mathcal{C}$ is a counter. □

We now show a claim that is concerned with the relationship between DMA's with counters and corresponding DFA's with counters. We first need some definitions. Let $\equiv^{\text{Conf}}_{\mathcal{A}}$ be the equivalence relation on the configurations of a DMA $\mathcal{A}$ defined by: $(p, \bar{a}) \equiv^{\text{Conf}}_{\mathcal{A}} (q, \bar{b})$ iff for all words w

> the configuration of $\mathcal{A}$ after reading w from $(p, \bar{a})$ is accepting iff
> the configuration of $\mathcal{A}$ after reading w from $(q, \bar{b})$ is accepting.

We note that even if $\mathcal{A}$ is a minimal DMA then $\equiv^{\text{Conf}}_{\mathcal{A}}$ might not be trivial. For example the canonical DMA $\mathcal{A}_{abc}$ accepting the language $\{abc \in D^* \mid c \in \{a, b\}\}$ has two different configurations after reading ab and ba respectively, that are $\equiv^{\text{Conf}}_{\mathcal{A}_{abc}}$-equivalent.

Claim 3. *Let $\mathcal{A}_L$ be a canonical l-local DMA with k registers accepting a language L. For all $m \geq lk+1$, $\mathcal{A}_L$ has a counter consisting of $\equiv^{Conf}_{\mathcal{A}_L}$-distinct configurations iff the minimal DFA $\mathcal{B}_{L \cap \Delta_m}$ accepting $L \cap \Delta_m$ has a counter.*

Proof of Claim 3. We first show the more interesting direction from left to right. Assume that $m \geq lk + 1$ and that $\mathcal{A}_L$ has a counter $\mathcal{C}$ consisting of $\equiv^{\text{Conf}}_{\mathcal{A}_L}$-distinct configurations. By Claim 2, $\mathcal{C}$ is also a counter for a word over the alphabet Δ_n for $n = kl+1$. Now consider the finite automaton $\mathcal{A}_{\Delta_n}$ that is obtained from restricting $\mathcal{A}$ to the alphabet Δ_n: The states of $\mathcal{A}_{\Delta_n}$ consist of pairs of the form $(q, \bar{a})$ where q is a state of $\mathcal{A}$ and $\bar{S}$ is a register assignment over values in Δ_n. The transitions $\mathcal{A}_{\Delta_n}$ can also be inferred

from the transitions of $\mathcal{A}$ in an obvious way. It is clear that $\mathcal{C}$ is also a counter of $\mathcal{A}_{\Delta_n}$. It is also clear that $\mathcal{A}_{\Delta_n}$ accepts the language $L \cap \Delta_n$, but it is not the minimal DMA accepting $L \cap \Delta_n$ in general. However we know from the theorem of Myhill and Nerode, that there is a congruence $\equiv$ on the states of $\mathcal{A}_{\Delta_n}$ such that the minimal DMA $\mathcal{B}_{L\cap\Delta_n}$ that accepts $L \cap \Delta_n$ can be obtained from $\mathcal{A}_{\Delta_n}$ by collapsing $\equiv$-equivalent states. As the states of $\mathcal{A}_{\Delta_n}$ are configurations of $\mathcal{A}$, we can extend the definition of $\equiv^{\mathrm{Conf}}_{\mathcal{A}_L}$ the states of $\mathcal{A}_{\Delta_n}$ in the obvious way. It is obvious from the definition of $\equiv^{\mathrm{Conf}}_{\mathcal{A}_L}$ that $\equiv^{\mathrm{Conf}}_{\mathcal{A}_L}$ coincides with $\equiv$. Thus the fact that the states of $\mathcal{A}_{\Delta_n}$ that are in involved in the counter $\mathcal{C}$ are $\equiv^{\mathrm{Conf}}_{\mathcal{A}_L}$-distinct, means that if we minimize $\mathcal{A}_{\Delta_n}$ by collapsing states, then the states in $\mathcal{C}$ are not collapsed. It follows that $\mathcal{B}_{L\cap\Delta_n}$ has a counter.

The proof of the other direction is omitted, because it is similar to the direction from left to right, but slightly simpler. □

We are finally ready to prove Theorem 3.4, that is that one can decide, given a DMA $\mathcal{A}$, whether $L(\mathcal{A})$ is definable in NUFO$(\sim, <)$. Hence let $\mathcal{A} = (Q, q_I, F, T)$ be an l-local DMA with memory size k that accepts the language $L = L(\mathcal{A})$.

$$L \text{ is definable in NUFO}(\sim, <) \text{ iff } L_n \text{ is definable in FO}(\Delta_n, <) \text{ for } n = lk + 1. \quad (*)$$

Here FO$(\Delta_n, <)$ denotes classical first-order logic with predicates of the form $x < y$ and $a(x)$, for all $a \in \Delta_n$, where Δ_n is an alphabet of size n. Note that Theorem 3.4 follows from $(*)$ because (i) L_n is a regular language over a finite alphabet and (ii) it is known from [Sch65, MP71] that the problem of checking whether a given regular language is definable in classical first-order logic is decidable.

Hence we will prove $(*)$. If L is definable in NUFO$(\sim, <)$, then L_n is definable in FO$(\sim, <)$ by definition, and in FO$(\Delta_n, <)$ by Claim 1. To show the other direction assume that L is not definable in NUFO$(\sim, <)$. Then, there is some (possibly big) number $N \in \mathbb{N}$ such that L_N cannot be defined in FO$(\sim, <)$. Due to Claim 1 we know that in fact L_N cannot be defined in FO$(\Delta_N, <)$. If $N \leq n$ then we are done. Hence we will assume that $N > n$. As L_N is a regular language there is a minimal finite automaton $\mathcal{B}_{L_N}$ accepting L_N. Due to results of McNaughton, and Papert we know that $\mathcal{B}_{L_N}$ must have a special cycle called "counter" [MP71]. By Claim 3 we know that $\mathcal{A}$ has a counter $\mathcal{C}$ involving only $\equiv^{\mathrm{Conf}}_{\mathcal{A}}$-distinct configurations. Now we can use Claim 2 to deduce that $\mathcal{C}$ is a counter for a word $\tilde{w}$ over the alphabet Δ_n with $n = kl + 1$. We apply Claim 3 for a second time to deduce that the minimal

DFA $\mathcal{B}_{L_n}$ that accepts the language $L \cap \Delta_n$ has a counter. Hence it follows that $L \cap \Delta_n$ cannot be defined in $\mathrm{FO}(\Delta_n, <)$ [MP71]. Applying Claim 1 for a second time we get that $L \cap \Delta_n$ cannot be defined in $\mathrm{FO}(\sim, <)$. We have just shown $(*)$, which concludes the proof of Theorem 3.4. □

Below, we show that the analogous problem is decidable for 1-memory DMA. Observe that 1-DMA are incomparable with local DMA: On the one hand, the language of all words where the first value is equal to the last one is recognizable by 1-DMA, but not by local DMA. On the other hand, the language of all words where the third value is equal to either the first value or the second value is recognizable by local DMA, but not by 1-DMA.

Theorem 3.10. *The following problem is decidable: Given a DMA $\mathcal{A}$ with at most one register, is $L(\mathcal{A})$ definable in NUFO$(\sim, <)$?*

The proof exploits, first, the fact that it is decidable whether a given DMA $\mathcal{A}$ is local. If $\mathcal{A}$ is local, then Theorem 3.4 can be applied and we are done. If $\mathcal{A}$ is not local, then $\mathcal{A}$ must contain certain 'non-local cycles'. By distinguishing several cases, depending on the way these cycles occur in $\mathcal{A}$, it can be decided whether $\mathcal{A}$ is definable in $\mathrm{NUFO}(\sim, <)$.

Proof of Theorem 3.10. The proof is by case analysis. Let $\mathcal{A} = (Q, q_I, F, T)$ be a canonical 1-DMA that recognizes a language L.

We distinguish two types of transitions in $\mathcal{A}$: those for which the target memory content is non-empty and it does not depend on the input value (namely, those triples $(p, [ab], E, q)$, with $p, q \in Q_1$, $a \neq b \in D$, and $E = \{2\}$), and those for which the target memory content is either empty or it is uniquely determined by the input value. We call the former type of transitions *non-local* and the latter type *local*. From the proof of Proposition 3.4, we recall that the period of a simple loop C in $\mathcal{A}$ is the number of its states and C is *local* iff it contains *at least one local transition*. Moreover, we say that two simple loops intersect if they share at least one control state.

We now distinguish between the following three cases:

1. there is a non-local simple loop in $\mathcal{A}$ with period strictly greater than 1;
2. there is a simple loop in $\mathcal{A}$ with period strictly greater than 1 that intersects a non-local simple loop (with period 1);

3. for every pair of intersecting simple loops in $\mathcal{A}$, either both loops are local, or both have period 1.

In the sequel, we prove that, in each of the above cases, there is an effective procedure that decides whether the language L is definable in NUFO$(\sim, <)$ (in fact, in the first two cases, the procedures turns out to be trivial, namely, they always return false).

Claim 1. *If $\mathcal{A}$ contains a non-local simple loop of period strictly greater than 1, then L is not definable in NUFO$(\sim, <)$.*

Proof of Claim 1. Let C be a non-local simple loop in $\mathcal{A}$ of period $m > 1$ and let $p_1, \ldots, p_m$ be the sequence of states in C ordered according to the transitions that connect them. Since non-local transitions do not modify the memory content and C contains only non-local transitions, we know that there exist two words w and v such that

(i) $\mathcal{A}$ reaches C after reading w, formally $w^A(q_I, \varepsilon) = (p_1, \bar{a})$

(ii) $\mathcal{A}$ cycles through C by reading repetitions of v, formally $v^A(p_i, \bar{a}) = (p_{i+1 \bmod m}, \bar{a})$ for all $i \leq m$.

Let N be the number of distinct values that occur in w and in v. Below, we prove that $L_N = L \cap \Delta_N$ is not definable in FO$(\Delta_N, <)$, and hence, by Theorem 3.4, L is not definable in NUFO$(\sim, <)$.

Suppose, by way of contradiction, that there is an FO$(\Delta_N, <)$ formula ϕ of quantifier depth d that defines the language L_N. Since $\mathcal{A}$ is in canonical form C features at least two different states, say p_1 and p_2, we know from the results presented in Section 2.5 that there is a word $s \in \Delta_N^*$ such that

$$\left(w \cdot v^{m^d} \cdot s\right) \not\equiv_L \left(w \cdot v^{m^d+1} \cdot s\right).$$

To obtain a contraction it is sufficient to show that ϕ cannot distinguish between the two words $\left(w \cdot v^{m^d} \cdot s\right)$ and $\left(w \cdot v^{m^d+1} \cdot s\right)$. This can be done by exploiting the fact that ϕ has quantifier depth d and by using the following game argument: Duplicator has a winning strategy in the d-round Ehrenfeucht-Fraïssé game for FO$(\Delta_N, <)$ interpreted over $\left(w \cdot v^{m^d} \cdot s\right)$ and $\left(w \cdot v^{m^d+1} \cdot s\right)$. □

Claim 2. *If $\mathcal{A}$ contains a simple loop of period strictly greater than 1 that intersects a non-local simple loop with period 1, then L is not definable in NUFO$(\sim, <)$.*

Proof of Claim 2. Let C be a simple loop with period $m > 1$ and let C' be a non-local simple loop with period 1 that intersects C. Let p be a control state shared by the two loops C and C' and let q be the successor of p in C. Clearly, since C and C' are distinct, we have $p \neq q$. Now, let u be an infinite word that eventually cycles through C. By using a simple counting argument, one can find a memory content $\bar{a}$, a prefix w and a subword v of u such that

(i) $w^A(q_I, \varepsilon) = (p, \bar{a})$,

(ii) $v^A(p, \bar{a}) = (p, \bar{a})$.

This means that the infinite word $w \cdot v^\omega$ eventually cycles through C, exactly as u does. Let N be the number of distinct values that occur in w and in v. Below, we prove that $L_{2N} = L \cap \Delta_{2N}$ is not definable in $\mathrm{FO}(\Delta_{2N}, <)$, and hence, by Theorem 3.4, L is cannot be defined in $\mathrm{NUFO}(\sim, <)$.

Suppose, by way of contradiction, that there is an $\mathrm{FO}(\Delta_{2N}, <)$ formula ϕ that defines the language $L_{2N} = L \cap \Delta_{2N}^*$. Let d be the quantifier depth of ϕ and let v' be an isomorphic copy of v consisting of fresh values from $\Delta_{2N} \setminus \Delta_N$ (thus, v and v' feature disjoint sets of values). By construction, we have that $(v')^A(p, \bar{a}) = (p, \bar{a})$, as it happens for v. However, while v follows the transitions in C, v' follows the unique transition of C'. In particular, we have $(v(1))^A(p, \bar{a}) = (q, \bar{b})$, for some memory content $\bar{b}$, and $(v'(1))^A(p, \bar{a}) = (p, \bar{a})$ (note that since C' is a non-local simple loop, the transition that consumes the symbol $v'(1)$ does not modify the memory content). We can then proceed as in the proof of Lemma 3.5.3, first showing that there is a word $s \in \Delta_{2N}^*$ such that

$$\left(w \cdot (v \cdot v')^{2^d} \cdot s\right) \not\equiv_L \left(w \cdot (v \cdot v')^{2^d+1} \cdot s\right).$$

and then showing that ϕ cannot distinguish between these words. □

Claim 3. *Suppose that for every pair of intersecting simple loops in $\mathcal{A}$, either both loops are local, or both have period 1. Let $\mathcal{B}$ be the local DMA obtained from $\mathcal{A}$ by removing all transitions of non-local simple loops (of period 1) and of simple loops (of period 1) that intersect non-local simple loops. We have that L is definable in NUFO$(\sim, <)$ iff $L(\mathcal{B})$ is definable in NUFO$(\sim, <)$.*

Proof of Claim 3. Let us fix, in $\mathcal{A}$, a simple loop C of period 1 and a non-local simple loop C' of period 1 that intersects C. Furthermore, let (p, α, E, p) and (p, α', E', p) be the transitions of C and C', respectively, and let $\mathcal{A}'$ be the DMA obtained from $\mathcal{A}$ by removing these two transitions. Since C' is non-local, we have that α' is the $\cong$-type of a word of the form ab, with $a \neq b$. Moreover, since $\mathcal{A}$ is deterministic and C intersects C', α is the $\cong$-type of a word of the form aa (hence C is local). This implies that $\mathcal{A}$ contains no transitions, other than (p, α, E, p) and (p, α', E', p), departing from state p (namely, p is a sink state in $\mathcal{A}'$). Therefore, depending on whether p is final or not, we have $L = L(\mathcal{A}') \cdot D^*$ or $L = L(\mathcal{A}')$. In both cases, we have that $\mathcal{A}$ is definable in NUFO$(\sim, <)$ iff $L(\mathcal{A}')$ is definable in NUFO$(\sim, <)$.

To conclude the proof, it is sufficient to observe that the local DMA $\mathcal{B}$ is obtained from $\mathcal{A}$ by iterating the above construction on each pair of intersecting simple loops C and C' of period 1, where C is local and C' is non-local. □

Theorem 3.4, together with Claim 1, Claim 2, and Claim 3 finally gives a proof of Theorem 3.10. □

3.5.4 Characterization of FO with Two Variables

We now turn to effective characterizations of data languages that are accepted by orbit finite data monoids. We start with an effective characterization of the two variable fragment of first-order logic, denoted FO$^2(\sim, <)$.

Theorem 3.11. *Given a plain language L that is accepted by an orbit finite data monoid, it is decidable whether L can be defined in FO$^2(\sim, <)$.*

We first give a quick overview of what lies ahead. We define two further classes of data languages – those that are accepted by a special class of data monoids, called DA, and a class called "testable fringe languages". We will show that membership in the class accepted by monoids in DA is decidable. Then we will show that these classes coincide with the data languages definable in FO$^2(\sim, <)$. This will conclude the proof of Theorem 3.11.

Data Monoids in DA

It is known that for each finite monoid M there is a number ω such that x^ω is idempotent for each $x \in M$. We show that this statement carries over to finite orbit data monoids. We note that this has already been observed by Bojańczyk [Boj11].

Lemma 3.16. *For each orbit finite data monoid there is a number ω such that x^ω is idempotent for each $x \in M$.*

Proof. We first show that for each $x \in M$ there is an $\omega_x \in \mathbb{N}$ such that x^{ω_x} is idempotent. As M has finitely many orbits there are distinct numbers n and m (we assume $n < m$) such that x^n and x^m are in the same orbit. Then there is a data renaming τ such that $x^m = \tau(x^n)$. As a data renaming can only move finitely many data values there must be a k such that $\tau^k = \mathrm{id}$. Simple calculations show that if $\omega_x = knm$ then $x^{2\omega_x} = x^{\omega_x}$.

It is easy to see that if x and y are on the same orbit then $\omega_x = \omega_y$. Hence we can chose ω to be the least common multiple of $\omega_{x_1}, \ldots, \omega_{x_o}$ where $x_1, \ldots, x_o$ contains an element in each orbit of M. With this choice we get that $x^\omega = x^{2\omega}$ for each $x \in M$. □

Definition 3.17. *For an orbit finite data monoid M, we denote by ω the smallest number such that x^ω is idempotent for each $x \in M$.*

Definition 3.18 (DA)**.** *A data monoid $(M, \cdot, \hat{})$ is in* DA *iff for all $x, y, z \in M$,*

$$(xyz)^\omega = (xyz)^\omega y (xyz)^\omega.$$

A language L is *DA-recognizable* if the syntactic monoid of L is in DA. The goal of this subsection is to show that given an orbit finite data monoid M, it is decidable whether M is in DA. The key ingredient is the following lemma.

Lemma 3.19. *Let M be a finite orbit data monoid. For each number n there is a finite subset N of M such that for all $x_1, \ldots, x_n \in M$ there is a data renaming τ such that for all $i \leq n$, $\hat{\tau}(x_i) \in N$.*

Proof. Let M be a data monoid with o orbits. We define two infinite sequences of finite subsets of M, $K_0 \subseteq K_1 \subseteq \ldots \subseteq M$ and $N_0 \subseteq N_1 \subseteq \ldots \subseteq M$, by simultaneous induction. First, $K_0 = N_0 = \varnothing$. For all $j \in \mathbb{N}$, $K_j = N_{j-1} \cup O_j$ where O_j is a finite set disjoint with $K_{j-1} \cup N_{j-1}$ that contains an element in each orbit of M. In addition we require that the elements in O_j have pairwise disjoint memorable values and that for all $x \in O_j$, the memorable values of x are disjoint from the union of the memorable

values of N_{j-1}. Let $C_j = \bigcup_{x \in K_j} \mathrm{mem}(x)$ and let Π_j be the set of all permutations on C_j. Let N_j be the closure of K_j under Π_j, that is $N_j = \{\hat{\pi}(x) \mid \pi \in \Pi_j, x \in K_j\}$.

We now prove by induction on n that for all $x_1, \ldots, x_n \in M$ there is a data renaming τ_n such that for all $i \leq n$, $\hat{\tau}_n(x_i) \in N_n$. The base case, $n = 1$, holds because N_1 contains an element in every orbit of M.

For the induction step assume that there is a data renaming τ_{n-1} such that for all $i < n$, $\hat{\tau}_{n-1}(x_i) \in N_{n-1}$. Let x_n be given and let x be an element in O_n that is in the same orbit as x_n. Then there is a data renaming σ such that $x = \hat{\sigma}(x_n)$. Let C be the intersection of $\bigcup_{i<n} \mathrm{mem}(x_i)$ with the memorable values of x_n. Also, let π be the data renaming that is the identity everywhere outside $\sigma(C) \cup \tau_{n-1}(C)$ and that swaps $\sigma(c)$ with $\tau_{n-1}(c)$ for all $c \in C$. We define

$$\tau_n(a) = \begin{cases} \tau_{n-1}(a) & \text{if } a \in \bigcup_{i<n} \mathrm{mem}(x_i) \\ \pi \circ \sigma(a) & \text{if } a \in D \setminus \bigcup_{i<n} \mathrm{mem}(x_i) \end{cases}$$

We need to show that for all $i \leq n$, $\hat{\tau}_n(x_i) \in N_n$. First consider the case $i < n$. As $\tau_n(a) = \tau_{n-1}(a)$ for all $a \in \bigcup_{i<n} \mathrm{mem}(x_i)$ it follows from the Memory Lemma 3.11 that $\hat{\tau}_n(x_i) = \hat{\tau}_{n-1}(x_i) \in N_{n-1} \subseteq N_n$. We now show that $\hat{\tau}_n(x_n) \in N_n$. First note that $\tau_{n-1}(c) = \pi \circ \sigma(c)$ for all $c \in C$. Thus $\tau_n(d) = \pi \circ \sigma(d)$ for all $d \in \mathrm{mem}(x_n)$ which implies that $\hat{\tau}_n(x_n) = \hat{\pi} \circ \hat{\sigma}(x_n)$ because of the memory lemma. Next note that $\hat{\sigma}(x_n) = x \in O_n$. As $\pi \in \Pi_n$, it follows that $\hat{\tau}_n(x_n) = \hat{\pi} \circ \hat{\sigma}(x_n) \in N_n$. □

Note that N can be constructed effectively from M. The previous lemma is of interest because of the following corollary.

Corollary 3.20. *Given an orbit finite data monoid M, one can decide whether M is in DA.*

Proof. Let an orbit finite data monoid M be given and assume that N is the subset of M that exists for the number 3 according to Lemma 3.19. We claim that M is in DA iff for all $x, y, z \in N$, $(xyz)^\omega = (xyz)^\omega y (xyz)^\omega$.

The direction from left to right is trivial. For the other direction we assume that for all $x, y, z \in N$, $(xyz)^\omega = (xyz)^\omega y (xyz)^\omega$. Let $x', y', z' \in M$ be given. By Lemma 3.19 there is a data renaming τ such that $\hat{\tau}(x')$, $\hat{\tau}(y')$, and $\hat{\tau}(y')$ are all in N. Thus $\hat{\tau}((x'y'z')^\omega) = \hat{\tau}((x'y'z')^\omega y' (x'y'z')^\omega)$. By applying the inverse of $\hat{\tau}$ to both sides of this equation we deduce that M is in DA. □

Testable Fringe Languages

The set of symbols in a word u is denoted by $\alpha(u)$. We say that (u_0, a, u_1) is the *a-left decomposition* (resp. *a-right decomposition*) of u if $u = u_0 a u_1$ and $a \notin \alpha(u_0)$ (resp. $a \notin \alpha(u_1)$). We define for each number n and finite subset A of D a relation $\equiv_n^A$ on D^* by induction on $|A| + n$. First, $u \equiv_0^A v$ iff $u, v \in A^*$. For $n > 0$ we define $u \equiv_n^A v$ iff

1. $\alpha(u) = \alpha(v) \subseteq A$,
2. for all $a \in A$, if (u_0, a, u_1) is the a-left decomposition of u and (v_0, a, v_1) is the a-left decompositions of v then $u_0 \equiv_n^{A \setminus \{a\}} v_0$ and $u_1 \equiv_{n-1}^A v_1$, and
3. for all $a \in A$, if (u_0, a, u_1) is the a-right decomposition of u and (v_0, a, v_1) is the a-right decompositions of v then $u_0 \equiv_{n-1}^A v_0$ and $u_1 \equiv_n^{A \setminus \{a\}} v_1$.

It is known that a language L over a finite alphabet A is $\mathrm{FO}^2(<)$-definable iff L is the union of $\equiv_n^A$-classes for some n [TW98]. In contrast, the combination of finitely many orbits and no successor relation makes the logic $\mathrm{FO}^2(\sim, <)$ fairly weak: already very simple languages such as 'the first symbol reappears' cannot be defined as they have infinitely many orbits. Basically, we will show that for languages over an infinite alphabet, language membership only depends on the prefix and suffix with a bounded number of distinct symbols. More precisely, we define the *k-symbol beginning* of a word u as the largest prefix of u that contains at most k distinct values. The *k-symbol end* of u is defined symmetrically. The *k-fringe* of u is the word obtained from u by erasing all positions that are neither in the k-beginning nor in the k-end of u. For example *aabacadc* is the 3-symbol fringe of *aabacdeadc*. We say that a language is *k-fringe* if for all words u, $u \in L$ iff $\text{k-fringe}(u) \in L$. A language L is *fringe* if there is an k such that L is k-fringe.

Definition 3.21. *Let u, v be data words, $A = \alpha(\text{k-fringe}(u) \cdot \text{k-fringe}(v))$, and $n, k \in \mathbb{N}$. Then we define*

$$u \approx_{n,k} v \quad \textit{iff} \quad \text{k-fringe}(u) \equiv_n^A \text{k-fringe}(v)$$
$$u \hat{\approx}_{n,k} v \quad \textit{iff} \quad \textit{there is a data renaming } \tau \textit{ such that } u \approx_{n,k} \tau(v)$$

It is easy to check that $\hat{\approx}_{n,k}$ is an equivalence relation. We say that a language L is *$\hat{\approx}$-testable* if there is are n, k such that L is the union of $\hat{\approx}_{n,k}$-classes.

Equivalence of FO^2-Definable, DA-Recognizable and Testable Fringe Languages

Recall that we showed in Corollary 3.20 that it is decidable whether a finite orbit data monoid is in DA. Thus Theorem 3.11 follows from the next lemma.

Lemma 3.22. *For any language L, the following is equivalent:*

(a) L is orbit finite and can be defined in $FO^2(\sim, <)$.

(b) L is $\hat{\approx}$-testable.

(c) M_L is orbit finite and belongs to DA.

We will prove Lemma 3.22 in the rest of this Subsection. The proof is decomposed into three lemmas, showing the implications (c) to (b), (b) to (a), and (a) to (c) respectively.

From Monoids in DA to Testable Fringe Languages. The following lemma is sufficient for the direction from (c) to (b).

Lemma 3.23. *Let L be a data language that is accepted by a monoid in DA. Then there are n, k such that $u \mathrel{\hat{\approx}}_{n,k} v$ implies $u \mathrel{\hat{\equiv}}_L v$.*

Before we delve into details the proof of Lemma 3.23, we give a quick overview. In the main part of the proof we show that if k is bigger than the number of orbits of M then k-fringe$(u) \equiv_L u$. In addition it can be shown that there is an n such that if $u \equiv_n^A v$ then $u \equiv_L v$. This lets us conclude that $u \approx_{n,k} v$ implies that $u \equiv_L v$. Lemma 3.23 follows as L is isomorphism closed.

The next lemma expresses a characteristic property of languages recognized by data monoids in DA.

Lemma 3.24. *Let M be a data monoid in DA. Then for each $\mathcal{R}$-class R there is a set $M_R \subseteq M$ such that for all $r \in R$, $rx \in R$ iff $x \in M_R$. In addition, if a and b are on the same data orbit with respect to r and if R is the $\mathcal{R}$-class of r then $x \in M_R$ iff $\hat{\tau}_{a,b}(x) \in M_R$.*

Proof. Let $M \in$ DA and an $\mathcal{R}$-class R be given. We define

$$M_R = \{x \in M \mid \text{there is } s \in R \text{ such that } sx \in R\}.$$

Now let $r \in R$ be given. Clearly if $x \notin M_R$ then $rx \notin R$. Now consider $x \in M_R$. There must be an $s \in R$ such that $sx \in R$. As r is also in R, $s \mathrel{\mathcal{R}} r$, and hence, there must

be $t \in M$ such that $s = rt$. Putting these equations together we get $r \mathcal{R} s \mathcal{R} sx = rtx$. By iterating this equation we obtain that $r \mathcal{R} r(tx)^\omega$. As M is a member of DA it follows that $(tx)^\omega = (tx)^\omega x(tx)^\omega$. Thus $r \mathcal{R} r(tx)^\omega x(tx)^\omega$ and by substituting $r(tx)^\omega$ by r (this is possible since $r \mathcal{R} r(tx)^\omega$) we get $r \mathcal{R} rx(tx)^\omega \leq_{\mathcal{R}} rx$.

We now show that if a and b are on the same data orbit with respect of r and if R is the $\mathcal{R}$-class of r, then $x \in M_R$ iff $\hat{\tau}_{a,b}(x) \in M_R$ where $\tau_{a,b}$ is the data renaming that swaps a with b. Thus let appropriate r, a, b be given and assume that $x \in M_R$. By the above argument $r =_{\mathcal{R}} rx$, and thus there is $s \in M$ such that $r = rxs$. As a and b are on the same data orbit with respect of r it holds that $r = \hat{\tau}_{a,b}(r)$. Thus $r = \hat{\tau}_{a,b}(r) = \hat{\tau}_{a,b}(rxs)$ which means that $r \mathcal{R} r\tau_{a,b}(x)$ and hence $\tau_{a,b}(x) \in M_R$. □

We call M_R *R's stable set*. The next lemma is the central argument in the proof that if k is bigger than the number of orbits of M then k-fringe$(u) \equiv_L u$.

Lemma 3.25. *Let L be a language whose syntactic monoid M_L is orbit finite and in DA. If $[u]_{\equiv_L} \mathcal{R} [ua]_{\equiv_L}$ for some value a that does not occur in the word u, then $[ua]_{\equiv_L} \mathcal{R} [uaw]_{\equiv_L}$ for all words w.*

Proof. We drop the subscript $\equiv_L$ in this proof. Let L be a language whose syntactic monoid M_L is orbit finite and in DA. We assume towards a contradiction that there are words u, v and values a, b such that a does not occur in u and

$$[u] \mathcal{R} [ua] \mathcal{R} [uav] >_{\mathcal{R}} [uavb]$$

We will show that for each number n there is a word u_n such that $|\text{mem}(u_n)| = n$, contradicting the orbit finiteness of M_L.

We fix a given n. Let R be the $\mathcal{R}$-class of $[u]$ and let $M_R \subseteq M$ be R's stable set. Let $a_1, \ldots, a_n$ be distinct values that do not occur in u. Clearly $a, a_1, \ldots, a_n$ are all on the same data orbit of $[u]$. By Lemma 3.24 $[a], [a_1], \ldots, [a_n]$ are all in M_R. This implies that $[uav] \mathcal{R} [uava_1]$ and hence there is a word v_1 such that $[uava_1v_1] = [uav]$. In fact, for each $n \in \mathbb{N}$, there are words $v_1, \ldots, v_n$ such that $[uava_1v_1 \ldots a_iv_i] = [uav]$ for all $i \leq n$.

We define $u_n = a_1v_1 \ldots a_nv_n$ and we show that for all $i \leq n$, $a_i \in \text{mem}(u_n)$. As $[uav] >_{\mathcal{R}} [uavb]$ and $[uav] \in R$, $[b] \notin M_R$. Note that for all symbols c that occur in u_n, $[c] \in M_R$. Thus b cannot occur in u_n, and hence it is in the unique infinite data orbit of $[u_n]$. As a_i occurs in u_n and $[b] \notin M_R$, it follows from Lemma 3.24 that $[uav] >_{\mathcal{R}} [uav\tau_{a_i,b}(u_n)]$. In contrast $[uav] \mathcal{R} [uavu_n]$, and thus $[uavu_n] \neq [uav\tau_{a_i,b}(u_n)]$, which implies that $[u_n] \neq [\tau_{a_i,b}(u_n)]$. Thus a_i and b are not on the same data orbit of $[u_n]$. This shows that the data orbit of a_i must be finite, because we observed before that b is on the unique infinite data orbit of $[u_n]$. □

We next show two well known propositions (see e.g. [Pin11]):

Proposition 3.26. *Any monoid in DA is aperiodic.*

Proof. Let M be a monoid in DA. Then $(xyz)^{omega} = (xyz)^{omega}y(xyz)^{omega}$ for all $x, y, z \in M$. Now let some $m \in M$ be given. If we choose x, y, z to be m in the above statement, we get

$$m^\omega = m^\omega m^\omega m^\omega = (mmm)^\omega = (mmm)^\omega m(mmm)^\omega = m^{\omega+1}$$

This shows that M is aperiodic □

Proposition 3.27. *Let M be an aperiodic monoid and $x, y \in M$. If $x \mathcal{R} y$ and $x \mathcal{L} y$ then $x = y$.*

Proof. Assume that $x \mathcal{R} y$ and $x \mathcal{L} y$. Then there are s, t such that $x = sy$ and $xt = y$. Thus $x = sxt$, and hence $x = s^\omega x t^\omega$ and by aperiodicity $x = s^\omega x t^{\omega+1} = xt = y$. □

We define the $\mathcal{R}$-*decomposition* of a word u with respect to a language L as the unique factorization $u = u_0 a_1 u_1 \dots a_n u_n$ such that

1. $[\epsilon]_{\equiv_L} \mathcal{R} [u_0]_{\equiv_L}$
2. $[u_0 a_1 u_1 \dots a_i]_{\equiv_L} \mathcal{R} [u_0 a_1 u_1 \dots a_i u_i]_{\equiv_L}$ for all $i \leq n$
3. $[u_0 a_1 u_1 \dots a_i u_i]_{\equiv_L} >_{\mathcal{R}} [u_0 a_1 u_1 \dots a_{i+1}]_{\equiv_L}$ for all $i < n$

We call n the *size* of the $\mathcal{R}$-decomposition of u. The $\hat{\mathcal{R}}$-*decomposition* is defined like the $\mathcal{R}$-decomposition if we replace $\mathcal{R}$ everywhere by $\hat{\mathcal{R}}$.

Lemma 3.28. *Let L be a language whose syntactic monoid is orbit finite and in DA. If k is bigger than the number of orbits of M_L then* k-fringe$(u) \equiv_L u$.

Proof. Let M_L be a syntactic data monoid with o orbits and let some $k > o$ be given. If u contains at most k symbols then the k-fringe of u is equal to u and we are done.

Hence assume that u contains more than k symbols. As $k > o$ the common prefix of u and k-fringe(u) contains a position x that is fresh (that is no position to the left of x has the same value as x) and there are exactly o positions to the left of x are also fresh. Let $u_0 a_1 u_2 \dots a_n u_n$ be the $\hat{\mathcal{R}}$-decomposition of u. By Lemma 3.25 $u_0 a_1 \dots u_{n-1} a_n$ contains only symbols in $a_1, \dots a_n$, in particular at most o distinct symbols. Thus $x > |u_0 a_1 \dots u_{n-1} a_n|$ which implies that $[u(1, x-1)]_{\equiv_L} \mathcal{R} [u(1, x)]_{\equiv_L}$ (in the rest of the proof we will drop $\equiv_L$ as a subscript). Then it follows from Lemma 3.25 that $[u(1, x)] \mathcal{R} [u(1, x)w]$ for all words w, and as $u(1, x)$ is a prefix

of both u and k-fringe(u) it follows that $[u]\,\mathcal{R}\,[\text{k-fringe}(u)]$. By symmetry we obtain that $[u]\,\mathcal{L}\,[\text{k-fringe}(u)]$. Now it follows from Proposition 3.27 that $[u] = [\text{k-fringe}(u)]$. □

We note that there are finite orbit languages with infinitely many $\mathcal{R}$-classes (for an example consider the language 'the first symbol is equal to the last symbol'). Hence it is not clear whether there is a uniform bound on the size of the $\mathcal{R}$-decompositions. It follows from the next lemma that the number of orbits is such a bound.

Lemma 3.29. *For all elements s, t of a data monoid, $s\,\hat{\mathcal{R}}\,st$ iff $s\,\mathcal{R}\,st$.*

Proof. It is sufficient to show that $s\,\hat{\mathcal{R}}\,st$ implies that $s\,\mathcal{R}\,st$, because the other direction is trivial. Assume that $s\,\hat{\mathcal{R}}\,st$. Then there is a data renaming τ and a monoid element r such that $\tau(s) = str$. As each data renaming only modifies a finite number of data values, there must be some n such that $\tau^n = \mathrm{id}$. We fix this n. Note that $\tau^i(s) = \tau^{i-1}(str)$ for all i. Thus

$$s = \tau^n(s) = \tau^{n-1}(str) = \tau^{n-2}(str)\,\tau^{n-1}(tr) = \ \ldots\ = str\,\tau(tr)\,\ldots\,\tau^n(tr)$$

which means that $s \leq_{\mathcal{R}} st$. As $s \geq_{\mathcal{R}} st$ by definition it follows that $s\,\mathcal{R}\,st$. □

Note that it follows that if M is a data monoid with o orbits, then the size of the $\mathcal{R}$-decompositions of a word u cannot be bigger than o.

The next lemma has been shown in the context of finite monoids [TW98]. Exploiting that there is a bound on the size of the $\mathcal{R}$-decompositions, we can reuse the proof in [TW98].

Lemma 3.30 ([TW98])**.** *Let L be a DA-recognizable language whose syntactic monoid has o orbits. Then $n \geq o|\alpha(u)|$ and $u \equiv_n^{\alpha(u)} v$ imply that $u \equiv_L v$.*

Proof. Let L be a language whose syntactic monoid M_L hat $o \in \mathbb{N}$ orbits and let u and v be words such that $u \equiv_n^{\alpha(u)} v$ for some $n \geq o|\alpha(u)|$. If $\alpha(u) = \varnothing$ then $u = v = \epsilon$ and the statement is trivially satisfied. Hence assume that $\alpha(u) \neq \varnothing$. Let $(u_0, a_1, \ldots, a_l, u_l)$ be the $\mathcal{R}$-decomposition of u. It follows from Lemma 3.29 that $l \leq o$. If we denote $u_i a_{i+1} \ldots a_l u_l$ by x_i then it follows from Lemma 3.25 that (u_i, a_{i+1}, x_{i+1}) is the a_{i+1}-left-decomposition of x_i for $i < l$. As $u \equiv_n^{\alpha(u)} v$ and $n \geq l$ there is a decomposition $(v_0, a_1, \ldots, a_l, v_l)$ of v such that for all $i < l$, $(v_i, a_{i+1}, v_{i+1} a_{i+2} \ldots a_l v_l)$ is the a_{i+1}-left-decomposition of $v_i a_{i+1} \ldots a_l v_l$ and $u_i \equiv_{n-i}^{A} v_i$ for $A = \alpha(u) \setminus \{a_{i+1}\}$. As $n - i > o|A|$ for all $i < l$ it follows from the induction hypothesis that $[u_i] = [v_i]$ for all $i < l$

(as in previous proof we omit $\equiv_L$ as a subscript). Thus $[u] \mathcal{R} [u_0 a_1 \dots u_{l-1} a_l] = [v_0 a_1 \dots v_{l-1} a_l] \geq_{\mathcal{R}} [v]$. By symmetry we get $[u] \leq_{\mathcal{R}} [v]$, and hence $[u] \mathcal{R} [v]$. By left-right symmetry we get $[u] \mathcal{L} [v]$ and by Proposition 3.27, $u \equiv_L v$. □

We are finally ready to prove Lemma 3.23 which concludes the proof of the implication (c) to (b).

Proof. (of Lemma 3.23). Let k be bigger than the number of orbits of M_L, and let $n = k^2$. Assume that $u \mathrel{\hat{\approx}}_{n,k} v$, that is, there is a data renaming τ such that k-fringe$(u) \equiv_n^A$ k-fringe$(\tau(v))$ where $A = \alpha$(k-fringe(u)). Then k-fringe$(u) \equiv_L$ k-fringe$(\tau(v))$ by Lemma 3.30. If we combine this statement with Lemma 3.28 we get $u \equiv_L \tau(v)$ and hence $u \mathrel{\hat{\equiv}}_L v$. □

From Testable Fringe Languages to FO^2. In this section we show the implication from (b) to (a). We use two fairly similar games which we define first.

The FO$^2(\sim, <)$-game is played by two players, called *Spoiler* and *Duplicator* on two words $u, v \in D^*$. For each of the words there is a red pebble and a blue pebble. Initially these pebbles are placed somewhere outside of the words. In a round of the game, Spoiler picks up one pebble and places it on one of the words, say he places it on u. Then Duplicator picks up the other pebble of the same color and places it on v. If Spoiler had placed his pebble on v than Duplicator would have had to place the pebble on u. Duplicator wins the game if she can maintain the following invariant in each round:

1. The red pebble is to the left of the blue pebble on u iff the red pebble is to the left of the blue pebble on v.

2. The red pebble and the blue pebble are on positions with the same label on u iff the red pebble and the blue pebble are on positions with the same label on v.

It is known that Duplicator wins the r-round FO$^2(\sim, <)$ game on u, v iff u and v cannot be distinguished by a FO$^2(\sim, <)$ formula with quantifier rank r.

The other game we consider is the *FO$^2(<, A)$-game* where A is a finite subset of D. The rounds of the FO$^2(<, A)$-game are played in the same way as the rounds of the FO$^2(\sim, <)$-game. But there are two differences: first, the FO$^2(\sim, <)$-game is played on strings over A; second, to win the FO$^2(<, A)$ game Duplicator must assure that the following conditions are maintained:

1. The red pebble is to the left of the blue pebble on u iff the red pebble is to the left of the blue pebble on v.

2. The values of the position of the red pebble (resp. blue pebble) on u is equal to the value of the position of the red pebble (resp. blue pebble) on v.

Note that any fringe language is orbit finite. Hence the following lemma is sufficient for the direction from (b) to (a) because winning strategies for the $\mathrm{FO}^2(\sim, <)$-game are preserved under data renamings.

Lemma 3.31. *Let L be a data language whose syntactic monoid is orbit finite. If $u \not\approx_{n,k} v$ then Spoiler wins the $n + 5k$-round $FO^2(\sim, <)$-game on (u, v).*

Proof. Assume that $u \not\approx_{n,k} v$. Then $\text{k-fringe}(u) \not\equiv_n^A \text{k-fringe}(v)$ where A is the alphabet $\alpha(\text{k-fringe}(u) \cdot \text{k-fringe}(v))$. It is has been shown in [TW98] that Spoiler has an $n + 4k$-move winning strategy for the $\mathrm{FO}^2(<, A)$-game on $\text{k-fringe}(u), \text{k-fringe}(v)$ (note that $|A| \leq 4k$). It follows that Spoiler has an winning strategy for the $\mathrm{FO}^2(\sim, <)$-game on $\text{k-fringe}(u), \text{k-fringe}(v)$ with the same number of moves. Spoiler will use the strategy of the latter game for his game on u, v, thereby only placing pebbles on the k-fringes of the words. If Duplicator only pebbles positions on the fringes then he will lose within $n + 4k$ rounds. Hence assume that Duplicator places a pebble outside the fringe of a word, say on v. Then Duplicator uses his remaining k moves to show that one pebble has more than k fresh position to its left and the other does not. □

From FO^2 to DA. To prove the implication from (a) to (c) it is sufficient to show the following.

Lemma 3.32. *Let u, v, w be words. For each n, there is a number m such that Duplicator wins the m-round $FO^2(\sim, <)$-game on $(uvw)^n$ and $(uvw)^n v (uvw)^n$.*

Proof. It has been shown in [TW98] that for each n, there is an m such that Duplicator wins the m-round $\mathrm{FO}^2(<, \alpha(uvw))$-game on $(uvw)^n$ and $(uvw)^n v (uvw)^n$. Clearly this implies that Duplicator wins the m-round $\mathrm{FO}^2(\sim, <)$-game on $(uvw)^n$ and $(uvw)^n v (uvw)^n$. □

3.5.5 Characterization of Rigidly-Guarded FO

In this subsection we show the following theorem:

Theorem 3.7. *A language can be defined in rigidly-guarded first-order logic iff it is accepted by an aperiodic orbit finite data monoid. In addition, the translations between the two formalisms are effective.*

The proof of this theorem is quite involved, and presents one of the main technical challenges of this thesis. Before we begin with its proof, we recall the overview of the proof that we presented in Section 3.3.

The direction from left to right is proved by structural induction on the rigidly guarded MSO formulas: the translation of the atomic formulas $x < y$, $a(x)$, $x \in Y$ are easy (at least towards non-aperiodic monoids) and the translations of the Boolean connectives are as in the classical case. The translation of the existential closures uses a powerset construction on orbit finite data monoids. Since data monoids are in general infinite objects, the standard powerset construction would yield infinitely many orbits even if the original data monoid has finitely many of them. We address this issue by showing that the languages defined by rigidly guarded FO are accepted by orbit finite monoids via special morphisms, called 'projectable morphisms'. This allows us to preserve orbit finiteness. The most technical part of this direction 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 for this result: if $\varphi(x, y)$ were not rigid, then the data monoid recognizing $[\![\varphi(x, y) \wedge x \sim y]\!]$ would still be orbit finite, but the morphism would in general not be projectable. The proof that $[\![\varphi(x, y) \wedge x \sim y]\!]$ is recognized via a projectable morphism requires a bit of analysis since rigidity is a semantic assumption and hence one cannot directly deduce from it a property for the data monoid. However, one can use the rigidity property for "normalizing" the data monoid, allowing the construction to go through. We will reuse this proof technique in the proof of our result concerning the relationship between non-deterministic finite memory automata and rigidly guarded logics (see Theorem 3.9 in Subsection 3.6.2).

The proof of the other direction follows a structure similar to Schützenberger's proof that languages recognized by aperiodic monoids are definable by star-free expressions (i.e., in first-order). Namely, the proof relies on an induction on the structure of ideals of the data-monoid, the so called *Green's relations*. This requires specific study of this theory for orbit finite data-monoids. Such a study was initiated by Bojańczyk [Boj11], but we had to develop several new tools for our proof to go

through. As opposed to the classical case, the proof is significantly more involved for MSO compared to FO.

Presentations of Data Monoids

Since orbit finite data monoids are infinite objects, we need suitable representations that ease algorithmic manipulation. We will first show that every orbit finite data monoid can be presented in a finite way. Later we will define a more explicit presentation of data monoids where the elements of a data monoid are presented by a set of terms.

Finite presentations of data monoids The basic idea is to consider the restriction of an orbit finite data monoid to a finite set of data values:

Definition 3.33. *Given a data monoid $\mathcal{M} = (M, \cdot, \hat{\ })$ and $C \subseteq D$, we define the* restriction *of $\mathcal{M}$ to C to be $\mathcal{M}|_C = \big(M|_C, \cdot|_C, \hat{\ }|_C\big)$, where $M|_C$ consists of all elements $s \in M$ such that $\mathrm{mem}(s) \subseteq C$, $\cdot|_C$ is the restriction of $\cdot$ to $M|_C$, and $\hat{\ }|_C$ is the restriction of $\hat{\ }$ to G_C and $M|_C$.*

Note that $s \cdot t \in M|_C$ and $\hat{\tau}(s) \in M|_C$ for all $s, t \in M|_C$ and $\tau \in G_C$. Hence, if C is finite, $\mathcal{M}|_C$ is a finite data monoid.[1] Hereafter, we denote by $\|\mathcal{M}\|$ the maximum cardinality of the memories of the elements of an orbit finite data monoid $\mathcal{M}$.

Proposition 3.34. *Let $\mathcal{M}$, $\mathcal{M}'$ be orbit finite data monoids such that $\|\mathcal{M}\| = \|\mathcal{M}'\|$ and let $C \subseteq D$ be of cardinality at least $2\|\mathcal{M}\|$. If $\mathcal{M}|_C$ and $\mathcal{M}'|_C$ are isomorphic, then so are $\mathcal{M}$ and $\mathcal{M}'$.*

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

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

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

[1] One has to keep in mind that data monoids over finite sets do not satisfy the same properties as those over infinite sets. For instance, the Memory Theorem, as stated in [Boj11], does not hold for data monoids over finite sets.

prove that $t = t'$. Consider the data renaming $\pi := \sigma \circ \tau^{-1}$. As in this case $\sigma = \pi \circ \tau$, we get

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

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

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

which proves that the function f is well defined.

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

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

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

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

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

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

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

Term-based presentations of data monoids We have just shown how we can represent an infinite data monoid by a finite one. Bellow, we give a more explicit presentation of orbit finite data monoids in which elements of a data monoid are represented by equivalence classes of terms.

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, with associated arity k, and $d_1, \ldots, d_k$ are pairwise distinct data values from a fixed (possibly infinite) set C.

Definition 3.35. *Let O be a set of orbit names and let C be a (finite or infinite) set of data values. A* term-based presentation system *$\mathcal{S}$ over (O, C) consists of a set of terms $T = T_{O,C}$, a binary operation $\odot$ on T, an action $\check{}$ defined by $\check{\tau}(o(d_1, \ldots, d_n)) = o(\tau(d_1), \ldots, \tau(d_n))$, and a congruence $\approx$ for $\odot$ on T such that, for all terms $s, t, u \in T$ and all renamings $\tau \in G_C$,*

1. *Equivariance: $\check{\tau}(s) \odot \check{\tau}(t) = \check{\tau}(s \odot t)$,*
2. *Congruence for renamings: if $s \approx t$ then $\check{\tau}(s) \approx \check{\tau}(t)$,*

3. *Associativity up to* $\approx$: $(s \odot t) \odot u \approx s \odot (t \odot u)$,

4. *Identity: there is a distinguished term* $1 \in T$ *satisfying* $1 \odot t = t \odot 1 = t$.

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

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

1. M is the set of $\approx$-equivalence classes of terms from T,

2. $\cdot$ is a binary product on M defined by $[s]_\approx \cdot [t]_\approx = [s \odot t]_\approx$,

3. $\hat{}$ maps any renaming $\tau \in G_C$ to the function on M defined by $\hat{\tau}([s]_\approx) = [\check{\tau}(s)]_\approx$

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

Proposition 3.36. *Every term-based presentation system represents a data monoid. Conversely, every orbit finite data monoid is represented by a term-based presentation system.*

Proof. We start with the first part of the proposition. Let $\mathcal{S} = (T, \odot, \check{}, \approx)$ be a term-based presentation system over C and let $\mathcal{M} = (M, \cdot, \hat{})$ be the tuple represented by $\mathcal{S}$. It follows from Conditions 3. and 4. of the definition of a term-based presentation system that $(M, \cdot)$ is a monoid. We verify the other properties of data monoids:

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

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

2. Clearly if τ_{id} is the identity data renaming then $\hat{\tau}_{\mathrm{id}}([o(\bar{d})]) = [o(\tau_{\mathrm{id}}(\bar{d}))] = [o(\bar{d})]$.

3. Let $[s], [t] \in M$ and a data renaming $\tau \in G_C$ be given. We need to check that $\hat{\tau}([s] \cdot [t]) = \hat{\tau}([s]) \cdot \hat{\tau}([t])$. We assume that $s = o(\bar{d})$, $t = p(\bar{e})$, and $s \odot t = q(\bar{f})$. Then

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

From Condition 1. of of Definition 3.35 we know that $\check{\tau}(s \odot t) \approx \check{\tau}(s) \odot \check{\tau}(t)$. Thus, we can continue our calculation as follows

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

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

4. We finally show that $\hat{\tau}(1) = 1$, where 1 is the identity of $\mathcal{M}$. This follows from Condition 4. of Definition 3.35.

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

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

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

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

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

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

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

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

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

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

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

$\mathcal{S}$ is a term-based presentation system. Before we prove that $\mathcal{S}$ satisfies the conditions of Definition 3.35, we establish the following claim.

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

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

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

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

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

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

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

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

We now prove Condition C2:

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

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

Turning to the main proof of the proposition, we show that $\mathcal{S}$ is a presentation system by checking the conditions from Definition 3.35.

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

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

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

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

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

4. *Associativity up to* $\approx$*:* Let $s, t, u \in T$ be given. We show that $(s \odot t) \odot u = s \odot (t \odot u)$ using Condition C2 and the associativity of the product $\cdot$ of $\mathcal{M}$:

$$\begin{aligned} f((s \odot t) \odot u) &= f(s \odot t) \cdot f(u) = (f(s) \cdot f(t)) \cdot f(u) \\ &= f(s) \cdot (f(t) \cdot f(u)) = f(s) \cdot f(t \odot u) = f(s \odot (t \odot u)). \end{aligned}$$

5. *Identity:* Recall that we defined $1_{\mathcal{S}}$ to be $g(1_{\mathcal{M}})$. As $1_{\mathcal{M}}$ has empty memory, it follows that $g(1_{\mathcal{M}}) = f^{-1}(1_{\mathcal{M}})$. Let t be a term. Then we have

$$\begin{aligned} 1_{\mathcal{S}} \odot t &= \check{\sigma}^{-1}_{1_{\mathcal{S}},t} \circ g\ (f \circ \hat{\sigma}_{1_{\mathcal{S}},t}\ (1_{\mathcal{S}})\ \cdot\ f \circ \hat{\sigma}_{1_{\mathcal{S}},t}\ (t)) \\ &= \check{\sigma}^{-1}_{1_{\mathcal{S}},t} \circ g\ (f \circ \hat{\sigma}_{1_{\mathcal{S}},t}\ (f^{-1}(1_{\mathcal{M}}))\ \cdot\ f \circ \hat{\sigma}_{1_{\mathcal{S}},t}\ (t)) \\ &= \check{\sigma}^{-1}_{1_{\mathcal{S}},t} \circ g\ (1_{\mathcal{M}}\ \cdot\ f \circ \hat{\sigma}_{1_{\mathcal{S}},t}\ (t)) \\ &= \check{\sigma}^{-1}_{1_{\mathcal{S}},t} \circ g\ (f \circ \hat{\sigma}_{1_{\mathcal{S}},t}\ (t)) \\ &= t. \end{aligned}$$

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

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

$$\begin{aligned} [s] \mathbin{\tilde{\cdot}} [t] &= [s \odot t] \\ \tilde{\hat{\tau}}([s]) &= [\check{\tau}(s)] \\ 1_{\widetilde{\mathcal{M}}} &= [1_{\mathcal{S}}] \end{aligned}$$

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

1. We need to show that $h([s] \mathbin{\tilde{\cdot}} [t]) = h([s]) \cdot h([t])$. Using Condition C2 we can calculate

$$h([s] \mathbin{\tilde{\cdot}} [t]) \;=\; h([s \odot t]) \;=\; f(s \odot t) \;=\; f(s) \cdot f(t) \;=\; h([s]) \cdot h([t])$$

2. Next we show that $h(1_{\widetilde{\mathcal{M}}}) = 1_{\mathcal{M}}$:

$$h(1_{\widetilde{\mathcal{M}}}) \;=\; h([1_{\mathcal{S}}]) \;=\; h(g(1_{\mathcal{M}})) \;=\; 1_{\mathcal{M}}.$$

3. Finally, we need to show that for all terms $s \in T$ and all renamings $\tau \in G_C$, $h(\tilde{\hat{\tau}}([s])) = \hat{\tau}(h([s]))$. This holds because of the following:

$$h(\tilde{\hat{\tau}}([s])) \;=\; h([\check{\tau}(s)]) \;=\; f(\check{\tau}(s)) \;=\; \hat{\tau}(f(s)) \;=\; \hat{\tau}(h([s])).$$

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

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

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

Therefore $f(t) = m$ which shows that h is surjective. This completes the proof of the proposition. □

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

From First-Order Logic to Aperiodic Monoids

In this section, we show that every data language defined by a rigidly guarded first-order sentence is recognized by an aperiodic orbit finite data monoid. In fact we will show a more general statement that allows to translate a rigidly guarded MSO sentence into a (possibly periodic) orbit finite data monoid. The reason is that the proof involving MSO (in this direction) is not any more complicated than the FO case, and we will be able us to reuse the result about MSO in a proof in Section 3.6.1. 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 uppercase letters, without explicitly saying whether these are first-order or second-order variables (their correct types can be inferred from the atoms they appear in). As usual, we write $\varphi(X_1, \dots, X_m)$ whenever we want to make explicit that the free variables of φ are among $X_1, \dots, X_m$. Moreover, given a formula $\varphi(X_1, \dots, X_m)$, a data word $u \in (D \times A)^*$, and some unary predicates $U_1, \dots, U_m \subseteq \mathrm{dom}(u)$, we write $u \vDash \varphi(U_1, \dots, U_m)$ whenever φ holds on u by interpreting the free variables $X_1, \dots, X_m$ with the predicates $U_1, \dots, U_m$.

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

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

where B denotes the binary alphabet $\{0, 1\}$ and $\langle u, U_1, \dots, U_m\rangle$ is the word over the alphabet $D \times A \times B^m$ that has letter $(d, a, b_1, \dots, b_m)$ at position i iff (d, a) is the i-th letter of u, and for all $j = 1 \dots m$, 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 $[\![\varphi]\!]$ is recognized by an orbit finite data monoid. Though this statement is true, it cannot be used – as it is the case in the standard theory – as an induction hypothesis. The problem is that the operation that corresponds to existential quantification (i.e. projection) transforms an orbit finite data monoid into a data monoid which is not orbit finite, in general. That is why our induction hypothesis is stronger, and states that $[\![\varphi]\!]$ is recognized by an orbit finite data monoid via a *projectable* morphism, to be defined below (we write $s \doteq t$ whenever the elements s and t are in the same orbit):

Definition 3.37. *Let h be a morphism from the free data monoid $(D \times A \times B^m)^*$ to a data monoid $\mathcal{M} = (M, \cdot, \hat{\ })$. We say that h is* projectable *if for all data words $u \in (D \times A)^*$ and all tuples of predicates $\bar{U} = (U_1, \dots, U_m)$ and $\bar{V} = (V_1, \dots, V_m)$,*

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

The following lemma implies the direction from left to right of Theorem 3.7. It is at the same time our induction hypothesis:

Lemma 3.38. *For all rigidly guarded MSO formulas $\varphi(\bar{X})$, the language $[\![\varphi]\!]$ is effectively recognized by an orbit finite data monoid with a projectable morphism.*

Before we prove Lemma 3.38, we note the following important corollary:

Corollary 3.39. *Every data language definable in rigidly guarded MSO (resp., rigidly guarded FO) is recognizable by an orbit finite data monoid (resp., aperiodic orbit finite data monoid).*

Proof. The case of rigidly guarded MSO corresponds just to Lemma 3.38 in the case of a sentence φ.

The case of rigidly guarded FO could be proved by establishing the aperiodicity at the same time. However, in our case, it is sufficient to remark that, according to [Boj11], every data language definable in (non-necessarily rigidly guarded) FO is recognized by an aperiodic data monoid (in particular the syntactic one). Hence, if we consider the syntactic data monoid (we do not develop this object further here) of a language definable in rigidly guarded FO, it is aperiodic from [Boj11], and it is finite orbit as a quotient of the finite orbit data monoid obtained from Lemma 3.38. □

The proof of Lemma 3.38 is by structural induction on the rigidly guarded MSO formulas: the translation of the atomic formulas $x < y$, $a(x)$, $x \in Y$ are easy (at least towards non-aperiodic monoids) and the translations of the Boolean connectives are as in the classical case.

The translation of the existential closures (Lemma 3.40) uses a powerset construction on orbit finite data monoids. Since data monoids are in general infinite objects, the standard powerset construction would yield infinitely many orbits even if the original data monoid has finitely many of them. In our case, the construction remembers all possible elements of the original monoid, but since the morphism is projectable, one never has to store more than one element per orbit. Indeed, whenever another element in the same orbit is encountered, it has to be equal to the one already present: this limitation allows us to preserve orbit finiteness.

The most technical part concerns the translation of the rigidly guarded data tests $\varphi(x, y) \wedge x \sim y$ (Lemma 3.41). The rigidity assumption on the guard $\varphi(x, y)$ is crucial for this result: if $\varphi(x, y)$ were not rigid, then the data monoid recognizing $[\![\varphi(x, y) \wedge x \sim y]\!]$ would still be orbit finite, but the morphism would in general not be projectable. The proof that $[\![\varphi(x, y) \wedge x \sim y]\!]$ is recognized via a projectable morphism requires a bit of analysis since rigidity is a semantic assumption and hence one cannot directly deduce from it a property for the data monoid. However,

one can use the rigidity property for "normalizing" the data monoid, allowing the construction to go through.

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

Lemma 3.40. *Let $\varphi(\bar{X}, X_{m+1})$ be a formula and let $\varphi'(\bar{X}) = \exists X_{m+1} \,.\, \varphi(\bar{X}, X_{m+1})$. If $[\![\varphi]\!]$ is recognized by an orbit finite data monoid $\mathcal{M}$ via homomorphism h, then one can compute an orbit finite data monoid $\mathcal{M}'$ and a morphism h' such that $\mathcal{M}'$ recognizes $[\![\varphi']\!]$ via h'.*

Proof. For the sake of brevity, we denote the language $[\![\varphi]\!]$ over $D \times A \times B^{m+1}$ defined by the formula $\varphi(\bar{X}, X_{m+1})$ by L, and the language $[\![\varphi']\!]$ over $D \times A \times B^{m}$ defined by $\varphi'(\bar{X}) = \exists X_{m+1} \,.\, \varphi(\bar{X}, X_{m+1})$ by L' . We assume that L is recognized by an orbit finite data monoid $\mathcal{M}$ via a morphism h. We will exploit a variant of the powerset construction applied to the orbit finite data monoid $\mathcal{M}$ in order to obtain an orbit finite data monoid $\mathcal{M}'$ that recognizes L'. The same construction can be applied to a given restriction $\mathcal{M}|_C$ that represents $\mathcal{M}$ (see Proposition 3.34) in order to compute a restriction $\mathcal{M}'|_C$ that represents $\mathcal{M}'$. However, note that the cardinality of the set C must be at least twice the maximal size of the memories of the elements in $\mathcal{M}'$.

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

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

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

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

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

  for all $S \in M'$.

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

Another consequence of the above construction is that $\mathcal{M}'$ has finitely many orbits. Suppose, by way of contradiction, that there exist infinitely many sets $S_1, S_2, \ldots \in M'$ that are pairwise orbit distinct. First of all, we can turn each set S_i into an ordered sequence $\bar{s}_i = (s_{i,1}, \ldots, s_{i,k_i})$ (the choice of the order in which we list the elements is arbitrary and it is not relevant here). We can then denote by M_i the union of the memories of the elements in each tuple $\bar{s}_i$, namely, we let $M_i = \mathrm{mem}(s_{i,1}) \cup \ldots \cup \mathrm{mem}(s_{i,k})$. Without loss of generality, we can assume that

(i) all tuples $\bar{s}_1, \bar{s}_2, \ldots$ have the same cardinality k,

(ii) all sets $M_1, M_2, \ldots$ have the same cardinality l, and

(iii) for every index $1 \leq j \leq k$, the elements $s_{1,j}, s_{2,j}, \ldots$ are in the same orbit

(note that we can always enforce these assumptions by restricting to infinite subsequences $\bar{s}_{i_1}, \bar{s}_{i_2}, \ldots$). Now, for every index $i > 1$, we fix a renaming π_i such that $\pi_i(M_i) = M_1$ (recall that $|M_i| = |M_1| = l$). Similarly, for every pair of indices $i > 1$ and $1 \leq j \leq k$, we fix a renaming $\tau_{i,j}$ such that $\hat{\tau}_{i,j}(\hat{\pi}_i(s_{i,j})) = s_{1,j}$ (recall that $s_{1,j}$ and $s_{i,j}$ are in the same orbit). We then consider the functions $\tau_{i,j}$ restricted to the set M_1. From the Pigeonhole Principle, we have that there exist two indices $i \neq i'$ such that, for every $1 \leq j \leq k$,

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

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

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

Since $\hat{\pi}_i(s_{i,j}) = s_{1,j}$ and $\hat{\tau}_{i',j}^{-1} \circ \hat{\tau}_{i,j}$ is the identity on the set $M_1 \supseteq \mathrm{mem}(s_{1,j})$, we have that $\big(\hat{\tau}_{i',j}^{-1} \circ \hat{\tau}_{i,j}\big)(s_{1,j}) = s_{1,j}$ and hence $\big(\hat{\pi}_{i'}^{-1} \circ \hat{\pi}_i\big)(s_{i,j}) = s_{i',j}$ for all $1 \leq j \leq k$. In

conclusion, we have just shown that there exist two indices $i \neq i'$ and a renaming $\pi_{i,i'}$ $(= \pi_{i'}^{-1} \circ \pi_i)$ such that

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

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

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

$$h'(\langle u, U_1, \ldots, U_m\rangle) \;=\; \big\{h(\langle u, U_1, \ldots, U_m, U_{m+1}\rangle) \mid U_{m+1} \subseteq \mathrm{dom}(u)\big\}$$

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

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

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

Moreover, by definition of h', we have

$$\big\{h(\langle u, \bar{V}, V_{m+1}\rangle) \mid V_{m+1} \subseteq \mathrm{dom}(u)\big\} \;=\; \big\{\hat{\tau}(h(\langle u, \bar{U}, U_{m+1}\rangle)) \mid U_{m+1} \subseteq \mathrm{dom}(u)\big\}.$$

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

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

$$\big\{h(\langle u, \bar{U}, U_{m+1}\rangle) \mid U_{m+1} \subseteq \mathrm{dom}(u)\big\} \;=\; \big\{h(\langle v, \bar{V}, V_{m+1}\rangle) \mid V_{m+1} \subseteq \mathrm{dom}(v)\big\}$$

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

Guarded Equality Tests. We now turn to the translation of rigidly guarded data equality tests.

Lemma 3.41. *Given a rigid formula $\varphi(x,y)$, an orbit finite data monoid $\mathcal{M}$ and a projectable morphism h that recognizes $[\![\varphi]\!]$, one can compute an orbit finite data monoid $\mathcal{M}'$ and a projectable morphism h' that recognizes $[\![\varphi(x,y) \wedge x \sim y]\!]$.*

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

> ***Claim 1.*** *Given an orbit finite data monoid $\mathcal{M}$ and a morphism $h : (D \times A)^* \to \mathcal{M}$, one can compute the data sub-monoid $h((D \times A)^*)$.*
>
> *Proof of Claim 1.* Suppose that the orbit finite data monoid $\mathcal{M} = (M, \cdot, \hat{\ })$ is represented by its restriction $\mathcal{M}|_C$, for some finite subset C of D such that $|C| \geq 2|\mathrm{mem}(\mathcal{M})|$. Let $\mathcal{M}' = h\big((D \times A)^*\big)$ be the data sub-monoid induced by h. Clearly, we have $|\mathrm{mem}(\mathcal{M})| = |\mathrm{mem}(\mathcal{M}')|$ and hence, by Proposition 3.34, the data sub-monoid $\mathcal{M}'$ is uniquely determined by its restriction $\mathcal{M}'|_C$. Moreover, the domain of $\mathcal{M}'|_C$ is the finite set $h\big((A \times C)^*\big)$, which is computable from $\mathcal{M}|_C$ and $h|(A \times C)$. Finally, the product and the group action of the data sub-monoid $\mathcal{M}'|_C$ are the restrictions of the product and the group action of $\mathcal{M}$ to the finite set $h\big((A \times C)^*\big)$. This shows that $\mathcal{M}'|_C$ can be computed from $\mathcal{M}|_C$ and $h|(A \times C)$. This completes the proof of Claim 1. □

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

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

Definition 3.42. *Let h be a morphism from $(D \times A \times B^2)^*$ to a data monoid $\mathcal{M} = (M, \cdot, \hat{})$. We say that h is* 0-reduced *if for all data words $u \in (D \times A)^*$ and all positions $x, x', y, y' \in \mathrm{dom}(u)$, the following implications hold:*

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

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

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

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

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

data monoid $\mathcal{M}'$ as the quotient of $\mathcal{M}$ with respect to $\approx_G$, namely, as the triple $\mathcal{M}' = (M/_{\approx_G}, \odot, \check{})$, where

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

Note that both functions $\odot$ and $\check{}$ are well defined.

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

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

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

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

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

This shows that h' recognizes the language L.

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

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

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

$h(\langle u, V_1, V_2\rangle)$ belong to the ideal G and hence $h'(\langle u, U_1, U_2\rangle) = \{h(\langle u, U_1, U_2\rangle)\}$ and $h'(\langle u, V_1, V_2\rangle) = \{h(\langle u, V_1, V_2\rangle)\}$. Moreover, we have

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

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

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

We can now start with the main part of the proof of Lemma 3.41. Recall that $\mathcal{M} = (M, \cdot, \hat{\ })$ is the data monoid that recognizes $L = [\![\varphi(x, y)]\!]$ via the projectable morphism $h : (D \times A \times B^2)^* \to \mathcal{M}$. We denote by $\mathcal{N} = (N, \odot, \check{\ })$ the syntactic data monoid of the language defined by $x \sim y$ and by $g : (D \times A \times B^2)^* \to \mathcal{N}$ the corresponding morphism that recognizes $[\![x \sim y]\!]$. The data monoid $\mathcal{N}$ has finitely many orbits and its elements can be assumed to be terms of one the following forms:

1. $o(\varepsilon)$, which plays the role of the identity $1_{\mathcal{N}}$ in $\mathcal{N}$ and which corresponds to the image of the empty word under g;

2. $o(d)$, for any $d \in D$, which corresponds to the image of the data words expanded by a singleton predicate $U = \{x\}$ and the empty predicate $V = \varnothing$ under g;

3. $p(d)$, for any $d \in D$, which corresponds to the image of the data words expanded by the empty predicate $U = \varnothing$ and a singleton predicate $V = \{y\}$ under g;

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

5. $s(\varepsilon)$, which plays the role of the null element $0_{\mathcal{N}}$ in $\mathcal{N}$ and which corresponds to the image under g of the data words expanded with two non-empty predicates U, V that do not satisfy $x \sim y$

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

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

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

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

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

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

prove that $g(\langle u, U_1, U_2\rangle) = g(\langle u, V_1, V_2\rangle)$ holds as well. To do that, we distinguish between the following subcases:

1. $U_1 = U_2 = \varnothing$. We have $g(\langle u, U_1, U_2\rangle) = 1_{\mathcal{N}}$ and hence, since $1_{\mathcal{N}}$ has empty memory and $g(\langle u, U_1, U_2\rangle) \doteq g(\langle u, V_1, V_2\rangle)$, we immediately obtain $g(\langle u, V_1, V_2\rangle) = 1_{\mathcal{N}}$.

2. Both U_1 and U_2 are non-empty. In this case $g(\langle u, U_1, U_2\rangle)$ must be either the null element $0_{\mathcal{N}}$ or the term $r(\varepsilon)$ (recall that this term represents all expanded data words that satisfy $x \sim y$). Both elements have empty memory and hence from $g(\langle u, U_1, U_2\rangle) \doteq g(\langle u, V_1, V_2\rangle)$ we devise $g(\langle u, U_1, U_2\rangle) = g(\langle u, V_1, V_2\rangle)$.

3. $U_1 \neq \varnothing$ and $U_2 = \varnothing$. Clearly, U_1 is a singleton of the form $\{x\}$. Similarly, since $g(\langle u, U_1, U_2\rangle) \doteq g(\langle u, V_1, V_2\rangle)$, we have that V_1 is a singleton of the form $\{x'\}$ and $V_2 = \varnothing$. We then recall that $h(\langle u, U_1, U_2\rangle) = h(\langle u, V_1, V_2\rangle) \neq 0_{\mathcal{M}}$ and that the morphism h is 0-reduced, which implies that $x = x'$. This shows that $g(\langle u, U_1, U_2\rangle) = g(\langle u, V_1, V_2\rangle)$.

4. $U_1 = \varnothing$ and $U_2 \neq \varnothing$. This case is symmetric to the previous one and can be dealt with by similar arguments.

It follows that h' is a projectable morphism. □

Proof of Lemma 3.38. We are now ready to prove the Lemma 3.38.

Proof of Lemma 3.38. As already mentioned, the proof is by structural induction on the rigidly guarded MSO formula $\varphi(\bar{X})$. As for the base cases, we observe that the languages defined by the atomic formulas $x < y$, $a(x)$, and $x \in Y$ are recognized by suitable orbit finite data monoids and projectable morphisms. As for the inductive step, we suppose to be given a formula φ (resp., two formulas φ_1 and φ_2) with m free variables $X_1, \ldots, X_m$, an orbit finite data monoid $\mathcal{M}$ (resp., two orbit finite data monoids $\mathcal{M}_1$ and $\mathcal{M}_2$), and a projectable morphism $h : (D \times A \times B^m)^* \to \mathcal{M}$ (resp., two projectable morphisms $h_1 : (D \times A \times B^m)^* \to \mathcal{M}_1$ and $h_2 : (D \times A \times B^m)^* \to \mathcal{M}_2$) recognizing the languages defined by φ (resp., the languages defined by φ_1 and φ_2). We then observe that the language defined by the formula $\neg\varphi$ (resp., $\varphi_1 \wedge \varphi_2$) is recognized by the orbit finite data monoid $\mathcal{M}$ (resp., $\mathcal{M}_1 \times \mathcal{M}_2$) via the projectable morphism h (resp., $h_1 \times h_2$). As for the existential closure, Lemma 3.40 implies that the language defined by the formula $\exists X_m \,.\, \varphi$ is recognized by a suitable orbit finite data monoid $\mathcal{M}'$ via a projectable morphism h', both computable from $\mathcal{M}$ and h. Finally, if $m = 2$ and $\varphi(x_1, x_2)$ is a rigid formula, then we know from Lemma 3.41

how to compute an orbit finite data monoid $\mathcal{M}'$ and a projectable morphism h' that recognizes the language defined by $\varphi(x_1, x_2) \wedge x_1 \sim x_2$. This concludes the proof of the theorem. □

From Aperiodic Monoids to First-Order Logic

Having shown that every language defined by a rigidly guarded first-order logic formula is recognized by an aperiodic orbit finite data monoid, we now show the converse. The following lemma implies the direction from right to left of Theorem 3.7

Lemma 3.43. *Given an aperiodic orbit finite data monoid $\mathcal{M}$, a morphism h from the free data monoid to $\mathcal{M}$, and an orbit o, one can compute a rigidly guarded FO sentence φ that defines the data language $L = h^{-1}(o)$.*

Our proof follows a structure similar to Schützenberger's proof that languages recognized by aperiodic monoids are definable by star-free expressions (i.e., in first-order). Namely, the proof relies on an induction on the structure of ideals of the data-monoid, the so called *Green's relations*. This requires specific study of this theory for orbit finite data-monoids. Such a study was initiated by Bojańczyk [Boj11], but we had to develop several new tools for our proof to go through. As opposed to the classical case, the proof is significantly more involved for MSO compared to FO.

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

Definition 3.44. *We say that a formula* $\varphi(x_1, \dots, x_n)$ determines x_j *from* x_i *if for all words* w *and positions* x *in* w*, there is* y *such that* $w \vDash \varphi(x_1, \dots, x_n)$ *and* $x_i = x$ *implies* $x_j = y$.
The formula $\varphi(x_1, \dots, x_n)$ *is* rigid *if* x_i *determines* x_j *for all* i *and all* j *among* $1, \dots, n$ *(note that this is consistent with our previous definition).*

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

Definition 3.45. *A formula* $\varphi(x, y, z_1, \dots, z_n)$ *is a* witnessing *formula if it determines all variables from* x *and, symmetrically, all variables from* y*, and whenever* $w \vDash \varphi(x, y, z_1, \dots, z_n)$*, then* $x \leq z_i \leq y$ *for all* $1 \leq i \leq n$.
A formula witnesses the orbit o *if it is a witnessing formula and*

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

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

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

The proof of Lemma 3.43 is an induction based on the $\hat{\mathcal{J}}$-classes of $\mathcal{M}$ (see Section 3.2.3 for a definition of Green's relation $\hat{\mathcal{J}}$):

Lemma 3.46 (Inductive statement)**.** *For every* $\hat{\mathcal{J}}$*-class* $\hat{J}$ *of an aperiodic data monoid* $\mathcal{M}$*, the following claims hold:*

1. *there is a formula* $\varphi_{\hat{J}}(x, y)$ *such that* $w \vDash \varphi_{\hat{J}}(x, y)$ *iff* $h(w[x, y]) \in \hat{J}$*;*
2. *for every guard* $\alpha(x, y)$ *such that* $w \vDash \alpha(x, y)$ *implies* $h(w[i, j]) \geq_{\hat{\mathcal{J}}} \hat{J}$*, there exists effectively a family of formulas* F^α *computing the orbits under* α.

In the proof, a careful analysis of Green's relations and of memories of elements in an orbit finite data monoid is required.

Green's Theory and Memorable Values Here we focus our attention on the memorable values of the elements of orbit finite data monoids.

Definition 3.47. *Given an element m of an orbit finite data monoid $\mathcal{M}$, we define by* $\mathrm{mem}_{\mathcal{R}}(s)$ *(resp.,* $\mathrm{mem}_{\mathcal{L}}(s)$*) to be the intersection of* $\mathrm{mem}(t)$ *for all elements t in the $\mathcal{R}$-class (resp., $\mathcal{L}$-class) of s.*

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

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

Proposition 3.48. *For every element s of an orbit finite data monoid $\mathcal{M}$, we have* $\mathrm{mem}(s) = \mathrm{mem}_{\mathcal{R}}(s) \cup \mathrm{mem}_{\mathcal{L}}(s)$.

Before turning to the proof Proposition 3.48, let us show that a similar result fails for data monoids with infinitely many data orbits. Consider the data language $L_{\text{even}} \subseteq D^*$ that consists of all words $u \in D^*$ where every value $d \in D$ occurs in u an even number of times. The syntactic data monoid of the language L_{even} consists of one element m_C for each finite subset C of D. The product corresponds to the symmetric difference of sets. It is easy to see that the memorable values of m_C are exactly the values in C, which are neither $\mathcal{L}$-memorable nor $\mathcal{R}$-memorable (the data monoid is a group).

In order to prove Proposition 3.48, we need to introduce a couple of other concepts. An *inverse* of an element s of a monoid, is an element t such that $s{\cdot}t = t{\cdot}s = 1_{\mathcal{M}}$. If the inverse of s exists, then it can be easily proven to be unique and hence it can be denoted by s^{-1}. A *data group* is simply a *data monoid* where all elements have an inverse. The next lemma shows that orbit finiteness is, quite surprisingly, a severe restriction for data groups.

Lemma 3.49. *Every orbit finite data group is finite.*

Proof. We first observe two claims:

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

Claim 2. $\mathrm{mem}(s) \setminus \mathrm{mem}(t) \subseteq \mathrm{mem}(s \cdot t)$.

Proof of Claim 2. Indeed, $\mathrm{mem}(s) = \mathrm{mem}(s \cdot t \cdot t^{-1}) \subseteq \mathrm{mem}(s \cdot t) \cup \mathrm{mem}(t^{-1}) = \mathrm{mem}(s \cdot t) \cup \mathrm{mem}(t)$. Hence, $\mathrm{mem}(s) \setminus \mathrm{mem}(t) \subseteq \mathrm{mem}(s \cdot t)$. □

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

It is known that every $\mathcal{H}$-class H of a monoid is associated with a group $\Gamma(H)$ called the Schützenberger group [Pin11] (in fact there exist two such groups, but we will only consider one of them here). We define $T(H)$ to be the set of all elements $t \in H$ such that $t \cdot H$ is a subset of H. For each $t \in T(H)$ we let γ_t be the transformation on H that maps $h \in H$ to $t \cdot h$. Formally the Schützenberger group $\Gamma(H)$ consists of the set of all transformations γ_t with $t \in T(H)$. The multiplication operation of the group is the functional composition $\circ$. Moreover, there is a natural way to extend the action $\hat{}$ of the data monoid $\mathcal{M}$ to an action $\tilde{}$ on $\Gamma(H)$ by simply letting $\tilde{\tau}(\gamma_s) = \gamma_{\hat{\tau}(s)}$ for all renamings $\tau \in \Gamma_{D \setminus (\mathrm{mem}_{\mathcal{R}}(H) \cup \mathrm{mem}_{\mathcal{L}}(H))}$, where $\mathrm{mem}_{\mathcal{R}}(H) = \mathrm{mem}_{\mathcal{R}}(h)$ and $\mathrm{mem}_{\mathcal{L}}(H) = \mathrm{mem}_{\mathcal{L}}(h)$ for some arbitrary element $h \in H$ (note that all elements of H have the same set of $\mathcal{R}$-memorable values and the same set of $\mathcal{L}$-memorable values). The following lemma shows that $\tilde{}$ is indeed a group action on the Schützenberger group $\Gamma(H)$.

Lemma 3.50. *If $\mathcal{M} = (M, \cdot, \hat{})$ is a data monoid over the set D of data values and H is an $\mathcal{H}$-class of $\mathcal{M}$, then $(\Gamma(H), \circ, \tilde{})$ is a data group over the set $D \setminus (\mathrm{mem}_{\mathcal{R}}(H) \cup \mathrm{mem}_{\mathcal{L}}(H))$. Moreover, if $\mathcal{M}$ is orbit finite, then $(\Gamma(H), \circ, \tilde{})$ is orbit finite as well.*

Proof. It is known that $(\Gamma(H), \circ)$ is a group. We only need to verify that $\tilde{}$ is an action on $\Gamma(H)$. We first show that $\Gamma(H)$ is closed under the action $\tilde{}$ of the renamings over $D \setminus (\mathrm{mem}_{\mathcal{R}}(H) \cup \mathrm{mem}_{\mathcal{L}}(H))$. By definition of $\tilde{}$, this is equivalent to verifying that H is closed under the action $\hat{}$ of renamings over $D \setminus (\mathrm{mem}_{\mathcal{R}}(H) \cup \mathrm{mem}_{\mathcal{L}}(H))$. The following proof is similar to proof of the Memory Theorem for $\mathcal{J}$-classes in [Boj11]; however, we give a complete proof here for the sake of self-containment.

Suppose that H is the intersection of an $\mathcal{R}$-class R and an $\mathcal{L}$-class L. As a renaming is a permutation that is the identity on all but finite many values, any

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

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

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

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

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

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

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

Corollary 3.51. *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 3.50, we can associate with H an orbit finite data group $(\Gamma(H), \circ, \check{})$, where $\Gamma(H)$ is the Schützenberger group of H. By Lemma 3.49, it follows that $\Gamma(H)$ is finite. Moreover, it is known from classical results in algebra (see, for instance, [Pin11]), that the Schützenberger group $\Gamma(H)$ has the same cardinality as the $\mathcal{H}$-class H. We thus conclude that H is finite. □

We are now ready to prove Proposition 3.48.

Proof of Proposition 3.48. We aim at $\mathrm{mem}(s) \subseteq \mathrm{mem}_{\mathcal{R}}(s) \cup \mathrm{mem}_{\mathcal{L}}(s)$. Assume towards a contradiction that there is a value $d \in \mathrm{mem}(s) \setminus (\mathrm{mem}_{\mathcal{R}}(s) \cup \mathrm{mem}_{\mathcal{L}}(s))$. As $d \notin \mathrm{mem}_{\mathcal{R}}(s)$, there is an s' in the $\mathcal{R}$-class of s such that $d \notin \mathrm{mem}(s')$. Then there are $u, u' \in \mathcal{M}$ such that $s \cdot u = s'$ and $s' \cdot u' = s$. Symmetrically, as $d \notin \mathrm{mem}_{\mathcal{L}}(s)$, s has an $\mathcal{L}$-equivalent element s'' such that $d \notin \mathrm{mem}(s'')$, and there are $v, v'' \in \mathcal{M}$ such that $v \cdot s = s''$ and $v'' \cdot s'' = s$.

Let $d_1, d_2, \ldots$ be an infinite sequence of pairwise distinct values that are not in the memory of either s, s', or s''. We denote by π_i the transposition of d with d_i. As neither d nor $d_1, d_2, \ldots$ are in the memory of s', $\hat{\tau}_i(s') = s'$ and hence $\hat{\tau}_i(s \cdot u) = \hat{\tau}_i(s') = s'$. Combining this with $s' \cdot u' = s$ we obtain $\hat{\tau}_i(s) \cdot \hat{\tau}_i(u) \cdot u' = s' \cdot u' = s$ and hence $\hat{\tau}_i(s) \geq_{\mathcal{R}} s$. Similarly, one proves that $\hat{\tau}_i(s) \leq_{\mathcal{R}} s$ and hence $\hat{\tau}_i(s) \mathrel{\mathcal{R}} s$. By symmetry $\hat{\tau}_i(s) \mathrel{\mathcal{L}} s$. Hence $\hat{\tau}_i(s)$ belongs to the $\mathcal{H}$-class of s. As d is memorable in s, $\hat{\tau}_i(s)$ is different from s. Thus the $\mathcal{H}$-class of s is infinite, contradicting Corollary 3.51. □

Proof of the Inductive Statement Through the rest of this section we assume that $\mathcal{M}$ is an *aperiodic* orbit finite data monoid $\mathcal{M}$ and we prove the inductive statement given in Lemma 3.46.

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

It is also convenient to assume that $\mathcal{M}$ is represented by a term-based presentation system $\mathcal{S} = (T, \odot, \check{}, \approx)$. This means that the elements of $\mathcal{M}$ are the $\approx$-equivalence classes of terms in T. However, by a slight abuse of notation, we shall

often identify the elements of the data monoid $\mathcal{M}$ with the terms in T, writing, for instance, $h(w[x,y]) = o(d_1,\dots,d_k)$.

We start by presenting an special, but important, case of Lemma 3.46, which shows that the orbits of infixes of length 1 can be computed (this will serve as our base case for the inductive construction):

Lemma 3.52. *Let $\alpha^1(x,y) := x = y$. One can compute the orbits under the guard α^1.*

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

$$\begin{aligned}
\phi^1_o(x,y) &:= \bigvee_{h((d,a))=o()} a(x) \wedge x = y && \text{(if } o \text{ has memory size 0)}\\
\phi^1_o(x,y,z) &:= \bigvee_{h((d,a))=o(d)} a(x) \wedge x = y \wedge x = z && \text{(if } o \text{ has memory size 1)}\\
\phi^1_o(x,y) &:= \texttt{false}. && \text{(otherwise)}
\end{aligned}$$

□

The following lemma discloses another key argument in the proof of Lemma 3.46, which is obtained by syntactic transformations of formulas.

Lemma 3.53 (Sub-Definability Lemma)**.** *For all formulas $\varphi(x,y)$ that determine y from x, there exist finitely many formulas $\beta_i(z,y)$ that determine y from z, and such that for all $x \le z \le y$,*

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

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

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

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

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

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

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

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

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

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

and most importantly

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

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

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

$$\psi^*(x,y) \;:=\; \varphi^*(x,y) \wedge \forall\, y' \; \varphi^*(x,y') \to y' = y.$$

This formula ψ^* is equivalent to φ^* as long as φ^* determines y from x for a given choice of w, x. Otherwise, it simply does not hold. Hence x determines y in $\psi^*(x, y)$ by construction. This means that $\psi^*(x, y)$ is subject to the application of the standard MSO/FO case. Thus let $\beta_1^*(z, y), \ldots, \beta_k^*(z, y)$ be the corresponding formulas.

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

An immediate consequence of the above lemma is the following:

Corollary 3.54. *Every witnessing formula is equivalent to a finite disjunction of rigid formulas.*

Proof. Consider a witnessing formula $\varphi(x, y, z_1, \ldots, z_n)$. Since $\varphi(x, y, *, \ldots, *)$ determines y from x, one can apply Lemma 3.53 and get some formulas $\alpha_1, \ldots, \alpha_k$.

The desired rigid formulas are

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

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

Of course, $\varphi_{i_1,\ldots,i_n}$ implies φ by definition. Conversely, given some $x, y, z_1, \ldots, z_n$ such that $w \vDash \varphi(x, y, z_1, \ldots, z_n)$. For each $k = 1, \ldots, n$, by Lemma 3.53, there exists i_k such that $s \vDash \alpha_{i_k}(z_k, y)$. It follows that $w \vDash \varphi_{i_1,\ldots,i_n}(x, y, z_1, \ldots, z_n)$. Overall, φ is equivalent to a disjunction of rigid formulas. □

Corollary 3.54 can be used for composing families of formula that compute orbits, as shown by the following lemma:

Lemma 3.55. *If F and F' compute the orbits under the guards $\alpha(x, y)$ and $\alpha'(x, y)$, respectively, there exists effectively a family $F \cdot F'$ which computes the orbit under the guard*

$$(\alpha \cdot \alpha')(x, y) \;:=\; \exists z \,.\, \alpha(x, z) \wedge \alpha'(z + 1, y).$$

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

The orbit resulting from the product of an element in orbit o with an an element in orbit o' depends on the relative data values. For each possible relationship, we will produce a formula. This relationship will be represented by using explicit terms, which contain data values. Consider two terms of the form $t = o(c_1,\dots,c_k)$ and $t' = o'(c'_1,\dots,c'_{k'})$ (up to renaming, there are only finitely many possibilities for the pair (t,t')). Their product is $o''(d_1,\dots,d_n) \coloneqq t\cdot t'$. Call $(\varphi_p)_{p=1\dots m}$ (similarly $(\varphi'_{p'})_{p'=1\dots m'}$) the rigid formulas obtained from φ_o (resp. from $\varphi'_{o'}$) by Corollary 3.54. Consider now the formula:

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

where

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

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

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

Finally, the formula ψ_o is simply defined as:

$$\psi_o(x, y, \bar{z}) \coloneqq \bigvee_{t \cdot t' \in o} \psi_{t \cdot t'}(x, y, \bar{z}).$$

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

Using Lemma 3.52 and Lemma 3.55, one can compute the orbits of infixes of fixed length:

Corollary 3.56. *Let $\alpha^k(x, y) \coloneqq x + k - 1 = y$, there exists effectively a family of formulas which computes orbits under α^k.*

We now recall a direct consequence of Theorem V.1.9 from [Pin11]:

Lemma 3.57. *For every pair of elements s, t of $\mathcal{M}$, if $s \mathcal{J} t$ and $s \leq_{\mathcal{R}} t$, then $s \mathcal{R} t$ (and symmetrically for $\mathcal{L}$).*

The above lemma immediately implies the following:

Lemma 3.58. *Let $[x, y]$ be a minimal interval such that $h(w[x, y]) \not\leq_{\hat{J}} \hat{J}$, then either*

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

From now on, we assume that Claims C1 and C2 of Lemma 3.46 hold for every $\hat{\mathcal{J}}$-class above $\hat{J}$ (which is the induction hypothesis for classes above $\hat{J}$).

Lemma 3.59. *There exists a formula $\alpha^{\min}_{\not\leq_{\hat{\mathcal{J}}} \hat{J}}(x, y)$ such that $w \vDash \alpha^{\min}_{\not\leq_{\hat{\mathcal{J}}} \hat{J}}(x, y)$ iff $[x, y]$ is a minimal interval such that $h(w[x, y]) \not\leq_{\hat{\mathcal{J}}} \hat{J}$. Furthermore, it is rigid, and one can compute the orbits under $\alpha^{\min}_{\not\leq_{\hat{\mathcal{J}}} \hat{J}}$.*

Proof. Lemma 3.58 describes what can be such intervals $[x, y]$. Naturally, the last case is the most interesting. Let $\alpha(x, y)$ be a formula stating that $[x, y]$ is a maximal interval such that $h(w[x, y]) >_{\hat{\mathcal{J}}} \hat{J}$ and $x < y$. It is possible to write such a formula using the induction hypothesis C1. This formula is rigid, by definition. Hence, using this time Claim C2, one has a family of formulas F which computes the orbit under the guard α.

The formula $\alpha^{\min}_{\not\leq_{\hat{\mathcal{J}}} \hat{J}}(x, y)$ is then simply:

$$\begin{aligned} & \alpha^1(x, y) \wedge F^1(x, y) \not\leq_{\hat{\mathcal{J}}} \hat{J} \\ \vee\ & \alpha^2(x, y) \wedge F^2(x, y) \not\leq_{\hat{\mathcal{J}}} \hat{J} \\ \vee\ & ((\alpha^1 \cdot \alpha) \cdot \alpha^1)(x, y) \wedge ((F^1 \cdot F) \cdot F^1)(x, y) \not\leq_{\hat{\mathcal{J}}} \hat{J} \end{aligned}$$

where we use families of formulas $F, \dots$ as if they were functions computing orbits. This shorthand of notation should be clear to understand, and can be transformed into regular formulas by explicitly unfolding all cases. This is correct by Lemma 3.58.

Finally, this formula $\alpha^{\min}_{\not\in \hat{\mathcal{J}}\, \hat{J}}(x,y)$ is rigid by definition, and a family of formulas computing the orbits under this guard is also easy to obtain using the same kind of constructions. □

We are now ready to prove the induction steps for both Claim C1 and Claim C2 of Lemma 3.46 with respect to the $\hat{\mathcal{J}}$ -class $\hat{J}$ (we remark that only the proof of C2 relies on the fact that the orbit finite data monoid is aperiodic).

Lemma 3.60 (Induction step for C1)**.** *There exists a formula $\varphi_{\hat{J}}(x,y)$ such that $w \vDash \varphi_{\hat{J}}(x,y)$ iff $h(w[x,y]) \in \hat{J}$.*

Proof. The formula disproves the existence of a minimal interval $[x',y']$ included in $[x,y]$ such that $\alpha^{\min}_{\not\in \hat{\mathcal{J}}\, \hat{J}}(x',y')$ holds, using Lemma 3.59. This implies $h(w[x,y]) \geq_{\hat{\mathcal{J}}} \hat{J}$. The formula then excludes the case $h(w[x,y]) >_{\hat{\mathcal{J}}} \hat{J}$ using the induction hypothesis C1 for all $\hat{K} >_{\hat{\mathcal{J}}} \hat{J}$. □

Lemma 3.61 (Induction step for C2)**.** *For every guard $\alpha(x,y)$ such that $w \vDash \alpha(x,y)$ implies $h(w[x,y]) \geq_{\hat{\mathcal{J}}} \hat{J}$, one can compute the orbits under α.*

Proof. Of course, it is sufficient to prove the lemma for the case when $w \vDash \alpha(x,y)$ implies $h(w[x,y]) \in \hat{J}$, since using Claim C2 it is possible to compute the orbits in the other cases. Let $\alpha^{\min}_{\hat{J}}(x,y)$ be the formula expressing that $[x,y]$ is minimal such that $h(w[x,y]) \in \hat{J}$ (doable thanks to Lemma 3.60). From its definition, $\alpha^{\min}_{\hat{J}}$ is rigid.

Consider the formula $\beta(x,y,z_1,z_2,z_3,z_4)$ which holds if (i) $\alpha(x,y)$, (ii) $z_1 \geq x$ is minimal such that $\alpha^{\min}_{\hat{J}}(z_1,z_2)$, and (iii) $z_4 \leq y$ is maximal such that $\alpha^{\min}_{\hat{J}}(z_3,z_4)$. Remark first that since (i) implies $h(w[x,y]) \in \hat{J}$, all points z_1, z_2, z_3, and z_4 belong to $[x,y]$. Hence, β is a witnessing formula, and by Corollary 3.54, it is equivalent to a disjunction of rigid formulas, say $\beta_1, \dots, \beta_n$.

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

1. It detects for which i, $w \vDash \beta_i(x,y,z_1,z_2,z_3,z_4)$ holds, and guesses the corresponding variables z_1, z_2, z_3, z_4 (this implies in particular $w \vDash \alpha(x,y)$, hence $h(w[x,y]) \in \hat{J}$).

2. It computes the orbit of $h(w[x, z_2])$ (and $h(w[z_3, x])$ in a similar way). This is doable since:

   By definition of β, if $w \vDash \beta_i(x, *, y-1, *, *, *)$ then $h(w[x, y-1]) >_{\hat{\mathcal{J}}} \hat{J}$. Hence, using the induction hypothesis C2, one can compute the orbits under the guard $\beta_i(x, *, y-1, *, *, *)$. Furthermore by Lemma 3.59, one can compute the orbits under $\alpha^{\min}_{\hat{\jmath}}$. Overall, one can compute the orbits under $\beta_i(x, *, y-1, *, *, *) \cdot \alpha_{\hat{\jmath}}(y, z)$, i.e., under $\beta_i(x, *, *, y, *, *)$. This means that one can compute the orbit of $h(w[x, z_2])$.

3. Recall that in any aperiodic orbit finite data monoid all $\mathcal{H}$-classes are singletons. Since the considered data monoid is aperiodic, we have that $t = h(w[x, y])$ is the only element (up to $\approx$) such that $t \in \mathcal{R}(h(w[x, z_2])) \cap \mathcal{L}(h(w[z_3, y]))$. It follows that we know the orbit of $h(w[x, y])$. It remains to provide witnesses for the data values.

   Since $h(w[x, z_2]) \; \mathcal{J} \; h(w[x, y])$ and $h(w[x, z_2]) \geq_{\mathcal{R}} h(w[x, y])$, we have that $h(w[x, z_2]) \; \mathcal{R} \; h(w[x, y])$. In the symmetric way, $h(w[z_3, y]) \; \mathcal{L} \; h(w[x, y])$. Hence, by Proposition 3.48, each memorable value from t occurs either in $h(w[x, z_2])$ or in $h(w[z_3, y])$. This means that every data value in t is already witnessed at step 2.

□

The above arguments prove Lemma 3.46. We observe that the lemma directly implies Lemma 3.43 that deals with the aperiodic case. The following Corollary completes the proof of Theorem 3.7.

Corollary 3.62. *Every data language recognized by a finite orbit aperiodic data monoid is definable by a rigidly guarded FO sentence.*

Proof. One should provide, given an orbit o, a formula which holds over a word w iff $h(w) \in o$. Other situations are obtained by disjunction of this single orbit case. Consider the guard $\alpha(x, y) = (\neg \exists z \,.\, z < x) \wedge (\neg \exists z \,.\, z > y)$. It holds iff x is the first position and y the last position of the word. Since by Claim C2 of Lemma 3.46 one can compute the orbits under α, the language $h^{-1}(o)$ is definable. □

3.6 Towards Büchi for Data

Having shown characterizations of first-order logic and variants thereof in the last chapter, we now turn to monadic second-order logic. Over finite alphabets, the expressiveness of monoids and finite automata coincides. In addition, both formalisms are precisely as expressive as monadic second-order logic.

Over infinite alphabets, monoids are much weaker than automata. We have seen that orbit finite data monoids are strictly contained in deterministic finite memory automata, which are in turn strictly contained in non-deterministic finite memory automata. In this section we present two logics that correspond to orbit finite monoids and non-deterministic finite memory automata respectively.

3.6.1 A Logic For Orbit Finite Data Monoids

In this section, we show that rigidly guarded first-order logic, extended with monadic second-order quantification, corresponds exactly to orbit finite data monoids. That is, we show the following theorem

Theorem 3.8. *A language can be defined in rigidly-guarded monadic second-order logic iff it is accepted by an orbit finite data monoid. In addition, the translations between the two formalisms are effective.*

Before we start with the proof of Theorem 3.8 we note the following key corollary:

Corollary 3.13. *The satisfiability problem for rigidly guarded MSO logic is decidable. Moreover, one can decide whether a formula belongs to the rigidly guarded MSO logic, and in this case whether the formula is rigid.*

Proof. By Lemma 3.38, any formula of rigidly guarded MSO can be effectively transformed into an orbit finite data monoid. Satisfiability of this formula then corresponds to language non-emptiness. This can be tested by a simple saturation argument on the algebraic object.

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

$$\alpha^{\text{rigid?}} \;:=\; \forall\, x, x', y, y'\; \alpha(x, y) \wedge \alpha(x', y') \to (x = x' \leftrightarrow y = y')$$

holds on all words. The latter condition can be tested using the first claim of the corollary. □

We have already shown the direction from left to right of Theorem 3.8 in Lemma 3.38. For the other direction, we prove a lemma that is similar to Lemma 3.43, the difference being that it is concerned with monadic second-order logic and arbitrary orbit finite data monoids, instead of first-order logic and aperiodic orbit finite data monoids.

Lemma 3.63. *Given an orbit finite data monoid $\mathcal{M}$, a morphism h from the free data monoid to $\mathcal{M}$, and an orbit o, one can compute a rigidly guarded MSO sentence φ that defines the data language $L = h^{-1}(o)$.*

We have seen in the previous chapter how to establish Lemma 3.63 in the aperiodic case. The other part, stating that every data language recognized by an orbit finite data monoid is definable in rigidly guarded MSO logic, is proved by following the same structure, i.e., by relying on the same induction on $\hat{\mathcal{J}}$-classes and on similar constructions. Only two things need to be changed. The first one is that Lemma 3.53 needs to be reproven for rigidly guarded MSO (doable with the exact same proof). The second issue lies in the proof of Lemma 3.61, when the hypothesis of aperiodicity is used for the first time, namely at step 3.

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

The objective is to reprove Lemma 3.46, but this time without assuming that the underlying monoid is aperiodic. We note that the proof of Claim C1 given in Section 3.5.5 does not exploit the assumption that the monoid is aperiodic (as far as the induction hypothesis is admitted). Hence we can reuse this part of the proof for the data monoid $\mathcal{M}$. It thus remains to prove Claim C2 for a given $\hat{\mathcal{J}}$-class $\hat{J}$.

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

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

Definition 3.64. *A* partition *of a word w is a decomposition of w into $u_0, \ldots, u_{n+1}$ where each u_i with $1 \leq i \leq n$ is non-empty. We say that a partition $u_0, \ldots, u_{n+1}$ of a word w is an* $\hat{J}$-partition *if*

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

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

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

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

Definition 3.65. *A partition $u_0, \ldots, u_{n+1}$ of w is* rigidly guarded *under a guard $\alpha(x, y)$ if*

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

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

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

Definition 3.66. *A* presentation *for a partition $u_0, \ldots, u_{n+1}$ of w is a pair (X, Y) of monadic second-order variables such that*

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

Lemma 3.67. *For every rigid guard $\alpha(x,y)$, there is a formula $\alpha_{almost}(X,Y)$ such that for all words w, $w \vDash \alpha_{almost}(X,Y)$ iff (X,Y) is a presentation for an almost-$\hat{J}$-partition of w. In addition, such a partition is orbit-computable under the guard α. Furthermore, for all words w, if $w \vDash \alpha(x,y)$, then there is a pair (X,Y) such that $w \vDash \alpha_{almost}(X,Y)$.*

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

- $h(u_1 \cdot \ldots \cdot u_n) \in \hat{J}$. This can be done using the formula $\varphi_{\hat{J}}$ from Claim C1 of Lemma 3.46.
- For all $1 \le i \le n-1$, either $h(u_i) \in \hat{J}$ or $h(u_{i+1}) \in \hat{J}$. This can be checked by a combination of $\alpha^{\min}_{\not\in \hat{\mathcal{J}}\hat{J}}$ from Lemma 3.59 and of $\varphi_{\hat{J}}$ from Claim C1.
- For all $1 \le i \le n-1$, either u_i is a minimal interval in $\hat{J}$ or $h(u_i)$ is in a $\hat{\mathcal{J}}$-class above $\hat{J}$. The first case can be checked by $\alpha^{\min}_{\not\in \hat{\mathcal{J}}\hat{J}}$. The second case can be checked using the formulas $\varphi_{\hat{J}}$.

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

We now show how α_{almost} can check that the partition $u_0,\ldots,u_{n+1}$ is rigidly guarded under α. Clearly it can check that $w \vDash \alpha(x,y)$, where x' is the first position of u_1 and y is the last position of u_n. To check the second condition of Definition 3.65 we define the following finite set $G^{\rightarrow}$ of formulas:

1. The formula $\alpha^{\min}_{\not\in \hat{\mathcal{J}}\hat{J}}(x',y')$ from Lemma 3.59 is in $G^{\rightarrow}$.
2. Recall that $\alpha(x,y)$ is a rigid guard. By the Sub-definability Lemma 3.53 there exist finitely many formulas $\beta_i(x',y')$ for α that determine y' from x'. We let all these formulas β_i be contained in $G^{\rightarrow}$.
3. Let $\beta^{\leftarrow}_1(y',z'),\ldots,\beta^{\leftarrow}_k(y',z')$ be the formulas obtained from the rigid formula $\alpha^{\min}_{\not\in \hat{\mathcal{J}}\hat{J}}(x',z')$ using again Lemma 3.53 (we consider here the version of the lemma for formulas that determine y' from z'). $G^{\rightarrow}$ contains all formulas of the form

$$\beta^{\rightarrow}_j(x',y') \;:=\; \exists\, z'\; \alpha^{\min}_{\not\in \hat{\mathcal{J}}\hat{J}}(x',z') \wedge \beta^{\leftarrow}_j(z',y').$$

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

- Case: u_i is a minimal interval in $\hat{J}$. In this case the formula $\alpha^{\min}_{\not\leq_{\hat{\mathcal{J}}}\hat{J}}(x', y')$ has the desired properties.
- Case $i = n$. Then there is a formula among the $\beta_i(x', y')$ with the desired properties.
- Otherwise u_i is an interval in a $\hat{J}$-class above $\hat{J}$, and $1 \leq i \leq n-1$. In this case we claim that there is an index j such that $\beta^{\rightarrow}_j \in G^{\rightarrow}$ is the suitable formula: Assume that x' is the position at the beginning of u_i and that y' is the position at the end of u_i. Then $\alpha^{\min}_{\not\leq_{\hat{\mathcal{J}}}\hat{J}}(x', z')$ determines some z' from x' such that $w[x', y']$ is a minimal interval in $\hat{J}$. As u_i is in a higher $\hat{J}$-class than $\hat{J}$ and $1 \leq i \leq n-1$, it follows that z' must be in u_{i+1}. As $u_1, \ldots, u_n$ is an almost-$\hat{J}$ partition, u_{i+1} must be a minimal interval in $\hat{J}$. We observed before that these intervals are rigidly guarded by $\alpha^{\min}_{\not\leq_{\hat{\mathcal{J}}}\hat{J}}(x', y')$. By the Sub-definability Lemma 3.53, there must be a formula $\beta^{\rightarrow}_j(x', y')$ that determines y' from x'.

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

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

Clearly each formula in G is rigid (possibly unsatisfiable). It follows from the above observation that G satisfies the second condition of Definition 3.65.

Let us sum up what we have proven so far: The partition $u_0, \ldots, u_{n+1}$ of w induced by (X, Y) is a special almost-$\hat{J}$-partition where each interval is either in a $\hat{\mathcal{J}}$-class above $\hat{J}$, or it is a minimal interval in $\hat{J}$. In addition, the partition is guarded by G. Hence we can compute the orbit of each of the u_i under the respective guard by using Lemma 3.59 and the induction hypothesis. This means that $u_0, \ldots, u_{n+1}$ is orbit-computable.

What remains to be done is to show that such a partition exists. Consider an almost-$\hat{J}$-partition $u_0, \ldots, u_{n+1}$ such that for every $1 \leq i \leq n-1$, either u_i or u_{i+1} is minimal in $h^{-1}(\hat{J})$. If there exists still some u_i such that $h(u_i) \in \hat{J}$, but which is not minimal, then such u_i can be decomposed into $vv'v''$, where v' is minimal such that $h(v') \in \hat{J}$ and either v or v'' is non-empty. We substitute in the partition $vv'v''$

for u_i, yielding a finer partition which still satisfies the property. This refinement process is iterated, starting from the trivial partition of w into a single word, and it stops when there are no more non-minimal u_i's such that $h(u_i) \in \hat{J}$. If no extra refinement steps are possible, this means that the expected form for the partition has been obtained. □

Lemma 3.68. *For every rigid guard $\alpha(x,y)$, there is a formula $\alpha_{partition}(X,Y)$ that defines rigidly guarded $\hat{J}$-partitions orbit-computable under $\alpha(x,y)$. Furthermore, for all words w, if $w \vDash \alpha(x,y)$, then there is at least one pair (X,Y) such that $w \vDash \alpha_{partition}(X,Y)$.*

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

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

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

What remains to be done is to prove that this partition is rigidly guarded and that one can compute the orbits under α. However, this is very simple. Indeed, since each the u_i's are rigidly guarded by one of the α_i's, each of the v_i's is rigidly guarded by one of the $\alpha_i \cdot \alpha_j$ for some $1 \leq i,j \leq k$. One can also compute the orbits using Lemma 3.55. □

At this point we know how the formula will be looking for a witness of the value of $h(w[x,y])$ by guessing a partition $u_0, u_1, \ldots, u_n, u_{n+1}$ such that $h(w[x,y] = h(u_1 \ldots u_n)$ using Lemma 3.68. It remains to develop a tool for being able to evaluate $h(w[x,y]) = h(u_1 \ldots u_n)$, knowing $h(u_1), \ldots, h(u_n)$. Of course, this cannot be done by comparing all the data values involved in $h(u_1), \ldots, h(u_n)$. There are too many of them, and it is impossible to do it using rigidly guarded tests. We show below that it is sufficient to perform some sort of an approximation of this computation. The necessary machinery is defined below.

Definition 3.69. *Two sequences of terms $t_1, \ldots, t_n$ and $t'_1, \ldots, t'_n$ are* locally consistent *if*

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

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

Lemma 3.70. *Let $t_1, \ldots, t_n$ and $t'_1, \ldots, t'_n$ be two sequences of terms.*

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

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

> ***Claim 1.*** *If two sequences of terms are locally consistent, then they define the same domain $\mathcal{D}$ and the same equivalence E, and, furthermore, the corresponding colorings are border-equal.*

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

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

Claim 2. *Two locally consistent sequences of terms that have a small difference have the same value.*

Proof of Claim 2. Let $t_1, \ldots, t_n$ and $t'_1, \ldots, t'_n$ be locally consistent sequences of terms that have a small difference. Let C be the equivalence class for which the two colorings differ. Let i be the smallest number such that there is $(i,k) \in C$ and let j be the biggest number such that $(j,k') \in C$ for some $k, k' \in \mathbb{N}$. As $t_1, \ldots, t_n$ and $t'_1, \ldots, t'_n$ are locally consistent, it follows that $1 < i$ and $j < n$. Let d be the color of C in $t_1, \ldots, t_n$, and d' be the color of C in $t'_1, \ldots, t'_n$. We define τ to be the permutation that swaps the data values d and d' and is the identity elsewhere. We observe that:

- $\tau \circ \tau$ is the identity.
- $\hat{\tau}(t_i) = t'_i$ for all $i \in \{i, \ldots, j\}$.
- $t_i = t'_i$ for all $i \in \{1, \ldots, i-1\} \cup \{j+1, \ldots, n\}$.
- $\hat{\tau}(s') = s'$ for $s' = t'_{i-1} \cdot \ldots \cdot t'_{j+1}$. This is because neither d nor d' are memorable in s': by Proposition 3.48 every value is either an $\mathcal{L}$-memorable value or a $\mathcal{R}$-memorable value. As $t'_{i-1}, \ldots, t'_{j+1}$ are all in the same $\hat{\mathcal{J}}$-class, it follows that all $\mathcal{R}$-memorable values in s' must occur in t'_{i-1} and all $\mathcal{L}$-memorable values in s' must occur in t'_{j+1}. As neither d nor d' occur in either t'_{i-1} or t'_{j+1} it follows that d and d' are not memorable in s'.

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

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

□

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

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

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

The general idea for proving the induction step of Claim C2 of Lemma 3.46 is that the formula that computes the orbits under a rigid guard α will guess a suitable $\hat{J}$-partition $u_0, u_1, \ldots u_n, u_{n+1}$ of the underlying data word w. For evaluating the product $h(u_1) \cdot h(u_2) \cdot h(u_n)$, the formula will guess a sequence of terms $t_1 \ldots t_n$ which is locally consistent with $h(u_1) \ldots h(u_n)$. This may look difficult a priori, because there are infinitely many possible terms (as there are infinitely many data values) and an MSO formula would not be able to guess such a sequence. However, our last preparatory lemma shows that it suffices to considers sequences of terms built over a set of data values of bounded cardinality.

Lemma 3.71. *For each sequence of terms $t_1 \ldots t_n$, there is a sequence of terms $t'_1 \ldots t'_n$ that is locally consistent with $t_1 \ldots t_n$ and uses at most $4\|\mathcal{M}\|$ distinct data values.*

Proof. We can reuse the objects in the proof of Lemma 3.70, namely, we can view a sequence of terms as a domain, an equivalence relation, and a coloring. We can assume without loss of generality that:

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

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

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

We can prove the induction step for Claim C2 of Lemma 3.46.

Lemma 3.72 (Induction step for C2)**.** *For every guard $\alpha(x, y)$ such that $w \vDash \alpha(x, y)$ implies $h(w[x, y]) \geq_{\hat{\mathcal{J}}} \hat{J}$, one can compute the orbits under α.*

Proof. The construction starts as in the proof of Lemma 3.61, but things change at point 3 since the construction there exploits aperiodicity. We continue the construction, without the assumption of aperiodicity, as follows.

3. One guesses a $\hat{J}$-partition which is rigidly guarded and orbit-computable using the formula $\alpha_{\text{partition}}(X, Y)$ of Lemma 3.68. We denote by $u_0, u_1, \ldots, u_n, u_{n+1}$ this partition.
4. One guesses some terms $t_1, \ldots, t_n$ over a finite set data set C of size $4\|\mathcal{M}\|$. There are finitely many such terms, so this can be done using monadic quantifications. This must be done in such a way that the information concerning t_i is

located on the factor u_i, say on the first letter. One then computes $t = t_1 \cdot \ldots \cdot t_n$. This can be done inductively using MSO since the partial products of the terms $t_1, \ldots, t_n$ range over a finite set.

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

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

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

   - if $d_l = d'_{l'}$, then $(\exists\, y\ \varphi(*, y, \bar{*}, z_l, \bar{*}) \wedge \varphi'(y+1, *, \bar{*}, z'_{l'}, \bar{*})) \wedge z_l \sim z'_{l'}$ holds (the guard is rigid since φ and φ' are rigid),
   - if $d_l \neq d'_{l'}$, then $(\exists\, y\ \varphi(*, y, \bar{*}, z_l, \bar{*}) \wedge \varphi'(y+1, *, \bar{*}, z'_{l'}, \bar{*})) \wedge z_l \not\sim z'_{l'}$ holds (same argument for the rigidity).

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

   - if $d_l = d'_{l'}$, then $(\exists\, x, y\ \varphi(x, *, \bar{*}, z_l, \bar{*}) \wedge \alpha(x, y) \wedge \varphi(*, y, \bar{*}, z'_{l'}, \bar{*})) \wedge z_l \sim z'_{l'}$ holds (rigidity holds as above, since α is rigid),
   - if $d_l \neq d'_{l'}$, then $(\exists\, x, y\ \varphi(x, *, \bar{*}, z_l, \bar{*}) \wedge \alpha(x, y) \wedge \varphi(*, y, \bar{*}, z'_{l'}, \bar{*})) \wedge z_l \not\sim z'_{l'}$ holds (same argument for the rigidity).

6. At this point, $h(w)$ has to be in the same orbit as t. What remains to be done is witness the data values. For this, once more one uses Proposition 3.48. Since $h(w)\ \hat{\mathcal{J}}\ h(u_1)$, and $h(w) \leq_{\hat{\mathcal{J}}} h(u_1)$, we have $h(w)\ \hat{\mathcal{R}}\ h(u_1)$. In the same way, we have $h(w)\ \hat{\mathcal{L}}\ h(u_n)$. This means that every memorable value of $h(w)$ is already present either in $h(u_1)$ or in $h(u_n)$. Hence, the formula can locate in a rigid way the positions of the memorable values of $h(u_1)$ and $h(u_n)$. If $t_1 \cdot \ldots t_n$

contains a data value d and d occurs in t_1, then the corresponding position in u_1 that witnesses the memorable value d is used, and symmetrically for u_n.

□

3.6.2 A Logic For Non-Deterministic Finite Memory Automata

In this section, we present a logic that defines exactly the languages that are accepted by non-deterministic finite memory automata. This can be considered as an example of how rigid guards can help constructing decidable logics.

Backward-Rigidly Guarded MSO

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

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

where $a \in A$ and $\varphi_{\text{backward}}$ is a backward-rigid formula generated from the same grammar (as usual, we can enforce backward-rigidity syntactically).

Note that, like rigidly guarded MSO, backward-rigidly guarded MSO is a logic for defining adorned data languages over the alphabet $D \times A$. Hence if we seek a correspondence to finite memory automata, we need to consider a variant of finite memory automata that accept adorned data languages. Such an model can easily obtained from the one given in Definition 2.1 by allowing transition of the form (p, a, α, E, q), where p, q are states, $a \in A$, α is the isomorphism type of a data word of length $\mathrm{ar}(p) + 1$, $E \subseteq \{1, \ldots, \mathrm{ar}(p) + 1\}$,and $\mathrm{ar}(q) = \mathrm{ar}(p) + 1 - |E|$. The details of this definition are omitted. In the rest of this section, we will only consider finite memory automata over adorned words.

One can translate backward-rigidly guarded MSO formulas to equivalent deterministic FMA, but not the other way around:

Proposition 3.73. *Every language definable in backward-rigidly guarded MSO is recognizable by deterministic FMA. There is a language recognized by a deterministic FMA which cannot be defined in backward-rigidly guarded MSO.*

We divide the proof of Proposition 3.73 into two lemmas, one dealing with the translation of backward-rigidly guarded MSO formulas into equivalent deterministic FMA, and the other one dealing with the counterexample for the converse translation.

Lemma 3.74. *Every language definable in backward-rigidly guarded MSO is recognizable by deterministic FMA.*

Proof. We want to exploit closure properties of (suitable forms of) deterministic FMA under complementation (corresponding to negations of formulas), intersection (corresponding to conjunctions of formulas), and projection (corresponding to existential quantifications).

Let us first discuss the operation of projection on deterministic FMA. In general, this operation maps a deterministic FMA to a non-deterministic FMA. Moreover, unlike classical finite automata, arbitrary non-deterministic FMA cannot be determinized: the standard subset construction might turn a non-deterministic FMA into one with an infinite state space (note that the same issue occurs when projecting from an orbit finite data monoid). We solve this problem by restricting to a special form of deterministic FMA, which is similar to the notion of projectability for data monoid morphisms given (see Section 3.5.5).

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

Definition 3.75. *An FMA $\mathcal{A}$ over the alphabet $D \times A \times B^m$ is* projectable *if (i) it is deterministic and complete and (ii) for all data words $w \in (D \times A)^*$ and for all tuples of predicates $\bar{U} = (U_1, \dots, U_m)$ and $\bar{V} = (V_1, \dots, V_m)$, if the configurations $\mathcal{A}(\langle w, \bar{U}\rangle)$ and $\mathcal{A}(\langle w, \bar{V}\rangle)$ have the same control state, then they coincide.*

We now prove the relevant closure properties of projectable FMA.

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

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

$$L' \coloneqq \left\{\langle w, \bar{U}\rangle \mid \exists\, U_{m+1} \subseteq \mathrm{dom}(w)\ \ \langle w, \bar{U}, U_{m+1}\rangle \in \mathscr{L}(\mathcal{A})\right\}$$

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

- $k' = k \cdot |Q_0 \cup \ldots \cup Q_k|$, namely, $\mathcal{A}'$ uses up to k registers for each state of $\mathcal{A}$;
- for all $0 \leq i \leq k'$, the set Q'_i contains all partial functions f from $Q_0 \cup \ldots \cup Q_k$ into the powerset of $\{1, \ldots, i\}$ such that
  1. if $q \in Q_j$ and $f(q)$ is defined then $|f(q)| = j$;
  2. $\bigcup_{q \in \mathrm{dom}(f)} f(q) = \{1, \ldots, i\}$.

  Intuitively, a configuration (f, r) of $\mathcal{A}'$, with $f \in Q'_i$ and $r \in D^i$, represents the set of all configurations of $\mathcal{A}$ of the form $(q, r|_{f(q)})$ with $q \in \mathrm{dom}(f)$.
- The transition relation T' is defined in such a way that whenever $\mathcal{A}'$ is in a configuration (f, r) and it reads $(d, a, b_1, \ldots, b_m)$, then it can move to any configuration (g, s) such that
  1. if a state q of $\mathcal{A}$ is in the domain of g, then there is a symbol $b_{m+1} \in \{0, 1\}$ and a control state $p \in \mathrm{dom}(f)$ such that $\mathcal{A}$ moves from $(p, r|_{f(p)})$ to $(q, s|_{g(q)})$ when reading $(d, a, b_1, \ldots, b_m, b_{m+1})$;
  2. if a state q of $\mathcal{A}$ is not in the domain of g, then there is no symbol $b_{m+1} \in \{0, 1\}$, no control state $p \in \mathrm{dom}(f)$, and no set $I \subseteq \{1, \ldots, k'\}$ such that $\mathcal{A}$ moves from $(p, r|_{f(p)})$ to $(q, s|_I)$ when reading $(d, a, b_1, \ldots, b_m, b_{m+1})$.

  We observe that, according to the above explanation, whether $\mathcal{A}'$ moves from a configuration (f, r) to a configuration (g, s) by reading $(d, a, b_1, \ldots, b_m)$ depends only on the control states f and g and on the *equality relationships* between the value d and the values in r and s. In particular, it depends neither on the concrete data value d, nor on the memory contents r and s. For this reason, one can turn the above description into a formal definition for the transition relation T' that uses isomorphism types and that satisfies the sanity conditions introduced at the beginning of this section (we omit the tedious details of such a definition). Below, we verify that these transition rules are deterministic when restricted to reachable control states.
- I' is the partial function that maps every initial state in I of $\mathcal{A}$ to the empty set and that is undefined on all other states;
- F' is the set of all partial functions $f \in Q'_0 \cup \ldots \cup Q'_{k'}$ such that $\mathrm{dom}(f) \cap F \neq \varnothing$.

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

We now argue that $\mathcal{A}'$ can be turned into a projectable FMA by simply pruning the unreachable control states (this can be done by a simple reachability analysis, see for instance [BLP10b]). In the following we assume that $\mathcal{A}'$ contains no unreachable states.

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

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

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

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

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

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

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

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

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

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

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

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

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

Closure under intersection and complementation of projectable FMA is proved, as in the classical case, by computing "products" of projectable FMA and by complementing the set of final control states, respectively.

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

Definition 3.76. *An FMA $\mathcal{A}$ over the alphabet $D \times A \times B^2$ is* 0-reduced projectable *if (i) it is projectable and (ii) for all data words $w \in (D \times A)^*$ and all positions $x, x', y \in \mathrm{dom}(w)$, if the two configurations $\mathcal{A}(\langle w, \{x\}, \varnothing \rangle)$ and $\mathcal{A}(\langle w, \{x'\}, \varnothing \rangle)$ coincide, then $x = x'$ or both configurations are a 0-configuration.*

We now prove the following:

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

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

We need to prove that $\mathcal{A}'$ is 0-reduced projectable. For this we use the fact that the automaton recognizes the language $[\![\varphi]\!]$, which is defined by the *backward-rigid* formula $\varphi(x, y)$. Let $w \in (D \times A)^*$ be a data word and let $x, x' \in \mathrm{dom}(w)$ be two positions in it. By way of contradiction, we assume that (i) $x \neq x'$ and (ii) the two configurations $\mathcal{A}'(\langle w, \{x\}, \varnothing\rangle)$ and $\mathcal{A}'(\langle w, \{x'\}, \varnothing\rangle)$ coincide but they are not the 0-configuration $(q_{\text{sink}}, \varepsilon)$. From this we argue that $\varphi(x, y)$ is not backward-rigid. Since the configuration $\mathcal{A}'(\langle w, \{x\}, \varnothing\rangle)$ is not a 0-configuration, it can reach a final configuration, namely, there is an expanded data word $\langle v, U_1, U_2\rangle \in (D \times A \times B^2)$ such that

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

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

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

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

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

1. a control state q_{wait}, with empty memory, for parsing the prefix up to $\min(x, y)$,
2. a control state q_{check}, with memory size 1, for parsing the infix from $\min(x, y)+1$ to $\max(x, y)$,

3. a final control state q_{accept}, with empty memory, for parsing the suffix from $\max(x, y) + 1$ to the end, in the case where the data test $x \sim y$ was successful,

4. a sink control state q_{reject}, with empty memory, for parsing the same suffix above, but in the case where the data test $x \sim y$ failed.

One can construct an FMA $\mathcal{C}$ recognizing $[\![\varphi(x, y) \wedge x \sim y]\!]$ using a suitable product between the 0-reduced projectable FMA $\mathcal{A}'$ and the deterministic (but not projectable) FMA $\mathcal{B}$ described above. This product is very similar to the 0-collapse product of orbit finite data monoids (see the proof of Lemma 3.41 in Section **??**). Intuitively, $\mathcal{C}$ mimics the computations of $\mathcal{A}'$ and $\mathcal{B}$ on the input word, but as soon as $\mathcal{A}'$ reaches the (unique) 0-configuration, the component that simulates the computation of $\mathcal{B}$ is reset to a special void configuration with empty memory. Of course the defined FMA $\mathcal{C}$ is deterministic and complete, as $\mathcal{A}'$ and $\mathcal{B}$ are so.

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

1. If $U_1 = U_2 = \varnothing$, then $\mathcal{B}(\langle u, U_1, U_2\rangle) = (q_{\text{wait}}, \varepsilon)$. Moreover, since $\mathcal{C}(\langle u, U_1, U_2\rangle)$ and $\mathcal{C}(\langle u, V_1, V_2\rangle)$ share the same control state, the two components of $\mathcal{C}(\langle u, U_1, U_2\rangle)$ and $\mathcal{C}(\langle u, V_1, V_2\rangle)$ for simulating $\mathcal{B}$ encode the same control state of $\mathcal{B}$. In particular, it follows that $\mathcal{B}(\langle v, V_1, V_2\rangle)$ has control state q_{wait} and hence $\mathcal{B}(\langle u, U_1, U_2\rangle) = \mathcal{B}(\langle v, V_1, V_2\rangle)$.

2. If both U_1 and U_2 are non-empty, then $\mathcal{B}(\langle u, U_1, U_2\rangle)$ is either $(q_{\text{accept}}, \varepsilon)$ or $(q_{\text{reject}}, \varepsilon)$. Using arguments similar to the previous case, one obtains $\mathcal{B}(\langle u, U_1, U_2\rangle) = \mathcal{B}(\langle v, V_1, V_2\rangle)$.

3. If $U_1 \neq \varnothing$ and $U_2 = \varnothing$, then clearly U_1 is a singleton of the form $\{x\}$ and the configuration $\mathcal{B}(\langle u, U_1, U_2\rangle)$ contains the control state q_{check} and the data value at the position x of u. Similarly, since the configurations $\mathcal{B}(\langle u, U_1, U_2\rangle)$ and $\mathcal{B}(\langle u, V_1, V_2\rangle)$ share the same control state, we have that V_1 is a singleton of the form $\{x'\}$ and $V_2 = \varnothing$. Moreover, since the two configurations $\mathcal{A}'(\langle u, U_1, U_2\rangle)$ and $\mathcal{A}'(\langle u, U_1, U_2\rangle)$ coincide and they are different from the 0-configuration, we have $x = x'$. This implies $\mathcal{B}(\langle u, U_1, U_2\rangle) = \mathcal{B}(\langle v, V_1, V_2\rangle)$.

4. The last case $U_1 = \varnothing$ and $U_2 \neq \varnothing$ is similar to the previous one.

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

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

Lemma 3.77. *There is a language recognized by a deterministic FMA which cannot be defined in backward-rigidly guarded MSO.*

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

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

for which there is a sequence of indices $1 = i_1 < i_2 < \ldots < i_k = n$ satisfying $d_{i_j+1} = d_{i_{j+1}}$ for all $1 \leq j < k$ and $d_{i_j+1} \neq d_h$ for all $i_j + 1 < h < i_{j+1}$ [2].

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

Below, we fix a generic formula φ of backward-rigidly guarded MSO and we show that it cannot define the language L''. For this, we let k be the number of

[2] This language is a variant of an example given in [Tan11] to study data languages recognized by pebble automata.

occurrences of data comparisons of the form $\alpha(y,z) \wedge y \sim z$ in φ. We then consider a family of data words of the form

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

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

Using an encoding similar to the proof of Corollary 3.54, one can turn each data word w_n into a classical word w_n^* over the finite alphabet $\{0,1\}^k$ where every position y of w_n^* is labeled by a k-tuple of boolean values, each of them representing the "observable" equality between the value at the position y of w_n and the value at the unique position x determined from y via the backward-rigid guard of a corresponding data test $\alpha(x,y) \wedge x \sim y$ of φ. Accordingly, we can rewrite every sub-formula β of φ into a formula β^* of standard MSO by replacing each backward-rigid guard $\alpha(x,y)$ of a data test in β by a fresh unary predicate

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

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

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

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

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

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

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

carries the value n, determines the position x of the other occurrence of the value n in w_n. This shows that

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

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

We are now ready to prove Proposition 3.73.

Proof. The first claim follows from Lemma 3.74. The second claim follows from Lemma 3.77. □

∃-Backward-Rigidly Guarded MSO

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

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

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

Theorem 3.9. *A language is definable in ∃backward-rigidly guarded MSO iff it is recognizable by non-deterministic FMA.*

Proof. Having established Lemma 3.74, the translation from an ∃backward-rigidly formula $\exists\, \bar{Z}\, \varphi$ to an equivalent non-deterministic FMA that recognizes the language $L = [\![\exists\, \bar{Z}\, \varphi]\!]$ is straightforward. For this, we first translate φ into a *deterministic* FMA $\mathcal{A}$ that recognizes the language $L' = [\![\varphi]\!]$ (note that this language is over the alphabet $D \times A \times B^m$, where $m = |\bar{Z}|$). After that, we project $\mathcal{A}$ onto the alphabet $D \times A$: the resulting automaton is a non-deterministic FMA that guesses the correct

interpretation for the second-order variables $\bar{Z}$ on any input data word that belongs to the language $L = [\![\exists\, \bar{Z}\ \varphi]\!]$.

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

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

1. For all positions $1 \leq x \leq |w|$, exactly one variable Z_t with $t \in T$ holds at x (i.e., $\forall\ x\ \bigvee_{t \in T} x \in Z_t \wedge \bigvee_{t \neq t' \in T} x \notin Z_t \vee x \notin Z_{t'}$). We will say that a transition rule t is *encoded* at a position x if $x \in Z_t$ holds.

2. For all positions $1 \leq x < |w|$, the target state of the transition rule encoded at position x coincides with the source state of the transition rule encoded at position $x + 1$ (i.e., $\forall\ x\ \bigvee_{t=(p,\theta,E,q), t'=(q,a,\theta',E',q') \in T} x \in Z_t \wedge x + 1 \in Z_{t'}$).

3. The transition rule encoded at the first position of the word starts with an initial control state and the transition rule encoded at the last position of the word ends with a final control state.

4. For all positions $1 \leq x \leq |w|$, if $t = (p, a, \theta, E, q)$ is the transition rule encoded at position x, then $a = a_x$, where a_x is the symbol that appears at position x of the input word.

5. For all positions $1 \leq x \leq |w|$, if $t = (p, a, \theta, E, q)$ is the transition rule encoded at position x, then θ is the isomorphism type of $r_{x-1} \cdot d_x$, where r_x is the memory content at position $x - 1$ in the encoded run of $\mathcal{A}$ and d_x is the data value at position x of the input word .

   To see how this check is done, consider an example of a partial run of $\mathcal{A}$:

$$(q_0, r_0) \xrightarrow{(d_1, a_1)} (q_1, r_1) \xrightarrow{(d_2, a_2)} \ldots \xrightarrow{(d_{x-1}, a_{x-1})} (q_{x-1}, r_{x-1}).$$

For each register i of r_{x-1}, we need to determine the "provenance" of the value $r_{x-1}[i]$ in the prefix $(d_1, a_1)(d_2, a_2)\dots(d_{x-1}, a_{x-1})$ of the input word, namely, we need to locate some position $y \leq x-1$ such that $d_y = r_{x-1}[i]$. We thus reconstruct, for each $z \leq x-1$, the position $f_{x,i}(z)$ of the occurrence of $r_{x-1}[i]$ (if any) in the memory r_z. This can be done by a suitable formula that first guesses some monadic variables $W_1, \dots, W_k$ encoding the partial function $f_{x,i}(z)$, and then verifies, using the run encoded by the variables $\bar{Z}$, that the guess is correct. One can also do this using only first-order quantifications, since the partial function $f_{x,i}(z)$ can only decrease as z increases; however, we prefer not to discuss this option in detail. We then compute the leftmost position y such that $f_{x,i}(z)$ is defined on all $z = y, \dots, x-1$. Clearly, this implies $d_y = r_{x-1}[i]$.

The formula that determines y from x using $\bar{Z}$ is of the form

$$\alpha_i(y, x, \bar{Z}) \;:=\; \beta_i(y, x, \bar{Z}) \wedge \forall\, y'\; y' < y \rightarrow \neg\beta_i(y', x, \bar{Z})$$

where

$$\begin{aligned}\beta_i(y, z, \bar{Z}) \;:=\; & \exists\, W_1, \dots, W_k\; \forall\, z \\ & y \leq z \leq x-1 \rightarrow \Big(\bigvee_{1\leq j\leq k} z \in W_j \wedge \neg \bigwedge_{1\leq j<j'\leq k} z \in W_j \wedge z \in W_{j'} \\ & \qquad \wedge \bigvee_{\substack{1\leq j\leq k,\ p\in Q_j \\ t=(p,a,\theta,E,q)\in T}} z \in Z_t \wedge z \in W_j \wedge z-1 \in W_{j-|E\cap\{1,\dots,j\}|} \Big)\end{aligned}$$

Note that $\alpha_i(y, x, \bar{Z})$ is an $\exists$backward-rigid formula. We can thus use α_i as a guard to test equality between the data value d_x at position x and the data value d_i $(= r_{x-1}[i])$ at position y.

Using the above constructions we can check that the isomorphism type of the transition rule encoded at position x coincides with the isomorphism type of $r_{x-1} \cdot d_x$.

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

As it happens for rigidly guarded MSO logic, one can derive from Theorem 3.9 the following decidability results:

Corollary 3.78. *The satisfiability problem for ∃backward-rigidly guarded MSO is decidable. Moreover, one can decide whether a formula belongs to the ∃backward-rigidly guarded MSO.*

Proof. By Theorem 3.9, any formula of ∃backward-rigidly guarded MSO can be effectively transformed into a non-deterministic FMA. Satisfiability of ∃backward-rigidly guarded MSO then corresponds to non-emptiness of languages recognized by non-deterministic FMA. The latter problem is known to be decidable from [KF94].

The second claim follows from arguments similar to the proof of Corollary 3.13, that is, one can inductively check that every sub-formula satisfies the syntactic restrictions given by the grammar of ∃backward-rigidly guarded MSO. For the sub-formulas $\alpha(x, y, \bar{Z})$ that are guards of data equality tests, one has also to verify that α is ∃backward-rigid by checking the validity of the formula

$$\alpha^{\exists\text{backward?}} \;:=\; \forall\, x, x', y, \bar{Z}\; \alpha(x, y, \bar{Z}) \wedge \alpha(x', y, \bar{Z}) \rightarrow x = x'. \quad \square$$

It is also easy to generalize both Theorem 3.9 and Corollary 3.78 to data *tree* languages recognized by non-deterministic finite memory tree automata [KT08]. For this we use a natural variant of ∃backward-rigidly guarded MSO on data trees. The guarded tests in this case are of the form

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

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

Finally, it is natural to look for effective characterizations of data languages that are both recognizable by non-deterministic FMA and definable in (unrestricted) FO. However, it is known that such characterization cannot be achieved: in Section 3.5.1 it has been shown that the problem of determining whether a language recognized by a non-deterministic FMA is definable in FO is undecidable. The problem of characterizing FO within the class of languages recognizable by deterministic FMA is still open.

3.7 Conclusions

In this chapter we have studied logics for data words, mostly first-order logic and variants thereof. We have focused on the relationship of the expressiveness of

	local-DMA	1-register DMA	o.f. data monoids	DMA
NUFO	✓	✓		
rigidly guarded FO			✓	
FO^2			✓	
local-NUFO				✓
local FO				✓

Table 3.1: Effective Characterizations of data languages.

these logics with language accepting devices such as automata and monoids. We overviewed the relationships of the logics to one another, the relationships of the automata and monoids to one another, and the relationships of the logics, the automata, and the monoids.

We then investigated the decidability of definability in logical classes within automata for data word languages. We show that this problem is undecidable for natural extensions of deterministic finite memory automata, such as two-way deterministic memory automata, non-deterministic finite memory automata, and a very weak version of pebble automata. For this reason we focused on deterministic finite memory automata, and restrictions thereof. We consider the following restrictions: local automata, whose transitions can only depend a bounded suffix of the input word, and orbit finite data monoids, a kind of infinite monoids whose expressiveness lies strictly between local automata and deterministic finite memory automata.

We considered two versions of logical definability: uniform definability, where a language is defined by a single formula, and non-uniform definability, where the formula can vary for different sizes of the alphabet. Within these classes we consider several restrictions of monadic second-order logic. We consider both restrictions that have been considered previously, such as first-order logic, as well as new restrictions that can be obtained by guarding the use of the data equality predicate.

We were able to prove several results of the form: Given a language L that is accepted by a deterministic finite memory automaton or a restriction thereof, and a logic $\mathcal{L}$, it is decidable whether L can be defined in $\mathcal{L}$. In some of these results, the class of languages defined by the logic $\mathcal{L}$ and the class of automata (or monoids) are incomparable. Table 3.1 shows an overview over the effective characterizations that have been obtained in these cases. In other cases we were able to show that the expressiveness of logics corresponds precisely with the expressiveness of models of automata or classes of monoids (see Figure 3.10). These results show that it is

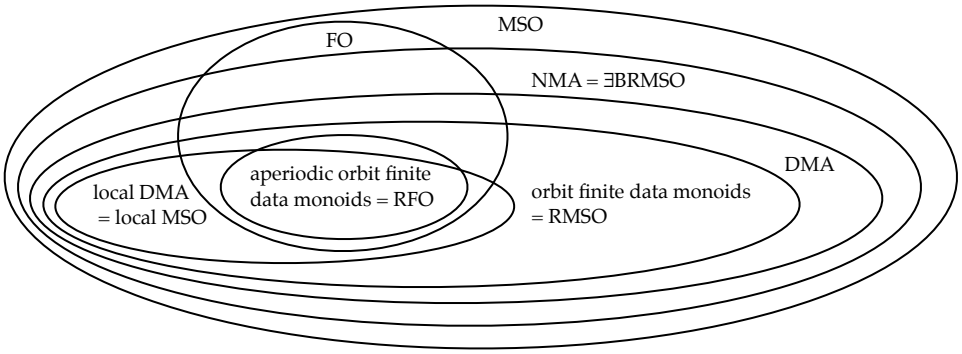


Figure 3.10: A overview over correspondences between automata and logics for data words.

in principle possible to extend the fundamental aspects of the theory of the regular languages to data languages.

One question that is left open is that of an effective characterization of $\mathrm{FO}^2(\sim, <)$ over *adorned* data words. Over plain words we have shown that whether a $\mathrm{FO}^2(\sim, <)$ formula holds on a data word depends only on a prefix and a suffix that contains a certain number of distinct values. This is not true for adorned languages. Consider for example the language "the value at leftmost a-labeled position must coincide with the value at the rightmost a-labeled position". This language can be defined in $\mathrm{FO}^2(\sim, <)$, but membership might depend on positions that are not in any prefix or suffix with a bounded number of distinct values. A related problem that is also open is an effective characterization of $\mathrm{FO}^2(\sim, +1)$.

Other open questions concern the relationship between logics and automata. We defined a logic that has exactly the same expressiveness as non-deterministic finite memory automata. We have shown that a natural restriction of our logic for non-deterministic finite memory automata is strictly weaker than deterministic finite memory automata. However, no logic is known that corresponds exactly to deterministic finite memory automata however.

The main open question is whether there are larger classes of data languages for which effective characterizations can be established. Many quite simple languages, such as "all values are different", cannot be recognized by orbit finite data monoids. It would be interesting to find a class of data monoids that includes such simple languages but still allows for effective characterizations. Ideally, such a class of languages should include all languages that can be defined by deterministic finite memory automata.

Conclusions and Open Questions

In this thesis we have investigated the question whether fundamental properties of the regular word languages can be extended to richer structures. We have both positive and negative results:

In the first chapter, we showed that first-order complete query languages for trees easily become incomplete if their navigational capabilities are restricted. This is problematic as query languages that cannot navigate in all directions of a tree can be evaluated more efficiently in some settings. For example, the unidirectional fragment of XPath is favorable to full XPath when evaluated over XML streams [Olt07, BJLW08]. Our results show that it is not clear how XPath could be extended to a first-order complete query language that can still be evaluated efficiently over XML streams. This clearly raises the question: What is a first-order complete unidirectional query language? To the best of our knowledge, no such language is known to date.

Other interesting open questions are concerned with the precise expressiveness of the languages we consider in the first chapter. Thérien and Wilke [TW04] have given an algebraic characterization of restrictions of LTL that are obtained by restricting the until-depth. It would be interesting to know whether such a characterization is also possible for the languages $\mathrm{CTL}^*_{\triangleleft\triangleright}(\mathrm{ud} \leq k)$ that we define in the first chapter. In addition, it is not known to which fragments of first-order logics these languages correspond to.

In the second part of the thesis we considered word languages over an infinite alphabet. Much work has been devoted towards extending properties of the regular word languages to data word languages. Most of this research, has focused on three aspects of the regular languages: decidability, closure properties, and the relationship between logics and automata.

We considered a different aspect of regularity: minimization and the ability to decide membership in logics. We gave a counterpart of the theorem of Myhill and Nerode for deterministic finite memory automata. Our theorem characterizes the

class of languages that are accepted by deterministic finite memory automata if the alphabet is associated with either data equality or a linear order on the data values. In addition, in the special case where the underlying alphabet is equipped with the equality relation, we provided an algorithm to minimize deterministic finite memory automata and we investigated the complexity of this minimization algorithm.

It would be interesting to know whether these results can be extended to automata that can access an arbitrary order on the data values. In addition other generalizations of finite memory automata could be taken into consideration. One possibility would be to extend finite memory automata with the ability to store sets of unbounded sizes in their registers. One could imagine several different operations to update these sets: In one version an automaton could check which sets contain the current input value on each transition. It could then store this value in some sets or move this value from one set to a new one. This model seems closely related to class memory automata [BS10]. In a second, possibly more expressive version, the automaton could, in addition, merge two sets when transitioning into a new state. Both kinds of automata accept some languages that are not accepted by finite memory automata. For example the language "all values are different" can be accepted by storing all values that have been encountered in a prefix of a word in one set. It would be interesting to know whether unique minimal representatives exists for these kinds of automata and whether they can be computed effectively.

In the last chapter of the thesis we proved several theorems that are inspired by results of Schützenberger, McNaughton, and Papert. Their results provide an effective characterization of first-order logic with respect to the regular languages. We have shown that effective characterizations are possible for logics that define data languages. We gave several effective characterizations of logics with respect to deterministic finite memory automata, and restrictions of them. In particular, we give a counterpart of the result of Schützenberger, McNaughton, and Papert for orbit finite data monoids: We show that a language can be defined in rigidly guarded first-order logic iff it is accepted by an aperiodic orbit finite data monoid. We also show that natural extensions of this logic correspond exactly to orbit finite data monoids and non-deterministic finite memory automata.

The main question that remains open is whether there is an effective characterization of (unrestricted) first-order logic with respect to deterministic finite memory automata. More generally one could try to find a maximal class of data word languages for which effective characterizations are possible. It is likely that this would

be a very well behaved class of languages, and possibly, a candidate for a truly regular class of data word languages.

Bibliography

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[ASW09] Mihhail Aizatulin, Henning Schnoor, and Thomas Wilke. Computationally sound analysis of a probabilistic contract signing protocol. pages 571–586, 2009.

[AZ08] Jorge Almeida and Marc Zeitoun. Description and analysis of a bottom-up dfa minimization algorithm. *Inf. Process. Lett.*, 107:52–59, July 2008.

[BBCF10] Jean Berstel, Luc Boasson, Olivier Carton, and Isabelle Fagnot. Minimization of automata. *CoRR*, abs/1010.5318, 2010.

[BC08] Marie-Pierre Béal and Maxime Crochemore. Minimizing incomplete automata. In *Workshop on Finite State Methods and Natural Language Processing*, 2008.

[BH10] Benedikt Bollig and Loïc Hélouët. Realizability of dynamic msc languages. In *CSR*, pages 48–59. 2010.

[BJ07] Michael Benedikt and Alan Jeffrey. Efficient and expressive tree filters. In *FSTTCS*, pages 461–472, 2007.

[BJLW08] Michael Benedikt, Alan Jeffrey, and Ruy Ley-Wild. Stream firewalling of xml constraints. In *SIGMOD Conference*, pages 487–498, 2008.

[BKL11] Mikolaj Bojanczyk, Bartek Klin, and Slawomir Lasota. Automata with group actions. pages 355–364, 2011.

[BL05] Pablo Barceló and Leonid Libkin. Temporal logics over unranked trees. In *LICS*, pages 31–40, 2005.

[BL10] Mikolaj Bojanczyk and Slawomir Lasota. An extension of data automata that captures xpath. In *LICS*, pages 243–252, 2010.

[BLP10a] Michael Benedikt, Clemens Ley, and Gabriele Puppis. Automata vs. logics on data words. In *CSL*, pages 110–124, 2010.

[BLP10b] Michael Benedikt, Clemens Ley, and Gabriele Puppis. Minimal memory automata. Technical report, Oxford University Computing Laboratory, 2010. Available at `http://www.comlab.ox.ac.uk/michael.benedikt/papers/myhilldata.pdf`.

[BLP10c] Michael Benedikt, Clemens Ley, and Gabriele Puppis. What you must remember when processing data words. 2010.

[BM08] Henrik Björklund and Wim Martens. The tractability frontier for nfa minimization. In *ICALP (2)*, pages 27–38, 2008.

[BMS+06] Mikolaj Bojanczyk, Anca Muscholl, Thomas Schwentick, Luc Segoufin, and Claire David. Two-variable logic on words with data. In *LICS*, pages 7–16, 2006.

[Boj08] Mikołaj Bojańczyk. Effective Characterizations of Tree Logics. In *PODS*, 2008.

[Boj11] Mikolaj Bojanczyk. Data monoids. In *Symposium on Theoretical Aspects of Computer Science*, pages 105–116, 2011.

[Bol11] Benedikt Bollig. An automaton over data words that captures EMSO logic. F.4, 2011.

[BP89] Daniele Beauquier and Jean-Eric Pin. Factors of words. In *ICALP*, 1989.

[BPT03] Patricia Bouyer, Antoine Petit, and Denis Thérien. An algebraic approach to data languages and timed languages. *Inf. Comput.*, 182:137–162, May 2003.

[BS10] Henrik Björklund and Thomas Schwentick. On notions of regularity for data languages. *Theor. Comput. Sci.*, 411:702–715, January 2010.

[Büc60] J. Richard Büchi. Weak second-order logic and finite automata. *S. Math. Logik Grunlagen Math.*, 6:66–92, 1960.

[CLP11] Thomas Colcombet, Clemens Ley, and Gabriele Puppis. On the use of guards for logics with data. *Mathematical Foundations of Computer Science*, 2011.

[CLT05] Julien Cristau, Christof Löding, and Wolfgang Thomas. Deterministic automata on unranked trees. In *FCT*, pages 68–79, 2005.

[Dav10] Julien David. The average complexity of moore's state minimization algorithm is o(n log log n). In *MFCS*, pages 318–329, 2010.

[DL09] Stéphane Demri and Ranko Lazić. Ltl with the freeze quantifier and register automata. *ACM Trans. Comput. Logic*, 10:16:1–16:30, April 2009.

[Eme90] E. Allen Emerson. Temporal and modal logic. In Jan van Leeuwen, editor, *Handbook of theoretical computer science (vol. B)*, pages 995–1072, Cambridge, MA, USA, 1990. MIT Press.

[EW00] Kousha Etessami and Thomas Wilke. An Until Hierarchy and Other Applications of an Ehrenfeucht-Fraïsse Game for Temporal Logic. *Information and Computation*, 160:88–108, 2000.

[Fig10] Diego Figueira. Forward-xpath and extended register automata on data-trees. In *ICDT*, pages 231–241, 2010.

[FK03] Nissim Francez and Michael Kaminski. An algebraic characterization of deterministic regular languages over infinite alphabets. *Theor. Comput. Sci.*, 306(1-3):155–175, 2003.

[GP02] Murdoch J. Gabbay and Andrew M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13(3-5):341–363, 2002.

[GPSS80] Dov Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. On the temporal analysis of fairness. In *POPL*, 1980.

[Gre51] J. A. Green. On the structure of semigroups. *Annals of Mathematics*, 54:163–172, 1951.

[HMU06] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation).* Addison-Wesley Longman, 2006.

[HNSY92] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111:394–406, 1992.

[Hop71] John E. Hopcroft. An n log n algorithm for minimizing states in a finite automaton. Technical report, Stanford University, Stanford, CA, USA, 1971.

[HT87] Thilo Hafer and Wolfgang Thomas. Computation Tree Logic CTL* and Path Quantifiers in the Monadic Theory of the Binary Tree. In *ICALP*, 1987.

[JR93] Tao Jiang and Bala Ravikumar. Minimal nfa problems are hard. volume 22, pages 1117–1141, 1993.

[Kam68] Hans Kamp. *Tense logic and the theory of linear order*. PhD thesis, University of California, Los Angeles, 1968.

[KF94] Michael Kaminski and Nissim Francez. Finite-memory automata. *TCS*, 134(2), 1994.

[KT08] Michael Kaminski and Tony Tan. Tree automata over infinite alphabets. In *Pillars of Computer Science*, pages 386–423, 2008.

[LB09] Clemens Ley and Michael Benedikt. How big must complete xml query languages be? In *ICDT*, pages 183–200, 2009.

[Lib04] Leonid Libkin. *Elements of Finite Model Theory*. Springer, August 2004.

[LPS11] Alexei Lisitsa, Igor Potapov, and Rafiq Saleh. Planarity of knots, register automata and logspace computability. In *LATA*, pages 366–377. 2011.

[LS08] Leonid Libkin and Cristina Sirangelo. Reasoning about xml with temporal logics and automata. In *LPAR*, pages 97–112, 2008.

[Mar04] Maarten Marx. Conditional xpath, the first order complete xpath dialect. In *PODS*, pages 13–22, 2004.

[Mar05a] Maarten Marx. Conditional xpath. *ACM Trans. Database Syst.*, 30:929–959, December 2005.

[Mar05b] Maarten Marx. First order paths in ordered trees. In *ICDT*, pages 114–128, 2005.

[MdR05] Maarten Marx and Maarten de Rijke. Semantic characterizations of navigational xpath. *SIGMOD Rec.*, 34:41–46, June 2005.

[MN07] Wim Martens and Joachim Niehren. On the minimization of xml schemas and tree automata for unranked trees. *J. Comput. Syst. Sci.*, 73:550–583, June 2007.

[Moo56] Edward F. Moore. Gedanken Experiments on Sequential Machines. In *Automata Studies*, pages 129–153. Princeton U., 1956.

[MP71] Robert McNaughton and Seymour A. Papert. *Counter-Free Automata (M.I.T. research monograph)*. The MIT Press, 1971.

[MR99] Faron Moller and Alexander Moshe Rabinovich. On the expressive power of ctl. In *LICS*, pages 360–369, 1999.

[MT11] Andrzej S. Murawski and Nikos Tzevelekos. Algorithmic nominal game semantics. In *ESOP*, pages 419–438. 2011.

[NSV04] Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Logic*, 5:403–435, July 2004.

[Olt07] Dan Olteanu. Spex: Streamed and progressive evaluation of xpath. *IEEE Trans. on Knowl. and Data Eng.*, 19:934–949, July 2007.

[Pin11] J.E. Pin. Mathematical foundations of automata theory. Available on: `http://www.liafa.jussieu.fr/~jep/MPRI/MPRI.html`, 2011.

[Pot95] Andreas Potthoff. First-order logic on finite trees. In *TAPSOFT*, pages 125–139, 1995.

[Rab02] Alexander Moshe Rabinovich. Expressive power of temporal logics. In *CONCUR*, pages 57–75, 2002.

[Rab08] Alexander Rabinovich. Personal communciation, 2008.

[Rev92] Dominique Revuz. Minimisation of acyclic deterministic automata in linear time. *Theor. Comput. Sci.*, 92:181–189, January 1992.

[RM00] Alexander Moshe Rabinovich and Shahar Maoz. Why so many temporal logics climb up the trees? In *MFCS*, pages 629–639, 2000.

[Sch65] M.P. Schtzenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190 – 194, 1965.

[Sch07] Thomas Schwentick. Automata for xml—a survey. *J. Comput. Syst. Sci.*, 73:289–315, May 2007.

[Seg06] Luc Segoufin. Automata and logics for words and trees over an infinite alphabet. In *CSL*, pages 41–57, 2006.

[SM73] Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In *STOC*, pages 1–9, 1973.

[Sto74] L. J. Stockmeyer. *The Complexity of Decision Problems in Automata Theory*. PhD thesis, Department of Electrical Engineering, MIT, 1974.

[Str94] Howard Straubing. *Finite automata, formal logic, and circuit complexity*. Birkhauser Verlag, Basel, Switzerland, Switzerland, 1994.

[SZ11] Thomas Schwentick and Thomas Zeume. Two-variable logic with two order relations. volume abs/1110.1439, 2011.

[Tan10] Tony Tan. On pebble automata for data languages with decidable emptiness problem. *J. Comput. Syst. Sci.*, 76:778–791, December 2010.

[Tan11] Tony Tan. Graph reachability and pebble automata over infinite alphabets. volume abs/1110.2776, 2011.

[TT02] Pascal Tesson and Denis Therien. Diamonds are forever: The variety da. In *Semigroups, Algorithms, Automata and Languages*, pages 475–500. World Scientific, 2002.

[TW98] Denis Thérien and Thomas Wilke. Over words, two variables are as powerful as one quantifier alternation. In *STOC*, pages 234–240, 1998.

[TW04] Denis Thérien and Thomas Wilke. Nesting until and since in linear temporal logic. *Theory Comput. Syst.*, 37(1):111–131, 2004.

[Tze11] Nikos Tzevelekos. Fresh-register automata. In *POPL*, pages 295–306, 2011.

[VL08] Antti Valmari and Petri Lehtinen. Efficient minimization of dfas with partial transition. In *STACS'08*, pages 645–656, 2008.

[Wil99] Thomas Wilke. Classifying discrete temporal properties. In *STACS*, pages 32–46, 1999.

[Wor99] World Wide Web Consortium. Xml path language (xpath) recommendation, November 1999. http://www.w3c.org/tr/xpath.

[Wor00] World Wide Web Consortium. Extensible markup language (xml) 1.0, October 2000. http://www.w3.org/tr/rec-xml.