Automata vs. Logics on Data Words

Michael Benedikt, Clemens Ley, and Gabriele Puppis

Oxford University Computing Laboratory - Parks Road, Oxford OX13QD UK

Abstract. The relationship between automata and logics has been investigated since the 1960s. In particular, it was shown how to determine, given an automaton, whether or not it is definable in first-order logic with label tests and the order relation, and for first-order logic with the successor relation. In recent years, there has been much interest in languages over an infinite alphabet. Kaminski and Francez introduced a class of automata called finite memory automata (FMA), that represent a natural analog of finite state machines. A FMA can use, in addition to its control state, a (bounded) number of registers to store and compare values from the input word. The class of data languages recognized by FMA is incomparable with the class of data languages defined by first-order formulas with the order relation and an additional binary relation for data equality.

We first compare the expressive power of several variants of FMA with several data word logics. Then we consider the corresponding decision problem: given an automaton A and a logic, can the language recognized by A be defined in the logic? We show that it is undecidable for several variants of FMA, and then investigate the issue in detail for deterministic FMA. We show the problem is decidable for first-order logic with local data comparisons – an analog of first-order logic with successor. We also show instances of the problem for richer classes of first-order logic that are decidable.

Logics are natural ways of specifying decision problems on discrete structures, while automata represent natural processing models. On finite words from a fixed (finite) alphabet, Büchi [1] showed that monadic second-order logic has the same expressiveness as deterministic finite state automata, while results of Schützenberger and McNaughton and Papert showed that first-order logic with the label and order predicates has the same expressiveness as counter-free automata [2, 3]. The latter theorem gives a decidable characterization of which automata correspond to first-order sentences. Decidable characterizations have also been given for first-order logic with the label and successor predicates [4]. These characterizations have been extended to many other contexts; for example there are characterizations of the tree automata that correspond to sentences in logics on trees [5].

Automata processing finite words over infinite alphabets (so called *data words*) are attracting significant interest from the database and verification communities, since they can be often used as low-level formalisms for representing and reasoning about data streams, program traces, and serializations of structured documents. Moreover, properties specified using high-level formalisms (for

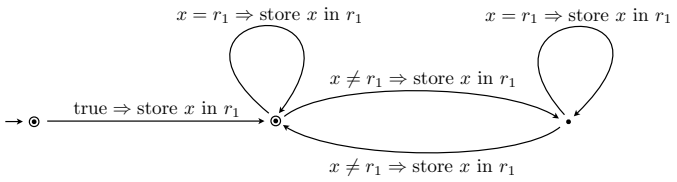

Fig. 1. A finite-memory automaton.

instance, within suitable fragments of first-order logic) can be often translated into equivalent automaton-based specifications, easing, in this way, the various reasoning tasks.

Different models of automata which process words over infinite alphabets have been proposed and studied in the literature (see, for instance, the surveys [6, 7]). *Pebble automata* [8] use special markers to annotate *locations* in a data word. The *data automata* of [9] 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. Of primary interest to us here will be a third category, the *finite memory automata* [10], also called *register automata*, which make use of a finite number of registers in order to store and eventually compare values in the precessed data word.

It is known that the languages accepted by finite memory automata are strictly contained in the languages definable in monadic second-order logic with the successor relation and a binary relation to test data equality [8]. The first order variant of this logic is incomparable in expressive power with deterministic finite memory automata: The set of words of even length can be recognized by a finite memory automaton but can not be defined in first-order logic; on the other hand the set of words that have two positions with the same value can be expressed in first-order logic, but it can not be recognized by any deterministic finite memory automaton.

We will compare the expressive power of several restrictions of deterministic finite memory automata with restrictions of MSO. We consider the class of finite memory automata with a bounded number of registers as well as the class that can only perform "local" data comparisons (within a fixed distance). We will also look at several variants of first-order logic – we will look at logic where we can use equality between any two symbols in the word, as well as logics where one can only compare symbols "locally". We will look at logics where the word ordering relation is present, as well as logics where only the successor relation is available. We will also consider "non-uniform first-order definability" where a different formula is used depending on the alphabet.

Our main goal is to find effective characterisations for these logics with respect to the automata models described above. That is, we present algorithms that can decide questions of the form: Given an automaton and a logic, can the language of the automaton be defined in the logic?

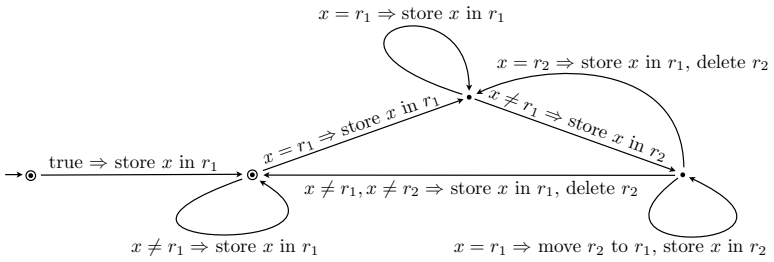

Fig. 2. A finite-memory automaton recognizing a first-order definable language.

Example 1. Consider the automaton from Figure 1. We have had used an intuitive notation in the figure – a more precise syntax is given in Section 1. An edge is labeled with $g \Rightarrow a$ where, g is a guard (precondition) and a an action (postcondition); both g and a refer to the current symbol as x, and the i^{th} register as r_i. This automaton accepts exactly the data words w such that there are an even number of places $n \leq |w|$ with $w(n) \neq w(n-1)$. Our algorithms can check that this language is not definable in first-order logic with order and data equality, even in the non-uniform model. The automaton from Figure 2 accepts words such that for every n with $w(n) = w(n-1)$ there is $y > x$ such that $w(y) \neq w(y+1) \neq w(y+2)$ and $w(y) \neq w(y+2)$. Our techniques can determine that this language is first-order definable: in fact, definable using only local data comparisons.

We first show that one can not hope to characterize logical classes for many powerful classes of automata on infinite alphabets – for example, we show this for non-deterministic memory automata, and for two-way deterministic memory automata.

We thus focus on Deterministic Memory Automata (DMAs). We give a method for deciding non-uniform FO definability for two special classes of DMAs – 1-memory DMA and window automata (automata that can only make data comparisons locally). We then provide a decidable criterion for a DMA being expressible within the local variants of first-order logic. We then turn to non-local FO definability. The general question of decidability of non-local definability is open; however, we provide effective necessary and sufficient conditions for a subclass of DMA, the window automata, to be non-locally FO definable.

Organization: Section 1 explains the automata and logic formalisms that are the core topic of this paper, and their relationships. Section 2 gives undecidability results for several powerful models. Section 3 gives decidable criteria for non-uniform first order definability within certain classes of memory automata. Section 4 gives a decision procedure for first-order definability with only local data comparisons. Section 5 investigates the broader question of deciding first-order definability with unrestricted data comparisons. We do not resolve this question, but provide effective necessary conditions and effective sufficient criteria. Section 6 gives conclusions. All proofs can be found in the appendix.

1 Preliminaries

We fix an infinite alphabet D of (*data*) *values*. A (*data*) *word* is a finite sequence of values from the infinite alphabet D. Two words u and v are said to be isomorphic, and we denote it by $u \simeq v$, if $|u| = |v|$ and $u(i) = u(j)$ iff $v(i) = v(j)$ for all pairs of positions i, j in u. The $\simeq$-equivalence class of a word u, denoted by $[u]_\simeq$ or simply by $[u]$, is called the $\simeq$-*type of* u. A (*data*) *language* is a set of data words. Given two words u and v, we write $u =_L v$ if either both u and v are in L, or both are not. From now on, we tacitly assume that any data language L is closed under $\simeq$-preserving morphisms, namely, $\simeq$ refines $=_L$.

1.1 Finite-memory automata

In this section, we introduce a variant of Kaminski's finite-memory automata [10] that recognize data languages over an infinite alphabet D. These automata process data words by storing a bounded number values into their memory and by comparing them with the incoming input values.

Definition 1. *A* k-memory automaton *(*k*-MA) is a tuple* $A = (Q_0, \ldots, Q_k, q_I, F, T)$*, where* $Q_0, \ldots, Q_k$ *are pairwise disjoint finite sets of states,* $q_I \in Q_0$ *is an initial state, and* $F \subseteq Q_0 \cup \ldots \cup Q_k$ *is a set of final states.* T *is a finite set of transitions of the form* (p, α, E, q)*, where* $p \in Q_i$ *for some* $i \leq k$*,* α *is the* $\simeq$*-type of a word of length* $i+1$*,* $E \subseteq \{1, \ldots, i+1\}$*, and* $q \in Q_j$ *with* $j = i + 1 - |E|$*.*

A *configuration* of a k-MA A is a pair $(p, \bar{a})$ consisting of a state $p \in Q_i$, with $0 \leq i \leq k$, and a memory content $\bar{a} \in D^i$. The initial configuration is (q_I, ε), where ε denotes the empty memory content. The automaton can move from a configuration $(p, \bar{a})$ to a configuration $(q, \bar{b})$ by consuming an input symbol a iff there is a transition $(p, \alpha, E, q) \in T$ such that the word $\bar{a} \cdot a$ has $\simeq$-type α and $\bar{b}$ is obtained from $\bar{a} \cdot a$ by removing all positions in E.

We enforce the following sanity conditions to every transition (p, α, E, q) of a k-MA. First, we assume that E is non-empty whenever $q \in Q_k$ (this is in order to guarantee that the length of the target memory content $\bar{b}$ never exceeds k). Then, we assume that if the $\simeq$-type α is of the form $[\bar{a} \cdot a]$, with $\bar{a}(j) = a$ for some $1 \leq j \leq |\bar{a}|$, then E contains at least the index j (this is in order to guarantee that the target memory content $\bar{b}$ contains *pairwise distinct* elements).

A *run* of A is defined in the usual way. If u is a data word and A has a run on u from a configuration $(p, \bar{a})$ to a configuration $(q, \bar{b})$, then we denote this by writing either $u^A(p, \bar{a}) = (q, \bar{b})$ or $(p, \bar{a}) \xrightarrow[A]{u} (q, \bar{b})$, depending on which is more convenient. The language recognized by A, denoted $L(A)$, is the set of all words u such that $u^A(q_I, \varepsilon) = (p, \bar{a})$, for some $p \in F$ and some $\bar{a} \in D^*$.

A finite-memory automaton $A = (Q_0, \ldots, Q_k, T, I, F)$ is *deterministic* if for each pair of transitions $(p, \alpha, E, q), (p', \alpha', E', q') \in T$, if $p = p'$ and $\alpha = \alpha'$, then $E = E'$ and $q = q'$. Similarly, A is *complete* if for every state $q \in Q_i$ and every $\simeq$-type α with $i+1$ variables, T contains a transition rule of the form (p, α, E, q). We abbreviate any *deterministic and complete* k-memory automaton by (k-)*DMA*.

Minimal deterministic finite-memory automata. Bouyer et. al. have given an algebraic characterization of the languages that are accepted by a generalization of DMA [11]. A Myhill-Neorde style characterization of the languages that are accepted by DMA was given by Francez and Kaminski in [12]. They state as an open question whether one can compute, given a DMA, a minimal DMA that accepts the same language. In [13] we gave a positive answer to this question. Precisely, we show that given a DMA that accepts a language L, one can compute a DMA A_L that has the minimum number of states and that stores the minimum number of data values for every consumed input word. A semantics-based definition of the set of values that need to be stored by any DMA A in order to recognize a given language L is as follows:

Definition 2. *Let L be a language. A value a is* L-memorable *in a word u if a occurs in u and there is a word v and a value $b \notin u \cdot v$ such that $u \cdot v \neq_L u \cdot v[a/b]$.*

We denote by $\mathsf{mem}_L(u)$ the sequence consisting of the L-memorable values of u in the order of their last appearance in u.

In [13], we showed that a language is DMA-recognizable iff the following equivalence relation has finite index:

Definition 3. *Given a language L, we define the equivalence $\equiv_L \subseteq D^* \times D^*$ such that $u \equiv_L v$ iff $\mathsf{mem}_L(u) \simeq \mathsf{mem}_L(v)$ and for all words $u', v' \in D^*$, if $\mathsf{mem}_L(u) \cdot u' \simeq \mathsf{mem}_L(v) \cdot v'$ then $u \cdot u' =_L v \cdot v'$.*

Let L be a language with equivalence $\equiv_L$ of finite index. We can define the *canonical automaton* for L as the k-memory automaton $A_L = (Q_0, \ldots, Q_k, q_I, F, T)$, where k is the maximum length of $\mathsf{mem}_L(u)$ over all words $u \in D^*$; Q_i, with $0 \leq i \leq k$, contains all $\equiv_L$-equivalence classes $[u]_{\equiv_L}$, with $u \in D^*$ and $|\mathsf{mem}_L(u)| = i$; q_I is the $\equiv_L$-equivalence class of the empty word; $F = \{[u]_{\equiv_L} \mid u \in L\}$; T contains all transitions of the form $([u]_{\equiv_L}, \alpha, E, [u \cdot a]_{\equiv_L})$, where $u \in D^*$, $a \in D$, α is the $\simeq$-type of $\mathsf{mem}_L(u) \cdot a$, and E is the set of positions in $\mathsf{mem}_L(u) \cdot a$ that have to be removed from $\mathsf{mem}_L(u) \cdot a$ to obtain $\mathsf{mem}_L(u \cdot a)$.

Theorem 1 ([13]). *Given a DMA A that recognizes L, the canonical automaton for L can be effectively computed from A.*

We will extensively exploit the following property of a canonical automaton A_L: after parsing an input word u, the automaton A_L stores exactly the sequence $\mathsf{mem}_L(u)$ of L-memorable values of u. It is also worth remarking that if two words lead to distinct states in the canonical automaton A_L, then they belong to distinct $\equiv_L$-equivalence classes.

Local finite-memory automata. A DMA A is *l-local* if for all configurations $(p, \bar{a})$ and all words u of length l, if $u^A(p, \bar{a}) = (q, \bar{b})$, then $\bar{b}$ contains only values that occur in u. We call *local* any DMA that is l-local for some $l \in \mathbb{N}$. Then we can show:

Proposition 1. *The following problem is decidable: Given a DMA A, is A local? If A is local then we can compute the minimum number l for which A is l-local.*

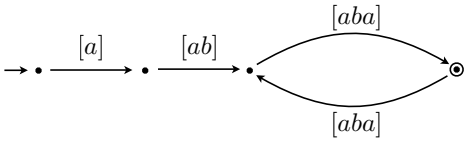

Fig. 3. A DWA that recognizes the language $L = \{aba \dots ba \in D^* \mid a \neq b\}$.

1.2 Window automata

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

Definition 4. *An* l-window automaton *(*l*-WA) is a tuple* $A = (Q, q_I, F, T)$*, 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* $\simeq$*-type of a word of length at most* l*.*

An l-WA $A = (Q, q_I, F, T)$ processes an input word $u = a_1 \dots a_n$ from left to right, starting form its initial state q_I, as follows. At each step of the computation, if 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 A consumes the next symbol a_{i+1} of u 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 shows an example of a 3-DWA.

A *path* is a sequence of consecutive transitions in an automaton. A path ρ in a DWA is *realizable* if there is a word u that induces a run along ρ. 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 $i-1, i, i+1$ have pairwise different values. Then the next transition can not be used, as it requires that positions $i, i+1, i+2$ have the same value. A DWA A is *realizable* if all paths in 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 $a_{n-m+2} \dots a_n \simeq b_1 \dots b_{m-1}$
2. if $n < m-1$, then $a_1 \dots a_n \simeq b_{m-n} \dots b_{m-1}$.

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 is realizable.

Proposition 2. *For every* l*-local DMA, there is an equivalent* l*-DWA and, vice versa, for every* l*-DWA, there is an equivalent* l*-local (*l*-)DMA.*

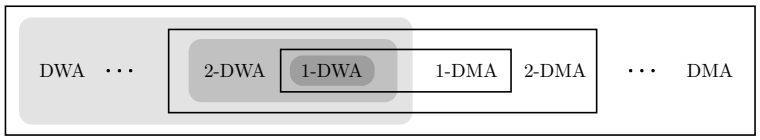

Fig. 4. The inclusions between DMA and DWA.

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 4 shows the inclusions that hold between DMA and DWA.

1.3 Logics for Data Words

$\mathrm{MSO}(\sim, <)$ denotes monadic second-order logic with predicates $\sim$ and $<$, interpreted respectively by the data-equality relation and by the total order relation over the positions of a given data word. $\mathrm{FO}(\sim, <)$ is the restriction of $\mathrm{MSO}(\sim, <)$ that only uses quantification over first-order variables. An example of an $\mathrm{FO}(\sim, <)$ formula is $\forall x, y \ \ (x \sim y \to x = y)$, which requires all values in a data word to be distinct (observe that $=$ can be defined in terms of $<$). Note that the language defined by the above formula is not DMA-recognizable.

We also consider fragments of $\mathrm{FO}(\sim, <)$ where the predicates are replaced by local variants. For instance, $+1$ denotes the successor relation, which holds between two positions x and y (denoted $y = x+1$) iff y is the immediate successor of x. We denote by $\mathrm{FO}(\sim, +1)$ the first-order logic where the total order $<$ has been replaced by the successor relation $+1$.

There is also a local variant of the data-equality relation: given $l \in \mathbb{N}$, $x \sim_l y$ can be viewed as a shorthand for the formula $y = x + l \wedge x \sim y$. We accordingly denote by $\mathrm{FO}(\sim_{\leq l}, <)$ the logic with the predicates $<$ and $\sim_i$, for all $i \leq l$. For example, the formula $\forall x, y \, \exists z \ \ (x \sim_5 y \to y \nsim z)$ requires that if position y has the same value as its fifth neighbor x to the left, then there is a position z that has a different value than x and y.

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

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 $\mathrm{NUFO}(\sim, <)$, if for each $n \in \mathbb{N}$, the language $L_N = L \cap D_n^*$ can be defined in $\mathrm{FO}(\sim, <)$ (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.

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

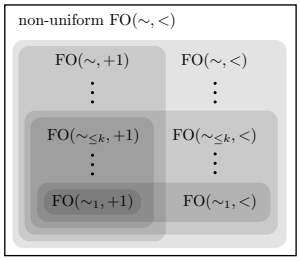

Fig. 5. An overview over some logics for data words.

The above example shows that the non-uniform definability is much weaker than uniform definability (indeed, it can easily be shown that there are continuumly many non-uniformly definable languages for any of our signatures). Nonetheless, the following proposition shows that definability in the "local logic" $\mathrm{FO}(\sim_{\leq k}, <)$ is equivalent to definability in $\mathrm{NUFO}(\sim_{\leq k}, <)$, provided that we restrict to DMA-recognizable languages.

Proposition 3. *Let L be a language recognized by a DMA and let $l \in \mathbb{N}$. L can be defined in $\mathrm{NUFO}(\sim_{\leq l}, <)$ iff it can be defined in $\mathrm{FO}(\sim_{\leq l}, <)$. An analogous result holds when $<$ is replaced by $+1$.*

We do not know whether Proposition 3 holds also for the unrestricted logics $\mathrm{FO}(\sim, <)$ and $\mathrm{FO}(\sim, +1)$.

Figure 5 gives an overview over the logics considered so far.

1.4 DMA vs Logics

Recall that the class of languages recognized by (deterministic) finite-state automata strictly includes the class of languages over finite alphabets defined with first-order logic. This result does not extend to languages over infinite alphabets: the language of all words with pairwise distinct symbols can be defined in first-order logic with the (unrestricted) data-equality relation, but it can not be recognized by any DMA. As with languages over finite alphabets, the set of all (data) words of even length is clearly recognized by a 0-DMA, but it can not be defined in first-order logic. Hence, first-order logics and DMA are, in general, expressively incomparable.

For the restricted logics the situation is different. It follows from the next proposition that any language that is definable in $\mathrm{MSO}(\sim_{\leq l}, +1)$ is recognized by an l-DMA. This also shows that local DMA are closed under the usual boolean operations, projection, concatenation, and Kleene star.

Proposition 4. *A language L can be defined in $\mathrm{MSO}(\sim_{\leq l}, +1)$ iff it can be recognized by an l-local DMA.*

Figure 6 gives an overview over the relationships between logics and DMA.

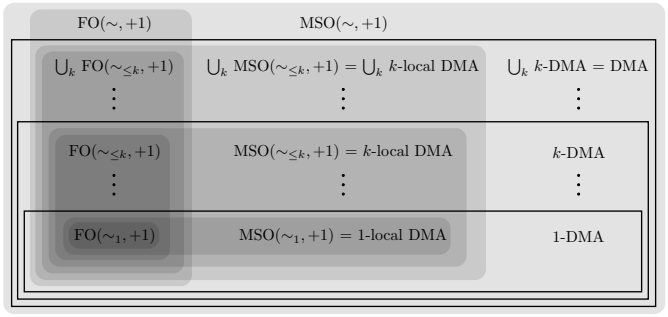

Fig. 6. Comparing the expressive power of automata with that of logics.

Given that logics and automata are incomparable, we will focus on the problem of deciding *when an automaton-recognizable language is definable within a given logic*. The dual problem of determining when a logical sentence corresponds to a logic is not explored here – but it is easily shown to be undecidable for our most powerful logics, since the satisfiability problem for these logics is undecidable [8].

2 Undecidability Results

In this section we show that there is no hope for achieving effective characterizations of fragments of $\mathrm{FO}(\sim, <)$ within several classes of languages recognized by automaton-based models stronger than DMA. We first consider the class of languages recognized by non-deterministic finite-memory automata (NMA):

Theorem 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 an 3-NMA A, can $L(A)$ be defined in $\mathcal{L}$?*

The proof (see the appendix) is by reduction from the 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 [8].

Below, we show that similar negative results hold for two-way deterministic finite-memory automata DMA (2-way DMA) and for the weakest variant of pebble automata, namely, weak one-way pebble automata (weak 1-way DPA). We briefly sketch the distinctive features of these two models of automaton (see [8] for formal definitions). A 2-way DMA can revisit the same positions in a given input word several times, by moving its head in either direction. A pebble automaton, on the other hand, has the ability to mark a finite number of word positions by placing pebbles on them. The guards of the transitions of a 1-way DPA allow it to compare the current input value with the values of the positions marked by pebbles, but only the most recently placed pebble can be moved and

only to the right. Moreover, in the *weak* variant of DPA, new pebbles are placed at the first position of the word. The proof of the following result is similar to that of Theorem 2 (and it can be found in the appendix).

Theorem 3. *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 3-DMA A or a weak 1-way DPA A with 3 pebbles, can $L(A)$ be defined in $\mathcal{L}$?*

3 Characterizations of Non-Uniform FO

In this section we will look for 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 4. *The following problem is decidable: Given a local DMA A, is $L(A)$ definable in NUFO$(\sim, <)$?*

The idea of the proof is to show that $L = L(A)$ is definable in NUFO$(\sim, <)$ iff $L_N = L \cap D_N$ is definable in FO$(D_N, <)$, where N is a suitable number that depends only on A. 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$(D_N, <)$ is known from [14]. One direction of this claim is straightforward: if L is definable in NUFO$(\sim, <)$, then L_N is clearly definable in FO$(D_N, <)$. 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 $L_n = L \cap D_n$ can not be defined in FO$(D_n, <)$. It follows from [14] that the minimal DFA A_n recognizing L_n has a counter. We then prove that there is a (potentially) much smaller alphabet D_N for which the minimal DFA A_N recognizing $L_N = L \cap D_N$ has a counter. Thus L_N can not be defined in FO$(D_N, <)$. The full proof is in the appendix.

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 5. *The following problem is decidable: Given a 1-DMA A, is $L(A)$ definable in NUFO$(\sim, <)$?*

The proof (see the appendix) exploits, first, the fact that it is decidable whether a given DMA A is local. If A is local, then Theorem 4 can be applied and we are done. If A is not local, then A must contain certain 'non-local cycles'. By distinguishing several cases, depending on the way these cycles occur in A, it can be decided whether A is definable in NUFO$(\sim, <)$.

4 Characterizations of Local FO

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

Theorem 6. *The following problem is decidable: Given a DMA A, is there an l such that $L(A)$ is definable in $FO(\sim_{\leq l}, <)$? If such an l exists, then we can compute the minimal l_0 such that $L(A)$ is definable in $FO(\sim_{\leq l_0}, <)$. Analogous results hold when $<$ is replaced by $+1$.*

The proof exploits the fact that it is decidable whether a given (canonical) DMA A is local and, in such a case, one can compute the smallest l_0 such that A is l_0-local. We first show that if A is not local, then $L(A)$ is not definable in $\mathrm{FO}(\sim_{\leq l}, <)$ (nor in $\mathrm{FO}(\sim_{\leq l}, +1)$). Otherwise, if A is l-local, then we can reduce the $\mathrm{FO}(\sim_{\leq l}, <)$ definability problem for L to a classical first-order definability problem for a regular language $\mathsf{abs}(L)$ over a finite alphabet, whose symbols are $\simeq$-types of words of length at most l. The argument for $\mathrm{FO}(\sim_{\leq l}, +1)$ is similar. The full proof is in the appendix.

As an example, consider the 3-local DMA A equivalent to the 3-DWA depicted in Figure 3: if thought of as a DFA, such an automaton contains a counter over the $\simeq$-type $[aba]$, where $a \neq b$. It can then be proved that the data language $L(A)$ can not be defined in $\mathrm{FO}(\sim_l, <)$, for any $l \in \mathbb{N}$.

The next corollary follows from Theorem 6 and Proposition 3.

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

5 Necessary and sufficient conditions for FO(~, <)

The ultimate goal would be to decide, given a DMA, whether or not its language can be defined in the unrestricted first-order logic $\mathrm{FO}(\sim, <)$. We present a partial decision procedure that classifies DWA (or, equivalently, local DMA) according to the $\mathrm{FO}(\sim, <)$ definability of their recognized languages. For certain DWA, the algorithm answers correctly whether the recognized languages are definable in $\mathrm{FO}(\sim, <)$ or not; for other DWA, the algorithm can only output "don't know".

Given a k-DWA A, we denote by $L^{\geq k}(A)$ the set of words in $L(A)$ that have length at least k. In the rest of this Section, we will only prove results about languages of the form $L^{\geq k}(A)$. This simplifies the presentation (for instance, we

Fig. 7. Two DWA. While $L(A_1)$ can be defined in FO$(\sim, <)$, $L(A_2)$ can not.

can assume that all $\simeq$-types in the transitions of a k-DWA have length exactly k) and or results easily generalize to arbitrary languages. As an example, the left DWA A_1 in Figure 7 recognizes language $\{u = aba \dots ba \mid |u| \geq 3, a \neq b\}$, which is $L^{\geq 3}(A)$ where A is the DWA shown in Figure 3. Similarly, the right DWA A_2 recognizes the language of all constant words of odd length at least 3. Note that neither $L^{\geq 3}(A_1)$ nor $L^{\geq 3}(A_2)$ are definable in FO$(\sim_{\leq l}, <)$ for any l. On the other hand, $L^{\geq 3}(A_1)$ can be defined in FO$(\sim, <)$, while $L^{\geq 3}(A_2)$ can not. For the sake of simplicity, we will often write $L(A)$ instead of $L^{\geq k}(A)$

To be able to effectively separate DWA recognizing languages in FO$(\sim, <)$ from DWA recognizing languages not in FO$(\sim, <)$, we extend DWA with some additional information. Precisely, we label the transitions with "parametrized types", which specify data-equalities between the local neighborhood of the current position and the local neighborhood of some other fixed position in the string (i.e., the parameter).

For $u \in D^k$ and $v \in D^l$, the *k-parametrized l-type* of (u, v) is the $\simeq$-type of $u \cdot v$, that is the set of words that are isomorphic to $u \cdot v$. We shortly denote this set by $[u; v]$. The set of all k-parametrized l-types is by $T_{k,l}$. To avoid confusion, we will refer to standard $\simeq$-types $[v]$ as *unparametrized* types through the rest of this section.

Definition 5. *A* parametrized k-window automaton *(k-PWA) is a tuple $A = (Q, q_I, F, T)$, 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 $T \subseteq Q \times T_{k-1,k} \times Q$ is a finite set of transitions.*

The input to a k-PWA A is a pair of words $(u, w) \in D^{k-1} \times D^*$, called a *parametrized word*. A configuration of A is a pair (p, i), where p is a state of A and i $(\geq k)$ is a position in w. The automaton A processes a parametrized word (u, w) in a single run, from left to right, starting from the initial configuration (q_I, k). At each step of the computation, A can move from a configuration (p, i) to a configuration $(q, i + 1)$ iff there is a transition of the form (p, α, q), with $u \cdot w[i - k + 2, i + 1] \in \alpha$. The notions of successful run and recognized (parametrized) language $L(A)$ are as usual. A k-PWA $A = (Q, q_I, f, T)$ is *deterministic* (k-DPWA) if for every pair of transitions (p, α, q) and (p, α', q') in T, $\alpha = \alpha'$ implies $q = q'$. Note that a parametrized k-window automaton can be thought of as an window automaton that has k predicates for the first k symbols in the input string which it can evaluate when transitioning to a new state.

A path ρ in a PWA A is *realizable* if there is a parametrized word (u, w) that induces a run of A along it. A PWA A is *realizable* if all paths in it are realizable.

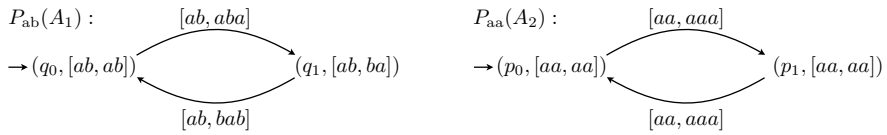

Fig. 8. The parametrized versions of the DWA of Figure 7

It is easy to see that for any given k-PWA A, there is an equivalent realizable k-PWA A', which can be computed from A. Moreover, it can be decided whether a given PWA A is realizable or not (this test is similar to that for WA described in Section 1).

Definition 6. *Given a k-DWA $A = (Q, q_I, F, T)$ and a word u of length $k-1$, the* u-parametrized version *of A is the k-DPWA $P_u(A) = (\tilde{Q}, \tilde{q}_I, \tilde{F}, \tilde{T})$, where $\tilde{Q} = Q \times T_{k-1,k-1}$, $\tilde{q}_I = (q_I, [u; u])$, $\tilde{F} = \{(p, [v; w]) \in \tilde{Q} \mid p \in F\}$, and $\tilde{T}$ contains $(p, [u; a_1 \dots a_{k-1}]) \xrightarrow{[u; a_1 \dots a_k]} (q, [u; a_2 \dots a_k])$ iff $(p, [a_1 \dots a_k], q) \in T$.*

Figure 8 shows the ab-parametrized version of A_1 and the aa-parametrized version of A_2. Note that both are realizable.

We call *counter* of a DPWA B any cycle of transitions of the form

$$p_1 \xrightarrow{\bar{\alpha}} \dots \xrightarrow{\bar{\alpha}} p_m \xrightarrow{\bar{\alpha}} p_1$$

where $m > 1$, $p_1, \dots, p_m$ are pairwise distinct states of B and $\bar{\alpha}$ is a non-empty sequence of $(k-1)$-parametrized k-types.

The following result gives a sufficient condition for a language recognized by a DWA to be definable in FO$(\sim, <)$.

Proposition 5. *Let A be a DWA. If $P_u(A)$ is counter-free for all $u \in D^{k-1}$, then $L(A)$ is definable in FO$(\sim, <)$.*

The converse of the above proposition does not hold. Consider, for instance, the DWA A_3 in Figure 9: although $P_{\mathrm{ab}}(A_3)$ has a counter (because $[ab, cdc] = [ab, dcd]$), $L(A_3)$ is still definable in FO$(\sim, <)$. We will thus distinguish between two kinds of counters, which we call "good" and "bad". We will show that if a DPWA contains a bad counter, then it recognizes a language that is not definable in FO$(\sim, <)$. In order to define bad counters, we need to consider a slightly modified (and more general) version of the automaton given in Definition 6.

Definition 7. *Let $A = (Q, q_I, F, T)$ be k-DWA and let $u, v \in D^{k-1}$. The (u, v)-*parametrized version *of A is the k-DPWA $P_{u,v}(A) = (\tilde{Q}, \tilde{q}_I, \tilde{F}, \tilde{T})$, where $\tilde{Q} = Q \times T_{k-1,k-1}$, $\tilde{q}_I = (q_I, [u; v])$, $\tilde{F} = \{(p, [v; w]) \in \tilde{Q} \mid p \in F\}$, and $\tilde{T}$ contains the transition $(p, [w; a_1 \dots a_{k-1}]) \xrightarrow{[w; a_1 \dots a_k]} (q, [w; a_2 \dots a_k])$ iff $w \in D^{k-1}$ and $(p, [a_1 \dots a_k], q) \in T$. We denote by $\mathcal{P}(A)$ the set $\{P_{u,v}(A) \mid u, v \in D^{k-1}\}$.*

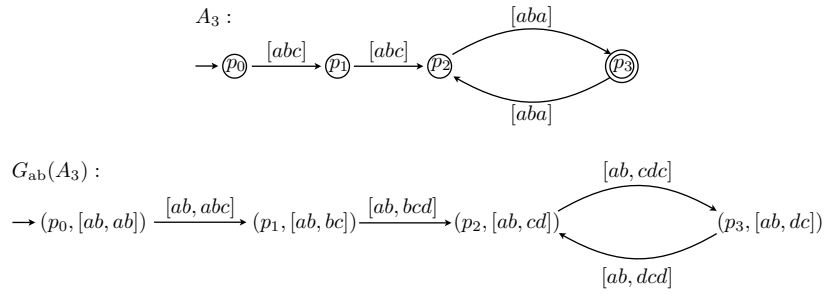

Fig. 9. A FO($\sim, <$) definable DWA and its parameterized version.

Let A be a k-DWA and B be the (u, v)-parametrized version of A. A *bad counter* of B is a sequence of transitions

$$p_1 \xrightarrow{\bar{\alpha}_1} \ \dots \ \xrightarrow{\bar{\alpha}_{n-1}} p_n \xrightarrow{\bar{\alpha}_n} p_{n+1} \xrightarrow{\bar{\alpha}} \ \dots \ \xrightarrow{\bar{\alpha}} p_{n+m} \xrightarrow{\bar{\alpha}} p_{n+1}$$

such that

1. $n \geq 0$ and $m \geq 2$,
2. $p_1, \dots, p_{n+m}$ are pairwise distinct states, and p_1 is of the form $(p, [u, u])$,
3. $\bar{\alpha}_1, \dots, \bar{\alpha}_n, \bar{\alpha} \in T^l_{k-1,k}$ for some $l > 0$, and $\mathsf{loc}(\bar{\alpha}_1) = \dots = \mathsf{loc}(\bar{\alpha}_n) = \mathsf{loc}(\bar{\alpha})$.

Here $\mathsf{loc} : T_{k-1,k} \to T_k$ is defined by $\mathsf{loc}([u, v]) = [v]$ and loc is extended to strings over $T_{k-1,k}$ in the usual way.

Similarly to a DWA, a DPWA A can be thought of as a deterministic finite-state automaton over the alphabet $T_{k-1,k}$ of $k-1$-parametrized k-types. We say that A is *canonical* if A is minimal as a DFA. Clearly, a canonical DPWA contains only reachable states and for all pairs of states $p \neq q$, there is a $\bar{\alpha}$-labelled path that starts at p leads to an accepting state, while the $\bar{\alpha}$-labelled path that starts at q leads to a rejecting state. We can finally show that the absence of bad counters is a necessary condition for FO definability:

Proposition 6. *Let A be a canonical DWA. If there a DPWA $B \in \mathcal{P}(A)$ that contains a bad counter, then $L(A)$ is not definable in FO($\sim, <$).*

6 Conclusion

In this work we have studied a number of variants of first-order logic, and also introduced several natural subclasses of memory automata. We overviewed the relationships of the logics to one another, the relationships of the automata to one another, and the relationships of the logic and the automata. We then investigated the decidability of definability in logical classes within memory automata. We have shown that the problem is undecidable for natural extensions of deterministic memory automata, and decidable with certain restrictions on the

logics or the automata. Finally, we provide necessary and sufficient conditions for determining when a memory automaton is definable within a logic.

The main question left open is an effective characterization of which deterministic memory automata are definable in first-order logic with unrestricted data comparison. The conditions we give in Section 5 for window automata are a step towards this. Another significant open question is the relationship between non-uniform and uniform (non-local) definability. Non-uniform first-order definability is weaker then first-order definability for the vocabularies with unrestricted data equality, but we do not know if this separation is witnessed by languages accepted by DMAs.

References

[1] Büchi, J. R. (1960) Weak second-order logic and finite automata. *S. Math. Logik Grunlagen Math.*, **6**, 66–92.

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

[3] McNaughton, R. and Papert, S. A. (1971) *Counter-Free Automata*. MIT.

[4] Beauquier, D. and Pin, J.-E. (1989) Factors of words. *ICALP*.

[5] Benedikt, M. and Segoufin, L. (2009) Regular Tree Languages Definable in FO and FOmod. *TOCL*, **11**.

[6] Segoufin, L. (2006) Automata and logics for words and trees over an infinite alphabet. *CSL*.

[7] Schwentick, T. (2007) Automata for XML - a survey. *JCSS*, **73**, 289–315.

[8] Neven, F., Schwentick, T., and Vianu, V. (2004) Finite state machines for strings over infinite alphabets. *TOCL*, **5**, 403–435.

[9] Bojanczyk, M., Muscholl, A., Schwentick, T., Segoufin, L., and David, C. (2006) Two-variable logic on words with data. *LICS*.

[10] Kaminski, M. and Francez, N. (1994) Finite-memory automata. *TCS*, **134**.

[11] Bouyer, P., Petit, A., and Thérien, D. (2003) An algebraic approach to data languages and timed languages. *Inf. Comput.*, **182**, 137–162.

[12] Francez, N. and Kaminski, M. (2003) An algebraic characterization of deterministic regular languages over infinite alphabets. *TCS*, **306**, 155–175.

[13] Benedikt, M., Ley, C., and Puppis, G. (2010), Minimal memory automata. Available at `http://www.comlab.ox.ac.uk/michael.benedikt/papers/myhilldata.pdf`.

[14] Wilke, T. (1999) Classifying discrete temporal properties. *STACS*.

[15] Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2006) *Introduction to Automata Theory, Languages, and Computation)*. Addison-Wesley Longman.

[16] Straubing, H. (1994) *Finite automata, formal logic, and circuit complexity*. Birkhauser.

A Appendix

A.1 Proofs from Section 1

Proposition 1. *The following problem is decidable: Given a DMA A, is A local? If A is local then we can compute the minimum number l for which A is l-local.*

Proof. We need some preliminary definitions. A *loop* C in a DMA A is a cycle of consecutive transitions in 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:

$$p_1 \xrightarrow[\phi_1, E_1]{} \cdots \xrightarrow[\phi_{m-1}, E_{m-1}]{} p_m \xrightarrow[\phi_m, E_m]{} p_1$$

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 $\simeq$-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 u that cycles through C and every value a that is stored at some point along the corresponding run, a occurs in every subword of u of length m. It is easy to see that A is local iff every simple loop in A is local. Moreover, the latter property can be effectively checked by examining all (finitely many) simple loops in the DMA A.

We now show how to compute, from a given local k-memory DMA A, the minimum number l such that A is l-local. From the definition of local simple loop, it immediately follows that A is local only if it is $k|T|$-local, where T is the set of transitions of A. This implies that every value that gets stored at some point along a computation of 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 $l = n + 1$. □

Proposition 2. *For every l-local DMA, there is an equivalent l-DWA and, vice versa, for every l-DWA, there is an equivalent l-local (l-)DMA.*

Proof. Let A be an l-local DMA. We sketch the construction of an l-DWA B equivalent to A. From the locality property of A, we know that every value that is stored by A must occur within the last l symbols of the consumed input word (hence A can store at most l values). We define the state space of B as $Q \times \{\bot, 1, \ldots, l\}^l$. B will maintain the following invariant while processing an input word u: if B is in state $(q, i_1, \ldots, i_l)$, then the last occurrence in u of the value stored by 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 B.

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

Proposition 3. *Let L be a language recognized by a DMA and let $l \in \mathbb{N}$. L can be defined in NUFO$(\sim_{\leq l}, <)$ iff it can be defined in FO$(\sim_{\leq l}, <)$. An analogous result holds when $<$ is replaced by $+1$.*

Proof. Let us fix a k-DMA that recognizes L. We prove the proposition in the case of signatures with the order $<$. The interesting direction is from left to right. We assume that L can not be defined in FO$(\sim_{\leq l}, <)$. Then, for each $d \in \mathbb{N}$, there exist two words u and v such that $u \in L$, $v \notin L$, but u and v can not be distinguished by FO$(\sim_{\leq l}, <)$ formulas of quantifier depth d.

Let $u = a_1 \dots a_n$ be a word with more than $k + l$ different symbols. Consider a run

$$(p_0, \bar{a}_0) \xrightarrow{a_1} \dots \xrightarrow{a_n} (p_n, \bar{a}_n)$$

of A on u. As A is k-memory DMA, there must be $x, y \leq |u|$ such that

1. neither a_x nor a_y occur in $\bar{a}_y$,
2. $x + l < y$,
3. $u(x) \neq u(y)$, and
4. for all z with $x < z < y$, $u(x) \neq u(z)$ and $u(z) \neq u(y)$.

We define $u' = a_1 \dots a_x \dots a_{y-1}(a_y \dots a_n)[a_y/a_x]$. Observe that at position y, A can not distinguish between a_x and a_y as both do not occur in $\bar{a}_y$. Hence $u' \in L$. Observe also that, for all positions x', y' in u, the two structures (u, x', y') and (u', x', y') satisfy the same set of atomic FO$(\sim_{\leq l}, <)$ formulas. Thus, u and u' can not be distinguished in FO$(\sim_{\leq l}, <)$.

By iterating the above construction one can obtain a word $\tilde{u} \in L$ that can not be distinguished from u by any FO$(\sim_{\leq l}, <)$ formula. Moreover, $\tilde{u}$ 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 can not be distinguished from v by any FO$(\sim_{\leq l}, <)$ formula. Since u and v can not be distinguished by FO$(\sim_{\leq l}, <)$ formulas of quantifier depth d, by transitivity the same holds for $\tilde{u}$ and $\tilde{v}$.

Now, let D_{2k+2l} be a finite alphabet that contains all symbols in $\tilde{u}$ and in $\tilde{v}$. Since $\tilde{u} \in L \cap D_{2l}$ and $\tilde{u} \notin L \cap D_{2l}$, it follows that no FO$(\sim_{\leq l}, <)$ formula can define $L \cap D_{2l}$ and hence L is not NUFO$(\sim_{\leq l}, <)$ definable. □

Proposition 4. *A language L can be defined in MSO$(\sim_{\leq l}, +1)$ iff it can be recognized by an 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 $\simeq$-types of words of length at most l. Given a data word $u = a_1 \dots a_n \in D^*$, we then denote by $\mathsf{abs}_{\leq l}(u)$ the word $\alpha_1 \dots \alpha_n$ over the finite alphabet $T_{\leq l}$, where α_i is either the $\simeq$-type of the prefix $a_1 \dots a_i$ or the $\simeq$-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 $\mathsf{abs}_{\leq l}(L)$ the language over $T_{\leq l}$ that contains all and only the words of the form $\mathsf{abs}_{\leq l}(u)$, with $u \in L$.

Now, it is easy to see 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 $\mathsf{abs}_{\leq l}$. More precisely, a data language L is definable in $\mathrm{MSO}(\sim_{\leq l}, <)$ iff $\mathsf{abs}_{\leq l}(L) = \{\mathsf{abs}_{\leq l}(u) \mid u \in L\}$ 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 $\mathrm{MSO}(T_{\leq l}, <)$ formulas by exploiting structural induction as follows. 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 $\simeq$-types of words u such that $i \leq |u| \leq l$ and $u(|u| - i) = u(|u|)$. 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 $x_1, \ldots, x_n, X_1, \ldots, X_m$, every data word $u \in D^*$, every n-tuple of positions $i_1, \ldots, i_n \leq |u|$, and all unary predicates $P_1, \ldots, P_m \subseteq \{1, \ldots, |u|\}$, we have

$$(u, i_1, \ldots, i_n, P_1, \ldots, P_m) \vDash \psi \quad \text{iff} \quad (\mathsf{abs}_{\leq l}(u), i_1, \ldots, i_n, P_1, \ldots, P_m) \vDash f(\psi).$$

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

Next, observe that any l-DWA 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 $\mathsf{abs}_{\leq l}(L)$ when it is considered as a DFA. From the equivalence of MSO and finite automata over finite alphabets, it follows that $\mathsf{abs}_{\leq l}(L)$ is definable in $\mathrm{MSO}(T_{\leq l}, <)$ iff $\mathsf{abs}_{\leq l}(L)$ is recognized by a realizable l-DWA A when considered as a DFA. Moreover, the latter property holds iff L is accepted by an l-local DWA. Finally, by Proposition 2, l-DWA are exactly as expressive as l-local DMA (in fact, a data language L is recognized by an l-DWA iff it is recognized by an l-local DMA). □

A.2 Proofs from Section 2

Theorem 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 an 3-MA A, can $L(A)$ be defined in $\mathcal{L}$?*

Proof. The proof is by reduction from the Post Correspondence Problem (PCP). An instance of the PCP is a finite set $I = \{(u_1, v_1), \ldots, (u_k, v_k)\}$ consisting of pairs of words over a binary alphabet $\{a, b\}$. We say that I has a solution if there is a sequence of indices $i_1, \ldots, i_m$ such that $u_{i_1} \cdot \ldots \cdot u_{i_m} = v_{i_1} \cdot \ldots \cdot v_{i_m}$. It is known that the following problem is undecidable [15]: Given a PCP instance I, does I have a solution?

Let us fix a PCP instance $I = \{(u_1, v_1), \ldots, (u_k, v_k)\}$. In [8] Neven et al. encode a candidate solution of I of the form $S = (u_{i_1}, \ldots, u_{i_m}; v_{j_1}, \ldots, v_{j_n})$,

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 $\mathsf{enc}(S) = \square\, u\, \square\, v$, where $\square \in D$ is used as a delimiter and u and v are two words, with no occurrences of $\square$, that represent, respectively, the sequence $u_{i_1}, \ldots, u_{i_m}$ and the sequence $v_{j_1}, \ldots, v_{j_m}$ in S. They 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:

$$L_I^{\text{no-sol}} \;=\; \Big\{ v \in D^* \;\Big|\; \text{if } v = \mathsf{enc}(S), \text{ then } S \text{ is not a solution of } I \Big\}$$

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

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

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 prove that L_I can be defined in the logic $\mathcal{L}$ iff I has no solution (this would complete the proof of the theorem). If I has a solution S, then we let u_i be an encoding of S of the form $\square^i u\, \square\, v$, 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} can not 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} can not be distinguished by $\mathrm{FO}(<)$ formulas of quantifier depth d — see, for instance, [8]). 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. □

Theorem 3. *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 3-DMA A or a weak 1-way DPA A with 3 pebbles, can $L(A)$ be defined in $\mathcal{L}$?*

Proof (sketch). We first prove the claim for 2-way 3-DMA A. We claim that there is a 2-way DMA A_{dup} that recognizes the language

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

Indeed, A_{dup} can check that the following properties hold:

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

Making use of a construction in [8], 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 2.

As for 1-way DPA, in [8] it is shown that there is a 1-way DPA A_I^{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 2. □

A.3 Proofs from Section 3

Theorem 4. *The following problem is decidable: Given a local DMA A, is $L(A)$ definable in NUFO$(\sim, <)$?*

Proof. Let $A = (Q, q_I, F, T)$ be an l-local DMA with memory size k, let $L = L(A)$ and $N = l + kl|Q| + 1$. Hereafter, we will abbreviate $L \cap D_n^*$ by L_n. Clearly, if L is definable in NUFO$(\sim, <)$, then L_N is definable in FO$(D_N, <)$. It remains to show the converse, namely:

$$\text{If } L_N \text{ is definable in FO}(D_N, <), \text{ then } L \text{ is definable in NUFO}(\sim, <). \qquad (\star)$$

(here FO$(D_N, <)$ denotes classical first-order logic with predicates of the form $x < y$ and $a(x)$, for all $a \in D_N$). This will complete the proof of the theorem because (i) L_N is a regular language over a finite alphabet and (ii) it is known from [14] that the problem of checking whether a given regular language is definable in classical first-order logic is decidable. Below, we show the contrapositive of the above property $(\star)$.

Assume that L is not definable in NUFO$(\sim, <)$. Then, there is some natural number n such that L_n can not be defined in FO$(\sim, <)$. The following lemma shows that FO$(\sim, <)$ interpreted by words over the finite alphabet D_n has the same expressive power as FO$(D_n, <)$.

Lemma 1. *Let $L \subseteq D^*$ be an isomorphism-closed language and let $n \in \mathbb{N}$. Then L_n can be defined in FO$(\sim, <)$ over D_n iff L_n can be defined in FO$(D_n, <)$.*

Proof. First assume that there is an FO$(\sim, <)$ formula ϕ that defines $L_n \subseteq D_n^*$. We will abbreviate $L \cap D_n^*$ by L_n throughout this proof. Let ψ be the FO$(D_n, <)$ obtained from ϕ by replacing every occurrence of $x \sim y$ by $\bigvee_{a \in D_n} a(x) \wedge a(y)$. A simple proof by structural induction shows that ψ defines $L \cap D_n^*$.

For the other direction, let ψ be an FO$(D_n, <)$ formula that defines L_n. We assume that ψ is of the form $Q_1 x_1 \ldots Q_n x_n . \theta$ where each Q_i 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 $\bigvee_{b \in D_n \setminus \{a\}} b(x)$. 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 D_n induce a partition of the variables that occur in each clause. Assume that θ contains exactly m clauses and, for each $1 \leq i \leq m$, let $X_{i,a}$ be the set of all variables that occur in the i-th clause with the predicate a. We define

$$\tilde{\theta} \;=\; \bigvee_{i \leq m} \left(\bigwedge_{\substack{a \in D_n \\ x,y \in X_{i,a}}} (x \sim y) \quad \wedge \bigwedge_{\substack{a,b \in D_n,\, a \neq b \\ x \in X_{i,a},\, y \in X_{i,b}}} (x \not\sim y) \right)$$

By exploiting the fact that L is closed under isomorphism, one can show that $Q_1 x_1 \dots Q_n x_n . \theta$ is equivalent to $Q_1 x_1 \dots Q_n x_n . \tilde{\theta}$. □

Now, we need to introduce some definitions and another technical lemma. Let $\bar{\mathbf{c}} = (\mathbf{c}_1, \dots, \mathbf{c}_m)$ be a tuple of configurations of A. Given a word $u = a_1 \dots a_n$, we define the $\bar{\mathbf{c}}$-*trace* of u as the sequence $\gamma = \bar{\mathbf{c}}_0 \dots \bar{\mathbf{c}}_n$ of tuples of configurations of A, where $\bar{\mathbf{c}}_j = (\mathbf{c}_{1,j}, \dots, \mathbf{c}_{m,j})$ and each $\mathbf{c}_{i,j}$ is the configuration reached by A after reading the prefix $a_1 \dots a_j$ of the word u, starting from the configuration $\mathbf{c}_i$ (by convention, we let $\mathbf{c}_{i,0} = \mathbf{c}_i$). Given two words u and u' with the corresponding $\bar{\mathbf{c}}$-traces γ and γ', we say that γ *mimics* γ' if (i) u and u' coincide on all positions that are associated with values in $\bar{\mathbf{c}}$ and (ii) the two sequences obtained from γ and γ' after projecting the states coincide (in other words, γ and γ' differ only in the memory contents). The following lemma shows that the $\bar{\mathbf{c}}$-trace of any given word u can be mimicked by the $\bar{\mathbf{c}}$-trace on another word u' over an alphabet of bounded size (note that this result does not depend on the locality of A).

Lemma 2. *Let A be a k-DMA and let $\bar{\mathbf{c}}$ be an m-tuple of configurations of A. For any subset Δ of D that contains at least the values in $\bar{\mathbf{c}}$ plus $mk+1$ additional symbols, one can find a word $u' \in \Delta^*$ such that the $\bar{\mathbf{c}}$-trace of u' mimics the $\bar{\mathbf{c}}$-trace of u.*

Proof. Let A, $\mathbf{c}$, and Δ be as in the statement of the lemma. As a preliminary step, we introduce a transformation f on $\bar{\mathbf{c}}$-traces, which only depends on $\bar{\mathbf{c}}$ and Δ. Given a word $v = b_1 \dots b_n$ and its $\bar{\mathbf{c}}$-trace $\gamma = \bar{\mathbf{c}}_0 \dots \bar{\mathbf{c}}_n$, we consider the leftmost position x in v that is labeled by a value $b_x \in D \setminus \Delta$ (if such a position does not exist, then we simply let $f(\gamma) = \gamma$). Let b'_x be a value from Δ that occurs neither in the memory contents of $\bar{\mathbf{c}}_0$, nor in the memory contents of $\bar{\mathbf{c}}_{x-1}$ (note that such a value exists since, by definition, Δ contains more values than those in $\bar{\mathbf{c}}_0$ and in $\bar{\mathbf{c}}_{x-1}$). We then define

$$f(\gamma) \;=\; \bar{\mathbf{c}}_0 \dots \bar{\mathbf{c}}_{x-1} \, (\bar{\mathbf{c}}_x \dots \bar{\mathbf{c}}_n) \, [b_x \swarrow\!\nearrow b'_x]$$

where $(\bar{\mathbf{c}}_x \dots \bar{\mathbf{c}}_n) \, [b_x \swarrow\!\nearrow b'_x]$ denotes the sequence obtained from $\bar{\mathbf{c}}_x \dots \bar{\mathbf{c}}_n$ by substituting every occurrence of b_x by b'_x and, vice versa, every occurrence of b'_x by b_x. By a slight abuse of notation, we also let $f(v) = b_1 \dots b_{x-1} (b_x \dots b_n) [b_x \swarrow\!\nearrow b'_x]$. Note that the position of the leftmost occurrence in $f(v)$ of a value from $D \setminus \Delta$

(if any) is strictly greater than the position of the leftmost occurrence in v of a value from $D \setminus \Delta$.

Below we prove that $f(\gamma)$ is exactly the $\bar{\mathbf{c}}$-trace of $f(v)$ and, furthermore, it mimics the $\bar{\mathbf{c}}$-trace γ of v. As $\gamma = (\mathbf{c}_{1,0}, \ldots, \mathbf{c}_{m,0}) \ldots (\mathbf{c}_{1,n}, \ldots, \mathbf{c}_{m,n})$ is a trace of $v = b_1 \ldots b_n$, we know that for each index $1 \leq i \leq m$, there is a run of A on v of the form

$$\mathbf{c}_{i,0} \xrightarrow[\alpha_{i,1}, E_{i,1}]{b_1} \mathbf{c}_{i,1} \xrightarrow[\alpha_{i,2}, E_{i,2}]{b_2} \cdots \xrightarrow[\alpha_{i,n}, E_{i,n}]{b_n} \mathbf{c}_{i,n}.$$

Let us consider the run of A on $f(v) = b'_1 \ldots b'_n$ that starts at configuration $\mathbf{c}_{i,0}$:

$$\mathbf{c}'_{i,0} \xrightarrow[\alpha'_{i,1}, E'_{i,1}]{b'_1} \mathbf{c}'_{i,1} \xrightarrow[\alpha'_{i,2}, E'_{i,2}]{b'_2} \cdots \xrightarrow[\alpha'_{i,n}, E'_{i,n}]{b'_n} \mathbf{c}'_{i,n}$$

(note that $\mathbf{c}'_{i,0} = \mathbf{c}_{i,0}$).
Recall that x is the leftmost position in v that is labeled by a value $b_x \in D \setminus \Delta$. Since Δ contains all values in $\bar{\mathbf{c}}$ and b'_x does not occur in $\mathbf{c}$, we have that v and $f(v)$ coincides on all positions that are associated with values in $\bar{\mathbf{c}}$. Moreover, by construction, all values that occur in the memory content of $\mathbf{c}'_{i,x}$ belong to the set Δ. We now exploit an induction on $j \leq n$ to show that (i) $\alpha'_{i,j} = \alpha_{i,j}$, (ii) $E'_{i,j} = E_{i,j}$, and (iii) $\mathbf{c}'_{i,j} = f(\gamma)(i)(j)$, where $f(\gamma)(i)(j)$ denotes the j-th configuration of the tuple that appears at position i of $f(\gamma)$. All cases with $j < x$ are trivial, by construction. For $j = x$, we observe that b_x does not occur in Δ. Since $\mathbf{c}_{i,x-1}$ contains only values from Δ, it follows that the guard $\alpha_{i,x}$ requires that "the input value is not stored in the current memory". Hence $\alpha_{i,x}$ is also satisfied by the memory content of $\mathbf{c}_{i,x-1}$ and by the input value b'_x Since A is deterministic, we obtain (i) $\alpha'_{i,x} = \alpha_{i,x}$, (ii) $E'_{i,x} = E_{i,x}$, and (iii) $\mathbf{c}'_{i,x} = f(\gamma)(i)(x)$. The invariant is thus preserved for $j = x$. As for the remaining case $j > x$, one can easily see that the invariant is preserved because b'_j satisfies the same $\sim$-relationships with each value of the memory content of $\mathbf{c}'_{i,j}$ as b_j does with each value of the memory content of $\mathbf{c}_{i,j}$. The above arguments prove that $f(\gamma)$ is the $\mathbf{c}$-trace of $f(v)$ and that it mimics the $\mathbf{c}$-trace γ of v.

We conclude the proof as follows. Let $u = a_1 \ldots a_n$ be as in the statement of the lemma and let γ be the corresponding $\bar{\mathbf{c}}$-trace. We define a new word u' starting from u by iteratively applying f until a fixed point is reached (this happens after at most $|u|$ iterations, since each iteration moves the first occurrence of a symbol from $D \setminus \Delta$ to the right). From the properties of f mentioned above, it follows that $u' \in \Delta^*$ and, by transitivity, the $\bar{\mathbf{c}}$-trace of u' mimics that of u. □

Now, recall that from previous assumptions, A recognizes L and $L_n = L \cap D_n^*$. Clearly, the language L_n is recognized by a deterministic finite state automaton A_n: the states of A_n are the (finitely many) configurations of A where the memory values are restricted to range over D_n, the transitions of A_n mimic those of A in the obvious way, and the initial and final states of A_n coincide with the initial and final configurations of A, respectively. The aboe defined DFA A_n is said to be the *projection* of A onto the finite alphabet D_n. We also denote by A'_n the *minimum* DFA that recognizes L_n (this can be obtained by collapsing states of A_n). The following result shows that the projection of a canonical DMA A' onto a sufficiently large alphabet D_n is a minimal DFA recognizing L_n.

Proposition 7. *Let A' be a canonical k-DMA that recognizes a language L. For every $n \geq 2k+1$, the minimal DFA A'_n that recognizes $L_n = L \cap D_n^*$ can be obtained from the projection of A' onto the finite alphabet D_n.*

Proof. Let us fix a natural number $n \geq 2k+1$ and let A'_n be the minimal DFA recognizing $L_n = L \cap D_n^*$. From standard results in automata theory, we can identify each state of A'_n with some equivalence class of the form $[u]_{\equiv_{L_n}}$, where $u \in D_n^*$ and $\equiv_{L_n}$ is the Myhill-Nerode equivalence such that $u \equiv_{L_n} v$ iff, for every word $s \in D_n^*$, $u \cdot s \in L_n$ iff $v \cdot s \in L_n$. Similarly, from the results in [13], we can identify each state of the canonical DMA A' with an equivalence class of form $[u]_{\equiv_L}$, where $u \in D^*$ and $\equiv_L$ is the equivalence over data words introduced in Section 1.1. In order to prove the proposition, it is sufficient to show that, for any pair of words $u, v \in D_n^*$, $u \equiv_{L_n} v$ iff $u \equiv_L v$. The right-to-left direction is trivial. Below, we prove the contrapositive of the left-to-right direction.

Suppose that u and v are two words over D_n such that $u \not\equiv_L v$. From Definition 3, we have that either $\mathsf{mem}_L(u) \not\simeq \mathsf{mem}_L(v)$ or there exist two words $u', v' \in D^*$ such that $\mathsf{mem}_L(u) \cdot u' \simeq \mathsf{mem}_L(v) \cdot v'$ and $u \cdot u' \not\equiv_L v \cdot v'$. We only consider the second, more interesting, case (the proof in the first case uses similar arguments). Suppose that u' and v' are two words over D^* such that $\mathsf{mem}_L(u) \cdot u' \simeq \mathsf{mem}_L(v) \cdot v'$ and $u \cdot u' \not\equiv_L v \cdot v'$. We first consider the $\mathbf{c}$-trace of u', where $\mathbf{c}$ is the configuration reached by A' after reading u, starting from its initial configuration. Since $n \geq 2k+1$ and the language L is closed under $\simeq$-isomorphisms, we can assume that the finite alphabet D_n^* contains at least the values in $\mathbf{c}$ and $k+1$ additional values not in $\mathbf{c}$. We can thus apply Lemma 2 (with $m = 1$) and devise the existence of another word $u'' \in D_n$ whose $\mathbf{c}$-trace mimics that of u'. Since A' is canonical, the L-memorable values in u coincide with the values stored in the configuration $\mathbf{c}$ and hence, since u' and u'' coincide on all positions associated with values in $\mathbf{c}$, it follows that $\mathsf{mem}_L(u) \cdot u' \simeq \mathsf{mem}_L(u) \cdot u''$. Moreover, since the $\mathbf{c}$-trace of u'' mimics that of u', we have $u \cdot u' =_L u \cdot u''$. By using analogous arguments, one can show that there is a word $v'' \in D_n$ such that $\mathsf{mem}_L(v) \cdot v' \simeq \mathsf{mem}_L(v) \cdot v''$ and $u \cdot u' =_L u \cdot u''$. Finally, by transitivity, we conclude that $u \cdot u'' \not\equiv_L v \cdot v''$. This implies that u and v are distinguishable by the Myhill-Nerode equivalence $\equiv_{L_n}$ and this completes the proof of the proposition. □

We turn back to the proof of Theorem 4. By previous assumptions and Lemma 1, we know that there is a natural number n such that L_n can not be defined in $\mathrm{FO}(D_n, <)$. Let A'_n be the minimal DFA that recognizes L_n. From [14], it follows that A'_n has a *counter* C over a word $u \in D_n^*$ (intuitively, C is a cycle of consecutive transitions that read non-trivial repetitions of the same word u). Moreover, let A' be the canonical DMA equivalent from A. From Proposition 7, A'_n coincides with the projection of A' onto the finite alphabet D_n. We can thus assume that the counter C of A'_n is of the form

$$\mathbf{c}_1 \xrightarrow{u} \ldots \xrightarrow{u} \mathbf{c}_m \xrightarrow{u} \mathbf{c}_1$$

where $m \geq 2$ and $\mathbf{c}_1, \ldots, \mathbf{c}_m$ are pairwise distinct configurations of A' featuring only values of u. Moreover, as A' is also l-local, the memory content in each configuration $\mathbf{c}_i$ contains only values that occur in the l-length suffix of u and hence $m \leq l|Q|$.

Now, let $N = l + kl|Q| + 1$ and $\bar{\mathbf{c}} = (\mathbf{c}_1, \ldots, \mathbf{c}_m)$. Since $N \geq l + mk + 1$ and L is closed under $\simeq$-isomorphisms, we can assume that D_N contains all values in $\bar{\mathbf{c}}$ plus $mk + 1$ additional values. We can thus apply Lemma 2 and devise the existence of a word $u' \in D_N$ whose $\bar{\mathbf{c}}$-trace mimics that of u. Let γ be the $\bar{\mathbf{c}}$-trace on u and let γ' be the $\bar{\mathbf{c}}$-trace on u'. We claim that γ' induces a counter of A_N on u'. Indeed, since γ' mimics γ, we know that the two words u and u' coincide on all positions that are associated with values in $\bar{\mathbf{c}}$. For the same reason, the two sequences γ and γ' coincide up to a renaming of the values that do not occur in $\bar{\mathbf{c}}$. Hence, since A_n moves from the configuration $\mathbf{c}_i$ to the configuration $\mathbf{c}_{(i \bmod m)+1}$ by reading u, A_N does the same by reading u'. This shows that C is a counter of A_N on u'.

Towards a conclusion, we observe that $N \geq mk + 1 \geq 2k + 1$ and hence, by Proposition 7, A_N is the minimal DFA that recognizes L_N. By applying again the results from [14], we conclude that L_N is not definable in $\mathrm{FO}(D_N, <)$. □

Theorem 5. *The following problem is decidable: Given a 1-DMA A, is $L(A)$ definable in NUFO$(\sim, <)$?*

Proof. The proof is by case analysis. Let $A = (Q, q_I, F, T)$ be a canonical 1-DMA that recognizes a language L.

We distinguish two types of transitions in 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 1, we recall that the period of a simple loop C in 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 A with period strictly greater than 1;
2. there is a simple loop in 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 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 turn out to be trivial, namely, they always return false).

Lemma 3. *If A contains a non-local simple loop of period strictly greater than 1, then L is not definable in NUFO$(\sim, <)$.*

Proof. Let C be a non-local simple loop in 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 u and v such that (i) $u^A(q_I, \varepsilon) = (p_1, \bar{a})$ (namely, A reaches C after reading u) and (ii) $v^A(p_i, \bar{a}) = (p_{(i \bmod m)+1}, \bar{a})$ for all $i \leq m$ (namely, after reading u, A cycles through C by reading repetitions of v). Let N be the number of distinct values that occur in u and in v. Below, we prove that $L_N = L \cap D_N$ is not definable in FO$(D_N, <)$, and hence, by Lemma 1, L is not definable in NUFO$(\sim, <)$.

Suppose, by way of contradiction, that there is an FO$(D_N, <)$ formula ϕ of quantifier depth d that defines the language L_N. Since 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 1.1 (see also [13]) that there is a word $s \in D_N^*$ such that

$$\left(u \cdot v^{m^d} \cdot s\right) \not\equiv_L \left(u \cdot v^{m^d+1} \cdot s\right).$$

To obtain a contraction it is sufficient to show that ϕ cannot distinguish between the two words $\left(u \cdot v^{m^d} \cdot s\right)$ and $\left(u \cdot v^{m^d+1} \cdot s\right)$. 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$(D_N, <)$ interpreted over $\left(u \cdot v^{m^d} \cdot s\right)$ and $\left(u \cdot v^{m^d+1} \cdot s\right)$. □

Lemma 4. *If 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. 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 w 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 u and a subword v of w such that (i) $u^A(q_I, \varepsilon) = (p, \bar{a})$, (ii) $|v|$ is a multiple of the period m of C, and (iii) $v^A(p, \bar{a}) = (p, \bar{a})$. This means that the infinite word $u \cdot v^\omega$ eventually cycles through C, exactly as w does. Let N be the number of distinct values that occur in u and in v. Below, we prove that $L_{2N} = L \cap D_{2N}$ is not definable in FO$(D_{2N}, <)$, and hence, by Lemma 1, L is not definable in NUFO$(\sim, <)$.

Suppose, by way of contradiction, that there is an FO$(D_{2N}, <)$ formula ϕ that defines the language $L_{2N} = L \cap D_{2N}^*$. Let d be the quantifier depth of ϕ and let v' be an isomorphic copy of v consisting of fresh values from $D_{2N} \setminus D_N$ (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, first showing that there is a word $s \in D_{2N}^*$ such that

$$\left(u \cdot (v \cdot v')^{2^d} \cdot s\right) \not\simeq_L \left(u \cdot (v \cdot v')^{2^{d+1}} \cdot s\right).$$

and then showing that ϕ cannot distinguish between these two words. □

Lemma 5. *Suppose that for every pair of intersecting simple loops in A, either both loops are local, or both have period 1. Let B be the local DMA obtained from 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 $L(B)$ is definable in NUFO$(\sim, <)$.*

Proof. Let us fix, in 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 A' be the DMA obtained from A by removing these two transitions. Since C' is non-local, we have that α' is the $\simeq$-type of a word of the form ab, with $a \neq b$. Moreover, since A is deterministic and C intersects C', α is the $\simeq$-type of a word of the form aa (hence C is local). This implies that 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 A'). Therefore, depending on whether p is final or not, we have $L = L(A') \cdot D^*$ or $L = L(A')$. In both cases, we have that A is definable in NUFO$(\sim, <)$ iff $L(A')$ is definable in NUFO$(\sim, <)$.

To conclude the proof, it is sufficient to observe that the local DMA B is obtained from 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 4, together with Lemma 3, Lemma 4, and Lemma 5, finally gives a proof of Theorem 5. □

A.4 Proofs from Section 4

Theorem 6. *The following problem is decidable: Given a DMA A, is there an l such that $L(A)$ is definable in FO$(\sim_{\leq l}, <)$? If such an l exists, then we can compute the minimal l_0 such that $L(A)$ is definable in FO$(\sim_{\leq l_0}, <)$. Analogous results hold when $<$ is replaced by $+1$.*

Proof. In [13] we have shown that, given a DMA A, one can compute an equivalent canonical DMA. We thus assume that A is a DMA already in canonical form. By Proposition 1, one can check whether A is local and, in such a case, compute the minimum number l_0 such that A is l_0-local. It follows from the next lemma that if A is not local, then $L(A)$ can not be defined in FO$(\sim_{\leq l'}, <)$ (nor in FO$(\sim_{\leq l'}, +1)$) for any $l' \in \mathbb{N}$. It also follows that if A is l_0-local but not l'-local for any $l' < l_0$, then $L(A)$ can not be defined in FO$(\sim_{\leq l'}, <)$ (nor in FO$(\sim_{\leq l'}, +1)$).

Lemma 6. *If a canonical DMA A is not l-local, then $L(A)$ is not definable in $FO(\sim_{\leq l}, <)$.*

Proof. Suppose that 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 u. Clearly, a occurs in $\bar{a}$. Let u be a word that induces a run of A from the initial configuration to the configuration $(p, \bar{a})$ (such a run exists since all states of A are reachable). Now, recall that a canonical DMA stores only memorable values. Since a is stored after reading $u \cdot v$, then a is L-memorable in $u \cdot v$. Thus, by definition of memorable value, there is a data word s and a value b such that $u \cdot v \cdot s \not\equiv_L u \cdot v \cdot (s[a/b])$. As a does not occur in v and $|v| = l$, one can show, using a simple game-theoretic argument, that $u \cdot v \cdot s$ and $u \cdot v \cdot (s[a/b])$ can not be distinguished in $\mathrm{FO}(\sim_l, <)$. □

We now consider the case of A being a l_0-local, but not l'-local for any $l' < l_0$. By Proposition 2, one can compute an l_0-DWA B equivalent to A. We can further assume that B is realizable. We denote by $\mathrm{FO}(T_{\leq l}, <)$ the first-order logic over the finite vocabulary $T_{\leq l}$. A similar argument as in the proof of Proposition 4 shows that $L(B)$ is definable in $\mathrm{FO}(\sim_{\leq l}, <)$ iff $\mathsf{abs}_{\leq l}(L(B))$ is definable in $\mathrm{FO}(T_{\leq l}, <)$ (abs is defined in the proof of 4). This reduces the problem of deciding whether $L(A)$ is $\mathrm{FO}(\sim_{\leq l_0}, <)$ decidable to an analogous problem over DFA. It was shown in [14] that the latter problem is decidable.

We recall the basic facts that have been disclosed so far: (i) one can decide whether the given canonical DMA A is local and, in such a case, compute the minimum l_0 such that A is l_0-local, (ii) if A is not local then $L(A)$ can not be defined in $\mathrm{FO}(\sim_{\leq l}, <)$ for any $l \in \mathbb{N}$ (iii) if A is l_0-local then $L(A)$ and l_0 is minimal then we know that $L(A)$ can not be defined in $\mathrm{FO}(\sim_{\leq l}, <)$ for any $l < l_0$ and we can also check whether $L(A)$ can be defined in $\mathrm{FO}(\sim_{\leq l_0}, <)$. Still, it is not clear how one can decide whether $L(A)$ is definable in $\mathrm{FO}(\sim_{\leq l}, <)$, for some $l > l_0$, if $L(A)$ is *not* definable in $\mathrm{FO}(\sim_{\leq l_0}, <)$. The following lemma gives a straightforward answer to such a question: since A is l_0-local by hypothesis it follows that if $L(A)$ is not definable in $\mathrm{FO}(\sim_{\leq l_0}, <)$, then it is not definable in $\mathrm{FO}(\sim_{\leq l}, <)$, for any $l > l_0$, either. This concludes the proof Theorem 6.

Lemma 7. *Let A be an l_0-local DMA. If $L(A)$ is not definable in $FO(\sim_{\leq l_0}, <)$, then $L(A)$ is not definable in $FO(\sim_{\leq l}, <)$ for all $l > l_0$.*

Proof. We exploit an induction on $l - l_0$. The base case $l - l_0 = 0$ is trivial. Let $l \geq l_0$ and assume that $L = L(A)$ is not definable in $\mathrm{FO}(\sim_{\leq l}, <)$. We have to prove that L is not definable in $\mathrm{FO}(\sim_{\leq l+1}, <)$ either. Let us fix an arbitrary natural number d for the quantifier depth of formulas. Since there exist no $\mathrm{FO}(\sim_{\leq l}, <)$ formula of quantifier depth d that defines L, there exist two words u and v such that (i) $u \in L$, (ii) $v \notin L$, and (iii) Duplicator has a winning strategy in the d-round $\mathrm{FO}(\sim_{\leq l}, <)$-game on u and v. Below, we show that there exist two corresponding words $\tilde{u} \in L$ and $\tilde{v} \notin L$ and a winning strategy for Duplicator in the d-round

$\mathrm{FO}(\sim_{\leq l+1}, <)$-game on $\tilde{u}$ and $\tilde{v}$ (this would immediately imply that L can not be defined by any $\mathrm{FO}(\sim_{\leq l+1}, <)$ formula of quantifier depth d).

We first define $\tilde{u}$ starting from u. Suppose that $u = a_1 \dots a_n$ 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 $u' = a_1 \dots a_{y-1} \cdot (a_y \dots a_n)[a_y/b]$, where b is a fresh value that does not occur in u. Observe that for any two positions x' and y', with $x' \leq y' \leq x' + l$, we have $u(x') = u(y')$ iff $u'(x') = u'(y')$. In addition, since $\mathsf{abs}_{\leq l_0}(u) = \mathsf{abs}_{\leq l_0}(u')$, we have $u' \in L$. By iterating the above construction, one obtain a word $\tilde{u}$ from u such that

1. $\tilde{u} \in L$;
2. if $y \leq x + l$, then $\tilde{u}(x) = \tilde{u}(y)$ iff $u(x) = u(y)$;
3. if $y = x + (l+1)$, then $\tilde{u}(x) \neq \tilde{u}(y)$.

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 $\mathrm{FO}(\sim_{\leq l}, <)$-game on u and v can be used to win the d-round $\mathrm{FO}(\sim_{\leq l+1}, <)$-game on $\tilde{u}$ and $\tilde{v}$. Suppose that the play in the $\mathrm{FO}(\sim_{\leq l+1}, <)$-game on $\tilde{u}$ and $\tilde{v}$ has progressed up to the i-th round and that the pebbles lie at positions $x_1, \dots, x_i$ in $\tilde{u}$ and at positions $y_1, \dots, y_i$ in $\tilde{v}$. Suppose that Spoiler places a new pebble at position x_{i+1} in $\tilde{u}$ (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 $(u, x_1, \dots, x_{i+1})$ and $(v, y_1, \dots, y_{i+1})$ satisfy the same atomic formulas in $\mathrm{FO}(\sim_{\leq l+1}, <)$. Due to property 2. above, the configuration reached after Spoiler action can be viewed as a configuration in the $\mathrm{FO}(\sim_{\leq l}, <)$-game on u and v. Duplicator can thus respond by using the winning strategy in that game. This guarantees that the two resulting structures $(u, x_1, \dots, x_{i+1})$ and $(v, y_1, \dots, y_{i+1})$ satisfy the same atomic formulas in $\mathrm{FO}(\sim_{\leq l}, <)$. From property 2., it then follows that $(\tilde{u}, x_1, \dots, x_{i+1})$ and $(\tilde{v}, y_1, \dots, y_{i+1})$ satisfy the same atomic formulas in $\mathrm{FO}(\sim_{\leq l}, <)$. Finally, from property 3., the same holds for the atomic formulas in $\mathrm{FO}(\sim_{l+1}, <)$. This shows that $\tilde{u} \in L$ and $\tilde{v} \notin L$ can not be distinguished by any $\mathrm{FO}(\sim_{\leq l+1}, <)$ formula of (arbitrary) quantifier depth d and hence L can not be defined in $\mathrm{FO}(\sim_{\leq l+1}, <)$. □

A.5 Proofs from Section 5

Proposition 5. *Let A be a DWA. If $P_u(A)$ is counter-free for all $u \in D^{k-1}$, then $L(A)$ is definable in FO$(\sim, <)$.*

Proof. We fix a k-DWA A that recognizes L. Observe that A accepts exactly the words w such that $(w[1, k-1], w)$ is accepted by $P_{w[1,k-1]}(A)$ (recall that $P_u(A)$ is the u-parametrized version of A). Then $L = \bigcup_{u \in D^{k-1}} L_u$ where

$$L_u = \{w \in D^* \mid w[1, k-1] \simeq u \text{ and } (u, w) \in L(P_u(A))\}.$$

The automaton $P_u(A)$, can be thought of as a deterministic finite-state automaton over the alphabet $T_{k-1,k}$ of $(k-1)$-parametrized k-types. We denote this automaton by B_u. Note that, since $P_u(A)$ is realizable, $L(B_u) \subseteq T^*_{k-1,k}$ coincides with the language $\mathsf{abs}_{k-1,k}(L(P_u(A)))$, where $\mathsf{abs}_{k-1,k}$ maps parametrized words to sequences of $T_{k-1,k}$-types in the obvious way (see the proof of Theorem 6 for similar notations and arguments).

Now fix an $u \in D^{k-1}$ and suppose that B_u is counter-free. It follows from standard results in automata theory [14] that the language $L(B_u)$ is definable in $\mathrm{FO}(T_{k-1,k}, <)$. Similar arguments as in Proposition 4 show that $L(B_u)$ is definable in $\mathrm{FO}(T_{k-1,k}, <)$ iff L_u is definable in $\mathrm{FO}(\sim, <)$.

In order to complete the proof, it is sufficient to observe that the above languages L_u are uniquely determined by the $\simeq$-type of u and hence there exist at most $|T_{k-1}|$ such languages. Since $L = \bigcup_{u \in D^{k-1}} L_u$ and $\mathrm{FO}(\sim, <)$ definable languages are closed under finite unions, we conclude that L can be defined in $\mathrm{FO}(\sim, <)$. □

Proposition 6. *Let A be a canonical DWA. If there a DPWA $B \in \mathcal{P}(A)$ that contains a bad counter, then $L(A)$ is not definable in $FO(\sim, <)$.*

Proof. Let A be a canonical k-DWA and assume that $\mathcal{P}(A)$ contains a k-DPWA $B = (Q, T, q_I, F) = P_{u_{\mathrm{in}}, v_{\mathrm{in}}}(A)$ with a bad counter

$$p_1 \xrightarrow{\bar\alpha_1} \ldots \xrightarrow{\bar\alpha_{n-1}} p_n \xrightarrow{\bar\alpha_n} p_{n+1} \xrightarrow{\bar\alpha} \ldots \xrightarrow{\bar\alpha} p_{n+m} \xrightarrow{\bar\alpha} p_{n+1}$$

By definition of bad counter, we have:

1. $n \geq 0$ and $m \geq 2$,
2. $p_1, \ldots, p_{n+m}$ are pairwise distinct states, and p_1 is of the form $(p, [u_{\mathrm{in}}, u_{\mathrm{in}}])$,
3. $\bar\alpha_1, \ldots, \bar\alpha_n, \bar\alpha \in T^l_{k-1,k}$ for some $l > 0$, and $\mathsf{loc}(\bar\alpha_1) = \ldots = \mathsf{loc}(\bar\alpha_n) = \mathsf{loc}(\bar\alpha)$.

Let $\bar\alpha_{\mathrm{in}}$ be a sequence of $(k-1)$-parameterized k-types that labels a path from q_I to p_1. As B is canonical, there is a sequence $\bar\alpha_{\mathrm{out}}$ of $(k-1)$-parameterized k-types such that there is a $\bar\alpha_{\mathrm{out}}$-labelled path that leads from p_{n+1} to an accepting state and there is a $\bar\alpha_{\mathrm{out}}$-labelled path that leads from p_{n+2} to a rejecting state. For all $r \in \mathbb{N}$, we define

$$\begin{aligned} \bar\gamma_r^+ &= \bar\alpha_{\mathrm{in}} \cdot \bar\alpha_1 \cdot \ldots \cdot \bar\alpha_n \cdot \bar\alpha^{mk2^r} \cdot \bar\alpha_{\mathrm{out}} \\ \bar\gamma_r^- &= \bar\alpha_{\mathrm{in}} \cdot \bar\alpha_1 \cdot \ldots \cdot \bar\alpha_n \cdot \bar\alpha^{mk2^r+1} \cdot \bar\alpha_{\mathrm{out}} \end{aligned}$$

Given a path ρ in a DPWA, we say that $(u, a_1 \ldots a_n)$ is a *free parametrized word realizing* ρ if $(u, a_1 \ldots a_n)$ induces a run along ρ and, for all $i \leq n$, if $(u, a_1 \ldots a_{i-1} a'_i \ldots a'_n)$ induces a run along ρ for some $a'_i \ldots a'_n$, with a'_i not occurring in $a_1 \ldots a_{i-1}$, then a_i does not occur in $a_1 \ldots a_{i-1}$ either.

Let (u_{in}, u_r^+) be the free parametrized word realizing the (unique) path in B that is labelled $\bar\gamma_r^+$. Similarly (u_{in}, u_r^-) is the free parametrized word realizing the $\bar\gamma_r^-$-labelled path in B. By definition the parametrized word (u_{in}, u_r^+) is accepted

by B and hence $u_r^+ \in L(A)$. Symmetrically, $(u_{\text{in}}, u_r^-) \notin L(B)$ and since both A and B are deterministic $u_r^- \notin L(A)$.

We will show that no FO($\sim, <$) formula of quantifier depth $r' = r - \lceil \log(k) \rceil$ can distinguish between u_r^+ and u_r^-. We will do so by showing that Duplicator has a winning strategy for the r'-round FO($\sim, <$)-game on u_r^+ and u_r^-. This game is defined as follows: There are two players, called *Spoiler* and *Duplicator*. At the beginning of round $i \leq r'$ the game boards consists of two structures $(u_r^+, x_1, \ldots, x_{i-1})$ and $(u_r^-, y_1, \ldots, y_{i-1})$. Spoiler places a pebble i either at a position of u_r^+ or at a position of u_r^-. If he picks position x_i in u_r^+, then Duplicator picks a position y_i in u_r^-. If Spoiler picks a position in u_r^-, then Duplicator picks a position in u_r^+. In either case, the new game board is $(u_r^+, x_1, \ldots, x_i)$ and $(u_r^-, y_1, \ldots, y_i)$. Note that multiple pebbles can be placed at the same position. This concludes round i. If $i < r'$ then the players proceed to play round $i+1$, and if $i = r'$ then the game ends. Duplicator wins the game if for all $i, j \leq r'$ the same atomic FO($\sim, <$) formulas are true on (u_r^+, x_i, x_j) as on (u_r^-, y_i, y_j). Otherwise Spoiler wins. It has been shown in [16] that Duplicator has a winning strategy for the r'-round FO($\sim, <$)-game on u_r^+ and u_r^- iff u_r^+ and u_r^- can not be distinguished by FO($\sim, <$) formulas of quantifier depth r'. The FO($<$)-game is played in the same way, the only difference being that the winning condition is with respect to FO($<$) instead of FO($\sim, <$).

Beside the *main game* on u_r^+, u_r^-, Duplicator will play a *surrogate* r-round FO($<$)-game on the structures a^{2^r}, a^{2^r+1}. It is known that Duplicator has a winning strategy for this game [16]. Duplicator will use this strategy for her strategy in the main game.

Assume that the players have played $i < r'$ rounds, the game board of the main game is $(u_r^+, x_1 \ldots x_i)$, $(u_r^-, y_1 \ldots y_i)$, and the board of the surrogate game is $(a^{2^r}, x'_1 \ldots x'_i)$, $(a^{2^r+1}, y'_1 \ldots y'_i)$. We call a position x on u_r^+ *small* if $x \leq |\bar{\alpha}_{\text{in}}\ \bar{\alpha}_1\ \ldots\ \bar{\alpha}_n|$. If $x > |\bar{\alpha}_{\text{in}}\ \bar{\alpha}_1\ \ldots\ \bar{\alpha}_n\ \bar{\alpha}^{mk2^r}|$ then we call x *big*. If x is neither small nor big then we call x *interesting*. Similarly we call y on u_r^- *small* if $y \leq |\bar{\alpha}_{\text{in}}\ \bar{\alpha}_1\ \ldots\ \bar{\alpha}_n|$, *big* if $y > |\bar{\alpha}_{\text{in}}\ \bar{\alpha}_1\ \ldots\ \bar{\alpha}_n\ \bar{\alpha}^{mk2^r+1}|$, and *interesting* otherwise. Duplicator will maintain the following invariant:

1. $(a^{2^{r+1}}, x'_1 \ldots x'_i)$, $(a^{2^{r+1}+1}, y'_1 \ldots y'_i)$ is a winning position for the FO($<$)-game with $r+1-i$ rounds left to play.
2. For all $j \leq i$, if either x_j or y_j is small then $x_j = y_j$.
3. For all $j \leq i$, if either x_j or y_j is big then $x_j + |\bar{\alpha}| = y_j$.
4. For all $j \leq i$, if either x_j or y_j is interesting then $x_j = y_j \bmod |\bar{\alpha}|$.
5. For all $j \leq i$, if x_j is interesting then $|\bar{\alpha}_{\text{in}}\ \bar{\alpha}_1\ \ldots\ \bar{\alpha}_n| + (x'_j - 1)|\bar{\alpha}| \leq x_j \leq |\bar{\alpha}_{\text{in}}\ \bar{\alpha}_1\ \ldots\ \bar{\alpha}_n| + x'_j|\bar{\alpha}|$.
6. For all $j \leq i$, if y_j is interesting then $|\bar{\alpha}_{\text{in}}\ \bar{\alpha}_1\ \ldots\ \bar{\alpha}_n| + (y'_j - 1)|\bar{\alpha}| \leq y_j \leq |\bar{\alpha}_{\text{in}}\ \bar{\alpha}_1\ \ldots\ \bar{\alpha}_n| + y'_j|\bar{\alpha}|$.

Lemma 8. *If x_i, x_j on u_r^+ and if y_i, y_j on u_r^- are all interesting and if $|x_i - x_j| \leq k|\bar{\alpha}|$ or $|y_i - y_j| \leq k|\bar{\alpha}|$ then $x_i - x_j = y_i - y_j$.*

Proof. Assume towards a contradiction that $|x_i - x_j| \leq k|\bar{\alpha}|$ and $x_i - x_j \neq y_i - y_j$. Then $|x'_i - x'_j| \leq k$. If follows from invariant 4 above that $x'_i - x'_j \neq y'_i - y'_j$. As

the surrogate game has $\lceil \log(k) \rceil$ more rounds than the main game, there are at least $\lceil \log(k) \rceil$ rounds left in the surrogate game. But then the position in the surrogate game is not a winning position, contradicting invariant 1. □

For all $i \leq r'$, Duplicator uses the following strategy in the i-th round:

1. If Spoiler picks a small position the Duplicator picks the same position in the other word.
2. If Spoiler picks a big position the Duplicator picks a position such that $x_i + |\bar{\alpha}| = y_i$
3. Now assume that Spoiler picks an interesting position x_i on u_r^+. Then there is an x' such that $|\bar{\alpha}_{\text{in}}\ \bar{\alpha}_1\ \ldots\ \bar{\alpha}_n| + (x'-1)|\bar{\alpha}| \leq x_i \leq |\bar{\alpha}_{\text{in}} f\ \bar{\alpha}_1\ \ldots\ \bar{\alpha}_n| + x'|\bar{\alpha}|$. Now Duplicator turns to the surrogate game and places pebble x'_i on position x' of the structure a^{2^r} – giving rise to the game board $(a^{2^r}, x'_1, \ldots, x'_i)$, $(a^{2^r+1}, y'_1, \ldots, y'_{i-1})$. She then uses her winning strategy for the surrogate game to determine the next move y'_i. Then Duplicator turns back to the main game, and places the pebble y_i such that $|\bar{\alpha}_{\text{in}}\ \bar{\alpha}_1\ \ldots\ \bar{\alpha}_n| + (y'_i - 1)|\bar{\alpha}| \leq y_i \leq |\bar{\alpha}_{\text{in}}\ \bar{\alpha}_1\ \ldots\ \bar{\alpha}_n| + y'_i|\bar{\alpha}|$ and $x_i = y_i \bmod |\bar{\alpha}|$.
4. The case where Spoiler picks an interesting position on u_r^- is symmetric to case 3.

To show that Duplicator's strategy is a winning strategy we will make use of the following lemma.

Lemma 9. *Let $i \in \mathbb{N}$. If x, x', and $x' + |\bar{\alpha}|$ are interesting positions in u_r^+ such that $|x - x'| \geq k|\bar{\alpha}|$ then*

$$u_r^+(x) = u_r^+(x') \quad \textit{iff} \quad u_r^+(x) = u_r^+(x' + |\bar{\alpha}|).$$

Proof. We only show the direction from left to right. The proof of the other direction is similar. We will call a value a that occurs in u_r^+ *recurrent* if there are two positions y, y' in u_r^+ that both have value a and $y = y' \bmod |\bar{\alpha}|$. We first show that if a is recurrent, x and $x + |\bar{\alpha}|$ are interesting positions in u_r^+, and $u_r^+(x) = a$ then $u_r^+(x + |\bar{\alpha}|) = a$. This proves the claim for recurrent values.

Assume that a is recurrent, $u_r^+(x) = a$, and x and $x + |\bar{\alpha}|$ are interesting positions in u_r^+. As a is recurrent there are positions y, y' (we assume $y < y'$) in u_r^+ with value a and $y = y' \bmod |\bar{\alpha}|$. Recall that (u_{in}, u_r^+) is a free realizing relativized word for a $\bar{\gamma}_r^+$-labelled path. As (u_{in}, u_r^+) is free y and y' can only have the same value if either a occurs in u_{in} or if there are positions $y = y_1 < \ldots < y_l = y'$ in $\bar{\gamma}_r^+$ that have value a and $y_{i+1} - y_i \leq |\bar{\alpha}|$ for all $i \leq l$.

We first consider the case where a occurs in u_{in}. As x and $x+|\bar{\alpha}|$ are interesting positions, $\bar{\gamma}_r^+(x)$ and $\bar{\gamma}_r^+(x + |\bar{\alpha}|)$ must be the same parameterized type. Hence this type must be of the form $[u_{\text{in}}, w]$ where the last letter of w is a. It follows that $u_r^+(x + |\bar{\alpha}|)$ must be a.

Now assume that there are positions $y = y_1 < \ldots < y_l = y'$ in $\bar{\gamma}_r^+$ that have value a and $y_{i+1} - y_i \leq |\bar{\alpha}|$ for all $i \leq l$. As $\mathsf{loc}(\bar{\alpha}_1) = \ldots = \mathsf{loc}(\bar{\alpha}_n) = \mathsf{loc}(\bar{\alpha})$ there are positions $y'' = y'_1 < \ldots < y'_{l'} = y$ that also have value a and $y_{i+1} - y_i \leq |\bar{\alpha}|$ for all $i \leq l'$. In addition $|\bar{\alpha}_{\text{in}} - \bar{\alpha}| \leq y'' < |\bar{\alpha}_{\text{in}}|$. As p_1 is of the form $(p, [u_{\text{in}}, u_{\text{in}}])$ it

follows from the definition of B that a must occur in u_{in}. We have already dealt with this case above.

We still need to consider non-recurrent values. Observe that a non-recurrent value a can occur at most $|\bar{\alpha}|-1$ times in u_r^+. As two occurrences of a are atmost $|\bar{\alpha}|$ symbols apart it follows that non-recurrent values are at most $k|\bar{\alpha}|$ positions apart. Hence the claim also holds for non-recurrent values. This completes the proof of the claim. □

We now show that Duplicator's strategy is a winning strategy. Assume that $(u_r^+, x_1 \dots x_{r'})$, $(u_r^i, y_1 \dots y_{r'})$ is the game board at the end of round r'. We need to check that for all $i, j \leq r'$, the same FO$(\sim, <)$ formulas are true on (u_r^+, x_i, x_j), (u_r^i, y_i, y_j).

It is easy to show that if $x_i < x_j$ and $y_i \not< y_j$ then the position in the surrogate game is not a winning position (the formal argument that involves invariant 1,4,5, and 6 is omitted). Now consider the atomic formula $x \sim y$. We proceed by case distinction. Observe that by the invariant, x_i is small iff y_i is small, x_i is interesting iff y_i is interesting, and x_i is big iff y_i is big. The same is true for x_j and y_j. We assume that $x_i < x_j$ and $y_i < y_j$.

1. The case where x_i, x_j, y_i, y_j are all small is trivial because of invariant 2.
2. Assume that x_i and y_i are small and x_j and y_j are interesting. Then the positions x_i in u_r^+ and y_i in u_r^- have the same value. By Lemma 8 it holds that $|x_i - x_j| > k|\bar{\alpha}|$ and $|y_i - y_j| > k|\bar{\alpha}|$ or $x_i - x_j = y_i - y_j$. If $x_i - x_j = y_i - y_j$ then we are done. Otherwise it follows from Lemma 9 and invariant 4 of the invariant that x_i and x_j have the same value iff y_i and y_j have the same value.
3. Now consider the case where x_i and y_i are small and x_j and y_j are big. In this case $x_j - x_i \geq k$ because by definition $|\bar{\alpha}| > 1$ and $\bar{\alpha}$ is iterated at least k times when constructing $\bar{\gamma}_r^+$. The same argument show that $y_j - y_i \geq k$. As u_r^+ and u_r^- are free words, either x_j and x_i are on positions with different values and y_j and y_i are on positions with different values, or there are interesting position with the same values as position x_j and y_j have in their respective words. In the first case we are done and we have dealt with the second case under 2.
4. The argument for the case where x_i, x_j, y_i, y_j are all interesting is the same as under 2. above.
5. In case that x_i and y_i are interesting and x_j and y_j are big we are done if $x_i - x_j = y_i - y_j$. Otherwise it follows from Lemma 8 that $|x_i - x_j| > k|\bar{\alpha}|$ and $|y_i - y_j| > k|\bar{\alpha}|$. Then it follows from Lemma 9 and invariant 4 that x_i and x_j have the same value iff y_i and y_j have the same value.
6. The case that x_i, x_j, y_i, y_j are all big is trivial because of invariant 3. □