Automata vs. Logics on Data Words

Michael Benedikt, Clemens Ley, Gabriele Puppis

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

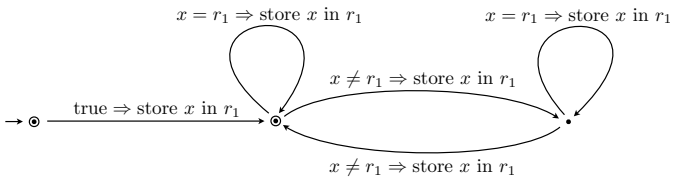

Fig. 1. A finite-memory automaton.

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?

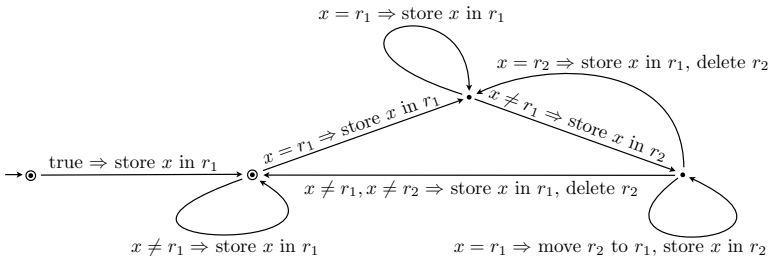

Fig. 2. A finite-memory automaton recognizing a first-order definable language.

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 [11].

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 [12]. A Myhill-Neorde style characterization of the languages that are accepted by DMA was given by Francez and Kaminski in [13]. They state as an open question whether one can compute, given a DMA, a minimal DMA that accepts the same language. In [14] 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 [14], 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 ([14]). *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 $l \in \mathbb{N}$. The proof of the following Proposition can be found in [11].

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.*

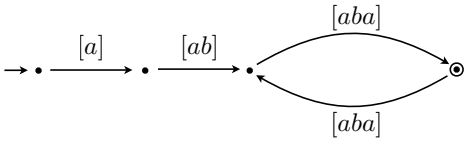

Fig. 3. A DWA that recognizes the language $L = \{aba \dots ba \in D^* \mid a \neq b\}$.

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.*

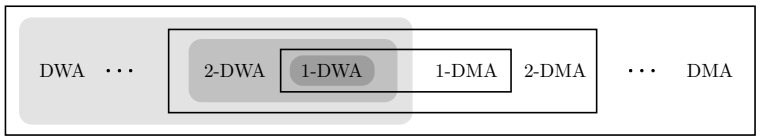

Fig. 4. The inclusions between DMA and DWA.

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

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. FO$(\sim, <)$ is the restriction of MSO$(\sim, <)$ that only uses quantification over first-order variables. An example of an FO$(\sim, <)$ formula is $\forall x, y \ \ (x \sim y \rightarrow x = y)$, 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 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 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 FO$(\sim_{\leq l}, <)$ the logic with the predicates $<$ and $\sim_i$, for all $i \leq l$. For example, the formula $\forall x, y \, \exists z \ \ (x \sim_5 y \rightarrow y \nsim z)$ 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 FO$(\sim_{\leq l}, <)$, but not in FO$(\sim_{\leq l-1}, <)$. Hence the family of logics FO$(\sim_{\leq l}, <)$, where l ranges over $\mathbb{N}$, forms an infinite (strictly increasing) hierarchy. Note also that FO$(\sim, +1)$ can express properties like "the first letter is equal to the last letter", which can not be expressed in 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 NUFO$(\sim, <)$, if for each $n \in \mathbb{N}$, the language $L_N = L \cap D_n^*$ can be defined in FO$(\sim, <)$ (under the assumption that input words are over the finite alphabet D_n). Non-uniform variants of the other logics considered above are defined similarly.

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 FO$(\sim, <)$, but it is definable in NUFO$(\sim, <)$ since $L_{2\times} \cap D_n^*$ is finite for every $n \in \mathbb{N}$.

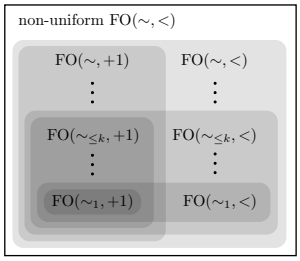

Fig. 5. An overview over some logics for data words.

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.

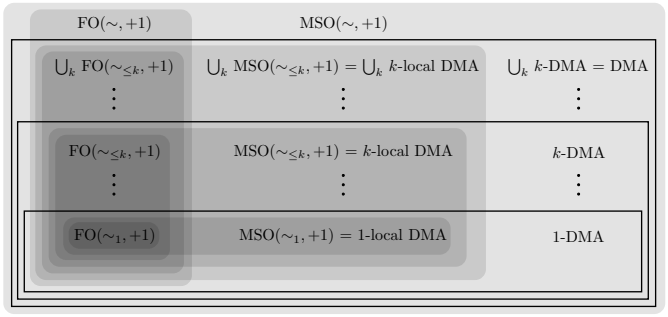

Fig. 6. Comparing the expressive power of automata with that of logics.

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 [11]) 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 [11]).

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 [15]. 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 [15] 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 [11].

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 [11]) 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 [11].

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.

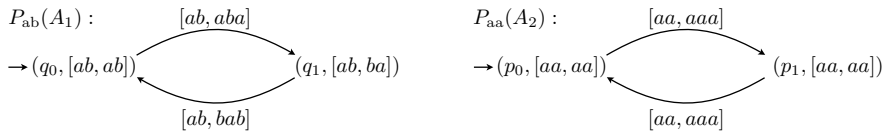

Fig. 8. The parametrized versions of the DWA of Figure 7

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 $\{P_{u,v}(A) \mid u, v \in D^{k-1}\}$.*

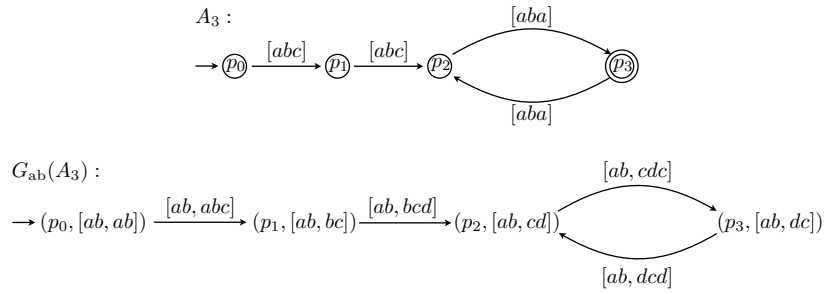

Fig. 9. A FO($\sim, <$) definable DWA and its parameterized version.

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] Benedikt, M., Ley, C., and Puppis, G. (2010), Automata vs. logics on data words. Available at `http://www.comlab.ox.ac.uk/publications/publication3616-abstract.html`.

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

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

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

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