Minimal Memory Automata

Michael Benedikt, Clemens Ley, and Gabriele Puppis

Oxford University Computing Laboratory, Park Rd, Oxford OX13QD UK

Abstract. We provide a Myhill-Nerode-like theorem that characterizes the class of data languages recognized by deterministic finite-memory automata (DMA). As a byproduct of this characterization result, we obtain a canonical representation for any DMA-recognizable language. We then show that this canonical automaton is minimal in a strong sense: it has the minimal number of control states and also the minimal amount of internal storage. We finally show how this minimal automaton can be computed.

1 Introduction

Automata processing words and trees over infinite alphabets 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 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]). Among them, we would like to mention a few interesting categories, which generalize the standard notion of regular language in several respects. *Pebble automata* [5] use special markers to annotate *locations* in a data word. The *data automata* of [1] parse data words in two phases, with one phase applying a finite-state transducer to the input data word and another deciding acceptance on the grounds of a classification of the maximal sub-sequences consisting of the same data values (such a classification is usually specified in terms of membership relationships with suitable regular languages). Of primary interest to us here will be a third category, the *finite memory automata* [4], also called *register automata*, which make use of a finite number of registers in order to store and eventually compare values in the processed data word.

One could hope that most of the fundamental results in standard (i.e., finite-state) automata theory can be carried on in the setting of words over infinite alphabets. However, prior work has shown that many elementary closure and decision properties of finite automata are absent in the infinite-alphabet case. For example, the equivalence of the non-deterministic and deterministic variants of automata is known to fail for both memory automata and pebble automata

[5]. While in the finite case the equivalence and universality problems for nondeterministic automata are decidable, for most of the infinite word models they are not [4, 5].

Among several paradigmatic problems in automata theory, a crucial one, for both theoretical and practical reasons, is certainly the minimization problem. Roughly speaking, it consists of determining the automaton-based representation that uses the "smallest space" for a given language. In the case of standard finite-state automata, minimal space usage is usually translated in terms of the minimum number of states. The well-known Myhill-Nerode theorem [3] gives a canonical automaton for every regular language, which is minimal among deterministic finite automata representing the same language. When dealing with more general models of automata, however, one may need to take into account different complexity measures at the same time, possibly yielding some tradeoffs between the amount of control state and the number of values/locations being stored.

In this paper, we consider minimization for a particular model of register automata, which process finite words over an infinite alphabet. On the one hand, the class of memory automata we are dealing with (DMA, for short) is very similar to that of deterministic finite memory automata introduced in [4]. Our notion of register automaton is slightly more general in allowing to compare values both with respect to a fixed equality relation on values (as in the standard class of finite memory automata) and with respect to a fixed total ordering relation. For instance, our model of register automaton can recognize the language of all strictly-increasing finite sequences of natural numbers, which can not be recognized by a finite memory automaton.

The first contribution of the paper is an isolation of the ideal "minimal storage" for a DMA. This is formalized in terms of the *memorable values* for any word in the language – the set of values that must be stored at any point. Using this we can give a characterization of the class of languages recognized by some DMA, which closely resembles the Myhill-Nerode theorem. Precisely, we associate with each language L a suitable equivalence $\equiv_L$, using the memorable values, and we characterize the class of DMA-recognizable languages as the class of languages L for which $\equiv_L$ has finite index. We remark that a similar result, but restricted to the class of register automata that can only compare values with respect to equality, has already appeared in [2]. In fact, our alternative characterization, besides relating equivalence to space-minimality, holds also for the larger class of DMA that compare values with respect to a fixed arbitrary total ordering relation.

As our second contribution, which stems directly from the previous characterization result, we show that the canonical DMA $\mathcal{A}_L$, which is obtained from a given language L when the corresponding equivalence $\equiv_L$ has finite index, satisfies a strong notion of minimality that takes into account both the number of control states and the number of values stored. Finally, we give an effective means for *minimizing* a DMA, presenting a procedure that begins with an arbitrary DMA and produces the minimal equivalent.

Organization: Section 2 gives preliminaries. Section 3 introduces the notion of memorable value that will be used throughout the paper. Section 4 presents our characterization of DMA-definable languages, along with the results on canonical and minimal automata. Section 5 gives our minimization algorithm, while Section 6 gives conclusions.

2 Preliminaries

From now on, we fix an infinite alphabet D of *values*. A *support* is a relational structure (D, R), where R is a binary relation on D. Through the rest of the paper, we will only consider supports of the form (D, R), where R is either the identity relation $\sim$ or a total order $\prec$. In some sections we will assume that $\prec$ is dense and we will point out this assumption explicitly.

A *(data) word* is a finite sequence consisting of values from the infinite alphabet D. In order to distinguish words up to R-preserving isomorphisms, where R is the binary relation of the underlying support, we introduce the equivalence relation $\simeq_R$ such that, for every $w, w' \in D^*$, $w \simeq_R w'$ whenever $|w| = |w'|$ and $w(i)\ R\ w(j)$ iff $w'(i)\ R\ w'(j)$ for all pairs of positions $1 \leq i, j \leq |w|$. The classes of the equivalence relation $\simeq_R$ are called $\simeq_R$*-types*. Note that the $\simeq_R$-type of a word w of length n can be represented by a first-order formula over the signature $(R, <)$, where $<$ is the word order. We will drop the subscript from $\simeq_R$ whenever the relation R of the underlying support is clear from the context.

A *(data) language* over (D, R) is a (possibly infinite) set of data words over (D, R). Given a language L, we say that two words w, w' are L*-distinct* if one is in L and the other is not, otherwise, we say that w, w' are L*-equivalent* and we shortly write $w =_L w'$ (clearly, $=_L$ is an equivalence relation with at most two classes). From now on, we tacitly assume that languages over the support (D, R) are closed under R-preserving isomorphisms, namely, for every language L over (D, R), we assume that $\simeq_R$ is a refinement of $=_L$. Note that any language L which is closed under R-preserving isomorphisms is also closed under substitutions of value occurrences, namely, for every word $w \in D^*$ and every permutation f of D, we have $w =_L f(w)$.

2.1 Finite-memory automata

In this section, we introduce a variant of Kaminski's finite-memory automata [4] that recognize data languages over supports of the form $(D, \sim)$ or $(D, \prec)$. These automata process data words by storing a bounded number of values into their memory and by comparing them with respect to the binary relation of the underlying support.

Definition 1. *A* (non-deterministic) finite-memory automaton *over a support* (D, R) *is a tuple of the form* $\mathcal{A} = (Q_0, \ldots, Q_k, T, I, F)$*, where*

- k *is the maximum number of stored values;*
- $Q_0, \ldots, Q_k$ *are pairwise disjoint finite sets of control states;*

- *T is a finite set of transition rules of the form (p, α, E, q), where $p \in Q_i$ for some $0 \leq i \leq k$, α is the $\simeq$-type of a word of length $i+1$, $E \subseteq \{1, .., i+1\}$, and $q \in Q_j$, with $j = i + 1 - |E|$;*
- *$I \subseteq Q_0$ is a set of initial states;*
- *$F \subseteq Q_0 \cup \ldots \cup Q_k$ is a set of final states.*

A *configuration* of $\mathcal{A}$ is defined as a pair of the form (q, σ) consisting of a control state $q \in Q_i$, with $0 \leq i \leq k$, and a memory content $\sigma \in D^i$. The meaning of a transition rule of the form (q, α, E, q') is that the automaton can move from a configuration (q, σ) to a configuration (q', σ') by consuming an input value a iff the word $\sigma \cdot a$ has $\simeq$-type α and σ' is obtained from $\sigma \cdot a$ by removing all positions in E.

We enforce two sanity conditions to every transition rule (q, α, E, q'). To guarantee that the length of the target memory content σ' never exceeds k, we assume that E is non-empty whenever $q \in Q_k$. Second, the memory is updated like a stack: if the $\simeq$-type α is of the form $[\sigma \cdot a]_{\simeq}$, with $\sigma(j) = a$ for some $1 \leq j \leq |\sigma|$, then E contains the index j. This has two advantages: The memory content σ' always contains pairwise distinct elements and the order of the data values in the memory is the order of their last occurrences in the input word. We will exploit the latter property when we show that for every FMA language L there is a canonical FMA recognizing L.

A *run* of $\mathcal{A}$ is defined in the usual way. If w is a data word and $\mathcal{A}$ has a run on w from a configuration (q, σ) to a configuration (q', σ'), then we write

$$(q, \sigma) \xrightarrow[\mathcal{A}]{w} (q', \sigma').$$

The language recognized by $\mathcal{A}$, denoted $\mathscr{L}(\mathcal{A})$, is the set of all words w such that $(q, \varepsilon) \xrightarrow[\mathcal{A}]{w} (q', \sigma')$, for some $q \in I$ and some $q' \in F$.

We say that a finite-memory automaton $\mathcal{A} = (Q_0, \ldots, Q_k, T, I, F)$ is *deterministic* if the set of initial states I is a singleton and there is no pair of transitions $(p, \alpha, E, q), (p, \alpha, E', q') \in T$, with either $q \neq q'$ or $E \neq E'$. Similarly, $\mathcal{A}$ is said to be *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 (q, α, E, q'). By a slight abuse of terminology, we abbreviate any *deterministic and complete* finite-memory automaton by *DMA*.

Our model of finite-memory automata is very similar the model of finite-memory automata introduced in [4]. There are several distinguishing elements though. The main difference is that while in the original model the number of registers is fixed throughout the run, the number of stored values can vary in our model. This flexibility will allow us to track space consumption more finely. In particular, our definition allows automata that are canonical in a strong sense, in that they store only the values that are essential for an automaton – the number of such values may vary with the input word. A second distinction is that the original model has an initial register assignment while the memory content is initially empty in our model. In addition, only the support $(D, \sim)$

has been considered previously. It should be pointed out that, over $(D, \sim)$, all models have the same expressive power (provided that, in the original model, all registers are initialized with a dummy value $\perp \notin D$).

3 Memorable Symbols

Given a DMA-recognizable language L and a prefix w of an input word, there exist some values in w that need to be stored by *any* DMA that recognizes L. We will call these values memorable. As an example, consider the language $L = \{xyzy : x < y < z\}$ over the support $(\mathbb{Q}, <)$ and the word $w = 123$. Observe that, after parsing w, any DMA $\mathcal{A}$ that recognizes L must be storing the value 2: otherwise $\mathcal{A}$ could not distinguish the two possible continuations $u = 2$ and $v = 2.5$ (this is necessary since $w \cdot u \not\equiv_L w \cdot v$). For this reason, we will define 2 to be memorable in w with respect to the language L.

We first give the definition of a memorable value in the case where the relation of the underlying support is either the identity or a *dense* total order. Later, we will consider the slightly more involved case of a non-dense total order.

Definition 2. *Let L be a language over (D, R), where R is either the identity or a dense total order. A value a is* L-memorable *in a word w if a occurs in w and there exists a word u and a value b such that*

$$\begin{cases} w \cdot u & \simeq_R & (w \cdot u)[a/b] \\ w \cdot u & \not\equiv_L & w \cdot u[a/b]. \end{cases}$$

Here $u[a/b]$ denotes the word obtained from u by replacing each occurrence of a with b. Note that it follows from the definition that b does not appear in any of w, u, and that a does appear in u.

It is convenient to fix a string-based representation for the set of L-memorable values of a word w. We thus denote by $\mathsf{mem}_L(w)$ the finite sequence that consists of all L-memorable values of w ordered according to the positions of their last occurrences in w (recall that every L-memorable value of w must occur at least once in w).

The following proposition (whose proof can be found in the Appendix) makes the intuition precise that any DMA has to store the memorable values of the input word. That is, if a DMA $\mathcal{A}$ reaches a configuration (q, σ) after reading a word w, then $\mathsf{mem}_L(w)$ must be a sub-sequence of σ.

Proposition 1. *Let $\mathcal{A}$ be a DMA over (D, R), where R is either the identify or a dense total order on D and let $L = L(\mathcal{A})$. Then, for every word w, $\mathsf{mem}_L(w)$ is a sub-sequence of the stored values of $\mathcal{A}$ after reading w. Moreover, if (q, σ) and (q', σ') are the configurations reached by $\mathcal{A}$ after reading words w and w', respectively, then*

$$\begin{cases} q & = & q' \\ \sigma & \simeq_R & \sigma' \end{cases} \qquad \textit{implies} \qquad \mathsf{mem}_L(w) \cdot \sigma \simeq_R \mathsf{mem}_L(w') \cdot \sigma'.$$

Hence every DMA must store the memorable values of an input word. We will show in Section 4 that there is a DMA that does not need to store more than the memorable values.

Intuitively, the next proposition shows that, if two words u and v are isomorphic with respect to the L-memorable values of a word w, then L can not distinguish between $w{\cdot}u$ and $w{\cdot}v$. Again, the proof can be found in the Appendix.

Proposition 2. *Let L be a language over (D, R), where R is a either the identity or a dense total order on D. Then, for all words w, u, v we have*

$$\mathsf{mem}_L(w) \cdot u \ \simeq_R \ \mathsf{mem}_L(w) \cdot v \qquad \textit{implies} \qquad w \cdot u \ =_L \ w \cdot v.$$

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

$$w \cdot a \ \simeq_\prec \ \tilde{w} \cdot \tilde{a} \ \simeq_\prec \ \tilde{w} \cdot \tilde{b}.$$

In particular, this implies that $\tilde{a}$ can always be substituted with $\tilde{b}$ in $\tilde{w}$ without changing the resulting $\simeq_\prec$-type. The following definition is the natural generalization of Definition 2.

Definition 3. *Let L be a language over support $(D, \prec)$, where $\prec$ is a total (possibly not dense) order. A value a is* L-memorable *in a word w if a occurs in w and there exist two words $\tilde{w}, \tilde{u}$ and two values $\tilde{a}, \tilde{b}$ such that*

$$\begin{cases} w \cdot a \ \simeq_\prec \ \tilde{w} \cdot \tilde{a} \\ \tilde{w} \cdot \tilde{u} \ \simeq_\prec \ (\tilde{w} \cdot \tilde{u})[\tilde{a}/\tilde{b}] \\ \tilde{w} \cdot \tilde{u} \ \neq_L \ \tilde{w} \cdot \tilde{u}[\tilde{a}/\tilde{b}]. \end{cases}$$

As an example, given $L = \{xyzy : xy < z\}$ over the support $(\mathbb{N}, <)$, we now have that $a = 2$ is L-memorable in $w = 123$, since there exist $\tilde{w} = 246$, $\tilde{u} = 4$, $\tilde{a} = 4$, and $\tilde{b} = 5$ such that (i) $w \cdot a \ \simeq_< \ \tilde{w} \cdot \tilde{a}$, (ii) $\tilde{w} \cdot \tilde{u} \ \simeq_< \ (\tilde{w} \cdot \tilde{u})[\tilde{a}/\tilde{b}]$, and (iii) $\tilde{w} \cdot \tilde{u} \ \neq_L \ \tilde{w} \cdot \tilde{u}[\tilde{a}/\tilde{b}]$.

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

4 Myhill-Nerode for Data Languages and Minimal Automata

This section is devoted to prove the main result that characterizes the class of DMA-recognizable languages. This result also shows that these languages have automata that are minimal in a strong sense: they have minimal number of control states and they store only the things that they must store, namely, the memorable values.

We begin by associating with each language L a new equivalence relation $\equiv_L$, which is finer than $=_L$, but coarser than $\simeq$. In a similar way, we associate with each DMA $\mathcal{A}$ a corresponding equivalence relation $\equiv_{\mathcal{A}}$.

Definition 4. *Given a language L over the support (D, R), we define $\equiv_L \subseteq D^* \times D^*$ by letting $w \equiv_L w'$ iff*

- $\mathsf{mem}_L(w) \simeq_R \mathsf{mem}_L(w')$,
- *for all words u, u' if $\mathsf{mem}_L(w) \cdot u \simeq_R \mathsf{mem}_L(w') \cdot u'$ then $w \cdot u =_L w' \cdot u'$.*

Definition 5. *Given a DMA $\mathcal{A}$ over the support (D, R), we define $\equiv_{\mathcal{A}} \subseteq D^* \times D^*$ by letting $w \equiv_{\mathcal{A}} w'$ iff, whenever $\mathcal{A}$ reaches the configurations (q, σ) and (q', σ') by reading w and w', respectively, then $q = q'$ and $\sigma \simeq_R \sigma'$ follow.*

It is easy to see that both $\equiv_L$ and $\equiv_{\mathcal{A}}$ are equivalence relations. In fact, $\equiv_L$ is also a congruence with respect to concatenation of words to the right, namely, $w \equiv_L w'$ implies $w \cdot u \equiv_L w' \cdot u$, under the assumption that $\mathsf{mem}_L(w) = \mathsf{mem}_L(w')$. Moreover, it is easy to see that, given a DMA $\mathcal{A}$ having n control states and storing at most k values, the corresponding equivalence $\equiv_{\mathcal{A}}$ has index at most $n \cdot k!$ (indeed, the $\equiv_{\mathcal{A}}$-equivalence class of any word w is uniquely determined by the control state q and by the $\simeq$-type of the register assignment σ of the configuration (q, σ) that is reached by $\mathcal{A}$ after reading w). If the underlying support contains only the identity relation $\sim$, then the upper bound for the number of $\equiv_{\mathcal{A}}$-equivalence classes drops down to n.

We are now ready to state the main characterization result.

Theorem 1. *Let L be a language over (D, R), where R is either the identity or a total order on D. Then, L is DMA-recognizable iff $\equiv_L$ has finite index.*

We briefly summarize the key ingredients of the proof of Theorem 1 (which is given in the Appendix). The left-to-right-direction is proved by assuming that L is recognized by a DMA $\mathcal{A}$ and by exploiting Proposition 1 in order to show that the corresponding equivalence relation $\equiv_{\mathcal{A}}$ refines $\equiv_L$ (from previous arguments it then follows that $\equiv_L$ has finite index). The converse direction is proved by assuming that $\equiv_L$ has finite index and by building a finite-memory automaton $\mathcal{A}_L$, called canonical automaton. Below, we give a formal definition of such an automaton. The fact that $\mathcal{A}_L$ is deterministic and complete follows from Proposition 2.

Definition 6. *Let L be a language over (D, R), where R is either the identity or a total order on D. If $\equiv_L$ has finite index, then we define the* canonical automaton *for L as the DMA $\mathcal{A}_L = (Q_0, ..., Q_k, T, \{q_I\}, F)$, where*

- *k is the maximum length of sequences of the form $\mathsf{mem}_L(w)$, with $w \in D^*$;*
- *for every $0 \leq i \leq k$, Q_i is the set of all $\equiv_L$-equivalence classes of the form $[w]_{\equiv_L}$, with $w \in D^*$ and $|\mathsf{mem}_L(w)| = i$;*
- *T is the set all transition rules of the form $\big([w]_{\equiv_L}, \alpha, E, [w \cdot a]_{\equiv_L}\big)$, with $w \in D^*$, $a \in D$, $\alpha = [\mathsf{mem}_L(w) \cdot a]_{\simeq}$, and $E \subseteq \{1, ..., |\mathsf{mem}_L(w)| + 1\}$ such that the sub-sequence obtained from $\mathsf{mem}_L(w) \cdot a$ by removing all positions of E coincides with the sequence $\mathsf{mem}_L(w \cdot a)$;*
- *q_I is the $\equiv_L$-equivalence class of the empty word ε;*
- *F is the set of all $\equiv_L$-equivalence classes of words $w \in L$.*

Minimal DMA. We now prove that the canonical automaton for a given language L is minimal among all equivalent DMA recognizing L. Here, we adopt a general notion of minimality for DMA that takes into account both the number of control states and the number of stored values on each input word. Precisely, we say that a DMA $\mathcal{A} = (Q_0, ..., Q_k, T, \{q_I\}, F)$ is *state-minimal* if for every equivalent DMA $\mathcal{A}' = (Q'_0, ..., Q'_{k'}, T', \{q'_I\}, F')$ that recognizes the same language, we have

$$|Q| = \sum_{0 \leq i \leq k} |Q_i| \quad \leq \quad \sum_{0 \leq i \leq k'} |Q'_i| = |Q'|.$$

Similarly, we say that $\mathcal{A}$ is *data-minimal* if, for every equivalent DMA $\mathcal{A}' = (Q'_0, ..., Q'_{k'}, T', \{q'_I\}, F')$ that recognizes the same language and every input word w, we have

$$\begin{cases} (q_I, \varepsilon) \xrightarrow[\mathcal{A}]{w} (q, \sigma) \\ (q'_I, \varepsilon) \xrightarrow[\mathcal{A}']{w} (q', \sigma') \end{cases} \qquad \text{implies} \qquad |\sigma| \leq |\sigma'|.$$

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

Theorem 2. *The canonical automaton $\mathcal{A}_L$ for a given DRA-recognizable language L is minimal.*

We conclude the section by proving that minimal DMA, and in particular canonical automata, are unique up to isomorphisms. Here we think of each DMA $\mathcal{A} = (Q_0, ..., Q_k, T, \{q_I\}, F)$ as a finite directed graph, whose vertices are labeled by indices $i \in \{0, ..., k\}$ and represent control states in Q_i and whose edges are labeled by pairs (α, E) and represent transitions of the form (q, α, E, q')). The proof of the following result is given in the Appendix.

Corollary 1. *Any minimal DMA recognizing a language L is isomorphic to the canonical automaton for L.*

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

input: a DMA $\mathcal{A}$ that accepts a language L over $(D, \sim)$
output: the canonical automaton $\mathcal{A}_L$ for L

let k = maximal number of values stored by $\mathcal{A}$
let n = number of control states of $\mathcal{A}$
let Σ = any subset of D of size $k+1$

$$\left.\begin{array}{l} \textbf{let } Q_1 \leftarrow \varnothing, \ldots, Q_k \leftarrow \varnothing \\ \textbf{for all } w \in \Sigma^{\leq n} \\ \quad \textbf{do} \begin{cases} \textbf{if } w \not\equiv_L w' \textbf{ for all } w' \in \bigcup_{i \leq k} Q_i \\ \quad \textbf{then} \begin{cases} i \leftarrow |\mathsf{mem}_L(w)| \\ Q_i \leftarrow Q_i \cup \{w\} \end{cases} \end{cases} \end{array}\right\} \textit{Compute the sets of states}$$

$$\left.\begin{array}{l} T \leftarrow \varnothing \\ \textbf{for all } w, w' \in Q_0 \cup \ldots \cup Q_k \textbf{ and for all } a \in \Sigma \\ \quad \textbf{do} \begin{cases} \textbf{if } w \cdot a \equiv_L w' \\ \quad \textbf{then} \begin{cases} \alpha \leftarrow [\mathsf{mem}_L(w) \cdot a]_{\simeq} \\ E \leftarrow \{i : \forall\, j.\ \mathsf{mem}_L(w \cdot a)(j) \neq (\mathsf{mem}_L(w) \cdot a)(i)\} \\ T \leftarrow T \cup \{(w, \alpha, E, w')\} \end{cases} \end{cases} \end{array}\right\} \textit{Compute the set of transitions}$$

return $\mathcal{A}_L = (Q_0, \ldots, Q_k, T, \{w \in Q_0 : w \equiv_L \varepsilon\}, \{w \in Q_0 \cup \ldots \cup Q_k : w \in L\})$

Fig. 1. Minimization algorithm for DMA over the support $(D, \sim)$.

5 Computing Minimal DMA Over $(D, \sim)$

In this section, we focus on the problem of computing the minimal DMA for a given language over $(D, \sim)$. That is, given a DMA $\mathcal{A}$ that recognizes L over $(D, \sim)$, we show how to compute the canonical automaton $\mathcal{A}_L$ for L. This is sufficient as by Theorem 2, the canonical automaton $\mathcal{A}_L$ is minimal among all equivalent DMA.

Algorithm 1 provides the pseudo-code of a procedure that receives a DMA $\mathcal{A}$ recognizing L as input and computes (a representation of) the corresponding canonical automaton $\mathcal{A}_L$. Such a procedure consists of two main loops: the first one computes a minimal and complete set $Q = Q_0 \cup \ldots \cup Q_k$ of representatives of $\equiv_L$-equivalence classes, which are then identified with the control states of the canonical automaton $\mathcal{A}_L$; the second loop computes the set of transition rules of $\mathcal{A}_L$.

We now show that Algorithm 1 computes the minimal automaton for the given DMA $\mathcal{A}$. Assume that $\mathcal{A}$ has n states and stores at most k values. We first show that at the end of the computation $Q = Q_0 \cup \ldots \cup Q_k$ contains exactly one representative for each $\equiv_L$-equivalence class. We claim that $\Sigma^{\leq n}$ contains at least one representative for each equivalence class of $\equiv_L$: observe that each control state q in $\mathcal{A}$ can be reached by a word in $\Sigma^{\leq n}$ (this is the case because

$|\Sigma| > k$ – see the proof of Lemma 1 for a similar argument). By the proof of Theorem 1 we know that $\equiv_{\mathcal{A}}$ refines $\equiv_L$, and hence $\Sigma^{\leq n}$ contains at least one representative of each $\equiv_L$-equivalence class. It is clear that, after the execution of the first loop of the procedure, the set Q contains pairwise non-$\equiv_L$-equivalent representants of all $\equiv_L$-equivalence classes. Hence Q contains exactly one word in each $\equiv_L$-equivalence class. It is clear from Definition 6 that the set T contains exactly the transitions of $\mathcal{A}$.

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

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

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

Lemma 1. *Let $\mathcal{A}$ be a DMA over $(D, \sim)$ that stores at most k values and let $L = \mathscr{L}(\mathcal{A})$. Let w, w' be two words and let Δ be a finite subset of D containing all values occurring in w or in w', plus $2k+1$ additional values. For every $u \in D^*$, there is $v \in \Delta^*$ such that*

$$\begin{cases} w \cdot u =_L w \cdot v \\ w' \cdot u =_L w' \cdot v. \end{cases}$$

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

Proposition 3. *The memorability problem is decidable in non-deterministic single-exponential time $\mathcal{O}(n^2 \cdot m^{2k} \cdot k^{2k})$, where m is the length of the input word w, n is the number of control states of the input DMA $\mathcal{A}$ and k is maximum number of values stored by $\mathcal{A}$.*

Proposition 4. *The word-equivalence problem is decidable in non-deterministic single-exponential time $\mathcal{O}(n^2 \cdot m^{2k} \cdot k^{2k})$, where m is the sum of the lengths of the two input words w and w', n is the number of control states of the input DMA $\mathcal{A}$ and k is maximum number of values stored by $\mathcal{A}$.*

6 Conclusion

Some problems still remain open. In particular, the most natural question is whether a minimization procedure, similar to the one described in Section 5, can be given also in the case where the alphabet is equipped with a total order. Moreover, it would be interesting to see whether or not analogous characterization results (and, possibly, a corresponding minimization procedure) can be given in the case of DMA-recognizable languages over an infinite alphabet

equipped with a *partial order*. This would enable us to consider supports of the form $(\mathscr{P}(D), \subset)$, where $\mathscr{P}(D)$ is the powerset of an infinite domain D and $\subset$ is the containment relation between subsets of D. Finally, more general models of automata could be taken into account, including, for instance, automata that process sequences of database instances and, possibly, use more powerful policies for updating their memory.

Bibliography

[1] Mikolaj Bojanczyk, Anca Muscholl, Thomas Schwentick, Luc Segoufin, and Claire David. Two-variable logic on words with data. In *Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science*, pages 7–16, Washington, DC, USA, 2006. IEEE Computer Society.

[2] Nissim Francez and Michael Kaminski. An algebraic characterization of deterministic regular languages over infinite alphabets. *Theoretical Computer Science*, 306(1-3):155–175, 2003.

[3] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

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

[5] Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Logic*, 5(3):403–435, 2004.

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

[7] Luc Segoufin. Automata and logics for words and trees over an infinite alphabet. In *Proceedings of the 15th Annual Conference of the EACLS, 20th International Workshop on Computer Science Logic*, volume 4207 of *Lecture Notes in Computer Science*, pages 41–57, Szeged, Hungary, 2006. Springer.

A Appendix

A.1 Proof of Proposition 1

Proposition 1. *Let $\mathcal{A}$ be a DMA over (D, R), where R is either the identity or a dense total order on D, and let L be the language accepted by $\mathcal{A}$. Then, for every word w, we have that $\mathsf{mem}_L(w)$ is a sub-sequence of the stored values of $\mathcal{A}$ after reading w. Moreover, if (q, σ) and (q', σ') are the configurations reached by $\mathcal{A}$ after reading words w and w', respectively, then*

$$\begin{cases} q & = & q' \\ \sigma & \simeq_R & \sigma' \end{cases} \qquad \textit{implies} \qquad \mathsf{mem}_L(w) \cdot \sigma \simeq \mathsf{mem}_L(w') \cdot \sigma'.$$

Proof. We first prove that $\mathsf{mem}_L(w)$ is a sub-sequence of the memory content σ of $\mathcal{A}$ after reading w. As in the proof of the previous proposition, it is sufficient to prove that every L-memorable value of w occurs in the memory content σ (indeed, by definition of transition relation of $\mathcal{A}$, the values in σ, exactly as those in $\mathsf{mem}_L(w)$, are ordered according to the positions of their last occurrences in w). We prove the claim by contraposition, namely, we fix a value a that occurs in w, but not in the memory content σ, and we prove that a is not L-memorable for w (that is for every word u and every value b, $w \cdot u \simeq_R (w \cdot u)[a/b]$ implies $w \cdot u =_L w \cdot u[a/b]$). Let us then assume that (i) a occurs in w but not in σ, (ii) $w \cdot u \simeq_R (w \cdot u)[a/b]$, and (iii) $w \cdot u \in L$ (the dual case $w \cdot u \notin L$ can be dealt with by similar arguments). We have to prove that $w \cdot u[a/b] \in L$ holds as well. Since $w \cdot u \in L$, we know that $\mathcal{A}$ admits a run on u of the form $(q_0, \sigma_0), \ldots, (q_n, \sigma_n)$, with $(q_0, \sigma_0) = (q, \sigma)$ and q_n being a final control state. Moreover, since a does not occur in the memory content σ and since b occurs neither in w nor in u (otherwise, we would have $w \cdot u \not\simeq w \cdot u[a/b]$), we know that $\mathcal{A}$ admits a run on $u[a/b]$ of the form $(q_0, \sigma_0[a/b]), \ldots, (q_n, \sigma_n[a/b])$, with $(q_0, \sigma_0[a/b]) = (q, \sigma)$. This shows that $w \cdot u[a/b]$ is accepted by $\mathcal{A}$ as well. Hence a is not L-memorable in w.

We now prove the second claim of the proposition. Let us consider two configurations (q, σ) and (q', σ') that are reached by $\mathcal{A}$ after reading two words w and w', respectively, and such that $q = q'$ and $\sigma \simeq_R \sigma'$. We define $w'' = w'[\sigma'(i)/\sigma(i)]_{1 \leq i \leq |\sigma|}$ (note that $w'' \simeq_R w'$). By exploiting arguments similar to above ones, one can show that $\mathcal{A}$ reaches the configuration (q, σ) after reading w''. This implies that $w \cdot u =_L w'' \cdot u$ for every word u and hence $\mathsf{mem}_L(w)$ coincides with $\mathsf{mem}_L(w'')$. Moreover, by construction, we have $\mathsf{mem}_L(w'') = \mathsf{mem}_L(w')[\sigma'(i)/\sigma(i)]_{1 \leq i \leq |\sigma|}$. We can then conclude that $\sigma \cdot \mathsf{mem}_L(w) = \sigma \cdot \mathsf{mem}_L(w'') \simeq \sigma' \cdot \mathsf{mem}_L(w')$. □

A.2 Proof of Proposition 2

Proposition 2. *Let L be a language over (D, R), where R is a either the identity or a dense total order on D. Then, for all words w, u, v we have*

$$\mathsf{mem}_L(w) \cdot u \ \simeq_R \ \mathsf{mem}_L(w) \cdot v \qquad \textit{implies} \qquad w \cdot u \ =_L \ w \cdot v.$$

Proof. We only prove the case where R is a total and dense order. The case where R is the identity on D is similar and slightly simpler. Suppose that $\mathsf{mem}_L(w) \cdot u \simeq \mathsf{mem}_L(w) \cdot v$ (we will drop the subscript $\prec$ within this proof). We prove that $w \cdot u =_L w \cdot v$ holds by exploiting a double induction on two parameters: the first, dominant, parameter is the number of positions on which u and v have different values; the second parameter takes into account the number of values c that occur in w such that $c \cdot u \not\simeq c \cdot v$.

We begin by considering the set I of all positions where u and v differ. Precisely, we let $I = \{i : 1 \leq i \leq |u|, u(i) \not\sim v(i)\}$ (this is well defined because $|u| = |v|$), $\{a_1, \ldots, a_n\} = \{u(i) : i \in I\}$ such that $a_1 \prec \ldots \prec a_n$, and $\{b_1, \ldots, b_n\} = \{u(i) : i \in I\}$ such that $b_1 \prec \ldots \prec b_n$. None of the values $a_1, b_1, ..., a_n, b_n$ are L-memorable in w because of $\mathsf{mem}_L(w) \cdot u \simeq \mathsf{mem}_L(w) \cdot v$. We define a sequence $(u_0, v_0), \ldots, (u_n, v_n)$ by $(u_0, v_0) := (u, v)$ and for all $1 \leq i \leq n$ if $a_i \prec b_i$ then $u_i := u_{i-1}$ and $v_i := v_{i-1}[b_i/a_i]$ and if $a_i \succ b_i$ then $u_i := u_{i-1}[a_i/b_i]$ and $v_i := v_{i-1}$. Note that $u_n = v_n$ and $\mathsf{mem}_L(w) \cdot u_{i-1} \simeq \mathsf{mem}_L(w) \cdot u_i$ and $\mathsf{mem}_L(w) \cdot v_{i-1} \simeq \mathsf{mem}_L(w) \cdot v_i$ for all indices $1 \leq i \leq n$. As $a_1, b_1, \ldots, a_n, b_n$ are not L-memorable in w it is sufficient to show the following:

Claim. *For every word u' and every pair of values a, b, we have that*

$$\mathsf{mem}_L(w) \cdot u' \ \simeq \ \mathsf{mem}_L(w) \cdot u'[a/b] \qquad \text{implies} \qquad w \cdot u' \ =_L \ w \cdot u'[a/b].$$

We assume that $\mathsf{mem}_L(w) \cdot u' \simeq \mathsf{mem}_L(w) \cdot u'[a/b]$ and that $a \prec b$ (the other case is symmetric). Let $\{c_1, \ldots, c_m\}$ be the set of values that occur in w such that $a \preceq c \preceq b$ and $c_1 \prec \ldots \prec c_m$. Note that none of these values is L-memorable in w. We will prove the claim by induction on m.

- (Base case $m = 0$) If there is no value c that occurs in w and that satisfies $a \preceq c \preceq b$ then $w \cdot u' \simeq w \cdot u'[a/b]$. Since L is closed under isomorphism we obtain $w \cdot u' =_L w \cdot u'[a/b]$.
- (Induction step $m > 0$) We distinguish two sub-cases:
  1. (case $a \sim c_1$) As $\prec$ is dense, there is a fresh value d such that $a \prec d$ and $w \cdot u' \simeq (w \cdot u')[a/d]$. Moreover, since a is not L-memorable in w, we have $w \cdot u' =_L w \cdot u'[a/d]$. Now let m' be the number of values c' that occur in w and that satisfy $d \preceq c' \preceq b$. Since $c_1 = a \prec d$, we know that $m' < m$. As d and b are not memorable for w $\mathsf{mem}_L(w) \cdot u'[a/d] \simeq \mathsf{mem}_L(w) \cdot u'[a/d][d/b]$. Hence, by inductive hypothesis, $w \cdot u'[a/d] =_L w \cdot u'[a/d][d/b]$.

2. (case $a \prec c_1$) Again exploiting that $\prec$ is dense, there must be a value d such that $a \prec d \prec c_1$, $w \cdot u' \simeq (w \cdot u')[a/d]$, and $(w \cdot u')[a/d] \simeq (w \cdot u')[a/c_1]$. As a is not memorable for w, $w \cdot u' =_L w \cdot u'[a/d]$. Similarly $(w \cdot u')[a/d] =_L w \cdot u'[a/c_1]$ because c_1 is not memorable for w. Hence $w \cdot u' =_L w \cdot u'[a/c_1]$. We can use the same argument as in case 1. to show that $w \cdot u'[a/c_1] =_L w \cdot u'[a/b]$. □

A.3 Proof of Theorem 1

Theorem 1. *Let L be a language over (D, R), where R is either the identity or a total order on D. Then, L is DMA-recognizable iff $\equiv_L$ has finite index.*

Lemma 2. *Let L be a language over (D, R), where R is either a dense total order or the identity on D. Then, for every word w and every value a, we have that $\mathsf{mem}_L(w \cdot a)$ is a sub-sequence of $\mathsf{mem}_L(w) \cdot a$.*

Proof. We show that every L-memorable value of $w \cdot a$ is either an L-memorable value of w or it coincides with a (the convention that the values in $\mathsf{mem}_L(w)$ are ordered according to the positions of their last occurrences in w will then imply the claim of the proposition). Suppose, by contraposition, that there is a value b that is L-memorable in $w \cdot a$, but not in w. Since b is L-memorable in $w \cdot a$, we know there exist a word u and a value c such that

$$\begin{cases} w \cdot a \cdot u \;\simeq_R\; (w \cdot a \cdot u)[b/c] \\ w \cdot a \cdot u \;\not\simeq_L\; w \cdot a \cdot u[b/c]. \end{cases}$$

Similarly, since b is not L-memorable in w, then we know that, for every word v and every value d, we have

$$\Big\{ w \cdot v \;\simeq_R\; (w \cdot v)[b/d] \qquad \text{implies} \qquad w \cdot v \;=_L\; w \cdot v[b/d].$$

In particular, by letting $v = a \cdot u$ and $d = c$, we obtain

$$w \cdot a \cdot u \;=\; w \cdot v \;=_L\; w \cdot v[b/d] \;=\; w \cdot (a \cdot u)[b/c].$$

Together with $w \cdot a \cdot u \not\equiv_L w \cdot a \cdot u[b/c]$, this can only be true if $a = b$. □

Proof (of Theorem 1). Let us first consider the direction from left to right. Let $\mathcal{A} = (Q_0, ..., Q_k, T, \{q_I\}, F)$ be a DMA that recognizes the language L. We prove that $\equiv_L$ has finite index by showing that $\equiv_L$ is refined by $\equiv_{\mathcal{A}}$ (we have already argued that the index of $\equiv_{\mathcal{A}}$ has size at most $|Q| \cdot k!$). Let us fix two words w and w' such that $w \equiv_{\mathcal{A}} w'$ and let (q, r) and (q', r') be the configurations reached by $\mathcal{A}$ after reading w and w', respectively. Since $w \equiv_{\mathcal{A}} w'$, we know that $q = q'$ and $r \simeq r'$. Thus, by Proposition 1, we have $\mathsf{mem}_L(w) \cdot r \simeq \mathsf{mem}_L(w') \cdot r'$ (we will drop R as a subscript in this proof). In particular, this shows that

the first condition of Definition 4 (i.e., $\mathsf{mem}_L(w) \simeq \mathsf{mem}_L(w')$) is satisfied. We now consider two words u, u' such that $\mathsf{mem}_L(w) \cdot u \simeq \mathsf{mem}_L(w') \cdot u'$ and we show that $w \cdot u =_L w' \cdot u'$ follows. For the sake of brevity, we denote by (p, s) the configuration reached by $\mathcal{A}$ after reading $w \cdot u$ and we define $v = u[r(i)/r'(i)]_{1 \leq i \leq |r|}$ and $s' = s[r(i)/r'(i)]_{1 \leq i \leq |r|}$. Since $q' = q$ and $r \cdot u \cdot s \simeq r' \cdot v \cdot s'$, we know that $\mathcal{A}$ reaches the configuration (p, s') after reading the word $w \cdot v$. Summing up, we have:

$$\begin{cases} (q_I, \varepsilon) \xrightarrow[\mathcal{A}]{w} (q, r) \xrightarrow[\mathcal{A}]{u} (p, s) \\ (q_I, \varepsilon) \xrightarrow[\mathcal{A}]{w'} (q', r') \xrightarrow[\mathcal{A}]{v} (p, s'). \end{cases}$$

In particular, since $\mathcal{A}$ recognizes L, we know that $w \cdot u =_L w' \cdot v$. Moreover, by definition of v, we have $\mathsf{mem}_L(w') \cdot v \simeq \mathsf{mem}_L(w) \cdot u$ and, from the previous assumptions, $\mathsf{mem}_L(w) \cdot u \simeq \mathsf{mem}_L(w) \cdot u'$. By exploiting the transitivity of the relation $\simeq$, we obtain $\mathsf{mem}_L(w') \cdot v \simeq \mathsf{mem}_L(w') \cdot u'$ and hence, by Proposition 2, $w' \cdot v =_L w' \cdot u'$ follows. Finally, by transitivity of $=_L$, we conclude that $w \cdot u =_L w' \cdot u'$.

We now prove the opposite direction, namely, we assume that $\equiv_L$ has finite index and we show that there is a DMA $\mathcal{A}_L = (Q_0, ..., Q_k, T, \{q_I\}, F)$ that recognizes L. We first define the maximum number k of active registers and the sets $Q_0, ..., Q_k$ of control states of $\mathcal{A}_L$. Note that for each $\equiv_L$-equivalence class $C = [w]_{\equiv_L}$, with $w \in D^*$, there is a number i_C such that $|\mathsf{mem}_L(w)| = i_C$ (by definition of $\equiv_L$, this number i_C does not depend on the choice of the representant w). We let k be the maximum number i_C, for all $\equiv_L$-equivalence classes C. Then, for each index $0 \leq i \leq k$, we define Q_i as the set of all $\equiv_L$-equivalence classes C such that $i_C = i$. Precisely, we let $Q_i = \{[w]_{\equiv_L} : w \in D^*, \mathsf{mem}_L(w) = i\}$. Accordingly, we let $q_I = [\varepsilon]_{\equiv_L}$ be the initial state and $F = \{[w]_{\equiv_L} : w \in L\}$ be the set of final states of $\mathcal{A}_L$. It remains to define the set T of transition rules of $\mathcal{A}_L$. We let T contain all and only the tuples of the form

$$\left([w]_{\equiv_L}, \alpha, E, [w \cdot a]_{\equiv_L}\right)$$

where $w \in D^*$, $a \in D$, α is the $\simeq$-type of the word $\mathsf{mem}_L(w) \cdot a$, and E is the set of positions of $\mathsf{mem}_L(w) \cdot a$ such that the removal of all positions $i \in E$ from $\mathsf{mem}_L(w) \cdot a$ yields exactly $\mathsf{mem}_L(w \cdot a)$ (note that, by Lemma 2, $\mathsf{mem}_L(w \cdot a)$ is a sub-sequence of $\mathsf{mem}_L(w) \cdot a$ and hence such a set E exists). Clearly, by definition of T, $\mathcal{A}_L$ is complete. It remains to prove that $\mathcal{A}_L$ is deterministic, namely, that for every pair of transition rules in T of the form $\left([w]_{\equiv_L}, [\mathsf{mem}_L(w \cdot a)]_{\simeq}, E, [w \cdot a]_{\equiv_L}\right)$ and $\left([w']_{\equiv_L}, [\mathsf{mem}_L(w' \cdot a')]_{\simeq}, E', [w' \cdot a']_{\equiv_L}\right)$, we have that

$$\begin{cases} w \equiv_L w' \\ \mathsf{mem}_L(w) \cdot a \simeq \mathsf{mem}_L(w') \cdot a' \end{cases} \quad \text{implies} \quad \begin{cases} w \cdot a \equiv_L w' \cdot a' \\ E = E'. \end{cases}$$

Suppose that $w \equiv_L w'$ and $\mathsf{mem}_L(w) \cdot a \simeq \mathsf{mem}_L(w') \cdot a'$ hold.

We first prove that $w \cdot a \equiv_L w' \cdot a'$ holds. By Lemma 2, $\mathsf{mem}_L(w \cdot a)$ is a sub-sequence of $\mathsf{mem}_L(w) \cdot a$ and $\mathsf{mem}_L(w' \cdot a')$ is a sub-sequence of $\mathsf{mem}_L(w') \cdot a'$.

Moreover, by hypothesis, $\mathsf{mem}_L(w)\cdot a \simeq \mathsf{mem}_L(w')\cdot a'$ holds. It thus follows that $\mathsf{mem}_L(w\cdot a) \simeq \mathsf{mem}_L(w'\cdot a')$. This shows that the first condition of Definition 4 is satisfied. As for the second condition, we consider two words u and u' such that $\mathsf{mem}_L(w\cdot a)\cdot u \simeq \mathsf{mem}_L(w'\cdot a')\cdot u'$. We then define v as the word obtained by replacing every occurrence of the value $\mathsf{mem}_L(w')(i)$, with $1 \leq i \leq |\mathsf{mem}_L(w')| = |\mathsf{mem}_L(w)|$, by the corresponding value $\mathsf{mem}_L(w)(i)$ and every occurrence of a' by a. Clearly, by construction and by previous assumptions, we have

$$\mathsf{mem}_L(w\cdot a)\cdot a\cdot u \;\simeq\; \mathsf{mem}_L(w'\cdot a')\cdot a'\cdot u' \;\simeq\; \mathsf{mem}_L(w\cdot a)\cdot a\cdot v.$$

Hence, by Proposition 2, we know that $(w\cdot a)\cdot u =_L (w\cdot a)\cdot v$. Moreover, since $w \equiv_L w'$ and $\mathsf{mem}_L(w)\cdot(a\cdot v) \simeq \mathsf{mem}_L(w)\cdot(a'\cdot u')$, we know that $w\cdot(a\cdot v) =_L w'\cdot(a'\cdot u')$. Therefore, by transitivity, we conclude that $(w\cdot a)\cdot u =_L (w'\cdot a')\cdot u'$. This shows that $w\cdot a \equiv_L w'\cdot a'$.

It remains to show that $E = E'$. We do that by considering an arbitrary position i, with $1 \leq i \leq |\mathsf{mem}_L(w)\cdot a|$ $(= |\mathsf{mem}_L(w')\cdot a'|)$, and by proving that the i-th value of the sequence $\mathsf{mem}_L(w)\cdot a$ is L-memorable in $w\cdot a$ iff the i-th value of the sequence $\mathsf{mem}_L(w')\cdot a'$ is L-memorable in $w'\cdot a'$. For the sake of simplicity, we only consider the case of a densely linearly ordered support (the proof for the more general case is a straightforward adaptation of the following one). Let us fix a position i in $\mathsf{mem}_L(w)\cdot a$ and the two values $b = (\mathsf{mem}_L(w)\cdot a)(i)$ and $b' = (\mathsf{mem}_L(w')\cdot a')(i)$. We assume that b is L-memorable in $w\cdot a$ and we prove that b' is L-memorable in $w'\cdot a'$ (the converse implication is by symmetric arguments). Since b is L-memorable in $w\cdot a$, we know from Definition 2 that there exist a word u and a value c such that $w\cdot a\cdot u \simeq (w\cdot a\cdot u)[b/c]$ and $w\cdot a\cdot u \neq_L w\cdot a\cdot u[b/c]$. We then define u' as the word obtained from u by replacing every occurrence of a value of the form $\mathsf{mem}_L(w\cdot a)(j)$ by the corresponding value $\mathsf{mem}_L(w'\cdot a')(j)$. Clearly, we have $\mathsf{mem}_L(w\cdot a)\cdot u \simeq \mathsf{mem}_L(w'\cdot a')\cdot u'$. In a similar way, we can find two values b' and c' such that $\mathsf{mem}_L(w\cdot a)\cdot u[b/c] \simeq \mathsf{mem}_L(w'\cdot a')\cdot u'[b'/c']$. Now, we recall from previous arguments that the two words $w\cdot a$ and $w'\cdot a'$ are $\equiv_L$-equivalent. This implies that both $(w\cdot a)\cdot u =_L (w'\cdot a')\cdot u'$ and $(w\cdot a)\cdot u[b/c] =_L (w'\cdot a')\cdot u'[b'/c']$ hold. We thus conclude that $w'\cdot a'\cdot u' \simeq (w'\cdot a'\cdot u')[b'/c']$ and $w'\cdot a'\cdot u' \neq_L w'\cdot a'\cdot u'[b'/c']$, which means that b' is L-memorable in $w'\cdot a'$. In this way, we have just shown that

$$\begin{aligned} E &= \left\{i : \forall\, j.\ \mathsf{mem}_L(w\cdot a)(j) \neq (\mathsf{mem}_L(w)\cdot a)(i)\right\} \\ &= \left\{i : \forall\, j.\ \mathsf{mem}_L(w'\cdot a')(j) \neq (\mathsf{mem}_L(w')\cdot a')(i)\right\} \;=\; E'. \end{aligned}$$

Summing up, we have that $\mathcal{A}_L$ is a DMA that recognizes the language L. □

A.4 Proof of Theorem 2

Theorem 2. *The canonical automaton $\mathcal{A}_L$ for a given DRA-recognizable language L is minimal.*

Proof. We first prove that $\mathcal{A}_L$ is state-minimal. Let us consider a DMA $\mathcal{A}'$ that recognizes L. We introduce a function f that maps control states of $\mathcal{A}_L$ to control states of $\mathcal{A}'$, as follows. First, we associate with each control state q of $\mathcal{A}_L$ a representant w_q of the corresponding $\equiv_L$-equivalence class, namely, we assume $q = [w_q]_{\equiv_L}$ for some word $w_q \in D^*$. Then, for each control state q of $\mathcal{A}_L$, we define $f(q)$ as the control state reached by $\mathcal{A}'$ after reading w_q. We now prove that f injective. Let us consider two control states q and q' of $\mathcal{A}_L$ such that $f(q) = f(q')$. By construction, we know that $\mathcal{A}'$ reaches two configurations of the form $(f(q), \sigma)$ and $(f(q'), \sigma')$ by reading w_q and $w_{q'}$, respectively. Since $f(q) = f(q')$, we know that w_q and $w_{q'}$ are in the same $\equiv_{\mathcal{A}'}$-equivalence class. Moreover, by recalling the proof of Theorem 1, we know that $\equiv_{\mathcal{A}'}$ refines $\equiv_L$. This shows that $w_q \equiv_L w_{q'}$ and hence $q = q'$. We just prove that the function f is injective and hence $\mathcal{A}'$ has at least as many control states as $\mathcal{A}_L$.

We now prove that $\mathcal{A}_L$ is data-minimal. Let us fix a DMA $\mathcal{A}'$ that recognizes L and let us consider a generic word w. Suppose that $\mathcal{A}_L$ reaches the configuration (q, σ) by reading w and, similarly, $\mathcal{A}'$ reaches the configuration (q', σ') by reading w. From the proof of Theorem 1, we recall that the stored values σ coincides with the sequence of L-memorable values of w. Moreover, from Proposition 1, we know that $\mathsf{mem}_L(w)$ is a sub-sequence of the stored values σ'. We thus conclude that $|\sigma| \leq |\sigma'|$. □

A.5 Proof of Corollary 1

Corollary 1. *Any minimal DMA recognizing a language L is isomorphic to the canonical automaton for L.*

Proof. Let $\mathcal{A}' = (Q'_0, ..., Q'_{k'}, T', \{q'_I\}, F')$ be a DMA recognizing a language L and let $\mathcal{A}_L = (Q_0, ..., Q_k, T, \{q_I\}, F)$ be the corresponding canonical automaton. The proof follows easily from that of Theorem 2. In particular, we first observe that $k = k'$ follows from the minimality of $\mathcal{A}_L$ and $\mathcal{A}'$. The bijection that maps control states of $\mathcal{A}_L$ to control states of $\mathcal{A}'$ is given by the function f introduced in the first part of the proof of Theorem 2. Note that this function satisfies $f(q) \in Q'_i$ for all $0 \leq i \leq k = k'$ and all $q \in Q_i$. Moreover, since $\mathcal{A}'$ is state-minimal, f must be surjective. Finally, the correspondence between the transitions of $\mathcal{A}_L$ and those of $\mathcal{A}'$ is uniquely determined by the function f, since both $\mathcal{A}_L$ and $\mathcal{A}'$ are deterministic and complete. □

A.6 Proof of Lemma 1

Lemma 1. *Let $\mathcal{A}$ be a DMA over $(D, \sim)$ that stores at most k values and let $L = \mathscr{L}(\mathcal{A})$. Let w, w' be two words and let Δ be a finite subset of D containing all values occurring in w or in w', plus $2k+1$ additional values. For every $u \in D^*$, there is $v \in \Delta^*$ such that*

$$\begin{cases} w \cdot u &=_L w \cdot v \\ w' \cdot u &=_L w' \cdot v. \end{cases}$$

Proof. Let $\mathcal{A}$, L, Δ, w, and w' be as in the claim of the proposition. We first introduce some definitions. The (w, w')-*trace* of a word $u = a_1...a_n$ is the sequence $\bar{\gamma} = (\sigma_0, \sigma'_0)...(\sigma_n, \sigma'_n)$, where, for every index $0 \leq i \leq n$, σ_i is the memory content reached by $\mathcal{A}$ after reading the word $w \cdot (a_1...a_i)$ and σ'_i is the memory content reached by $\mathcal{A}$ after reading the word $w' \cdot (a_1...a_i)$. We also introduce a transformation f on (w, w')-traces, which only depends on w, w', and Δ. Given a word $u = a_1...a_n$ and the corresponding (w, w')-trace $\bar{\gamma} = \gamma_0...\gamma_n$, we let m be the position of the first occurrence in u of a value from $D \setminus \Delta$, we let b be a value from D that occur neither in the memory content σ_{m-1} nor in the memory content σ'_{m-1} (note that such a value exists since $|\Delta| \geq |\sigma_{m-1}| + |\sigma'_{m-1}|$), and we accordingly define

$$f(\bar{\gamma}) \;=\; (\gamma_0...\gamma_{m-1}) \cdot (\gamma_m...\gamma_n)[a_m \nearrow\!\swarrow b]$$

where $(\gamma_m...\gamma_n)[a_m \nearrow\!\swarrow b]$ is the sequence obtained from $\gamma_m...\gamma_n$ by substituting, in every position $i \in \{m, ..., n\}$ and in both memory contents σ_i and σ'_i, every occurrence of a_m by b and, vice versa, every occurrence of b by a_m. By a slight abuse of notation, we also denote by $f(u)$ the word $(a_1...a_{m-1}) \cdot (a_m...a_n)[a_m \nearrow\!\swarrow b]$, where u, m and b are defined as above. Note that, in virtue of these definitions, the position of the first occurrence in $f(u)$ of a value from $D \setminus \Delta$ is *strictly greater* than the position of the first occurrence in u of a value from $D \setminus \Delta$. Below, we prove that $f(\bar{\gamma})$ is a (w, w')-trace of $f(u)$ and, furthermore, $w \cdot u =_L w \cdot f(u)$ and $w' \cdot u =_L w' \cdot f(u)$ hold. This would imply that that the folded iteration $f^i(u)$ of f on u is well-defined and it reaches a fixed point $f^\omega(u)$ in at most $|u|$ steps. Moreover, from previous arguments, the fixed point $f^\omega(u)$ contains only values from the finite alphabet Δ. By transitivity of $=_L$, we would then be able to conclude that, given $u \in D^*$, there is $v = f^\omega(u) \in \Delta^*$ such that $w \cdot u =_L w \cdot v$ and $w' \cdot u =_L w' \cdot v$.

We now fix a word $u \in D^*$ and we prove that $f(\bar{\gamma})$ is a (w, w')-trace of $f(u)$ and $w \cdot u =_L w \cdot f(u)$ holds (by analogous arguments, $w' \cdot u =_L w' \cdot f(u)$ would follow as well). For the sake of brevity, we assume that $u = a_1...a_n$, $\bar{\gamma} = \gamma_0...\gamma_n$, $f(u) = (a_1...a_{m-1}) \cdot (a_m...a_n)[a_m \nearrow\!\swarrow b]$, and $f(\bar{\gamma}) = (\gamma_0...\gamma_{m-1}) \cdot (\gamma_m...\gamma_n)[a_m \nearrow\!\swarrow b]$, where m is the position of the first occurrence in u of a value from $D \setminus \Delta$ and b is a value from D that is fresh for both memory contents contained in γ_{m-1}. Moreover, for every index $0 \leq i \leq n$, we denote by (q_i, σ_i) the configuration reached by $\mathcal{A}$ after reading the word $w \cdot (a_1...a_i)$. Similarly, for every index $m-1 \leq i \leq n$, we denote by (p_i, τ_i) the configuration reached by $\mathcal{A}$ after reading the word $w \cdot (a_1...a_{m-1}) \cdot (a_m...a_i)[a_m \nearrow\!\swarrow b]$. We can shortly write

$$\begin{cases} (q_I, \varepsilon) \xrightarrow[\mathcal{A}]{w} (q_0, \sigma_0) \xrightarrow[\mathcal{A}]{a_1...a_{m-1}} (q_{m-1}, \sigma_{m-1}) \xrightarrow[\mathcal{A}]{a_m...a_n} (q_n, \sigma_n) \\ (q_I, \varepsilon) \xrightarrow[\mathcal{A}]{w} (q_0, \sigma_0) \xrightarrow[\mathcal{A}]{a_1...a_{m-1}} (p_{m-1}, \tau_{m-1}) \xrightarrow[\mathcal{A}]{(a_m...a_n)[a_m \nearrow\!\swarrow b]} (p_n, \tau_n). \end{cases}$$

In order to show that $f(\bar{\gamma})$ is a (w, w')-trace of $f(u)$ and $w \cdot u =_L w \cdot f(u)$, it is sufficient to prove that $p_i = q_i$ and $\tau_i = \sigma_i[a_m \nearrow\!\swarrow b]$ for all $m - 1 \leq i \leq n$. We

prove this by exploiting an induction on i. The base case $i = m-1$ is trivial since neither a_m nor b occur in the memory content σ_i (in particular, this follows from the definition of m and from the fact that $a_m \notin \Delta$ and Δ contains all values that occur in w). As for the induction case, suppose that $p_{i-1} = q_{i-1}$ and $\tau_{i-1} = \sigma_{i-1}[a_m \nearrow b]$. If $a_i \not\sim a_m$ and $a_i \not\sim b$, then it immediately follows that $p_i = q_i$ and $\tau_i = \sigma_i[a_m \nearrow b]$. If $a_i \sim a_m$, then, by definition of m and b, we have

$$\sigma_{i-1} \cdot a_i \;=\; \sigma_{i-1} \cdot a_m \;\simeq\; (\sigma_{i-1} \cdot a_m)[a_m \nearrow b] \;=\; \tau_{i-1} \cdot a_i[a_m \nearrow b].$$

This shows that the same transition rule of $\mathcal{A}$ is activated in the configuration (q_{i-1}, σ_{i-1}) and in the configuration (p_{i-1}, τ_{i-1}) while consuming, respectively, the value a_i and the value $a_i[a_m \nearrow b]$. Therefore, $p_i = q_i$ and $\tau_i = \sigma_i[a_m \nearrow b]$ follow. The argument for the remaining case $a_i \sim b$ is symmetric. □

A.7 Proof of Proposition 3

Proposition 3. *The memorability problem is decidable in non-deterministic single-exponential time $\mathcal{O}(n^2 \cdot m^{2k} \cdot k^{2k})$, where m is the length of the input word w, n is the number of control states of the input DMA $\mathcal{A}$ and k is maximum number of values stored by $\mathcal{A}$.*

Proof. Let $\mathcal{A}$ be a DMA recognizing a language L and using n control states and at most k stored values, let w be a word of length m, and let a be a value that occurs in w. Moreover, let us fix a fresh value b not occurring in w and a subset Δ of D that contains the value b, all values occurring in w, and $2k+1$ additional values. We first prove that

$$a \text{ is } L\text{-memorable in } w \qquad \text{iff} \qquad a \text{ is } L \cap \Delta^*\text{-memorable in } w$$

(here, by a slight abuse of terminology, we adapt Definition 2 to languages over finite alphabets; precisely, we say that a value a is $L \cap \Delta^*$-memorable in w iff there exist $v \in \Delta^*$ and $c \in \Delta$ such that $w \cdot v \simeq (w \cdot v)[a/c]$ and $w \cdot v \not\equiv_{L \cap \Delta^*} w \cdot v[a/c]$).

The direction from right to left is trivial. Let us then assume that a is L-memorable in w. From Definition 2, we know that there exist a word u and a value b' such that $w \cdot u \simeq (w \cdot u)[a/b']$ and $w \cdot u \not\equiv_L w \cdot u[a/b']$. By Proposition 2, we can assume, without loss of generality, that $b' \sim b$. Now, let v be the (w, b)-surrogate of u over the finite alphabet Δ. Since b occurs neither in w nor in v, we know that $w \cdot v \simeq (w \cdot v)[a/b]$. It remains to prove that $w \cdot v \not\equiv_{L \cap \Delta^*} w \cdot v[a/b]$. By definition of surrogate, we have $w \cdot u =_L w \cdot v$. Moreover, given the construction of the surrogate v of u (cf., the proof of Lemma 1), we can assume, without loss of generality, that $w \cdot u[a/b] \simeq w \cdot v[a/b]$ and hence $w \cdot u[a/b] =_L w \cdot v[a/b]$. This shows that $w \cdot v \not\equiv_{L \cap \Delta^*} w \cdot v[a/b]$ holds and hence a is $L \cap \Delta^*$-memorable in w.

In order to decide whether the value a is $L \cap \Delta^*$-memorable in w it is sufficient to provide an upper bound to the length of a shortest word $v \in \Delta^*$ (if there is any) such that $w \cdot v \simeq (w \cdot v)[a/b]$. Let us assume that a is $L \cap \Delta^*$-memorable in w

and let $v \in \Delta^*$ be a word among the shortest ones witnessing $w \cdot v \simeq (w \cdot v)[a/b]$. Moreover, let $\mathcal{B}$ be a deterministic finite-state automaton that recognizes the language $L \cap \Delta^*$. Without loss of generality, we can assume that the number N of states of $\mathcal{B}$ does not exceed $n \cdot |\Delta|^k$ (indeed, the set of states of $\mathcal{B}$ can be obtained from the set of configurations of $\mathcal{A}$ that are reachable by reading words over Δ). We now prove that $|v| \leq N^2$ (note that since $N \leq n \cdot |\Delta|^k$ and $|\Delta| \leq m + 2k + 2$, this would immediately lead to a non-deterministic procedure that decides in time $\mathcal{O}(n^2 \cdot m^{2k} \cdot k^{2k})$ whether a is L-memorable in w). Suppose, by way of contradiction, that $|v| > N^2$. This implies that there exist two distinct positions $1 \leq i < j \leq |v|$ and two states s, s' of $\mathcal{B}$ such that

$$\begin{cases} s_I \xrightarrow[\mathcal{B}]{w \cdot (v(1)...v(i))} s \xrightarrow[\mathcal{B}]{(v(i+1)...v(j))} s \\ s_I \xrightarrow[\mathcal{B}]{w \cdot (v(1)...v(i))[a/b]} s' \xrightarrow[\mathcal{B}]{(v(i+1)...v(j))[a/b]} s' \end{cases}$$

where s_I is the initial state of $\mathcal{B}$. The above arguments show there is a shorter word $v' = v(1)...v(i)v(j+1)...v(|v|)$ such that $w \cdot v' \simeq (w \cdot v')[a/b]$, thus contradicting the minimality of v. □

A.8 Proof of Proposition 4

Proposition 4. *The word-equivalence problem is decidable in non-deterministic single-exponential time $\mathcal{O}(n^2 \cdot m^{2k} \cdot k^{2k})$, where m is the sum of the lengths of the two input words w and w', n is the number of control states of the input DMA $\mathcal{A}$ and k is maximum number of values stored by $\mathcal{A}$.*

Proof. Let us fix a DMA $\mathcal{A}$ recognizing a language L and using n control states and at most k stored values, and let us fix two words w and w'. As a first remark, note that $w \not\equiv_L w'$ holds only if $|\mathsf{mem}_L(w)| = |\mathsf{mem}_L(w')|$. In virtue of Proposition 3, checking whether $|\mathsf{mem}_L(w)| = |\mathsf{mem}_L(w')|$ can be done in non-deterministic time $\mathcal{O}(n^2 \cdot m^{2k} \cdot k^{2k})$. Thus, from now on, we can assume that $|\mathsf{mem}_L(w)| = |\mathsf{mem}_L(w')|$. Moreover, since $\simeq$ refines $\equiv_L$, we can assume, without loss of generality, that $\mathsf{mem}_L(w) = \mathsf{mem}_L(w')$ (indeed, if this were not the case, we could equivalently check whether $w \equiv_L w''$, where w'' is the word obtained from w' by first replacing all occurrences of values of the form $\mathsf{mem}_L(w)(i)$ that are not L-memorable in w' by corresponding fresh values, simultaneously for all indices $1 \leq i \leq |\mathsf{mem}_L(w)|$, and then replacing all occurrences of values of the form $\mathsf{mem}_L(w')(i)$ by $\mathsf{mem}_L(w)(i)$, simultaneously for all indices $1 \leq i \leq |\mathsf{mem}_L(w)|$). As usual, we introduce a finite set Δ that contains all values that occur in w or in w', plus $2k+1$ additional values. Moreover, we denote by $\approx_L$ the standard Myhill-Nerode equivalence between words over the finite alphabet Δ. In particular, we have

$$w \approx_L w' \qquad \text{iff} \qquad \forall\, u \in \Delta^* \quad w \cdot v =_L w' \cdot v.$$

We now prove that, under the assumption $\mathsf{mem}_L(w) = \mathsf{mem}_L(w')$, we have

$$w \equiv_L w' \qquad \text{iff} \qquad w \approx_L w'.$$

Assume that $w \equiv_L w'$. Since $\mathsf{mem}_L(w) = \mathsf{mem}_L(w')$, we know from the definition of $\equiv_L$ that $w \cdot v =_L w' \cdot v$ holds for every word $v \in D^*$ and hence, in particular, for every word $v \in \Delta^*$. This shows that $w \approx_L w'$. As for the other direction, we assume, by contraposition, that $w \not\equiv_L w'$. By definition, there exist two words $u, u' \in D^*$ such that $\mathsf{mem}_L(w) \cdot u \simeq \mathsf{mem}_L(w') \cdot u'$ and $w \cdot u \neq_L w' \cdot u'$. Moreover, in virtue of Proposition 2, we know that $w' \cdot u =_L w' \cdot u'$. This implies $w \cdot u \neq_L w' \cdot u$. Now, let v be the (w, w')-surrogate of u over the alphabet Δ. By construction, we have that $w \cdot u =_L w \cdot v$ and $w' \cdot u =_L w' \cdot v$. Therefore, we obtain $w \cdot v \neq_L w' \cdot v$ and hence $w \not\approx_L w'$.

To conclude the proof, it is sufficient to show that problem of checking whether $w \approx_L w'$ is decidable in non-deterministic time $\mathcal{O}(n^2 \cdot m^{2k} \cdot k^{2k})$. We omit this part of the proof since the arguments are very similar to the ones used in the proof of Proposition 3. □