What You Must Remember When Processing Data Words

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.

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 [7, 8]). Among them, we would like to mention a few interesting categories, which generalize the standard notion of regular language in several respects. *Pebble automata* [6] use special markers to annotate *locations* in a data word. The *data automata* of [2] 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* [5], 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.

Example 1. Consider the automaton from Figure 1. We have used an intuitive notation in the figure – a more precise syntax is given in Section 2. An edge 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 accepts exactly the data words w such that there are an even number of places $n \leq |w|$ with $w(n) \neq w(n-1)$.

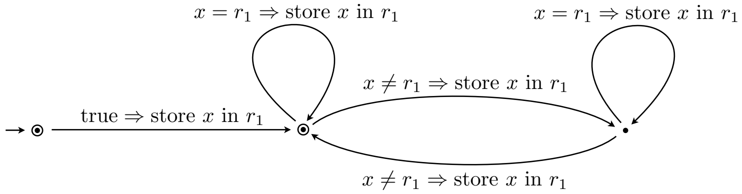
$x = r_1 \Rightarrow$ store x in r_1
$x = r_1 \Rightarrow$ store x in r_1
$x \neq r_1 \Rightarrow$ store x in r_1
true $\Rightarrow$ store x in r_1
$x \neq r_1 \Rightarrow$ store x in r_1

Fig. 1. A finite-memory automaton.

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 [6]. While in the finite case the equivalence and universality problems for non-deterministic automata are decidable, for most of the infinite word models they are not [5, 6].

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 [4] 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 [5]. 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.

A similar characterization, with a more algebraic flavor, has already appeared in [3]. The authors of [3] state as an open question whether the DMA obtained from the algebraic characterisation is minimal. We answer this question positively: 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.

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, while Section 5 gives conclusions. All proofs can be found in [1]

2 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 w and u are said to be isomorphic, and we denote it by $w \simeq u$, if $|w| = |u|$ and $w(i) = w(j)$ iff $u(i) = u(j)$ for all pairs of positions i, j in w. The $\simeq$-equivalence class of a word w, denoted by $[w]_\simeq$ or simply by $[w]$, is called the *$\simeq$-type of w*. A *(data) language* is a set of data words. Given two words w and u, we write $w =_L u$ if either both w and u 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 $=_L$.

2.1 Finite-memory automata

In this section, we introduce a variant of Kaminski's finite-memory automata [5]. These automata process data words by storing a bounded number of values into their memory and by comparing them with respect to the data-equality relation

Definition 1. *A* (non-deterministic) finite-memory automaton *is a tuple* $\mathcal{A} = (Q_0, \ldots, Q_k, I, F, T)$*, where* $Q_0, \ldots, Q_k$ *are pairwise disjoint finite sets of control states,* $I \subseteq Q_0$ *is a set of initial states,* $F \subseteq Q_0 \cup \ldots \cup Q_k$ *is a set of final states, and* T *is a finite set of transition rules of the form* (p, α, E, q)*, where* $p \in Q_i$ *for some* $0 \le i \le k$*,* α *is the* $\simeq$*-type of a word of length* $i+1$*,* $E \subseteq \{1, .., i+1\}$*, and* $q \in Q_j$*, with* $j = i + 1 - |E|$.

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 $0 \le i \le k$, 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 [5]. 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. It should be pointed out that, 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 Values

Given a DMA-recognizable language L and a prefix w of an input word, there exist some values in w that need to be stored by *any* DMA that recognizes L. We will call these values memorable. As an example consider the language $L = \{w \mid w(1) = w(|w|)\}$. Observe that any DMA that recognizes L must store the first symbol in its register. For this reason, we will define a to be memorable in the word $abcde$ with respect to the language L.

Definition 2. *Let L be a language. 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*

$$\begin{cases} w \cdot u & \simeq \quad (w \cdot u)[a/b] \\ w \cdot u & \neq_L \quad w \cdot u[a/b]. \end{cases}$$

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 of the 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. This is well defined because every L-memorable value of w must occur at least once in w.

The following proposition 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 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*

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

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 $w \cdot v$.

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*

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

4 Myhill-Nerode for Data Languages and Minimal Automata

This section is devoted to a characterization of 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 $\simeq$.

Definition 3. *Given a language L, we define $\equiv_L \subseteq D^* \times D^*$ by $w \equiv_L w'$ iff*

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

In a similar way, we associate with each DMA $\mathcal{A}$ a corresponding equivalence relation $\equiv_{\mathcal{A}}$.

Definition 4. *Given a DMA $\mathcal{A}$, we define $\equiv_{\mathcal{A}} \subseteq D^* \times D^*$ by $w \equiv_{\mathcal{A}} w'$ iff whenever $\mathcal{A}$ reaches the configurations (q, σ) and (q', σ') by reading w and w', respectively, then $q = q'$ and $\sigma \simeq \sigma'$ 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 $\mathsf{mem}_L(w) = \mathsf{mem}_L(w')$. Note that, 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. Hence, for any DMA $\mathcal{A}$ with n control states and storing at most k values, the corresponding equivalence $\equiv_{\mathcal{A}}$ has at most $n \cdot k!$ classes.

We are now ready to state the main characterization result.

Theorem 1. *A language L is DMA-recognizable iff $\equiv_L$ has finite index.*

We briefly summarize the key ingredients of the proof of Theorem 1. The full proof is given in [1]. 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 $\equiv_L$. This is sufficient because $\equiv_{\mathcal{A}}$ 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 5. *Let L be a language. If $\equiv_L$ has finite index, then we define the* canonical automaton *for L as the DMA $\mathcal{A}_L = (Q_0, \ldots, Q_k, I, F, T)$, where*

- *$k = \max\{|\mathsf{mem}_L(w)| \mid w \in D^*\}$;*
- *$Q_i = \{[w]_{\equiv_L} \mid w \in D^*, |\mathsf{mem}_L(w)| = i\}$ for all $0 \leq i \leq k$;*
- *$I = \{[\varepsilon]_{\equiv_L}\}$, where ε is the empty word;*
- *$F = \{[w]_{\equiv_L} \mid w \in D^*, w \in L\}$.*
- *T is the set transitions $([w]_{\equiv_L}, \alpha, E, [w \cdot a]_{\equiv_L})$, with $w \in D^*$, $a \in D$, $\alpha = [\mathsf{mem}_L(w) \cdot a]_{\simeq}$, and $E \subseteq \{1, \ldots, |\mathsf{mem}_L(w)| + 1\}$ such that $\mathsf{mem}_L(w \cdot a)$ is the sub-sequence obtained from $\mathsf{mem}_L(w) \cdot a$ by removing all positions in E;*

Minimal DMA. We now explain informally why it is 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 $\mathcal{A} = (Q_0, \dots, Q_k, T, \{q_I\}, F)$ is *state-minimal* if for every equivalent DMA $\mathcal{A}' = (Q'_0, \dots, Q'_{k'}, T', \{q'_I\}, F')$ that recognizes the same language, we have

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

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

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

Finally, we say that $\mathcal{A}$ is *minimal* if it is both state-minimal and data-minimal. Below, we state that the canonical automaton is minimal among all equivalent DMA. The proof is given in [1].

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

We conclude the section by explaining in what sense the minimal DMA, and in particular canonical automata, are unique up to isomorphisms. Here we think of each DMA $\mathcal{A} = (Q_0, \dots, Q_k, T, \{q_I\}, F)$ as a finite directed graph, whose vertices are labeled by indices $i \in \{0, \dots, k\}$ 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')).

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

5 Conclusion

We provide a Myhill-Nerode-like theorem that characterizes the class of DMA-recognizable languages. As in the classical Myhill-Nerode Theorem over finite alphabets, we show that for a given DMA language L there is an automaton – the canonical automaton for L – whose definition depends only on L. This answers a question left open in [3], since the canonical automaton is minimal in a strong sense: it has the minimal number of states and also the minimal amount of internal storage.

Some problems still remain open. It would be interesting to see whether or not analogous characterization results can be given in the case of DMA-recognizable languages over an infinite alphabet equipped with a partial order. 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.

References

[1] Michael Benedikt, Clemens Ley, and Gabriele Puppis. Minimal memory automata, 2010. Available at `http://www.comlab.ox.ac.uk/michael.benedikt/papers/myhilldata.pdf`.

[2] 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.

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

[4] 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.

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

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

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

[8] 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.