



**HAL**  
open science

## Walking on Data Words

Amaldev Manuel, Anca Muscholl, Gabriele Puppis

► **To cite this version:**

Amaldev Manuel, Anca Muscholl, Gabriele Puppis. Walking on Data Words. 8th International Computer Science Symposium in Russia, Jun 2013, Ekaterinburg, Russia. pp.64-75, 10.1007/978-3-642-38536-0\_6 . hal-00877065

**HAL Id: hal-00877065**

**<https://hal.science/hal-00877065>**

Submitted on 30 Oct 2013

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Walking on Data Words\*

Amaldev Manuel, Anca Muscholl, Gabriele Puppis

LaBRI, University of Bordeaux, France

**Abstract.** We see data words as sequences of letters with additional edges that connect pairs of positions carrying the same data value. We consider a natural model of automaton walking on data words, called Data Walking Automaton, and study its closure properties, expressiveness, and the complexity of paradigmatic problems. We prove that deterministic DWA are strictly included in non-deterministic DWA, that the former subclass is closed under all boolean operations, and that the latter class enjoys a decidable containment problem.

## 1 Introduction

Data words arose as a generalization of strings over finite alphabets, where the term ‘data’ denotes the presence of elements from an infinite domain. Formally, data words are modelled as finite sequences of elements chosen from a set of the form  $\Sigma \times \mathbb{D}$ , where  $\Sigma$  is a finite alphabet and  $\mathbb{D}$  is an infinite alphabet. Elements of  $\Sigma$  are called *letters*, while elements of  $\mathbb{D}$  are called *data values*. Sets of data words are called *data languages*.

It comes natural to investigate reasonable mechanisms (e.g., automata, logics, algebras) for specifying languages of data words. Some desirable features of such mechanisms are the decidability of the paradigmatic problems (i.e., emptiness, universality, containment) and effective closures of the recognized languages under the usual boolean operations and projections. The often-used idea is to enhance a finite state machine with data structures to provide some ability to handle data values. Examples of these structures include registers to store data values [5, 6], pebbles to mark positions in the data word [7], hash tables to store partitions of the data domain [1]. In [4] the authors introduced the novel idea of composing a finite state transducer and a finite state automaton to obtain a so-called Data Automaton. Remarkably, the resulting class of automata captures the data languages definable in two-variable first-order logic over data words. For all models except Pebble Automata and Two-way Register Automata the non-emptiness problem is decidable; universality and, by extension, equivalence and inclusion problems are undecidable for all non-deterministic models.

In this work we consider data words as sequences of letters with additional edges that connect pairs of positions carrying the same data value. This idea is

---

\* The research leading to these results has received funding from the ANR project 2010 BLAN 0202 01 FREC and from the European Union’s Seventh Framework Programme (FP7/2007-2013) under grant agreement n. 259454.

consistent with the fact that as far as a data word is concerned the actual data value at a position is not relevant, but only the relative equality and disequality of positions with respect to data values. It is also worth noting that none of the above automaton models makes any distinction between permutations of the data values inside data words. Our model of automaton, called Data Walking Automaton, is naturally two-way: it can roughly be seen as a finite state device whose head moves along successor and predecessor positions, as well as along the edges that connect any position to the closest one having the same data value, either to the right or to the left. Remarkably, emptiness, universality, and containment problems are decidable for Data Walking Automata. Our automata capture, up to letter-to-letter renamings, all data languages recognized by Data Automata. The deterministic subclass of Data Walking Automata is shown to be closed under all boolean operations (closure under complementation is not immediate since the machines may loop). Finally, we deduce from results about Tree Walking Automata [2, 3] that deterministic Data Walking Automata are strictly less powerful than non-deterministic Data Walking Automata, which in turn are subsumed by Data Automata.

## 2 Preliminaries

We use  $[n]$  to denote the subset  $\{1, \dots, n\}$  of the natural numbers. Given a data word  $w = (a_1, d_1) \dots (a_n, d_n)$ , a *class* of  $w$  is a maximal set of positions with identical data value. The set of classes of  $w$  forms a partition of the set of positions and is naturally defined by the equivalence relation  $i \sim j$  iff  $d_i = d_j$ .

The *global successor* and *global predecessor* of a position  $i$  in a data word  $w$  are the positions  $i + 1$  and  $i - 1$  (if they exist). The *class successor* of a position  $i$  is the least position after  $i$  in its class (if it exists) and is denoted by  $i \oplus 1$ . The *class predecessor* of a position  $i$  is the greatest position before  $i$  in its class (if it exists) and is denoted by  $i \ominus 1$ . The global and class successors of a position are collectively called successors, and similarly for the predecessors.

Using the above definitions we can identify any data word  $w \in (\Sigma \times \mathbb{D})^*$  with a directed graph whose vertices are the positions of  $w$ , each one labelled with a letter from  $\Sigma$ , and whose edges are given by the successors and predecessor functions  $+1, -1, \oplus 1, \ominus 1$ . This graph is represented in space  $\Theta(|w|)$ .

**Local types.** Given a data word  $w$  and a position  $i$  in it, we introduce local types  $\overrightarrow{\text{type}}_w(i)$  and  $\overleftarrow{\text{type}}_w(i)$  to describe if each of the successors and predecessors of  $i$  exist and whether they coincide. Formally, when considering the successors of a position  $i$ , four scenarios are possible: (1)  $i$  is the rightmost position and neither the global successor nor the class successor are defined (for short we denote this by  $\overrightarrow{\text{type}}_w(i) = \text{max}$ ), (2)  $i$  is not the rightmost position, but it is the greatest in its class, in which case the global successor exists but not the class successor ( $\overrightarrow{\text{type}}_w(i) = \text{cmax}$ ), (3) both global and class successors of  $i$  are defined and they coincide, i.e.  $i + 1 = i \oplus 1$  ( $\overrightarrow{\text{type}}_w(i) = \text{1succ}$ ), or (4) both successors of  $i$  are defined and they diverge, i.e.  $i + 1 \neq i \oplus 1$  ( $\overrightarrow{\text{type}}_w(i) = \text{2succ}$ ). We define

$\overrightarrow{\text{Types}} = \{\text{max}, \text{cmax}, \text{1succ}, \text{2succ}\}$  to be the set of possible right types of positions of data words. The analogous scenarios for the predecessors of  $i$  are determined by the left type  $\overleftarrow{\text{type}}_w(i) \in \overleftarrow{\text{Types}} = \{\text{min}, \text{cmin}, \text{1pred}, \text{2pred}\}$ . Finally, we define  $\text{type}_w(i) = (\overleftarrow{\text{type}}_w(i), \overrightarrow{\text{type}}_w(i)) \in \text{Types} = \overleftarrow{\text{Types}} \times \overrightarrow{\text{Types}}$ .

**Class-Memory Automata.** We depend on Data Automata [4] for our decidability results. For convenience we use an equivalent model called Class-Memory Automata [1]. Class-Memory Automata are finite state automata enhanced with memory-functions from  $\mathbb{D}$  to a fixed finite set  $[k]$ . On encountering a pair  $(a, d)$ , a transition is non-deterministically chosen from a set that may depend on the current state of the automaton, the memory-value  $f(d)$ , and the input letter  $a$ . When a transition on  $(a, d)$  is executed, the current state and the memory-value of  $d$  are updated. Below we give a formal definition of Class-Memory Automata and observe that this model is similar to that of Tiling Automata [9].

A *Class-Memory Automaton (CMA)* is formally defined as a tuple  $\mathcal{A} = (Q, k, \Sigma, \Delta, I, F, K)$ , where  $Q$  is the finite set of states,  $[k]$  is the set of memory-values,  $\Sigma$  is the finite alphabet,  $\Delta \subseteq Q \times \Sigma \times (\{0\} \cup [k]) \times Q \times [k]$  is the transition relation,  $I \subseteq Q$  is the set of initial states,  $F \subseteq Q$  is the set of accepting states, and  $K \subseteq [k]$  is the set of accepting memory-values. Configurations are pairs  $(q, f)$ , with  $q \in Q$  and  $f$  partial function from  $\mathbb{D}$  to  $[k]$  (for the sake of brevity, we write  $f(d) = 0$  whenever  $f$  is undefined on  $d$ ). Transitions are of the form  $(q, f) \xrightarrow{(a,d)} (q', f')$ , with  $(q, a, f(d), q', h) \in \Delta$ ,  $f'(d) = h$ , and  $f'(e) = f(e)$  for all  $e \in \mathbb{D} \setminus \{d\}$ . Sequences of transitions are called runs. The initial configurations are the pairs  $(q_0, f_0)$ , with  $q_0 \in I$  and  $f_0(d) = 0$  for all  $d \in \mathbb{D}$ ; the final configurations are the pairs  $(q, f)$ , with  $q \in F$  and  $f(d) \in \{0\} \cup K$  for all  $d \in \mathbb{D}$ . The recognized language  $\mathcal{L}(\mathcal{A})$  contains all data words  $w = (a_1, d_1) \dots (a_n, d_n) \in (\Sigma \times \mathbb{D})^*$  that admit runs of the form  $(q_0, f_0) \xrightarrow{(a_1, d_1)} \dots \xrightarrow{(a_n, d_n)} (q_n, f_n)$ , starting in an initial configuration and ending in a final configuration.

It is known that CMA-recognizable languages are effectively closed under union, intersection, letter-to-letter renaming, but not under complementation. Their emptiness problem is decidable and reduces to reachability in vector addition systems, which is not known to be of elementary complexity. Inclusion and universality problems are undecidable. The following result, paired with closure under intersection, allows us to assume that the information about local types of positions of a data word is available to CMA:

**Proposition 1 (Björklund and Schwentick [1]).** *Let  $L$  be the set of all data words  $w \in (\Sigma \times \text{Types} \times \mathbb{D})^*$  such that, for all positions  $i$ ,  $w(i) = (a, \tau, d)$  implies  $\tau = \text{type}_w(i)$ . The language  $L$  is recognized by a CMA.*

**Tiling Automata.** We conclude this preliminary section by observing that CMA are similar to the model of *Tiling Automata on directed graphs* [9], restricted to a subclass of graphs, namely data words. We fix a finite set  $\Gamma$  of colours to be used in tiles. Given a type  $\tau = (\overleftarrow{\tau}, \overrightarrow{\tau}) \in \text{Types}$ , a  $\tau$ -tile associates colours to each position and to its neighbours (as specified by the type). For instance, a (1pred, 2pred)-tile is a tuple of the form  $t = (\gamma_0, \gamma_{-1}, \gamma_{+1}, \gamma_{\oplus 1}) \in \Gamma^4$  such

that, when associated to a position  $i$  in  $w$  with type  $\text{type}_w(i) = (\text{1pred}, \text{2pred})$ , implies that the colour of  $i$  is  $\gamma_0$ , the colour of  $i - 1$  ( $= i \ominus 1$ ) is  $\gamma_{-1}$ , the colour of  $i + 1$  is  $\gamma_{+1}$ , and the colour of  $i \oplus 1$  is  $\gamma_{\oplus 1}$ . A *Tiling Automaton* consists of a family  $\mathcal{T} = (T_{a,\tau})_{a \in \Sigma, \tau \in \text{Types}}$  of  $\tau$ -tiles for each letter  $a \in \Sigma$  and each type  $\tau \in \text{Types}$ . A tiling by  $\mathcal{T}$  of a data word  $w = (a_1, d_1) \dots (a_n, d_n)$  is a function  $\tilde{w} : [n] \rightarrow \Gamma$  such that, for all types  $\tau$  and all positions  $i$  of type  $\tau$ , the  $\tau$ -tile that is formed by  $i$  and its neighbours belongs to the set  $T_{a_i, \tau}$ . The language recognized by the Tiling Automaton  $\mathcal{T}$  consists of all data words that admit a valid tiling by  $\mathcal{T}$ .

The following result depends on the fact that CMA can compute the types of the positions in a data word and is obtained by simple translations of automata:

**Proposition 2.** *CMA and Tiling Automata on data words are equivalent.*

### 3 Automata walking on data words

An automaton walking on data words is a finite state acceptor that processes a data word by moving its head along the successors and predecessors of positions. We let  $\text{Axis} = \{0, +1, \oplus 1, -1, \ominus 1\}$  be the set of the five possible directions of navigation in a data word (0 stands for ‘stay in the current position’).

**Definition 1.** *A Data Walking Automaton (DWA for short) is defined as a tuple  $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ , where  $Q$  is the finite set of states,  $\Sigma$  is the finite alphabet,  $\Delta \subseteq Q \times \Sigma \times \text{Types} \times Q \times \text{Axis}$  is the transition relation,  $I \subseteq Q$  is the set of initial states,  $F \subseteq Q$  is the set of final states.*

Let  $w = (a_1, d_1) \dots (a_n, d_n) \in (\Sigma \times \mathbb{D})^*$  be a data word. Given  $i \in [n]$  and  $\alpha \in \text{Axis}$ , we denote by  $\alpha(i)$  the position that is reached from  $i$  by following the axis  $\alpha$  (for instance, if  $\alpha = 0$  then  $\alpha(i) = i$ , if  $\alpha = \oplus 1$  then  $\alpha(i) = i \oplus 1$ , provided that  $i$  is not the last element in its class). A configuration of  $\mathcal{A}$  is a pair consisting of a state  $q \in Q$  and a position  $i \in [n]$ . A transition is a tuple of the form  $(p, i) \xrightarrow{w} (q, j)$  such that  $(p, a_i, \tau, q, \alpha) \in \Delta$ , with  $\tau = \text{type}_w(i)$  and  $j = \alpha(i)$ . The initial configurations are the pairs  $(q_0, i_0)$ , with  $q_0 \in I$  and  $i_0 = 1$ . The halting configurations are those pairs  $(q, i)$  on which no transition is enabled; such configurations are said to be *final* if  $q \in F$ . The language  $\mathcal{L}(\mathcal{A})$  recognized by  $\mathcal{A}$  is the set of all data words  $w \in (\Sigma \times \mathbb{D})^*$  that admit a run of  $\mathcal{A}$  that starts in an initial configuration and halts in a final configuration.

We will also consider *deterministic* versions of DWA, in which the set  $I$  of initial states is a singleton and the transition relation  $\Delta$  can be seen as a partial function from  $Q \times \Sigma \times \text{Types}$  to  $Q \times \text{Axis}$ .

*Example 1.* Let  $L_1$  be the set of all data words that contain at most one occurrence of each data value (this language is equally defined by the formula  $\forall x \forall y \ x \sim y \rightarrow x = y$ ). A deterministic DWA can recognize  $L_1$  by reading the input data word from left to right (along axis  $+1$ ) and by checking that all positions except the last one have type  $(\text{cmin}, \text{cmax})$ . When a position with type  $(\text{cmin}, \text{max})$  or  $(\text{min}, \text{max})$  is reached, the machine halts in an accepting state.

*Example 2.* Let  $L_2$  be the set of all data words in which every occurrence of  $a$  is followed by an occurrence of  $b$  in the same class (this is expressed by the formula  $\forall x a(x) \rightarrow \exists y b(y) \wedge x < y \wedge x \sim y$ ). A deterministic DWA can recognize  $L_2$  by scanning the input data word along the axis  $+1$ . On each position  $i$  with left type  $\text{cmin}$ , the machine starts a sub-computation that scans the entire class of  $i$  along the axis  $\oplus 1$ , and verifies that every  $a$  is followed by a  $b$ . The sub-computation terminates when a position with right type  $\text{cmax}$  is reached, after which the machine traverses back the class, up to the position  $i$  with left type  $\text{cmin}$ , and then resumes the main computation from the successor  $i + 1$ . Intuitively, the automaton traverses the data word from left to right in a ‘class-first’ manner.

*Example 3.* Our last example deals with the set  $L_3$  of all data words in which every occurrence of  $a$  is followed by an occurrence of  $b$  that is *not* in the same class (this is expressed by the formula  $\forall x a(x) \rightarrow \exists y b(y) \wedge x < y \wedge x \not\sim y$ ). This language is recognized by a deterministic DWA, although not in an obvious way. Fix a data word  $w$ . It is easy to see that  $w \in L_3$  iff one following cases holds:

1. there is no occurrence of  $a$  in  $w$ ,
2.  $w$  contains a rightmost occurrence of  $b$ , say in position  $\ell_b$ , and all occurrences of  $a$  are before  $\ell_b$ ; in addition, we require that either the class of  $\ell_b$  does not contain an  $a$ , or the class of  $\ell_b$  contains a rightmost occurrence of  $a$ , say in position  $\ell_a$ , and another  $b$  appears after  $\ell_a$  but outside the class of  $\ell_b$ .

It is easy to construct a deterministic DWA that verifies the first case. We show how to verify the second case. For this, the automaton reaches the rightmost position  $|w|$  and searches backward, following the axis  $-1$ , the first occurrence of  $b$ : this puts the head of the automaton in position  $\ell_b$ . From position  $\ell_b$  the automaton searches along the axis  $\ominus 1$  an occurrence of  $a$ . If no occurrence of  $a$  is found before seeing the left type  $\text{cmin}$ , then the automaton halts by accepting. Otherwise, as soon as  $a$  is seen (necessarily at position  $\ell_a$ ), a second phase starts that tries to find another occurrence of  $b$  after  $\ell_a$  and outside the class of  $\ell_b$  (we call such an occurrence a *b-witness*). To do this, the automaton moves along the axis  $+1$  until it sees a  $b$ , say at position  $i$ . After that, it scans the class of  $i$  along the axis  $\oplus 1$ . If the right type  $\text{cmax}$  is seen before  $b$ , this means that the class of  $i$  does not contain a  $b$ : in this case, the automaton goes back to position  $i$  (which is now the first position along axis  $\ominus 1$  that contains a  $b$ ) and accepts iff  $b$  is seen along axis  $+1$  (thanks to the previous test, that occurrence of  $b$  must be outside the class of  $\ell_b$  and hence a *b-witness*). Otherwise, if a  $b$  is seen in position  $j$  before the right type  $\text{cmax}$ , this means that the class of  $i$  contains a  $b$ : in this case, the automaton backtracks to position  $i$  and resumes the search for another occurrence of  $b$  to the right of  $i$  (note that if  $i$  is a *b-witness*, then  $j$  is also a *b-witness*, which will be eventually processed by the automaton).

**Closure properties.** Closure of non-deterministic DWA under union is easily shown by taking a disjoint union of the state space of the two automata. Closure under intersection is shown by assuming that one of the two automata accepts only by halting in the leftmost position and by coupling its final states with the initial states of the other automaton.

Closure properties for deterministic DWA rely on the fact that one can remove loops from deterministic computations. The proof of the following result is an adaptation of Sipser’s construction for eliminating loops on configurations of deterministic space-bounded Turing machines [8].

**Proposition 3.** *Given a deterministic DWA  $\mathcal{A}$ , one can construct a deterministic DWA  $\mathcal{A}'$  equivalent to  $\mathcal{A}$  that always halts.*

**Proposition 4.** *Non-deterministic DWA are effectively closed under union and intersection. Deterministic DWA are effectively closed under union, intersection, and complementation.*

## 4 Deterministic vs non-deterministic DWA

We aim at proving the following separation results:

**Theorem 1.** *There exist data languages recognized by non-deterministic DWA that cannot be recognized by deterministic DWA. There also exist data languages recognized by CMA that cannot be recognized by non-deterministic DWA.*

Intuitively, the proof of the theorem exploits the fact that one can encode binary trees by suitable data words and think of deterministic DWA (resp. non-deterministic DWA, CMA) as deterministic Tree Walking Automata (resp. non-deterministic Tree Walking Automata, classical bottom-up tree automata). One can then use the results from [2, 3] that show that (i) Tree Walking Automata cannot be determinized and (ii) Tree Walking Automata, even non-deterministic ones, cannot recognize all regular tree languages. We develop these ideas below.

**Encodings of trees.** Hereafter we use the term ‘tree’ (resp. ‘forest’) to denote a generic finite tree (resp. forest) where each node is labelled with a symbol from a finite alphabet  $\Sigma$  and has either 0 or 2 children. To encode trees/forests by data words, we will represent the node-to-left-child and the node-to-right-child relationships via the predecessor functions  $\ominus 1$  and  $-1$ , respectively. In particular, a *leaf* will correspond to a position of the data word with *no class predecessor*, an *internal node* will correspond to a position where *both class and global predecessors* are defined (and are distinct), and a *root* will be represented either by the *rightmost position* in the word or by a position with no class successor that is immediately followed by a position with no class predecessor. As an example, given pairwise different data values  $d, e, f, g$ , the complete binary tree of height 2 can be encoded by the following data word:

$$w = d \quad e \leftarrow d \quad f \quad g \leftarrow f \leftarrow d$$

(to ease the understanding of the example, we drew only the instances of the predecessor functions  $\ominus 1$  and  $-1$  that represent left and right edges of the tree).

A formal definition of encoding of a tree/forest follows:

**Definition 2.** We say that a data word  $w \in (\Sigma \times \mathbb{D})^+$  is a forest encoding if there is no position  $i$  such that  $\overleftarrow{\text{type}}_w(i) = 1\text{pred}$  and no pair of consecutive positions  $i$  and  $i+1$  such that  $\overrightarrow{\text{type}}_w(i) = 2\text{succ} \wedge \overleftarrow{\text{type}}_w(i+1) = 2\text{pred}$ .

Given a forest encoding  $w$ , we denote by  $\text{forest}(w)$  the directed binary forest that has for nodes the positions of  $w$ , labelled over  $\Sigma$ , and such that:

- if  $\overleftarrow{\text{type}}_w(i) \in \{\text{min}, \text{cmin}\}$ , then  $i$  is a leaf in  $\text{forest}(w)$ ,
- if  $\overleftarrow{\text{type}}_w(i) = 2\text{pred}$ , then  $i \ominus 1$  and  $i - 1$  are left and right children of  $i$ ,
- if  $\overrightarrow{\text{type}}_w(i) = \text{max}$  or  $\overrightarrow{\text{type}}_w(i) = \text{cmax} \wedge \overleftarrow{\text{type}}_w(i+1) = \text{cmin}$ , then  $i$  is a root ( $\text{forest}(w)$  is clearly an acyclic directed graph; the fact that each node  $i$  has at most one parent follows from a case distinction based on the types of  $i$  and  $i+1$ ).

We let  $\text{tree}(w) = \text{forest}(w)$  if the forest encoded by  $w$  contains a single root, namely, it is a tree, otherwise, we let  $\text{tree}(w)$  be undefined.

We remark that there exist several encodings of the same tree/forest that are not isomorphic even up to permutations of the data values. For instance, the two data words below encode the same complete binary tree of height 2:

$$w = \begin{array}{ccccccc} & & \curvearrowright & & \curvearrowleft & & \\ & & d & e \leftarrow d & f & g \leftarrow f & \leftarrow d \\ & & & & & & \end{array} \quad w' = \begin{array}{ccccccc} & & \curvearrowright & & \curvearrowleft & & \\ & & d & f & e \leftarrow d & g \leftarrow f & \leftarrow d \\ & & & & & & \end{array}$$

Among all possible encodings of a tree/forest, we identify special ones, called *canonical encodings*, in which the nodes are listed following the *post-order* visit. Each tree  $t$  has a unique canonical encoding up to permutations of the data values, which we denote by  $\text{enc}(t)$ .

**Separations of tree automata.** We briefly recall the definition of a tree walking automaton and the separation results from [2, 3]. In a way similar to DWA, we first introduce local types of nodes inside trees. These can be seen as pairs of labels from the finite sets  $\text{Types}^\downarrow = \{\text{leaf}, \text{internal}\}$  and  $\text{Types}^\uparrow = \{\text{root}, \text{leftchild}, \text{rightchild}\}$ , and they allow us to distinguish between a leaf and an internal node as well as between a root, a left child, and a right child. We envisage a set  $\text{TAxis} = \{0, \uparrow, \swarrow, \searrow\}$  of four navigational directions inside a tree: 0 is for staying in the current node,  $\uparrow$  is for moving to the parent,  $\swarrow$  is for moving to the left child, and  $\searrow$  is for moving to the right child. A *non-deterministic Tree Walking Automaton (TWA)* is a tuple  $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ , where  $\Sigma$  is the finite alphabet,  $Q$  is the final set of states,  $\Delta \subseteq Q \times \Sigma \times \text{Types}^\downarrow \times \text{Types}^\uparrow \times Q \times \text{TAxis}$  is the transition relation, and  $I, F \subseteq Q$  are the sets of initial and final states. Runs of these automata are defined in a way similar to the runs of DWA. The sub-class of *deterministic TWA* is obtained by replacing the transition relation  $\Delta$  with a partial function from  $Q \times \Sigma \times \text{Types}^\downarrow \times \text{Types}^\uparrow$  to  $Q \times \text{TAxis}$  and by letting  $I$  consist of a single initial state  $q_0$ .

**Theorem 2 (Bojanczyk and Colcombet [2, 3]).** *There exist languages recognized by non-deterministic TWA that cannot be recognized by deterministic TWA. There also exist regular languages of trees that cannot be recognized by non-deterministic TWA.*

**Translations between TWA and DWA.** Hereafter, given a tree language  $L$ , we define  $L^{\text{enc}}$  to be the language of all data words that encode (possibly in a non-canonical way) trees in  $L$ , that is,  $L^{\text{enc}} = \{w : \text{tree}(w) \in L\}$ . To derive from Theorem 2 analogous separation results for data languages, we provide translations between TWA and DWA, as well as from tree automata to CMA:

**Lemma 1.** *Given a deterministic (resp. non-deterministic) TWA  $\mathcal{A}$  recognizing  $L$ , one can construct a deterministic (resp. non-deterministic) DWA  $\mathcal{A}^{\text{enc}}$  recognizing  $L^{\text{enc}}$ . Conversely, given a deterministic (resp. non-deterministic) DWA  $\mathcal{A}$ , one can construct a deterministic (resp. non-deterministic) TWA  $\mathcal{A}^{\text{tree}}$  such that, for all trees  $t$ ,  $\mathcal{A}^{\text{tree}}$  accepts  $t$  iff  $\mathcal{A}$  accepts the canonical encoding  $\text{enc}(t)$ .*

The proof of the first claim is almost straightforward: the DWA  $\mathcal{A}^{\text{enc}}$  is obtained by first transforming  $\mathcal{A}$  into a DWA  $\mathcal{A}'$  that mimics  $\mathcal{A}$  when the input is a valid encoding of a tree, and then intersecting  $\mathcal{A}'$  with a deterministic DWA  $\mathcal{U}$  that accepts all and only the valid encodings of trees. For the proof of the second claim, we observe that the navigational power of a DWA is generally greater than that of a TWA: when the input is a non-canonical encoding of a tree, a DWA may choose to move from a position  $i$  to the position  $i + 1$  even if  $i$  does not represent a right child; on the other hand, a TWA is only allowed to move from node  $i$  to node  $i + 1$  when the former is a right child of the latter. Nonetheless, when restricting to *canonical* encodings of trees, the successor  $i + 1$  of a position represents the node that immediately follows  $i$  in the post-order visit of the tree; in this case, any move of a DWA from  $i$  to  $i + 1$  can be mimicked by a maximal sequence of TWA moves of the form  $\uparrow \searrow \swarrow \dots \swarrow$ .

**Lemma 2.** *Given a tree automaton  $\mathcal{A}$  recognizing a regular language  $L$ , one can construct a CMA  $\mathcal{A}^{\text{enc}}$  recognizing  $L^{\text{enc}}$ .*

We are now ready to transfer the separation results to data languages:

*Proof (of Theorem 1).* Let  $L_1$  be a language recognized by a non-deterministic TWA  $\mathcal{A}_1$  that cannot be recognized by deterministic TWA (cf. first claim of Theorem 2). Using the first claim of Lemma 1, we construct a non-deterministic DWA  $\mathcal{A}_1^{\text{enc}}$  such that  $\mathcal{L}(\mathcal{A}_1^{\text{enc}}) = L_1^{\text{enc}}$ . Suppose by way of contradiction that there is a deterministic DWA  $\mathcal{B}_1$  that also recognizes  $L_1^{\text{enc}}$ . We apply the second claim of Lemma 1 and we obtain a deterministic TWA  $\mathcal{B}_1^{\text{tree}}$  that accepts all and only the trees whose canonical encodings are accepted by  $\mathcal{B}_1$ . Since  $L_1^{\text{enc}} = \{w : \text{tree}(w) \in L_1\}$  is invariant under equivalent encodings of trees (that is,  $w \in L_1^{\text{enc}}$  iff  $w' \in L_1^{\text{enc}}$  whenever  $\text{tree}(w) = \text{tree}(w')$ ), we have that  $t \in L_1$  iff  $\text{enc}(t) \in L_1^{\text{enc}}$ , iff  $t \in \mathcal{L}(\mathcal{B}_1^{\text{tree}})$ . We have just shown that the deterministic TWA  $\mathcal{B}_1^{\text{tree}}$  recognizes the language  $L_1$ , which contradicts the assumption on  $L_1$ .

By applying similar arguments to a regular tree language  $L_2$  that is not recognizable by non-deterministic TWA (cf. second claim of Theorem 2), one can separate CMA from non-deterministic DWA.  $\square$

We conclude by observing that if non-deterministic TWA were not closed under complementation, as one would reasonably expect, then, by Lemma 1, non-deterministic DWA would not be closed under complementation either.

## 5 Decision problems on DWA

We analyse in detail the complexity of the decision problems on DWA. We start by considering the simpler acceptance problem, which consists of deciding whether  $w \in \mathcal{L}(\mathcal{A})$  for a given a DWA  $\mathcal{A}$  and data word  $w$ . Subsequently, we move to the emptiness and universality problems, which consist of deciding, respectively, whether a given DWA accepts at least one data word and whether a given DWA accepts all data words. We will show that these problems are decidable, as well as the more general problems of containment and equivalence.

**Acceptance.** Compared to other classes of automata on data words (e.g. CMA, Register Automata), deterministic DWA enjoy an acceptance problem of very low time/space complexity, and the problem does not get much worse if we consider non-deterministic DWA:

**Proposition 5.** *The acceptance problem for a deterministic DWA  $\mathcal{A}$  and a data word  $w$  is decidable in time  $\mathcal{O}(|w| \cdot |\mathcal{A}|)$  and is LOGSPACE-complete under  $\text{NC}^1$  reductions. The acceptance problem for a non-deterministic DWA is NLOGSPACE-complete.*

**Emptiness.** We start by reducing the emptiness of CMA to the emptiness of deterministic DWA (or, equivalently, to universality of deterministic DWA). For this purpose, it is convenient to think of a CMA  $\mathcal{A}$  as a Tiling Automaton over a finite set  $\Gamma$  of colours and accordingly identify the set of all runs of  $\mathcal{A}$  with the set  $\text{Tilings}(\mathcal{A}) \subseteq (\Sigma \times \Gamma \times \mathbb{D})^*$  of all valid tilings of data words. Given a data word  $\tilde{w} \in (\Sigma \times \Gamma \times \mathbb{D})^*$ , checking whether  $\tilde{w}$  belongs to  $\text{Tilings}(\mathcal{A})$  reduces to checking constraints on neighbourhoods of positions. Since this can be done by a deterministic DWA, we get the following result:

**Proposition 6.** *Given a CMA  $\mathcal{A}$ , one can construct a deterministic DWA  $\mathcal{A}^{\text{tiling}}$  that recognizes the data language  $\text{Tilings}(\mathcal{A})$ .*

Two important corollaries follow from this observation:

**Corollary 1.** *Data languages recognized by CMA are projections of data languages recognized by deterministic DWA.*

**Corollary 2.** *Emptiness and universality of deterministic DWA is at least as hard as emptiness of CMA, which in turn is equivalent to reachability in VASS.*

We turn now to showing that languages recognized by non-deterministic DWA are also recognized by CMA, and hence emptiness of DWA is reducible to emptiness of CMA. Let  $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$  be a non-deterministic DWA. Without loss of generality, we can assume that  $\mathcal{A}$  has a single initial state  $q_0$  and a single final state  $q_f$ . We can also assume that whenever  $\mathcal{A}$  accepts a data word  $w$ , it does so by halting in the rightmost position of  $w$ . For the sake of brevity, given a transition  $\delta = (p, a, \tau, q, \alpha) \in \Delta$ , we define  $\text{source}(\delta) = p$ ,  $\text{target}(\delta) = q$ ,  $\text{letter}(\delta) = a$ ,  $\text{type}(\delta) = \tau$ , and  $\text{reach}(\delta) = \alpha$ . Below, we introduce the concept of min-flow, which can be thought of as a special form of tiling that witnesses acceptance of a data word  $w$  by  $\mathcal{A}$ .

**Definition 3.** Let  $w = (a_1, d_1) \dots (a_n, d_n)$  be a data word of length  $n$ . A min-flow on  $w$  is any map  $\mu : [n] \rightarrow 2^\Delta$  that satisfies the following conditions:

1. There is a transition  $\delta \in \mu(1)$  such that  $\text{source}(\delta) = q_0$ ;
2. There is a transition  $\delta \in \mu(n)$  such that  $\text{target}(\delta) = q_f$ ;
3. For all positions  $i \in [n]$ , if  $\delta \in \mu(i)$ , then  $\text{letter}(\delta) = a_i$  and  $\text{type}(\delta) = \text{type}_w(i)$ ;
4. For each  $i \in [n]$  and each  $q \in Q$ , there is at most one transition  $\delta \in \mu(i)$  such that  $\text{source}(\delta) = q$ ;
5. For each  $i \in [n]$  and each  $q \in Q$ , there is at most one position  $j \in [n]$  for which there is  $\delta \in \mu(j)$  such that  $\text{target}(\delta) = q$  and  $i = \text{reach}(\delta)(j)$ ;
6. For each  $i \in [n]$ , let  $\text{exiting}(i)$  be the set of all states of the form  $\text{source}(\delta)$  for some  $\delta \in \mu(i)$ ; similarly, let  $\text{entering}(i)$  be the set of all states of the form  $\text{target}(\delta)$  for some  $\delta \in \mu(j)$  and some  $j \in [n]$  such that  $i = \text{reach}(\delta)(j)$ ; our last condition states that for all positions  $i \in [n]$ ,
  - (a) if  $i = 1$ , then  $\text{entering}(i) = \text{exiting}(i) \setminus \{q_0\}$ ,
  - (b) if  $i = n$ , then  $\text{exiting}(i) = \text{entering}(i) \setminus \{q_f\}$ ,
  - (c) otherwise,  $\text{exiting}(i) = \text{entering}(i)$ .

**Lemma 3.**  $\mathcal{A}$  accepts  $w$  iff there is a min-flow  $\mu$  on  $w$ .

*Proof.* Let  $w = (a_1, d_1) \dots (a_n, d_n)$  be a data word of length  $n$  and let  $\rho$  be a successful run of  $\mathcal{A}$  on  $w$  of the form  $(q_0, i_0) \xrightarrow{w} (q_1, i_1) \xrightarrow{w} \dots (q_m, i_m)$  obtained by the sequence of transitions  $\delta_1, \dots, \delta_m$ . Without loss of generality, we can assume that no position in  $\rho$  is visited twice with the same state (indeed, if  $i_k = i_h$  and  $q_k = q_h$  for different indices  $k, h$ ,  $\rho$  would contain a loop that could be eliminated without affecting acceptance). We associate with each position  $i \in [n]$  the set  $\mu(i) = \{\delta_k : 1 \leq k \leq m, i_k = i\}$ . One can easily verify that  $\mu$  is a min-flow on  $w$ .

For the other direction, we assume that there is a min-flow  $\mu$  on  $w$ . We construct the edge-labelled graph  $G_\mu$  with vertices in  $Q \times [n]$  and edges of the form  $((p, i), (q, j))$  labelled by a transition  $\delta$ , where  $i \in [n]$ ,  $\delta \in \mu(i)$ ,  $p = \text{source}(\delta)$ ,  $q = \text{target}(\delta)$ , and  $j = \text{reach}(\delta)(i)$ . By construction, every vertex of  $G_\mu$  has the same in-degree as the out-degree (either 0 or 1), with the only exceptions being the vertex  $(q_0, 1)$  of in-degree 0 and out-degree 1, and the vertex  $(q_f, n)$  of in-degree 1 and out-degree 0. One way to construct a successful run of  $\mathcal{A}$  on  $w$  is to repeatedly choose the *only* vertex  $x$  in  $G_\mu$  with in-degree 0 and out-degree 1, execute the transition  $\delta$  that labels the *only* edge departing from  $x$ , and remove that edge from  $G_\mu$ . This procedure terminates when no edge of  $G_\mu$  can be removed and it produces a successful run on  $w$ .  $\square$

Since min-flows are special forms of tilings, CMA can guess them and hence:

**Theorem 3.** Given a DWA, one can construct an equivalent CMA.

**Universality.** Here we show that the complement of the language recognized by a DWA is also recognized by a CMA, and hence universality of DWA is reducible to emptiness of CMA. As usual, we fix a DWA  $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ , with  $I = \{q_0\}$

and  $F = \{q_f\}$ , and we assume that  $\mathcal{A}$  halts only on rightmost positions. Below we define max-flows, which, dually to min-flows, can be seen as a special forms of tilings witnessing non-acceptance.

**Definition 4.** Let  $w = (a_1, d_1) \dots (a_n, d_n)$  be a data word of length  $n$ . A max-flow on  $w$  is any map  $\nu : [n] \rightarrow 2^Q$  that satisfies the following conditions:

1.  $q_0 \in \nu(1)$  and  $q_f \notin \nu(n)$ ,
2. for all positions  $i \in [n]$  and all transitions  $\delta \in \Delta$ , if  $\text{source}(\delta) \in \nu(i)$ ,  $\text{letter}(\delta) = a_i$ , and  $\text{type}(\delta) = \text{type}_w(i)$ , then  $\text{target}(\delta) \in \nu\text{reach}(\delta)(i)$ .

**Lemma 4.**  $\mathcal{A}$  rejects  $w$  iff there is a max-flow  $\nu$  on  $w$ .

**Theorem 4.** Given a non-deterministic DWA  $\mathcal{A}$  recognizing  $L$ , one can construct a CMA  $\mathcal{A}'$  that recognizes the complement of  $L$ .

**Containment and other problems.** We conclude by mentioning a few interesting decidability results that follow directly from Theorems 3 and 4 and from the closure properties of CMA under union and intersection. The first result concerns the decidability of containment/equivalence of DWA. The second result concerns the property of language of being *invariant under tree encodings*, namely, of being of the form  $L^{\text{enc}}$  for some language  $L$  of trees.

**Corollary 3.** Given two non-deterministic DWA  $\mathcal{A}$  and  $\mathcal{B}$ , one can decide whether  $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$ .

**Corollary 4.** Given a non-deterministic DWA  $\mathcal{A}$ , one can decide whether  $\mathcal{L}(\mathcal{A})$  is invariant under tree encodings.

## 6 Discussion

We showed that the model of walking automaton can be adapted to data words in order to define robust families of data languages. We studied the complexity of the fundamental problems of word acceptance, emptiness, universality, and containment (quite remarkably, all these problems are shown to be decidable). We also analysed the relative expressive power of the deterministic and non-deterministic models of Data Walking Automata, comparing them with other classes of automata appeared in the literature (most notably, Data Automata and Class-Memory Automata). In this respect, we proved that deterministic DWA, non-deterministic DWA, and CMA form a strictly increasing hierarchy of data languages, where the top ones are projections of the bottom ones.

It follows from our results that DWA satisfy properties analogous to those satisfied by Tree Walking Automata – for instance deterministic DWA, like deterministic TWA, are effectively closed under all boolean operations, and are strictly less expressive than non-deterministic DWA. It turns out that DWA are also incomparable with one-way Register Automata [5]: on the one hand, DWA can check that all data values are distinct, whereas Register Automata

cannot; on the other hand, Register Automata can recognize languages of data strings that do not encode valid runs of Turing machines, while Data Walking Automata cannot, as otherwise universality would become undecidable.

Since moving along the axis  $\oplus 1$  (resp.  $\ominus 1$ ) can be simulated by storing the current data value or putting a pebble at the current position and moving along the axis  $+1$  (resp.  $-1$ ) searching for the nearest position with the stored data value or marked data value, it follows that DWA are subsumed by two-way 1-Register Automata and 2-Pebble Automata (note that in Pebble Automata one pebble is always used by the head). Other variants of DWA could have been considered, for instance, by adding registers, pebbles, alternation, or nesting. Unfortunately, none of these extensions yield a decidable containment problem. For instance, equipping DWA with a single pebble would enable encoding positive instances of the Post Correspondence Problem, thus implying undecidability of emptiness.

We leave open the following problems:

- Are non-deterministic DWA closed under complementation? (a similar separation result remains open for Tree Walking Automata [2, 3]).
- Do DWA capture all languages definable by two-variable first-order formulas using the predicates  $<$  and  $\sim$ .

As a matter of fact, we can easily show that DWA capture  $\text{FO}^2$  logic with predicates  $+1$  and  $\oplus 1$  (the proof relies on a variant of Gaifman's locality theorem).

**Acknowledgments.** The first author thanks Thomas Colcombet for detailed discussions and acknowledges that some of the ideas were inspired during these. The second author acknowledges Miłojaj Bojańczyk and Thomas Schwentick for discussions about the relationship between DWA and Data Automata.

## References

- [1] H. Björklund and T. Schwentick. On notions of regularity for data languages. *Theoretical Computer Science*, 411(4-5):702–715, 2010.
- [2] M. Bojańczyk and T. Colcombet. Tree-walking automata cannot be determinized. *Theoretical Computer Science*, 350(2-3):164–173, 2006.
- [3] M. Bojańczyk and T. Colcombet. Tree-walking automata do not recognize all regular languages. *SIAM Journal*, 38(2):658–701, 2008.
- [4] M. Bojańczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data words. *ACM Transactions on Computational Logic*, 12(4):27, 2011.
- [5] M. Kaminski and N. Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.
- [6] L. Libkin and D. Vrgoc. Regular expressions for data words. In *LPAR*, volume 7180 of *LNCS*, pages 274–288. Springer, 2012.
- [7] F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Transactions on Computational Logic*, 5(3):403–435, 2004.
- [8] M. Sipser. Halting space-bounded computations. *Theoretical Computer Science*, 10:335–338, 1980.
- [9] W. Thomas. Elements of an automata theory over partial orders. In *Partial Order Methods in Verification*, pages 25–40. American Mathematical Society, 1997.