Generalized Data Automata and Fixpoint Logic*

Thomas Colcombet and Amaldev Manuel

LIAFA, Université Paris-Diderot
{thomas.colcombet, amal}@liafa.univ-paris-diderot.fr

Abstract

Data ω-words are ω-words where each position is additionally labelled by a data value from an infinite alphabet. They can be seen as graphs equipped with two sorts of edges: 'next position' and 'next position with the same data value'. Based on this view, an extension of Data Automata called Generalized Data Automata (GDA) is introduced. While the decidability of emptiness of GDA is open, the decidability for a subclass class called Büchi GDA is shown using Multicounter Automata. Next a natural fixpoint logic is defined on the graphs of data ω-words and it is shown that the μ-fragment as well as the alternation-free fragment is undecidable. But the fragment which is defined by limiting the number of alternations between future and past formulas is shown to be decidable, by first converting the formulas to equivalent alternating Büchi automata and then to Büchi GDA.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Data words, Data Automata, Decidability, Fixpoint Logic

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.267

1 Introduction

Data words are words that can use symbols ranging over an infinite alphabet of 'data values'. Data values are meant to be tested for equality only. Hence, one is typically interested in languages such as 'no data value appears twice', or 'all consecutive data values in the word are distinct', etc. We can already see in these examples one specificity of data words, which is that the exact domain of data values does not matter, and these can be permuted without affecting the membership to a language.

Data values are particularly interesting in several modelling contexts. In particular, data values can be understood as identifiers in a database. The exact content of an identifier does not really matter. What is interesting is to be able to refer easily to the other places in the database/document where this identifier occurs. Another situation in which the data abstraction particularly makes sense is when considering the log of a system, say a server [1]. Such a log is a sequence (potentially infinite) of events that are generated by the different clients. The events produced by the various clients can be interleaved in any manner. Hence, a standard language theoretic approach does not help in verifying meaningful properties of such a log. Indeed, if the events of the sequence are anonymous – in the sense that the identity of the client that has produced it is not retained – then the interleaving obfuscates all relevant behavior of a specific client. Data languages, by annotating each action in this sequence by the unique identifier (the data) representing the client that has produced this action, give access to much more precise information. An interesting way to analyze the structure of the log is then the ability to navigate in its structure. Properties that we are

* The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 259454.

interested to express would typically consists of combinations of queries such as 'what is the next event in the log?', 'what is the next event in the log generated by the same user?', 'what is the last event that the client has generated?', 'has this client ever generated a given event before?', etc.

There are many different formalisms for describing properties of data words, i.e., for defining data languages. They include Data Automata [3], Register Automata [13, 7], Pebble Automata [17], Class Memory Automata [1], Class Automata [4], Walking Automata [16], Variable Automata [11], First-Order logic with two variables [3], Monadic Second Order logic [5], DataLTL [14], Freeze-LTL [7] and Freeze-μ [12], Logic of Repeating Values [6], XPath [8, 9], Regular expressions [15], Data Monoids [2], among others. As opposed to the case of the classical theory of regular languages, none of these formalisms can be considered to be a faithful data word counterpart of the notion of regular languages. This is due to the fact that undecidability arises very quickly in this context, and that many formalisms that turn out to be equivalent for standard words happen to have distinct expressiveness in the case of data words (a typical example is — data monoids [2], deterministic register automata and non-deterministic register-automata [13], that all have different expressiveness). In this contribution we are more interested in the kind of formalisms following the temporal logic approach. Temporal logics (LTL, CTL, CTL* and the μ-calculus) are formalisms that can describe properties of graphs (Kripke structures), by using operators that 'walk' in the structure, and can use all the Boolean connectives. This approach is particularly suitable for instance when one is interested in analysing the log of a system as described above: basic walking constructs are 'go to the next event', 'go to the next event of the same client', 'go to the previous event', and 'go to the previous event of the client'. More complex properties have also to be expressible such as 'go to the first event generated by the client'. Such advanced navigation can be achieved either using dedicated constructs (such as the 'since' and 'until' modalities of LTL), or using explicit fixpoints as done in μ-calculus. In practice, a data word consists of a linear order of positions together with an equivalence relation expressing that two given positions in the word carry the same data values (i.e., a binary relation that expresses that two events were generated by the same client). The walking modalities are then 'next', 'previous' (that we call the global modalities), 'next in the same class', and 'previous in the same class' (that we call the class modalities).

Formalisms that describe properties of data languages using temporal logics have been introduced in [7] and [14]. These two incomparable formalisms, namely DataLTL and Freeze-LTL, are related to two well-studied notions of automata, respectively Data Automata [3] and Register Automata[13, 7]. The logic in this paper is a notion along the lines of DataLTL. It is subsumed by Freeze-μ (which is undecidable over data ω-words) and is incomparable with the logics in [7, 6]. DataLTL is equipped with the four modalities described above, as well as until and since operators that can be used either with respect to global modalities or class modalities. Satisfiability of this logic is decidable by reduction to the decidability of the emptiness of Data Automata. This work was itself a continuation of another one [3] in which the satisfiability of first-order with two variables is shown, and Data Automata are introduced for this purpose. Though this logic is not syntactically a temporal logic, its behavior is in fact the one of a temporal logic.

Contribution. Our contributions are two fold. First, we introduce a generalization of Data Automata, called Generalized Data Automata. While the emptiness problem of GDA is open, we prove the decidability of a subclass of automata, namely the class of Büchi GDA via a reduction to Multicounter Automata. Secondly we generalize the notion of DataLTL by

introducing a natural fixpoint logic. It is shown that the μ-fragment, as well as alternation-free fragment, of this logic is undecidable. For this reason, we restrict our attention to the class of formulas in which the alternation between backward and forward modalities is bounded (this can be syntactically enforced very easily). It is shown that the satisfiability of the alternation-free fragment of this subclass is decidable by first translating the formula into an alternating automaton and then by simulating the alternating automaton by a Büchi GDA using games.

Organization of the paper. In Section 2 we introduce the basics of data ω-words and languages. In Section 3 we introduce generalized data automata and discuss their closure properties and subsequently prove the decidability of the emptiness problem for Büchi GDA. In Section 4 we define μ-calculus on data words and introduce the bounded-reversal alternation-free fragment. We then introduce alternating parity automata and prove the simulation theorem, which is followed by the decidability of the bounded reversal alternation-free fragment. Finally in Section 5 we discuss future work and conclude.

2 Data ω-words and Data Automata

We begin by recalling the basics of data words and Data Automata. Let Σ be a finite alphabet of *letters* and $\mathcal{D}$ be an infinite set. The elements of $\mathcal{D}$, often denoted by d_1, d_2, etc., are called *data values*. A *data word* is a finite sequence of pairs from the product alphabet $\Sigma \times \mathcal{D}$. Likewise a *data ω-word* is a sequence of length ω of pairs from $\Sigma \times \mathcal{D}$. A data language is a set of data words and likewise a data ω-language is a set of data ω-words.

We recall some standard notions related to data words. Let $w = (a_1, d_1)(a_2, d_2)\ldots(a_n, d_n)$ be a data word. The data values impose a natural equivalence relation $\sim$ on the positions in the data word, namely positions i and j are equivalent, i.e. $i \sim j$, if $d_i = d_j$. An equivalence class of the relation $\sim$ is called simply a *class*. The set of all positions in a data word is partitioned into classes. The *global successor* and *global predecessor* of a position i are the positions $i+1$ and $i-1$ respectively (if they exist). For convenience we use $g(i)$ and $g^{-1}(i)$ to denote the global successor and global predecessor of position i. The *class successor* of a position i (if it exists), denoted as $c(i)$, is the leftmost position after i in its class. Symmetrically *class predecessor* of a position i (if it exists), denoted as $c^{-1}(i)$, is the rightmost position before i in its class. These notions are naturally extended to the case of data ω-words.

To simplify the discussion we assume that *all classes in a data ω-word are infinite*. This assumption is similar to the one on infinite trees (that all maximal paths are infinite); by this assumption global successor and class successor relations become total functions. All the results presented later hold without this proviso as well.

Next we recall the notion of Data Automaton (DA for short) introduced in [3]. Originally it is formulated as a composition of two finite state automata. The definition here follows an equivalent presentation due to [1]. Intuitively it is a finite state machine that reads input pairs from $\Sigma \times \mathcal{D}$ and updates the state as follows. During the run the state after reading the pair at position i depends on the state at the class predecessor position of i in addition to the state and input letter at the position i. Formally a Data Automaton $\mathcal{A}$ is a tuple $(Q, \Sigma, \Delta, I, F_c, F_g)$ where Q is a finite set of states, Σ is the finite alphabet, $\Delta \subseteq Q \times (Q \cup \{\bot\}) \times \Sigma \times Q$ is the transition relation, I is the set of initial states, F_c is the set of class Büchi states, and F_g is the set of global Büchi states.

Next we define the run of a Data Automaton. A run $\rho \in (Q \times \mathcal{D})^\omega$ of the automaton

$\mathcal{A}$ on a data ω-word $w = (a_1, d_1)(a_2, d_2)\ldots$ is a relabelling of w by the states in Q, i.e. $\rho = (q_1, d_1)(q_2, d_2)\ldots$ such that the tuple $(q_0, \bot, a_1, q_1)$ is a transition in Δ for some $q_0 \in I$ and, for each position $i > 1$ with a class predecessor, say j, the tuple (q_{i-1}, q_j, a_i, q_i) is a transition in Δ, otherwise if $i > 1$ does not have a class predecessor, then the tuple $(q_{i-1}, \bot, a_i, q_i)$ is in Δ. The run ρ is *accepting* if there is a global Büchi state that occurs infinitely often in the sequence $q_1 q_2 \ldots$, and for every class $\{i_1, i_2, \ldots\}$ there is a class Büchi state occurring infinitely often in the sequence $q_{i_1} q_{i_2} \ldots$. The data ω-word w is accepted if the automaton $\mathcal{A}$ has an accepting run on w. The set of all data ω-words accepted by the automaton $\mathcal{A}$ is called the language of $\mathcal{A}$.

3 Generalized Data Automata

In this section we introduce a generalization of Data Automaton. For this purpose we view a data ω-word as a directed graph with positions as vertices and the global successor and class successor relations as edges. For convenience we refer to these edges as global and class edges. Since both global successor and class successor relations are functions any path in this graph is completely specified by the starting position and a sequence over the alphabet $\{g, c\}$ denoting which edge is taken. Formally a path $\pi = e_1 e_2 \ldots e_n \in \{g, c\}^*$ from the position i connects the sequence of vertices $i, e_1(i), e_2(e_1(i)), \ldots e_n(\ldots e_1(i))$. Similarly an infinite path is an ω-sequence over the alphabet $\{g, c\}$.

A given run of the Data Automaton is accepted or rejected based on two ω-regular conditions; one on the global path (composed only of global edges) and one on each class (composed only of class edges). Next we introduce a generalization of Data Automaton where an ω-regular condition is checked on all paths.

First we need the following definition. Let $w = (a_1, d_1)(a_2, d_2)\ldots$ be a data ω-word and $\pi = e_1 e_2 \ldots \in \{g, c\}^\omega$ be an infinite path starting from the first position. Let $i_0 = 1, i_1, i_2, i_3, \ldots$ be the sequence of positions that lie along the path π. The *path projection* of the data ω-word w w.r.t. the path π is the ω-word $a_{i_0} a_{i_1} a_{i_2} \ldots$. The *marked path projection* of the data ω-word w w.r.t. the path π, denoted as $mpp_w(\pi) \in (\Sigma \times \{\epsilon, g, c\})^\omega$, is obtained by annotating the path projection of w w.r.t. π by the path π, that is to say

$$mpp_w(\pi) = \begin{pmatrix} a_{i_0} \\ \epsilon \end{pmatrix} \begin{pmatrix} a_{i_1} \\ e_1 \end{pmatrix} \begin{pmatrix} a_{i_2} \\ e_2 \end{pmatrix} \ldots$$

Next we introduce the notion of Generalized Data Automaton that has the same transition structure as that of a Data Automaton but a more general acceptance criterion. A *generalized data automaton* $\mathcal{A}$ (for short GDA) $\mathcal{A}$ is a tuple $(Q, \Sigma, \Delta, I, L)$ where Q is the finite set of states, Σ is the finite alphabet, $\Delta \subseteq Q \times (Q \cup \{\bot\}) \times \Sigma \times Q$ is the transition relation, and I is the set of initial states and $L \subseteq (Q \times \{\epsilon, g, c\})^\omega$ is an ω-regular language.

Given a data ω-word $w = (a_1, d_1)(a_2, d_2)\ldots$ a run $\rho \in (Q \times \mathcal{D})^\omega$ of the automaton $\mathcal{A}$ on w is a relabelling $(q_1, d_1)(q_2, d_2)\ldots$ of w with states in Q that obeys all the consistency conditions as in the case of Data Automaton. The only difference is in the criterion of acceptance. The run ρ is accepting if for all paths π in the data ω-word ρ, the marked path projection $mpp_\rho(\pi)$ is in L. The set of all data ω-words on which $\mathcal{A}$ has an accepting run is called the language of $\mathcal{A}$.

The definition of GDA presented above is not concrete, however the acceptance criterion L can be presented as a Büchi automaton which we recall next. A Büchi automaton $\mathcal{B}$ is a tuple (S, A, T, s_{in}, G) where S is a finite set of states, A is the input alphabet, $T \subseteq S \times A \times S$ is the transition relation, s_{in} is the initial state and G is the set of Büchi states. A run r

of the automaton $\mathcal{B}$ on an ω-word $a_1a_2\ldots \in A^\omega$ is a sequence of states $s_0s_1\ldots \in Q^\omega$ such that $s_0 = s_{in}$ and for each $i \in \mathbb{N}$ the tuple (s_{i-1}, a_i, s_i) is a transition in T. The run r is accepting if there is a state in G that occurs infinitely often in it. To finitely present the GDA $\mathcal{A}$ it is enough to provide a Büchi automaton over the alphabet $Q \times \{\epsilon, g, c\}$ that accepts the language L. Next we introduce an important subclass of GDA, namely the class of *Büchi* GDA. A Büchi GDA is a special case of GDA where the acceptance criterion L is an ω-regular language that is furthermore accepted by a deterministic Büchi automaton; a deterministic Büchi automaton is a Büchi automaton whose transition relation T is a function, i.e. $T : S \times A \to S$. By definition Büchi GDA are subsumed by GDA. Our next lemma says that for every Data Automaton there is an equivalent Büchi GDA (hence a GDA as well).

▶ **Lemma 1.** *For every Data Automaton there is an equivalent Büchi GDA.*

In the following we briefly discuss the closure properties of GDA and Büchi GDA. The class of data languages accepted by Data Automata are closed under union, intersection (but not under complementation). The class of languages accepted by GDA and Büchi GDA also exhibit similar closure properties. The union of two GDA (as well as Büchi GDA) accepted languages is recognized by the disjoint union of the respective (Büchi) GDA. Closure under intersection is by usual product construction. (Both GDA and Büchi GDA are not closed under complementation, this follows from the fact that over finite data words GDA are equivalent to Data Automata.)

Two additional closure properties that are relevant for GDA (as well as for DA) are the closure under renaming and closure under composition which we recall now. For a map $h : \Sigma \to \Gamma$ and a data ω-word w over $\Sigma \times \mathcal{D}$, the renaming of w under h, denoted as $h(w)$, is obtained by replacing each letter $a \in \Sigma$ occurring in w by $h(a)$. For a language L of data ω-words over $\Sigma \times \mathcal{D}$, the renaming of L under h, in notation $h(L)$, is simply the set of all renamings $h(w)$ of each word $w \in L$.

Assume A, B, C are letter alphabets. A GDA over the alphabet $(A \times B) \times \mathcal{D}$ can be thought of as a machine that reads a data ω-word over the alphabet $A \times \mathcal{D}$ and applying a labelling of each position by a letter from the set B. In other words the machine can be thought of as a letter-to-letter transducer. The composition of languages correspond to the operation of cascading (feeding the output label of one machine into the input of another) the respective automata. Let L_1 and L_2 be two data ω-languages over the alphabets $(A \times B) \times \mathcal{D}$ and $(B \times C) \times \mathcal{D}$ respectively. The composition $Comp(L_1, L_2)$ of L_1 and L_2 is the set of data ω-words $((a_1, c_1), d_1), ((a_2, c_2), d_2)\ldots$ over the alphabet $(A \times C) \times \mathcal{D}$ such that there exists a data ω-word $((a_1, b_1), d_1), ((a_2, b_2), d_2)\ldots \in (A \times B) \times \mathcal{D}$ in L_1 and $((b_1, c_1), d_1), ((b_2, c_2), d_2)\ldots \in (B \times C) \times \mathcal{D}$ in L_2. The closure of GDA and Büchi GDA under renaming and composition can be shown by standard constructions (renaming of transitions and product construction respectively) as in the case of finite state automata. The following lemma summarizes the closure properties discussed above.

▶ **Lemma 2.** *GDA as well as Büchi GDA are closed under union, intersection, renaming and composition.*

3.1 Emptiness of Büchi GDA

The rest of this section is devoted to the emptiness problem of GDA, namely *is the language of a given GDA empty?*. We don't know if the emptiness of GDA is decidable. However, by extending the decidability proof of emptiness problem of Data Automata it can be shown

that the emptiness problem for Büchi GDA is decidable. As in the case of Data Automata [3], the emptiness problem of GDA is reduced to the emptiness problem of Multicounter Automata which is decidable.

The general idea is as follows. Given a Büchi GDA $\mathcal{A}$ we construct a Multicounter Automaton that guesses a data ω-word w and simulates the automaton $\mathcal{A}$ on w and accepts if and only if $\mathcal{A}$ accepts w. Since a data ω-word is an infinite object the Multicounter Automaton cannot guess the whole word w. Instead it guesses a finite data word satisfying certain conditions that guarantees the existence of a data ω-word in the language of the automaton $\mathcal{A}$.

Now we proceed with the proof. Fix a Büchi GDA $\mathcal{A} = (Q, \Sigma, \Delta, I, L)$ and a deterministic Büchi automaton $\mathcal{B} = (S, A = Q \times \{\epsilon, g, c\}, T, s_{in}, G)$ accepting the language L.

Let $w = (a_1, d_1)(a_2, d_2)\dots$ be a data ω-word accepted by the automaton $\mathcal{A}$ and let $\rho = (q_1, d_1)(q_2, d_2)\dots$ be a successful run of $\mathcal{A}$ on w. Therefore for every infinite path π the ω-word $mpp_\rho(\pi)$ is accepted by the Büchi automaton $\mathcal{B}$. Let π_1 and π_2 be two infinite paths. Their respective marked path projections agree on the common prefix of π_1 and π_2. Since the automaton $\mathcal{B}$ is deterministic the (unique) runs of $\mathcal{B}$ on $mpp_\rho(\pi_1)$ and $mpp_\rho(\pi_2)$ agree on the common prefix as well. This allows us to represent the runs of the automaton $\mathcal{B}$ on the marked path projections of ρ by a labelling by subsets of S in the following way.

Let $\pi = e_1 \dots e_n \in \{g, c\}^*$ be a finite path connecting the sequence of positions $j_0 = 1$, $j_1, \dots, j_n = i$. The marked path projection of ρ w.r.t. π is the word $(q_{j_0}, \epsilon)(q_{j_1}, e_1)\dots(q_{j_n}, e_n)$ over the alphabet $Q \times \{\epsilon, g, c\}$. By $\mathbf{P}(S)$ we denote the power set of S. Let $S_1 S_2 \dots \in (\mathbf{P}(S))^\omega$ be such that S_i is the set of all states q such that there is a finite path $\pi \in \{g, c\}^*$ ending in position i and the unique partial run of the automaton $\mathcal{B}$ on the marked path projection of π ends in state q. The ω-word $S_1 S_2 \dots \in (\mathbf{P}(S))^\omega$ can be seen as the superposition runs of the automaton $\mathcal{B}$ on each of the marked string projections. We call the data word $\zeta = ((q_1, S_1), d_1)((q_2, S_2), d_2)\dots \in ((Q \times \mathbf{P}(S)) \times \mathcal{D})^\omega$ the annotated run.

As we mentioned earlier a witness for non-emptiness of the language of the automaton $\mathcal{A}$ is an infinite object. Hence it is not possible to compute the witness algorithmically. Instead one has to define a finite object that witnesses the non-emptiness. In the case of a Büchi automaton over infinite words this finite object is a word of the form $u \cdot v$ such that $u \cdot v^\omega$ is in the language of the automaton. In the case of Büchi GDA, u and v are two finite data words such that $u \cdot v_1 \cdot v_2 \dots$ is in the language of the automaton where $v, v_1, v_2, \dots$ all have the same string projections and identical classes, in other words $v_1, v_2, \dots$ are obtained from v by renaming of data values.

We fix some notation. Let $w = (a_1, d_1)\dots(a_n, d_n)$ be a finite data word over the alphabet Σ. A position with no class successor is called a class-maximal position. Similarly a position with no class predecessor is called a class-minimal position. The class vector of w is vector $C(w) : \Sigma \to \mathbb{N}$ that maps each letter a in Σ to the number of class-maximal positions labelled by a.

Next we formally define the notion of the finite witness in the case of Büchi GDA. Let $u, v \in (\Sigma \times \mathcal{D})^*$ be two finite data words and let $w = u \cdot v$. Let $\rho = \rho_u \cdot \rho_v \in (\Delta \times \mathcal{D})^*$ be a partial run of the Büchi GDA on the finite data word w (A partial run is a finite prefix/infix/suffix of some run of the automaton under consideration). Let $\zeta = \zeta_u \cdot \zeta_v \in ((Q \times \mathbf{P}(S)) \times \mathcal{D})^*$ be the annotated run of the automaton $\mathcal{A}$ on the data word w. (Note that the definition of annotation extends to finitely data words naturally). We aim at constructing a data ω-word in the language of the automaton $\mathcal{A}$ by repeatedly appending the data word v (with possible renaming of data values) to the end of w. Therefore the 'configuration' of the automata $\mathcal{A}$ and $\mathcal{B}$, namely the states at which the partial runs of both automata end, have to be the same

at the end of the data words u and w. Moreover the number of class-maximal positions in ζ_w annotated with a pair $(q, S') \in Q \times \mathbf{P}(S)$ has to be at least the number of class-maximal positions in ζ_u annotated with the same pair for the pumping to work correctly. Finally for the acceptance criterion to be satisfied every partial run of the automaton $\mathcal{B}$ on the marked path projection of ρ w.r.t a path starting from a class-maximal position in u and ending in a class-maximal position in v (including the last position) has to see a Büchi state (in G). All these conditions are summarized below;

The triple w, ρ, ζ forms a *regular witness* if the following conditions are met.

- **(i)** The state at the end of the partial runs ρ_u and ρ_w are the same.
- **(ii)** $S_u = S_w$ where S_u and S_w are annotations at the last positions of ζ_u and ζ_w respectively.
- **(iii)** Let C_u and C_w be the class vectors of ζ_u and $\zeta_w$respectively. Then,
  - **(a)** $C_u \leq C_w$ in the componentwise ordering,
  - **(b)** for all $(q, S') \in Q \times \mathbf{P}(S)$, if $C_u((q, S')) = 0$ then it is the case that $C_w((q, S')) = 0$.
  - **(c)** Every partial run of the automaton $\mathcal{B}$ on the marked path projection of ρ w.r.t a path starting from a class-maximal position in u and ending in a class-maximal position in v (including the last position) has to see a Büchi state (in G).

In the subsequent lemma we prove the necessity and sufficiency of regular witnesses for deciding the nonemptiness. The proof rests on the following two standard lemmas.

▶ **Lemma 3** (Dickson's lemma). *Fix a $k \in \mathbb{N}$. Every infinite sequence of vectors $v_0, v_1, \ldots$ where $v_i \in (\mathbb{N}_0)^k$ contains an infinite nondecreasing subsequence $v_{i_0} \leq v_{i_1} \leq \ldots$ where the ordering $\leq$ is componentwise.*

▶ **Lemma 4** (König's lemma for words). *If A is a finite set and $L \subseteq A^*$ is infinite then there exists $x \in A^\omega$ such that x has infinitely many prefixes in L.*

▶ **Lemma 5.** *Automaton $\mathcal{A}$ accepts some data ω-word if and only if there is a regular witness for the non-emptiness of $\mathcal{A}$.*

Using Lemma 5 it is possible to decide if a given GDA accepts a non-empty language. This is achieved by a reduction to the non-emptiness problem of Multicounter Automata. A *Multicounter Automata* is a finite state machine equipped with a finite set $[k]$ of counters which hold positive integer values. During each step the machine reads a letter from the input and depending on the letter just read and the current state it performs a counter action and moves to a new state. The allowed operations on the counters are *increment counter i* and *decrement counter i*, but no zero tests are allowed. During the execution if a counter holding a zero value is decremented then the machine halts erroneously. Initially the machine starts in a designated initial state with all the counters set to value zero. An execution is accepting if the machine terminates in a state which belongs to a designated set of final states with all the counters being zero. We will be crucially making use of this final zero test. Non-emptiness of Multicounter Automata is decidable which implies by virtue of the following theorem that non-emptiness of Büchi GDA is decidable.

▶ **Theorem 6.** *Given a Büchi GDA $\mathcal{A}$ one can effectively construct an exponentially-sized Multicounter Automaton which accepts a word if and only if $\mathcal{A}$ has a regular witness.*

4 μ-calculus on data ω-words

In this section, we introduce μ-calculus over data words. Let $Prop = \{p, q, \ldots\}$ be a set of propositional variables. The formulas in the logic are the following. The atomic formulas are, $p \in Prop$, $\neg p$, and $\mathcal{S}, \mathcal{P}, \mathsf{first}^c, \mathsf{first}^g$ which are zeroary modalities. Also, $\mathtt{X}^g\varphi, \mathtt{X}^c\varphi, \mathtt{Y}^g\varphi, \mathtt{Y}^c\varphi$

$$
\begin{aligned}
[\![p]\!]_w &= \ell(p) & [\![\neg p]\!]_w &= \omega \setminus \ell(p)\\
[\![\mathcal{P}]\!]_w &= \{i \mid g^{-1}(i) = c^{-1}(i)\} & [\![\mathcal{S}]\!]_w &= \{i \mid g(i) = c(i)\}\\
[\![\mathsf{first}^g]\!]_w &= \{1\} & [\![\mathsf{first}^c]\!]_w &= \{i \mid \nexists j = c^{-1}(i)\}\\
[\![\mathtt{X}^g\varphi]\!]_w &= \{i \in \omega \mid g(i) \in [\![\varphi]\!]_w\} & [\![\mathtt{X}^c\varphi]\!]_w &= \{i \in \omega \mid c(i) \in [\![\varphi]\!]_w\}\\
[\![\mathtt{Y}^g\varphi]\!]_w &= \{i \in \omega \mid g^{-1}(i) \in [\![\varphi]\!]_w\} & [\![\mathtt{Y}^c\varphi]\!]_w &= \{i \in \omega \mid c^{-1}(i) \in [\![\varphi]\!]_w\}\\
[\![\mu p.\varphi]\!]_w &= \bigcap \left\{ S \subseteq \omega \;\middle|\; [\![\varphi]\!]_{w[\ell(p):=S]} \subseteq S \right\} & [\![\varphi_1 \wedge \varphi_2]\!]_w &= [\![\varphi_1]\!]_w \cap [\![\varphi_2]\!]_w\\
[\![\nu p.\varphi]\!]_w &= \bigcup \left\{ S \subseteq \omega \;\middle|\; S \subseteq [\![\varphi]\!]_{w[\ell(p):=S]} \right\} & [\![\varphi_1 \vee \varphi_2]\!]_w &= [\![\varphi_1]\!]_w \cup [\![\varphi_2]\!]_w
\end{aligned}
$$

Figure 1 Semantics of μ-calculus on a ω-word $w = (\omega, \ell, g, c)$.

are formulas whenever φ is a formula, and $\varphi_1 \vee \varphi_2, \varphi_1 \wedge \varphi_2$ are formulas whenever φ_1 and φ_2 are formulas. Finally, $\mu p.\varphi, \nu p.\varphi$ are formulas whenever φ is a formula and the variable p occurs positively in φ (i.e. $\neg p$ is not a subformula of φ).

Next we disclose the semantics; as usual, on a given structure each formula denotes the set of positions where it is true. The modality first^g holds (only) on the first position and first^c holds exactly on all the first positions of classes. The modality $\mathcal{S}$ is true at a position i if the successor and class successor of i coincide. Similarly $\mathcal{P}$ is true at i if the predecessor and class predecessor of i coincide. The modalities $\mathtt{X}^g\varphi$, $\mathtt{X}^c\varphi$, $\mathtt{Y}^g\varphi$, $\mathtt{Y}^c\varphi$ hold if φ holds on the successor, class successor, predecessor and class predecessor positions respectively. Coming to the fix-point formulas, each formula $\varphi(p)$, where p occurs positively, defines a function from sets of positions to sets of positions that is furthermore monotone. We define the semantics of $\mu p.\varphi(p)$ and $\nu p.\varphi(p)$ to be the least and greatest fix-points of $\varphi(p)$ that exists due to Knaster-Tarski theorem. To formally define the semantics we consider a data ω-word as a Kripke structure $w = (\omega, \ell, g, c)$ where $\ell : \mathit{Prop} \to \mathbf{P}(\omega)$ is valuation function giving for each $p \in \mathit{Prop}$ the set of positions where p holds, g is the global successor relation and c is the class successor relation. For $S \in \mathbf{P}(\omega)$ by $w[\ell(p) := S]$ we mean w with the new valuation function ℓ' that is defined as $\ell'(p) = S$ and $\ell'(q) = \ell(q)$ for all $q \in \mathit{Prop}$, $q \neq p$. The formal semantics $[\![\varphi]\!]_w$ of a formula φ over a data word w is described in Figure 1.

Note that we allow negation only on atomic propositions. However it is possible to negate a formula in the logic. For this, define the dual modalities $\tilde{\mathtt{X}}^g$, $\tilde{\mathtt{Y}}^g$, $\tilde{\mathtt{X}}^c$, $\tilde{\mathtt{Y}}^c$ of $\mathtt{X}^g, \mathtt{Y}^g, \mathtt{X}^c, \mathtt{Y}^c$ respectively and such that $\tilde{\mathtt{M}}\varphi = \neg\mathtt{M}\neg\varphi$, where $\neg$ stands for set complement. Since successor and class successor relations are total functions it follows that $\tilde{\mathtt{X}}^g\varphi \equiv \mathtt{X}^g\varphi$, $\tilde{\mathtt{X}}^c\varphi \equiv \mathtt{X}^c\varphi$. Similarly since predecessor and class predecessor relations are partial functions it follows that $\tilde{\mathtt{Y}}^g\varphi \equiv \mathsf{first}^g \vee \mathtt{Y}^g\varphi$, $\tilde{\mathtt{Y}}^c\varphi \equiv \mathsf{first}^c \vee \mathtt{Y}^c\varphi$. To negate a formula φ we take the dual of φ; this means exchanging in the formula $\wedge$ and $\vee$, μ and ν, p and $\neg p$, and all the modalities with their dual.

If $\varphi(p_1, \ldots, p_n)$ is a formula then by $\varphi(\psi_1, \ldots, \psi_n)$ we mean the formula obtained by substituting ψ_i for each p_i in φ. As usual the bound variables of $\varphi(p_1, \ldots, p_n)$ may require a renaming to avoid the capture of the free variables of ψ_i's. For a formula φ and a position i in the word w, we denote by $w, i \models \varphi$ if $i \in [\![\varphi]\!]_w$. The notation $w \models \varphi$ abbreviates the case when $i = 1$. The *data language* of a sentence φ is the set of data words w such that $w \models \varphi$, while the *data ω-language* of a sentence φ is the set of data ω-words w such that $w \models \varphi$.

Unfortunately, even the fragment of the logic containing only μ-fixpoints itself is undecidable.

▶ **Theorem 7.** *Satisfiability of the μ-fragment is undecidable.*

This also implies the undecidability of the alternation-free fragment (recalled below). One of the sources of undecidability is the presence of both future and past modalities, or in other words the two-way-ness of the logic. Therefore we can reclaim decidability of the logic if we restrict the number of times a formula is allowed to change direction. Next we formally define this fragment, namely the *bounded reversal alternation-free fragment*. We first recall the operation of composition of formulas. Let Ψ be a set of formulas. Define the set $\mathit{Comp}^i(\Psi)$ inductively as $\mathit{Comp}^0(\Psi) = \emptyset$ and

$$\mathit{Comp}^i(\Psi) = \{\psi(\varphi_1, \dots, \varphi_n) \mid \psi(p_1, \dots, p_n) \in \Psi,\ \varphi_1, \dots, \varphi_n \in \mathit{Comp}^{i-1}(\Psi)\} .$$

The set of formulas $\mathit{Comp}(\Psi)$ is defined as $\mathit{Comp}(\Psi) = \bigcup_{i\in\mathbb{N}} \mathit{Comp}^i(\Psi)$. For a formula $\psi \in \mathit{Comp}(\Psi)$ we define the *Comp-height of ψ in $\mathit{Comp}(\Psi)$* as the least i such that $\psi \in \mathit{Comp}^i(\Psi)$.

For $\lambda \in \{\mu, \nu\}$ let $\mathsf{Formulas}(\lambda)$ denote the formulas which uses only the fixpoint operator λ. Then the *alternation-free* fragment, denoted as AF, is the set of formulas $\mathrm{AF} = \mathit{Comp}(\mathsf{Formulas}(\mu) \cup \mathsf{Formulas}(\nu)))$; intuitively there does not exist a μ-subformula and a ν-subformula with intersecting scope in any formula of AF. We call the set of all μ-calculus formulas which does not use the modalities $\{\mathtt{Y}^c, \mathtt{Y}^g\}$ (*resp.* $\{\mathtt{X}^c, \mathtt{X}^g\}$) the forward (*resp.* backward) fragment. Forward (*resp.* backward) alternation-free fragment, denoted as $\mathrm{AF}_{\mathtt{X}}$ (*resp.* $\mathrm{AF}_{\mathtt{Y}}$) is the set of all formulas in the alternation-free fragment which are also in the forward (*resp.* backward) fragment. The *bounded reversal alternation-free fragment* of μ-calculus, denoted as BR, is the set of formulas $\mathrm{BR} = \mathit{Comp}(\mathrm{AF}_{\mathtt{X}} \cup \mathrm{AF}_{\mathtt{Y}})$. An example of a formula in AF but not expressible in the fragment BR (we do not prove it) is $\mu x.a \vee \mathtt{Y}^g\mathtt{X}^c x$. It tests whether an a-letter is reachable by successive steps of advancing to the next in the class, and going backward globally. An example of a formula that is in BR is $\mu y.(\nu x.a \vee \mathtt{X}^c x) \vee \mathtt{Y}^g y$.

Next we prove that the fragment BR is decidable by reducing the satisfiability problem for BR to the emptiness problem for Büchi GDA. Since both BR and Büchi GDA are closed under composition it is enough to prove that for every formula in the fragment $\mathrm{AF}_{\mathtt{X}}$ and $\mathrm{AF}_{\mathtt{Y}}$ there is a Büchi GDA that labels each position with the set of (sub)formulas true at that position.

▶ **Lemma 8.** *Given a formula φ in the backward fragment there is a Data Automaton that labels each position with the set of subformulas of φ true at that position.*

Next we show that for every formula in the forward alternation-free fragment there is a Büchi GDA that labels each position with the set of satisfied subformulas. For this, we recall the notion of alternating parity automaton over graphs (See [10] for a comprehensive presentation). First we need the basics of two player (namely *Adam* and *Eve*) games played on graphs. An arena $A = (V, E)$ is a set of positions $V = V_E \cup V_A$ partitioned into those of Adam (V_A) and those of Eve (V_E) along with a set of moves $E \subseteq (V_A \times V_E) \cup (V_E \times V_A)$ (we assume that there are no dead-ends in the game). A partial play $(v_0, v_1)(v_1, v_2) \dots (v_k, v_{k+1}) \subseteq E^*$ is a finite sequence of moves performed by the players. The position v_0 is the starting position of the play and v_{k+1} is the ending position of the play. A strategy for a player Eve (resp. Adam) σ maps a partial play ending in a position in V_E (resp. V_A) to a move in E. A play $\pi = (v_0, v_1)(v_1, v_2) \dots \in E^\omega$ is an ω-sequence of moves. We say π is a play according to the strategy σ of Eve if on all finite prefixes of π ending in V_E she plays according to σ. A winning condition $W \subseteq E^\omega$ is a set of plays which are winning for Eve. A game $\mathcal{G}$ is a triple $\mathcal{G} = (A = (V, E), v, W)$ where A is an arena, $v \in V$ is the initial position and W is the winning condition. The strategy σ is a winning strategy for Eve if all the plays according

to σ are winning for Eve. The strategy is positional if for all partial plays ending on the same vertex the strategy σ agrees on the next move. A *parity game* is a game where W is presented by means of a parity condition $\Omega : V \to \{0, \ldots, k\}$ for some $k \in \mathbb{N}$. Given Ω, the winning condition W is defined as the union of all plays $\pi = (v_0, v_1)(v_1, v_2)\ldots$ such that the maximal number occurring infinitely often in the sequence $\Omega(v_0), \Omega(v_1), \ldots$ is even. It is well-known that parity games are positionally determined. i.e. one of the players has a positional winning strategy.

Let P be a set of propositional variables. A positive conjunction $p_1 \wedge p_2 \ldots \wedge p_k, k \geq 1$ over P is identified with the subset $\{p_1, \ldots, p_k\}$ of P. A DNF formula over P is a disjunction $\varphi_1 \vee \varphi_2 \ldots \vee \varphi_l, l \geq 1$, where each φ_j is a positive conjunction over P, which is identified with a subset of the powerset of P, namely $\{\varphi_1, \ldots, \varphi_l\}$. The set of all DNF formulas over P is denoted by $\mathrm{DNF}^+(P)$. Let $\mathcal{M}$ be the set $\{\mathcal{S}, \neg\mathcal{S}, \mathcal{P}, \neg\mathcal{P}\}$. For a given a data ω-word w and a position i in w the *type* of i, denoted by $tp\,(i)$, is the subset of $\mathcal{M}$ satisfied at position i.

An *alternating parity automaton* on data ω-words $\mathcal{A}$ is a tuple $(Q, \Sigma, \Delta, q_0, \Omega)$ where Q is the finite set of states, Σ is the alphabet, q_0 is the initial state, $\Delta : Q \times \Sigma \times \mathbf{P}(\mathcal{M}) \to \mathrm{DNF}^+(\{\mathtt{X}^g p, \mathtt{X}^c p \mid p \in Q\})$ is the transition relation and $\Omega : Q \to \{0, \ldots, k\}$ is the parity condition. When Ω is such that all states have parity either 1 or 2 the automaton is called Büchi.

Fix an automaton $\mathcal{A}$. Given a data ω-word $w = (\omega, \lambda, g, c)$ (for convenience we let the labelling function $\lambda : \omega \to \Sigma$ map each position to its label), the acceptance of w by $\mathcal{A}$ is defined, as usual, in terms of a two-player parity game $\mathcal{G}_{\mathcal{A},w}$ (sometimes called the *membership game*) played between Adam and Eve on the arena with positions $V = V_E \cup V_A$ where $V_E = Q \times \omega$ and $V_A = \textit{co-Dom}(\Delta) \times \omega$. The moves E are the following. On every Eve position (p, i) she can make a move to an Adam position (φ, i) where φ is a conjunction over the set $\{\mathtt{X}^g p, \mathtt{X}^c p \mid p \in Q\}$ such that $\varphi \in \Delta(p, \lambda(i), tp\,(i))$. On every Adam position (φ, i) he can make a move to an Eve position (p, j) if j is the successor (*resp.* class successor) of i and $\mathtt{X}^g p$ (*resp.* $\mathtt{X}^c p$) is in φ. Observe that there are no dead-ends in the game. The parity of the game positions are defined as follows. For all Adam positions the parity is 0 and for all Eve positions (p, i) the parity is $\Omega(p)$. We say the automaton $\mathcal{A}$ accepts the data word w if in the game $\mathcal{G}_{\mathcal{A},w}$ the player Eve has a winning strategy from the position $(q_0, 1)$.

The following lemma follows from canonical connection between μ-calculus and alternating parity automata on any fixed class of graphs ([10]).

▶ **Fact 9.** *For every formula in the forward (resp. alternation-free) fragment there is an equivalent (which is effectively obtained) alternating parity (resp. Büchi) automaton. Moreover the states of the automaton are precisely the subformulas of the given formula.*

If a data ω-word w is accepted by $\mathcal{A}$ then there is a winning strategy for Eve in the game $\mathcal{G}_{\mathcal{A},w}$ which in turn implies that Eve has a positional winning strategy for the game. A positional strategy for Eve in $\mathcal{G}_{\mathcal{A},w}$ is a function $\sigma : \omega \to (Q \to \textit{co-Dom}(\Delta))$ such that for all i and for all $p \in Q$, $(\sigma(i))(p) \in \Delta(p, \lambda(i), tp\,(i))$. Once a strategy σ for Eve is fixed the game $\mathcal{G}_{\mathcal{A},w}$ can be seen as a game played by a single player (namely Adam) in the following way. Define $\mathcal{G}^{\sigma}_{\mathcal{A},w}$ as the subgame where the moves of Eve are limited to $\{(p, i) \to ((\sigma(i))(p), i) \mid i \in \omega\}$. Since the moves of Eve are fixed in the game $\mathcal{G}^{\sigma}_{\mathcal{A},w}$ $(\star)$ she wins if and only all the infinite paths in the graph $\mathcal{G}^{\sigma}_{\mathcal{A},w}$ are winning. A *local strategy* is a partial function $f : Q \to \textit{co-Dom}(\Delta)$ such that there exist $a \in \Sigma, \tau \in \mathbf{P}(\mathcal{M})$ such that for all $p \in \textit{Dom}(f)$, $f(p) = \Delta(p, a, \tau)$. A local strategy f is consistent at position i if $f(p) \in \Delta(p, \lambda(i), tp\,(i))$ for all $p \in \textit{Dom}(f)$. Observe that a positional strategy for Eve is a sequence of local strategies $(f_i)_{i \in \omega}$ such that each f_i is consistent at position i. Now we restate $(\star)$ in terms of local strategies. Let F be the set of local strategies.

A *local strategy annotation* of a data ω-word w is a sequence of local strategies $(f_i)_{i\in\omega}$ which are consistent at each position i and furthermore satisfy the following conditions. Let $(D_i)_{i\in\omega}$ be the sequence of subsets of states Q (called the set of *reachable states*) such that the local strategy f_i has domain D_i.

1. $D_1 = \{q_0\}$.
2. $q \in D_{\mathtt{M}(i)}$ iff there exists $p \in D_i$ such that $f_i(p) = \varphi$ and $\mathtt{M}q \in \varphi$ [When $\mathtt{M} = \mathtt{X}^g$ (*resp.* $\mathtt{M} = \mathtt{X}^c$) we use $\mathtt{M}(i)$ to denote the successor (*resp.* class successor) of i]. In this case we say that there is an *edge* between (p, i) and $(q, \mathtt{M}(i))$ in the strategy annotation.

A path in the strategy annotation is a sequence $(p_1, i_1)\dots(p_n, i_n)$ such that each successive tuples has an edge between them. The local strategy annotation $(f_i)_{i\in\omega}$ is *accepting* if for all infinite paths (starting from $(q_0, 1)$) it is the case that the maximal infinitely occurring parity is even.

It is straight-forward to see that Eve has a (positional) winning strategy σ in the game $\mathcal{G}_{\mathcal{A},w}$ iff all the paths in the $\mathcal{G}^{\sigma}_{\mathcal{A},w}$ are winning iff there is a local strategy annotation in which all paths are accepting. Thus we get,

▶ **Lemma 10.** *A data ω-word w is accepted by the automaton $\mathcal{A}$ if and only if there exist a local strategy annotation $(f_i)_{i\in\omega}$ of w which is accepting.*

Next we show the goal of this section namely that for every alternating Büchi automaton there is an equivalent Büchi GDA. Since we are converting an alternating automata to a non-deterministic automata (though not of the same kind) it can be seen as an analogue of the simulation theorem for alternating tree automata. A technicality here is that in the definition of GDA we don't have access to the type of a position. Therefore the GDA has to synthesize the type of every position. This is achieved by the following lemma due to Schwentick and Björklund.

▶ **Lemma 11** ([1])**.** *There is a Data automaton $\mathcal{A}$ which reads a data ω-word and outputs the type of each position.*

Now we present the simulation theorem. The proof is using the standard technique. The GDA guesses a local strategy annotation and verifies that all paths in the annotation are accepting. The only technicality is that the automaton has to rely on the marked path projection to verify that the paths are accepting.

▶ **Proposition 12.** *Given an alternating parity (resp. Büchi) automaton $\mathcal{A}$ there is an equivalent (resp. Büchi) GDA $\mathcal{A}'$.*

Finally we prove the main theorem of this section.

▶ **Theorem 13.** *Satisfiability of bounded-reversal alternation-free μ-calculus is decidable on data ω-words.*

5 Conclusion and future work

In this paper we have introduced a generalization of Data Automata. While the emptiness problem for GDA is open it is shown that the decidability of emptiness of a subclass, namely the class of Büchi GDA, is decidable. Next, a natural fixpoint logic on data words is defined and it is shown that the μ-fragment as well as the alternation-free fragment is undecidable. Then, by limiting the number of change of directions of formulas the class of bounded reversal alternation-free fragment is defined which subsumes other logics such DataLTL and FO^2.

It is shown that satisfiability problem for the bounded-reversal alternation-free fragment is decidable by extending the results for Data automata. In fact the latter result easily extends to the case of formulas with alternation depth $\nu\mu$.

Regarding future work, there are two interesting questions; namely *the decidability of the non-emptiness problem for GDA and the satisfiability problem of the forward fragment.* However these two problems are effectively equivalent since given a GDA $\mathcal{A}$ (*resp.* Büchi) there is an effectively constructed universal parity (*resp.* Büchi) automaton $\mathcal{A}'$ accepting the accepting runs of automaton $\mathcal{A}$. It is also interesting to know if DA is strictly included in (Büchi) GDA.

References

1 H. Björklund and T. Schwentick. On notions of regularity for data languages. *Theor. Comput. Sci.*, 411(4-5):702–715, 2010.
2 M. Bojańczyk. Data monoids. In *STACS*, pages 105–116, 2011.
3 M. Bojańczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Log.*, 12(4):27, 2011.
4 M. Bojańczyk and S. Lasota. An extension of data automata that captures xpath. In *Logic in Computer Science (LICS), 2010*, pages 243–252, July 2010.
5 T. Colcombet, C. Ley, and G. Puppis. On the use of guards for logics with data. In *MFCS*, volume 6907 of *LNCS*, pages 243–255. Springer, 2011.
6 S. Demri, D. Figueira, and M. Praveen. Reasoning about data repetitions with counter systems. In *Logic in Computer Science (LICS), 2013*, pages 33–42, June 2013.
7 S. Demri and R. Lazić. LTL with the freeze quantifier and register automata. *ACM Transactions on Computational Logic*, 10(3), April 2009.
8 D. Figueira. Alternating register automata on finite data words and trees. *Logical Methods in Computer Science*, 8(1), 2012.
9 D. Figueira. Decidability of downward XPath. *ACM Transactions on Computational Logic*, 13(4), 2012.
10 E. Grädel, W. Thomas, and T. Wilke, editors. *Automata Logics, and Infinite Games: A Guide to Current Research.* Springer-Verlag New York, Inc., New York, NY, USA, 2002.
11 O. Grumberg, O. Kupferman, and S. Sheinvald. Variable automata over infinite alphabets. In *Language and Automata Theory and Applications*, pages 561–572. Springer, 2010.
12 M. Jurdziński and R. Lazic. Alternating automata on data trees and xpath satisfiability. *ACM Trans. Comput. Log.*, 12(3):19, 2011.
13 M. Kaminski and N. Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.
14 A. Kara, T. Schwentick, and T. Zeume. Temporal logics on words with multiple data values. In *FSTTCS*, volume 8 of *LIPIcs*, pages 481–492, 2010.
15 L. Libkin and D. Vrgoc. Regular expressions for data words. In *LPAR*, volume 7180, pages 274–288, 2012.
16 A. Manuel, A. Muscholl, and G. Puppis. Walking on data words. In *Computer Science Theory and Applications*, volume 7913 of *LNCS*, pages 64–75. 2013.
17 F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. 5(3):403–435, 2004.