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

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 do 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 a lost information – then the interleaving obfuscates all relevant behaviour of a specific client. Data language, by annotating each action in this sequence by the unique identifier (the data) representing the client that has produced this action, gives access to much more precise informations. An interesting way to analyze the structure of the log is then the ability to navigate in its structure. Typical things that we would like to be able to express are 'what is the next event in the log?', 'what is the next

* 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.

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 that '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, the data words consist of a linear order of positions together with an equivalence relation expressing that to given positions in the word carry the same data values (i.e., a binary relation that expresses that two events where 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-words 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 behaviour 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 its 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. Due to space constraints the proofs have been moved to the appendix.

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-langauge is a set of data ω-words.

We recall some standard notions related to data words. Let $w = (a_1, d_1)(a_2, d_2) \dots (a_n, d_n)$ 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 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, $\Delta \subseteq Q \times (Q \cup \{\perp\}) \times \Sigma \times Q$ 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 $w = (a_1, d_1)(a_2, d_2) \dots$ is a relabelling of w by the states in Q, i.e. $\rho = (q_1, d_1)(q_2, d_2) \dots$ such that the tuple $(q_0, \perp, a_1, q_1)$ 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 $a_1 a_2 \ldots \in A^\omega$ is a sequence of states $s_0 s_1 \ldots \in Q^\omega$ 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 Automata there is an equivalent Bïchi 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 is 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 fsatisfying 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 $\pi = e_1 e_2 \dots e_n \in \{g, c\}^*$ 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_v 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 pairwise 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 lemmas we prove the necessity and sufficiecy of regular witness 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 pairwise.*

▶ **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.** *If the automaton $\mathcal{A}$ accepts some data ω-word then there is a regular witness for the non-emptiness of $\mathcal{A}$.*

▶ **Lemma 6.** *If $\mathcal{A}$ has a regular witness for its non-emptiness then $\mathcal{A}$ accepts some data ω-word.*

From the two lemmas above we conclude that,

▶ **Proposition 7.** *$\mathcal{A}$ accepts a data ω-word if and only if $\mathcal{A}$ has a regular witness for its non-emptiness.*

Using Proposition 7 it is possible to decide if a given GDA has non-empty language. This is achieved by reduction to 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 counters having value 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-empitness of Büchi GDA is decidable.

▶ **Theorem 8.** *Given a Büchi GDA $\mathcal{A}$ one can effectively construct a 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 $\mathit{Prop} = \{p, q, \ldots\}$ be a set of propositional variables. The formulas in the logic are the following. The atomic formulas are, $p \in \mathit{Prop}$, $\neg p$, and $\mathcal{S}, \mathcal{P}, \mathsf{first}^c, \mathsf{first}^g$ which are zeroary modalities. Also, $\mathtt{X}^g\varphi, \mathtt{X}^c\varphi, \mathtt{Y}^g\varphi, \mathtt{Y}^c\varphi$ 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.

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.

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

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

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 classs 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 9.** *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 $Comp^i(\Psi)$ inductively as $Comp^0(\Psi) = \emptyset$ and

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

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

For $\lambda \in \{\mu, \nu\}$ let $\mathsf{Formulas}(\lambda)$ denote the formulas which uses only the fixpoint operator λ. Then the *aternation-free* fragment, denoted as AF, is the set of formulas AF $= Comp\,(\mathsf{Formulas}\,(\mu) \cup \mathsf{Formulas}\,(\nu)))$; 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 BR $= Comp\,(\mathrm{AF}_{\mathtt{X}} \cup \mathrm{AF}_{\mathtt{Y}})$.

Next we prove that the frogment 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 10.** *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 $(v_0, v_1)(v_1, v_2) \ldots (v_k, v_{k+1}) \subseteq E^*$ 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 $\pi = (v_0, v_1)(v_1, v_2) \ldots \in E^\omega$ 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 is 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 powerset of P, namely $\{\varphi_1, \ldots, \varphi_l\}$. 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 $\mathit{tp}\,(i)$, 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 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 $V_A = \mathit{co\text{-}Dom}(\Delta) \times \omega$. 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 $\varphi \in \Delta(p, \lambda(i), \mathit{tp}\,(i))\}$. 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 11.** *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 $\sigma : \omega \to (Q \to \mathit{co\text{-}Dom}(\Delta))$ such that for all i and for all $p \in Q$, $(\sigma(i))(p) \in \Delta(p, \lambda(i), \mathit{tp}\,(i))$. 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 $f : Q \to \mathit{co\text{-}Dom}(\Delta)$ such that there exist $a \in \Sigma, \tau \in \mathbf{P}(\mathcal{M})$ such that for all $p \in \mathit{Dom}(f)$, $f(p) = \Delta(p, a, \tau)$. A local strategy f is consistent at position i if $f(p) \in \Delta(p, \lambda(i), \mathit{tp}\,(i))$ for all $p \in \mathit{Dom}(f)$. 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 futhermore 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 $\mathtt{M} = \mathtt{X}^c$ (*resp.* $\mathtt{M} = \mathtt{X}^g$) 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 annoation 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 12.** *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 dont 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 13** ([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 14.** *Given an alternating parity (resp. Buchi) automaton $\mathcal{A}$ there is an equivalent (resp. Buchi) GDA $\mathcal{A}'$.*

Finally we prove the main theorem of this section.

▶ **Theorem 15.** *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.

APPENDIX

A Proof of Lemma 1

Proof. Let $\mathcal{A} = (Q, \Sigma, \Delta, I, F_g, F_c)$ be a given Data Automaton. We observe that given a data ω-word w, every path of the form $g^k c^\omega$ eventually consists of positions from a single class. We make use of this observation to define a Büchi Generalized Data Automaton $\mathcal{A}' = (Q, \Sigma, \Delta, I, L)$ as follows. Note that the set of states, initial states and transition relation of the automaton $\mathcal{A}'$ are the same as that of the automaton $\mathcal{A}$. The deterministic Büchi language L is the set of all ω-words $u \in (Q \times \{\epsilon, g, c\})^\omega$ such that u belongs to one of the following (disjoint) languages,

(L1) $u \in (Q \times \{\epsilon\}) \cdot (Q \times \{g\})^\omega$ and some state state in F_g appears infinitely often in the first component of u.

(L2) $u = u_1 \cdot u_2$ where $u_1 \in (Q \times \{\epsilon\}) \cdot (Q \times \{g\})^*$ and $u_2 \in (Q \times \{c\})^\omega$ such that some state in F_c appears infinitely often in the first component of u_2.

(L3) $u \in (Q \times \{\epsilon\}) \cdot (Q \times \{g, c\})^\omega$ such that both g and c occurs infinitely often in the second component of u.

Each of (L1), (L2) and (L3) is the limit of a regular language of finite words and hence they are deterministic Büchi recognizable languages. Since finite union of deterministic Büchi recognizable languages is a deterministic Büchi recognizable language, we obtain that the language L is deterministic Büchi recognizable. It remains to show that the automata $\mathcal{A}$ and $\mathcal{A}'$ are equivalent. By virtue of definition every run of $\mathcal{A}$ is a run of $\mathcal{A}'$ and vice versa. Hence it suffices to show that a run ρ is accepting for automaton $\mathcal{A}'$ if and only if it is accepting for automaton $\mathcal{A}$. The left-to-right direction is easy; assume ρ is an accepting run of the automaton $\mathcal{A}$. Since the global path in the run ρ belongs to language (L1) and every path which eventually takes only class edges belongs to language (L2) the run ρ satisfies the acceptance criterion of the Data Automaton $\mathcal{A}$. Conversely, assume ρ is an accepting run of the Data Automaton $\mathcal{A}$. If π is a path that is composed eventually of only global edges (*resp.* class edges) then the marked path projection of the run ρ w.r.t. π contains infinitely often a state from F_g (*resp.* F_c). This takes care of Conditions (L1) and (L2). Otherwise the marked path projection of π contains infinitely many letters c (denoting class edges) and g (denoting global edges), in which case Condition (L3) is met. Hence every path in ρ satisfies one of Conditions (L1) or (L2) or (L3). Hence ρ is an accepting run of the GDA $\mathcal{A}'$. ◀

B Proof of Lemma 2

Proof. Let $\mathcal{A}_1 = (Q_1, \Sigma, \Delta_1, I_1, L_1)$ and $\mathcal{A}_2 = (Q_2, \Sigma, \Delta_2, I_2, L_2)$ be two GDA with disjoint sets of states.

The union of languages $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$ is accepted by the Generalized Data Automaton $\mathcal{A}_1 \cup \mathcal{A}_2 = (Q_1 \cup Q_2, \Sigma, \Delta_1 \cup \Delta_2, I_1 \cup I_2, L_1 \cup L_2)$. This claim also holds when $\mathcal{A}_1$ and $\mathcal{A}_2$ are Büchi GDA, since $L_1 \cup L_2$ is a deterministic Büchi recognizable language when L_1 and L_2 are deterministic Büchi recognizable.

Closure under intersection is by usual product construction. Define the product $\mathcal{A} = (Q = Q_1 \times Q_2, \Sigma, \Delta, I = I_1 \times I_2, L)$ where the set of transitions Δ and acceptance criterion

L are defined as follows;

$$((p_1,q_1),(p_2,q_2),a,(p_3,q_3)) \in \Delta \text{ if } (p_1,p_2,a,p_3) \in \Delta_1, (q_1,q_2,a,q_3) \in \Delta_2 \ ,$$
$$L = \{((p_i,q_i),e_i)_{i\in\omega} \in (Q \times \{\epsilon,g,c\})^\omega \mid (p_i,e_i)_{i\in\omega} \in L_1 \text{ and } (q_i,e_i)_{i\in\omega} \in L_2\} \ .$$

The language L is ω-regular (*resp.* deterministic Büchi recognizable) if L_1 and L_2 are ω-regular (*resp.* deterministic Büchi recognizable). The automaton $\mathcal{A}$ has an accepting run on a data ω-word w iff both automaton $\mathcal{A}_1$ and automaton $\mathcal{A}_2$ have accepting runs on w. Hence the automaton $\mathcal{A}$ accepts the intersection of the languages $L(\mathcal{A}_1)$ and $L(\mathcal{A}_2)$.

Next we consider closure under renaming. Let $h : \Sigma \to \Gamma$ be a renaming. The renaming of $\mathcal{A}_1$ under h is the automaton $h(\mathcal{A}_1) = (Q_1, \Gamma, \Delta_1', I_1, L_1)$ where Δ_1' is obtained by replacing each transition (p_1,p_2,a,p_3) in Δ_1 by $(p_1,p_2,h(a),p_3)$. The automaton $\mathcal{A}_1$ accepts a data ω-word $(a_1,d_1)(a_2,d_2)\ldots \in (\Sigma \times \mathcal{D})^\omega$ iff the automaton $h(\mathcal{A}_1)$ accepts the data ω-word $(h(a_1),d_1)(h(a_2),d_2)\ldots \in (\Gamma \times \mathcal{D})^\omega$. Hence the automaton $h(\mathcal{A}_1)$ accepts the language $h(L(\mathcal{A}_1))$. This claim also holds for Büchi GDA since construction leaves the acceptance criterion L_1 unchanged.

Finally we consider the closure under composition. Let $\mathcal{A}_1 = (Q_1, A \times B, \Delta_1, I_1, L_1)$ and $\mathcal{A}_2 = (Q_2, B \times C, \Delta_2, I_2, L_2)$ be two GDA (*resp.* deterministic Büchi GDA). Define the cascade comosition of $\mathcal{A}_1$ and $\mathcal{A}_2$, in notation $\mathit{Comp}(\mathcal{A}_1,\mathcal{A}_2)$, as the automaton $(Q = Q_1 \times Q_2, A \times C, \Delta, I_1 \times I_2, L)$ where

$$((p_1,q_1),(p_2,q_2),(a,c),(p_3,q_3)) \in \Delta \text{ if } \exists b \in B \ s.t. \ (p_1,p_2,(a,b),p_3) \in \Delta_1, \\ (q_1,q_2,(b,c),q_3) \in \Delta_2 \ .$$

$$L = \{((p_i,q_i),e_i)_{i\in\omega} \in (Q \times \{\epsilon,g,c\})^\omega \mid (p_i,e_i)_{i\in\omega} \in L_1 \text{ and } (q_i,e_i)_{i\in\omega} \in L_2\} \ .$$

The automaton $\mathit{Comp}(\mathcal{A}_1,\mathcal{A}_2)$ accepts the language $\mathit{Comp}(\mathcal{L}_1,\mathcal{L}_2)$ and moreover $\mathit{Comp}(\mathcal{A}_1,\mathcal{A}_2)$ is a Büchi GDA if $\mathcal{A}_1$ and $\mathcal{A}_2$ are Büchi GDA.

◀

C Proof of Lemma 5

Proof. Assume that the automaton $\mathcal{A}$ accepts some data ω-word $w = (a_1,d_1)(a_2,d_2)\ldots$ and let $\rho \in (Q \times \mathcal{D})^\omega$ be an accepting run of $\mathcal{A}$ on w and let $\zeta \in ((Q \times \mathbf{P}(S)) \times \mathcal{D})^\omega$ be the annotated run. Let the annotated run ζ be the word $((q_1,S_1),d_1)((q_2,S_2),d_2)\ldots$. By pigeonhole principle there is an infinite sequence of positions $\bar{i} = i_0, i_1, \ldots$ in the annotated run ζ such that $(q_{i_0}, S_{i_0}) = (q_{i_1}, S_{i_1}) = \ldots$. Conditions (i) and (ii) of regular witness are satisfied by the prefixes of the data ω-word w defined by any two positions in $\bar{i}$. Let $C_{i_0}, C_{i_1}, \ldots$ be the class vectors of the prefixes of the annotates run ζ at positions $i_0, i_1, \ldots$ respectively. By Dickson's lemma there exists an infinite subsequence $\bar{j} = j_0, j_1, \ldots$ of $\bar{i}$ such that the class vectors $C_{j_0} \le C_{j_1} \le \ldots$ in the pairwise ordering. Condition (iii.a) of the regular witness is satisfied by the prefixes of data ω-word w defined by any two positions in $\bar{j}$. Let x be a pair in $Q \times \mathbf{P}(B)$. Either for all positions $j \in \bar{j}$ it is the case that $C_j(x) = 0$ (in which case we set the variable $J(x)$ to be the position j_0), otherwise there is a position $j_k \in \bar{j}$ such that for all $j_k \le j_l \in \bar{j}$ it holds that $C_{j_l}(x) > 0$ (since the vectors are non-decreasing by construction). In the last case we set the variable $J(x)$ to the position j_k. We take $\mathbf{J}$ to the maximum over all positions $J(x)$ where $x \in Q \times \mathbf{P}(B)$. We denote the suffix of $\bar{j}$ defined by $\mathbf{J}$ (containing only positions after $\mathbf{J}$) by $\bar{l} = l_0, l_1, \ldots$. Condition (iii.b) of the regular witness is satisfied by the prefixes of data ω-word w defined by any two positions in $\bar{l}$. We take the finite data word u to be the prefix of the data ω-word defined by position l_0.

It only remains to define the finite data word v. We aim at defining the data word v to be the infix of w defined by positions l_0 (excluding) and a chosen (below) position such that Condition (iii.c) is satisfied. Choose an arbitrary position $1 \leq k \leq l_0$ which is class maximal in u such that the annotation at position k is $(q, S') \in Q \times \mathbf{P}(S)$. Let s be a state in S' and $\pi \in \{g, c\}^\omega$ be an infinite path starting from position k. We denote by $r_{\pi,s}$ the partial run of the automaton $\mathcal{B}$ on the marked path projection of the path π from the state s. Given a position i, let $\text{prefix}(r_{\pi,s}, i)$ denote the prefix of the partial run $r_{\pi,s}$ up to position i. We claim that $(\star)$ *there exist a position* $N(k) \in \bar{l}$ *such that for any state* s *in* S' *and any infinite path* $\pi \in \{g, c\}^\omega$ *from position* k *the prefix* $\text{prefix}(r_{\pi,s}, N(k))$ *contains a Büchi state.* Assume it is not the case. For $l \in \bar{l}$ let L_l be the set of all $\text{prefix}(r_{\pi,s}, l)$, where $\pi \in \{g, c\}^\omega$ and $s \in S'$, that does not contain a Büchi state . By assumption for each $l \in \bar{l}$ the set L_l is nonempty. Therefore the set $L = \cup_{l \in \bar{l}} L_l$ is infinite, hence by König's lemma there is some state s in S' and an infinite path π from position k such that the run $r_{\pi,s}$ of automaton $\mathcal{B}$ on the marked path projection of π has infinitely many prefixes in L. Therefore the partial run $r_{\pi,s}$ contains no Büchi states. This contradicts the assumption that the run ρ is accepting. Hence the claim $(\star)$ is established. Let N be the maximum of all $N(k) \in \bar{l}$ for all class maximal positions k in the data word u. We define the data word v to be the infix of the data ω-word w defined by positions l_0 (excluding) and N (including). By construction Condition (iii.c) is satisfied by the data word $u \cdot v$.

◀

D Proof of Lemma 6

Proof. Let $w = u \cdot v$ be a finite data word, ρ be a partial run of the automaton $\mathcal{A}$ on w and ζ be the corresponding annotated run. Our strategy, as mentioned already, is to construct a data ω-word inductively by appending a renamed copy (renaming of only data values) of the data word v to the end of the data word constructed so far. At each stage of the induction we maintain that there is a partial run of the automaton $\mathcal{A}$ on the data word constructed so far. Moreover each run of the automaton $\mathcal{B}$ on the marked path projections of the run of $\mathcal{A}$ sees a Büchi state.

Let w_i denote the data word constructed at induction step i. For the base case we take $w_0 = w$ and by definition there is a partial run ρ of $\mathcal{A}$ on w and an annotation ζ. For the inductive step assume the data word w_i has been constructed and let ρ_i be a partial run of automaton $\mathcal{A}$ on w_i and let ζ_i be an annotation. We aim at constructing a data word v' that is obtained from the data word v by renaming the data values, in other words v and v' has the same string projection and classes. For each data value d in v that does not appear in u, in other words a class that is composed of only positions in v, we label the class of d with a fresh data value (one that does not appear in w_i). Next assume there is a class, labelled by data value d, that contains both positions in u and v. Let k be the minimum position of this class in the data word v and let k' be the maximum position of this class in the data word u. Let $(q, S') \in Q \times \mathbf{P}(S)$ be the annotation of the position k'. To ensure consistency we have to choose a class-maximal position in the data word w_i that has the annotation (q, S'). Moreover this choice has to be done so that every class maximal position is assigned a class successor at some step of the induction (this is due to our assumption that all classes are infinite). Hence we choose the minimum class-maximal position in w_i, let it be labelled by data value d_1. We label every position in the class of d in the data word v'

with data value d_1. We repeat this procedure for every such class that contains positions in both u and v. This step is possible since it is guaranteed by the definition of regular witness (Condition (iii.a)). Finally we let the data word $w_{i+1} = w_i \cdot v'$. The run and the annotation are extended similarly, i.e. $\rho_{i+1} = \rho_i \cdot \rho_{v'}$ $\zeta_{i+1} = \zeta_i \cdot \zeta_{v'}$ where the data words $\rho_{v'}$ and $\zeta_{v'}$ are obtained by ρ_v and ζ_v by applying the relabelling of data values used to obtain the data word v'. The run and annotation ρ_{i+1} and ζ_{i+1} is consistent due to the Conditions (i) and (ii). This concludes the induction step.

This way we define the data ω-word w_ω, the run ρ_ω and the annotation ζ_ω. To see that ρ_ω is an accepting run of automaton $\mathcal{A}$ on w_ω it is enough to see that for every path π in ρ_ω, the marked path projection of π is accepted by the automaton $\mathcal{B}$. The fact that there is a run of automaton $\mathcal{B}$ on such a marked path projection follows from the existence of the annotation ζ_ω, and the fact that the run is accepting follows from Condition (iii.c). This is because at every induction step the partial run of automaton $\mathcal{B}$ on each path sees a Büchi state. ◀

E Proof of Theorem 8

Proof. We construct a Multicounter Automaton $\mathcal{M}$ that guesses and verifies a regular witness $w = u \cdot v, \rho, \zeta$ for the automaton $\mathcal{A}$. This guessing is done in a symbolic manner, that is to say, the automaton does not store the data word in its memory but incrementally guesses a data word and simulates both the automata ($\mathcal{A}$ and $\mathcal{B}$) on the guessed data word. The machine $\mathcal{M}$ works in two phases. The first phase ends when the machine $\mathcal{M}$ nondeterministically decides that it has guessed the data word u. Next we describe both the phases in detail.

In the first phase the states of the Multicounter Automaton $\mathcal{M}$ are pairs from the set $Q \times \mathbf{P}(S)$ and it has two counters (referred to as *copy 1* and *copy 2*) for each element in the set $Q \times \mathbf{P}(S)$. During the first phase the machine performs exactly the same operations on both copies of counters. So we describe only what happens to one of the copies. During the run the counter $(q, S') \in Q \times \mathbf{P}(S)$ maintains the number of class maximal positions i in the prefix guessed so ar that are labelled with the pair (q, S') by the annotation. The Multicounter Automaton starts in a state $(q_0, \{s_{in}\})$ where q_0 is an initial state of automaton $\mathcal{A}$ and s_{in} is the initial state of automaton $\mathcal{B}$. During the run let the machine $\mathcal{M}$ be in a state $(q_1, S_1) \in Q \times \mathbf{P}(S)$. It selects non-deterministically a letter $a \in \Sigma$ and executes one of the following actions.

- The machine $\mathcal{M}$ guesses that the next position in the data word u is the first position of a new class. In this case the next state of the machine is a state (q_2, S_2) such that q_2 is the state of automaton $\mathcal{A}$ and S_2 is the end states of runs of automaton $\mathcal{B}$ after reading the current position, in other words there is a transition $(q_1, \perp, a, q_2)$ in Δ and $S_2 = \{s \in S \mid (s_1, (q_2, g), s) \in T, s_1 \in S_1\}$. To indicate that number of the class-maximal positions labelled with (q_2, s_2) has gone up by one, the machine $\mathcal{M}$ increases the counter (q_2, S_2).
- The machine $\mathcal{M}$ guesses that the next position in data word u occurrs in an existing class. In this case the machine selects a counter (q', S') and decrements it by one, which symbolically corresponds to selecting a class maximal position with annotation (q', S'). The next state of the machine $\mathcal{M}$ machine is a pair (q_2, S_2) such that q_2 is the state of automaton $\mathcal{A}$ (i.e. there is a transition (q_1, q', a, q_2) in Δ) and S_2 is the end states of runs of automaton $\mathcal{B}$ after reading the current position. Note that the runs of the automaton $\mathcal{B}$ could take the global as well as the class edge to reach the position. Hence we define

$$S_2 = \{s \in S \mid (s_1, (q_2, g), s) \in T, s_1 \in S_1\} \bigcup \{s \in S \mid (s', (q_2, c), s) \in T, s' \in S'\} \ .$$

Finally the counter (q_2, S_2) is incremented by one.

At some point during the run the machine $\mathcal{M}$ guesses that the first phase is over and the machines prepares to enter the second phase in which it guesses the data word v. In the second phase the states as well as the counters of the machine are of the form $Q \times \mathbf{P}(S) \times \mathbf{P}(S)$. Intuitively the counter (p, S_1, S_2) counts the number of class-maximal positions i such that i is labelled with the state p (by the run of $\mathcal{A}$) and the set of states S_1 by the annotation (corresponding to the end states of runs of $\mathcal{B}$) and moreover the subset $S_2 \subseteq S_1$ of states is the end states of runs of automaton $\mathcal{B}$ that have not seen a Büchi state (in G) in a position in the second phase (in the data word v). Similarly if the state of the machine $\mathcal{M}$ is (p, S_1, S_2) (say on position i) then it means that i is labelled with the state p (by the run of $\mathcal{A}$) and the set of states S_1 by the annotation (corresponding to the end states of runs of $\mathcal{B}$) and moreover the subset $S_2 \subseteq S_1$ of states corresponds to the end states of runs of automaton $\mathcal{B}$ that has not seen a Büchi state (in G) in a position in the data word v.

During the preparation to second phase each counter $(q, S_1) \in Q \times \mathbf{P}(S)$ in copy 2 is copied to the counter $(q, S_1, S_1 \setminus G)$. This is achieved in the following way. The machine $\mathcal{M}$ nondeterministically decrements the counter (q, S_1) and increments the counter $(q, S_1, S_1 \setminus G)$ and repeats this operation until the machine $\mathcal{M}$ guesses that the counter (q, S_1) is zero. However the machine $\mathcal{M}$ cannot check that the counter (q, S_1) is zero, instead the counter is not touched till the run of $\mathcal{M}$ is finished at which point the counter is verified to be zero (by the zero test at the end of the run). After transferring the values from counters $(Q \times \mathbf{P}(B))$ to counters $(Q \times \mathbf{P}(B) \times \mathbf{P}(B))$ the machine $\mathcal{M}$ enters phase 2.

After finishing the preparation the machine $\mathcal{M}$ moves to the state $(q, S_1, S_1 \setminus G)$ given that it finished phase 1 in state (q, S_1). At any point during this phase if the machine is in state (q, S_1, S_2) it guesses the label a of the next position and one of the following two scenarios can happen.

- It is guessed that the next position is the beginning of a new class. In this case, the next state is of the form (q', S_1', S_2') where $(q, \perp, a, q')$ is a transition Δ and the set of states $S_1' = \{s' \in S \mid (s, (q', g), s') \in T, s \in S_1\}$ and $S_2' = S_1' \setminus G$. Subsequently the counter (q', S_1', S_2') is incremented.
- On the other hand if the the next position is guessed to be in an existing class a counter (p, R_1, R_2) is chosen nondeterministically and is decremented. The next state of the machine $\mathcal{M}$ is a triple (q', S_1', S_2') such that (q, p, a, q') is a transition in Δ, the set of states
$$S_1' = \{s' \in S \mid (s, (q', g), s') \in T, s \in S_1\} \bigcup \{s' \in S \mid (r, (q', c), s') \in T, r \in R_1\} ,$$
and $S_2' = S_1' \setminus G$. As before the counter (q', S_1', S_2') is incremented.

At the end of phase two the machine $\mathcal{M}$ enters a final verification phase. In this phase the machine decrements simultaneously counters (q, S_1) from copy 1 and the counter $(q, S_1, \emptyset)$. This is repeated until it is guessed that one of them is zero. This is repeated for all pairs of counters (q, S_1) from copy 1 and $(q, S_1, \emptyset)$. Finally the automaton accepts if the state is of the form $(q, S_1, \emptyset)$ where (q, S_1) is the state reached at the end of phase 1 and all counters are empty. ◀

F Proof of Theorem 17

First we discuss how to implement the standard temporal operators in the logic. The formula $\varphi \, \mathtt{U}^g \, \psi$ holds if ψ holds in the future, and φ holds in between. This can be implemented as

$\mu x.\psi \vee (\varphi \wedge \mathtt{X}^g x)$ The formula $\varphi\,\mathtt{U}^c\,\psi = \mu x.\psi \vee (\varphi \wedge \mathtt{X}^c x)$ is similar, but for the fact that it refers only to the class of the current position. The formula $\mathtt{F}^g\varphi$ abbreviates $\top\,\mathtt{U}^g\,\varphi$, and its dual is $\mathtt{G}^g\varphi = \neg\mathtt{F}^g\neg\varphi$. The constructs $\mathtt{S}^g$, $\mathtt{S}^c$, $\mathtt{P}^g$, $\mathtt{P}^c$, $\mathtt{H}^g$ and $\mathtt{H}^c$, are defined analogously, using past modalities, and correspond respectively to $\mathtt{U}^g$, $\mathtt{U}^c$, $\mathtt{F}^g$, $\mathtt{F}^c$, $\mathtt{G}^g$ and $\mathtt{G}^c$.

Consider a data word that uses, say, letters a, b, c, and such that the relation $\sim$ between positions is a bijection between a-labeled positions and b-labeled positions. It is easy to write a μ-calculus formula that checks this property. However, this is not yet sufficient for our purpose. We need the following lemma.

▶ **Lemma 16.** *The exists a formula in the μ-fragment that checks over finite data words the property that $\sim$ is an increasing bijection between a-labeled positions and b-labeled positions.*

Proof. For the sake of explanations, let us consider a data word u, and let A (*resp.* B) be the set of a-labeled (*resp.* b-labeled) positions in u. Let R be $\sim$ restricted to $A \times B$. We have to provide a formula that holds if R is a monotonic bijection between A and B. It is easy to write a formula of the μ-fragment that holds if and only if R is a bijection between A and B. We assume this is the case from now.

Consider now the binary relation $S \subseteq A^2$ such that $x\,S\,z$ if $x\,R\,x' < y'\,R^{-1}\,y < z$. An element $x \in A$ such that $x\,S\,x$ is called a *small witness*. Note first that the the existence of a small witness means that there exists $x > y$ and $x' < y'$ such that $x\,R\,x'$ and $y\,R\,y'$. Hence, there exists a small witness if and only if R is not increasing. Unfortunately, we are not able to directly detect the existence of a small witness using a μ-formula. Instead, we will search for 'big witnesses'. A *big witness* is a sequence $x_1, x_2, \ldots$ of elements of A such that

$$x_1\,S\,x_2\,S\ldots$$

We claim ($\star$) that there exists a small witness if and only if there exists a big witness. Of course, if there is a small witness, there is a big one. Assume now that there exists a big witness $x_1, \ldots$ Since the x_i's range over a finite domain, there exists i such that $x_{i+1} \leq x_i$. Thus, $x_i\,S\,x_{i+1} \leq x_i$ and hence $x_i\,S\,x_i$. we have found a small witness.

One easily verifies now that the μ-formula

$$\mathtt{F}^g\nu x.a \wedge \mathtt{F}^c\mathtt{P}^c(b \wedge \mathtt{X}^g\mathtt{F}^g(b \wedge \mathtt{F}^c\mathtt{P}^c(a \wedge \mathtt{X}^g\mathtt{F}^g x))))$$

expresses the existence of a big witness. Thus the non-existence of a big witness, hence of a small witness, hence the non increasing nature of R is definable by a μ-formula. *A priori*, this formula is a formula that uses both μ- and ν-fixpoints since the modalities $\mathtt{F}^c$ and $\mathtt{F}^g$ are in fact syntactic sugar for formulas of the μ-fragment. However, it is easy to check that, over *finite data words*, $\mathtt{F}^g(\varphi)$ is equivalent to $\nu x.\varphi \wedge \mathtt{X}^g x$ (the difference between least and greatest fixpoint does not exist when the fixpoints are reached within a finite number of steps). Thus, the above formula can be expressed in the ν-fragment, and hence its complement in the μ-fragment. ◀

Using this lemma we reduce the Post's correspondence problem to the satisfiability problem of the logic giving us,

▶ **Theorem 17.** *Satisfiability of the μ-fragment over finite data words is undecidable.*

Proof. The proof is by reduction from the Post's Correspondence Problem (PCP). An instance I of PCP is a finite set of tuples $I = \{(u_1, v_1), \ldots, (u_k, v_k) \mid u_j, v_j \in \Sigma^+\}$. A solution to I is a sequence $i_0 \ldots i_n \in [k]^+$ such that $u_{i_0} \ldots u_{i_n} = v_{i_0} \ldots v_{i_n}$. It is well known that the problem of determining if an instance of the PCP has a solution is undecidable.

Given an instance I of the PCP, we construct a formula in the μ-fragment that is satisfiable if and only if I has a solution. For this, we encode the solution of I as a data word u over the alphabet $\Sigma \uplus \{a, b\}$ (where a, b are assumed not present in Σ). Intuitively, u is $u_{i_0} \dots u_{i_n}$ in which are inserted letters a and b letters in order to describe the decomposition in $u_{i_0}, \dots, u_{i_n}$ (using a's) and in $v_{i_0}, \dots, v_{i_n}$ (using b's). The data values are required to induce an increasing bijection between a-labeled and b-labeled positions in order to be able to check the correctness of the solution. Formally, a data word u *encodes* the solution $i_0 \dots i_n$ to I if:

- the word has length at least 4, starts with letters ab and ends with ab, and
- $\sim$ induces an increasing bijection between a-labeled positions and b-labeled positions. Let $x_0 < \dots < x_n$ be the a-labeled positions and $y_0 < \dots < y_n$ be the b-labeled positions.
- Then for all $\ell = 1 \dots n$, the word obtained as the string projection of u restricted to the positions in $(x_\ell, x_{\ell+1})$ (*resp.* $(y_\ell, y_{\ell+1})$)to which b-letters (*resp.* a-letters) are removed is u_{i_ℓ} (*resp.* v_{i_ℓ}).

It is easy, from a solution to construct a data word that encodes it.

Hence, in order to guess a solution to I, it is sufficient to guess a data word over the alphabet $\Sigma \cup \{a, b\}$ such that (†):

- the word has length at least 4, starts with letters ab and ends with ab, and
- $\sim$ induces an increasing bijection between a-labeled positions and b-labeled positions, and there is at least one occurrence of a;
- for all occurrences x of an a-letter, but the last one, there exists $i \in [k]$ such that:
  - the string projection of u starting at position x belongs to $K_i = \{w : \overline{w}^b \in au_ia(\Sigma \cup a)^*\}$ where $\overline{w}^b$ is the word w with letter b removed, and
  - the string projection of u starting at position $R(x)$ belongs to $L_i = \{w : \overline{w}^a \in bv_ib(\Sigma \cup b)^*\}$ where $\overline{w}^a$ is the word w with letter a removed.

Quite naturally, if a data word encodes a solution to I then it satisfies (†). Conversely, if a data word satisfies (†), then there exists a solution to I that it encodes.

Thus, it is sufficient for us to write a formula of the μ-fragment for (†), which is easy using Lemma 16 for the second item, and the fact that the languages K_i and L_i are regular, thus definable by a formula of the μ-fragment. ◀

The above theorem extends to ω-words.

▶ **Corollary 18.** *Satisfiability of the μ-fragment over data ω-words is undecidable.*

Proof. Consider a formula φ of the μ-fragment, our goal is to construct a formula $\varphi^\sharp$ such that φ is satisfiable over data words if and only if $\varphi^\sharp$ is satisfiable over ω-data words. In combination with Theorem 17, this proves the statement.

The formula $\varphi^\sharp$ (for $\sharp$ a new fresh symbol) defines the data ω-words w such that:

- w contains at least one occurrence of the letter $\sharp$,
- the data ω-word w restricted to the positions that are to the left of all $\sharp$-occurrences satisfy φ.

Of course, if we can write such a formula, then it is satisfiable over data ω-words if and only if φ is satisfiable over data words. It is also clear that the first item is definable in the μ-fragment. Thus, we just have to turn φ into a formula that is sensitive only to the part of the word left of all $\sharp$'s. This is exactly the classical technique of relativization. Remark first that the property 'being at the left of all $\sharp$' is definable in the μ-fragment. Let ψ be such a formula. In our case, relativizing φ to ψ consists in replacing syntactically every subformula of the form $\mathsf{M}(\gamma)$ for some modality $\mathsf{M} \in \{\mathtt{X}^c, \mathtt{X}^g, \mathtt{Y}^c, \mathtt{Y}^g\}$ by $\mathsf{M}(\gamma \wedge \psi)$, last^g by $\mathtt{X}^g \sharp$ and last^c by $\mathsf{last}^c \vee \mathtt{X}^c\, \mathtt{S}^g\, \sharp$. The result is a formulas that holds over a word if and only if φ holds on the input restricted to its longest $\sharp$-free prefix. ◀

G Proof of Lemma 10

First we need the following lemma.

▶ **Lemma 19.** *Let $\varphi(x, \bar{y})$ be a formula such that the only unary modalities it uses are $\mathtt{Y}^g, \mathtt{Y}^c$ and furthermore any free occurrence of x appears in the scope of at least k nested modalities. Then for any data word (resp. data ω-word) w and valuation $S_1, \ldots, S_l$ of $\bar{y} = y_1, \ldots, y_l$, and S of x, and for all $i < k$,*

$$w[\ell(\bar{y}) := \bar{S}, \ell(x) = S], i \models \varphi \\ \Leftrightarrow w[\ell(\bar{y}) := \bar{S}, \ell(x) = \emptyset], i \models \varphi \ .$$

Proof. Without loss of generality assume that x is not a bound variable in $\varphi(x, \bar{y})$ (otherwise rename the occurrences of x). We proceed by an induction on the pair (k, i) ordered lexicographically (for all $i \geq k$ the claim holds trivially); For the base case when $k = 1$, the claim is vacuously true. For the inductive step assume the claim is true for pairs (k', i') where $k' < k$ or, $k' = k$ and $i' < i$. Let $\varphi(x, \bar{y})$ be a formula in which x appears with in the scope of $k + 1$ nested modalities. We do an induction on the structure of the formula. Let $\varphi(x, \bar{y})$ is of the form $\mathtt{M}\psi(x, \bar{y})$ where $\mathtt{M} \in \{\mathtt{Y}^g, \mathtt{Y}^c\}$. We do a case analysis on $\mathtt{M}$. Assume $\mathtt{M}$ is $\mathtt{Y}^g$ (the case when $\mathtt{M}$ is $\mathtt{Y}^c$ being analogous) then

$$\begin{aligned} & w[\ell(\bar{y}) := \bar{S}, \ell(x) = S], i \models \mathtt{M}\psi(x, \bar{y}) \\ & \quad \Leftrightarrow w[\ell(\bar{y}) := \bar{S}, \ell(x) = S], i-1 \models \psi(x, \bar{y}) && \text{(By defn. of } \mathtt{Y}^g\text{)} \\ & \quad \Leftrightarrow w[\ell(\bar{y}) := \bar{S}, \ell(x) = \emptyset], i-1 \models \psi(x, \bar{y}) && (i < k \Rightarrow i-1 < k-1, \text{ hence by IH}) \\ & \quad \Leftrightarrow w[\ell(\bar{y}) := \bar{S}, \ell(x) = \emptyset], i \models \mathtt{M}\psi(x, \bar{y}) \end{aligned}$$

The boolean cases are straightforward. Next assume $\varphi(x, \bar{y})$ is of the form $\theta y_i.\psi(x, \bar{y})$ ($\theta \in \{\mu, \nu\}$). We have to show that

$$w[\ell(\bar{y}) := \bar{S}, \ell(x) = S], i \models \theta y_i.\psi(x, \bar{y}) \\ \Leftrightarrow w[\ell(\bar{y}) := \bar{S}, \ell(x) = \emptyset], i \models \theta y_i.\psi(x, \bar{y}) \ .$$

By induction hypothesis (on the structure of the formula)

$$w[\ell(\bar{y}) := \bar{S}, \ell(x) = S], i \models \psi(x, \bar{y}) \\ \Leftrightarrow w[\ell(\bar{y}) := \bar{S}, \ell(x) = \emptyset], i \models \psi(x, \bar{y}) \ .$$

Hence S_i is a pre-fixpoint (*resp.* post-fixpoint) of $\psi(x, \bar{y})$ on $w[\ell(\bar{y}) := \bar{S}, \ell(x) = S]$ if and only if it is a pre-fixpoint (*resp.* post-fixpoint) of $\psi(x, \bar{y})$ on $w[\ell(\bar{y}) := \bar{S}, \ell(x) = \emptyset]$. Hence the claim is proved by Knaster-Tarski theorem. This concludes the induction. ◀

Next we prove that every formula in the backward fragment is equivalent to a *nu*-formula. This is done in two steps. The first step is to transform the formula in BR to an equivalent one that is furthermore guarded. This is achieved by the following lemma.

▶ **Lemma 20.** *Every formula is equivalent to a formula which is furthermore guarded.*

Proof. Proof is by induction on the structure of the formula. The atomic, boolean and modal cases are straightforward. The non-trivial case is when the formula is of the form $\lambda x.\varphi(x)$. Assume $\lambda x.\varphi(x)$ is unguarded and $\varphi(x)$ is guarded. We can furthermore assume that all unguarded occurrences of x is outside of any subformula $\theta y.\psi(x, y)$ of $\varphi(x)$, otherwise in $\varphi(x)$ we substitute for $\theta y.\psi(x, y)$ the equivalent formula $\psi(x, \theta y.\psi(x, y))$ which yields the desired form. Next we write $\varphi(x)$ is conjunctive normal form to obtain a formula of the form

$$\lambda x.(x \vee \alpha(x)) \wedge \beta(x),$$

where $\alpha(x)$ and $\beta(x)$ are guarded. It is left to the reader to check that

$$\mu x.(x \vee \alpha(x)) \wedge \beta(x) \equiv \mu x.\alpha(x) \wedge \beta(x)\,,$$

and

$$\nu x.(x \vee \alpha(x)) \wedge \beta(x) \equiv \nu x.\beta(x)\,.$$

◀

Thanks to the previous lemma it is enough to consider only guarded formulas,

▶ **Theorem 21.** *Every guarded formula in the backward fragment is has a unique fixpoint.*

Proof. Observe that by induction on the structure of the formula it is enough to verify that for every guarded formula $\psi = \mu x.\varphi(x, \bar{y})$ and for every data ω-word w and valuation $S_1, \ldots, S_k$ (all of them subsets of $[n]$) of $\bar{y} = y_1, \ldots, y_k$,

$$[\![\nu x.\varphi(x, \bar{y})]\!]_{w'} \subseteq [\![\mu x.\varphi(x, \bar{y})]\!]_{w'}$$

where $w' = w[\ell(y_1) := S_1, \ldots, \ell(y_k) := S_k]$, since the other inclusion follows from the fact that the least fixpoint is always included in the greatest fixpoint. This reduces to showing that

$$w', i \models \nu x.\varphi(x, \bar{y}) \Rightarrow w', i \models \mu x.\varphi(x, \bar{y})$$

This is exhibited by the following calculation,

$$\begin{aligned}
w', i \models \nu x.\varphi(x, \bar{y}) &\Leftrightarrow w', i \models \varphi(\nu x.\varphi(x, \bar{y}), \bar{y}) && \text{(By fixpoint iteration)}\\
&\Leftrightarrow w', i \models \varphi^{n+1}(\nu x.\varphi(x, \bar{y}), \bar{y}) && \\
&\Rightarrow w', i \models \varphi^{n+1}(\bot, \bar{y}) && \text{(By Lemma 19)}\\
&\Rightarrow w', i \models \mu x.\varphi(x, \bar{y}) && \text{(By Knaster-Tarski theorem)}
\end{aligned}$$

◀

We represent unique fixpoints using the symbol θ. Due to the above theorem it is enough to consider formulas in which every fixpoint subformula is of the form $\theta x.\varphi(x)$. Now we can prove our theorem,

▶ **Theorem 22.** *For every formula φ in the ν-fragment there is an effectively constructed Data ω-automaton $\mathcal{A}_\varphi$ such that $\mathcal{A}_\varphi$ labels each position with the set of subformulas of φ true at that position.*

Proof. We need the following definitions. Let $\mathrm{Prop}(\varphi)$ be the set of all propositional variables used in φ, and let $\mathrm{Sub}(\varphi)$ be the set of all subformulas of φ.

▶ **Definition 23.** The *closure* $\mathrm{CL}(\varphi)$ of φ is the smallest set such that,

1. $\mathrm{Prop}(\varphi) \cup \{\varphi, \mathcal{S}, \mathcal{P}, \mathsf{first}^c, \mathsf{first}^g\}$ and their negations belong to $\mathrm{CL}(\varphi)$,
2. If $\psi \in \mathrm{CL}(\varphi)$ then $\neg\psi$ (negation is pushed to the literals) belongs to $\mathrm{CL}(\varphi)$,
3. If $\varphi_1 \wedge \varphi_2 \in \mathrm{CL}(\varphi)$ or $\varphi_1 \vee \varphi_2 \in \mathrm{CL}(\varphi)$ then $\varphi_1 \in \mathrm{CL}(\varphi)$ and $\varphi_2 \in \mathrm{CL}(\varphi)$,
4. If one of $\mathtt{Y}^c\varphi_1, \mathtt{Y}^g\varphi_1$ is in $\mathrm{CL}(\varphi)$, then $\varphi_1 \in \mathrm{CL}(\varphi)$,
5. If $\theta x.\varphi_1(x) \in \mathrm{CL}(\varphi)$ then $\varphi_1(\theta x.\varphi_1(x)) \in \mathrm{CL}(\varphi)$.

▶ **Definition 24.** An *atom* A is a subset of $\mathrm{CL}(\varphi)$ that satisfies the following properties:

1. For all $\psi \in \mathrm{CL}(\varphi)$, $\psi \in A$ iff $\neg\psi \notin A$,
2. For all $\varphi_1 \vee \varphi_2 \in \mathrm{CL}(\varphi)$, $\varphi_1 \vee \varphi_2 \in A$ iff $\varphi_1 \in A$ or $\varphi_2 \in A$,
3. For all $\theta x.\varphi_1(x) \in \mathrm{CL}(\varphi)$, $\theta x.\varphi_1(x) \in A$ iff $\varphi_1(\theta x.\varphi_1(x)) \in A$.

Now we describe how the Data Automaton $\mathcal{A}_\varphi$ works on a given data ω-word w. The states of the automaton are precisely the atoms. To ensure that the labelling by atoms is correct the automaton has to verify the following consistency conditions,

(i) $\mathsf{first}^g \in A_i$ iff i is the first position, $\mathsf{first}^c \in A_i$ iff i is the first position of a class,
(ii) $p \in A_i$ iff the label at position i is p,
(iii) let $tp(i) = (p, s)$ then $\mathcal{S} \in A_i$ iff the marking $s = \mathcal{S}$, similarly, $\mathcal{P} \in A_i$ iff the marking p is $\mathcal{P}$,
(iv) $\mathtt{Y}^g\varphi_1 \in A_i$ iff $\varphi_1 \in A_{g^{-1}(i)}$,
(v) A_1 contains φ.
(vi) $\mathtt{Y}^c\varphi_1 \in A_i$ iff $\varphi_1 \in A_{c^{-1}(i)}$.

These consistency conditions can be checked easily by the Data Automaton. Now for correctness the labelling of every position by the set of subformulas true at that position is a valid run of the automaton. For the other direction it can be proved inductively that a formula belongs to an atom at position i iff it is true at position i. In particular the set of positions that contain a formula $\theta x.\varphi(x)$ is a fixpoint of $\varphi(x)$. Since $\varphi(x)$ has a unique fixpoint the completeness follows. ◀

H Proof of Proposition 14

Proof. Given an alternating parity automaton $\mathcal{A}$ we construct a GDA $\mathcal{A}'$ whose set of states Q is precisely the set of local strategies and given an input data ω-word w a run of $\mathcal{A}'$ labels every position with a local strategy $(f_i)_{i\in\omega}$ such that Condition (1) and (2) in the definition of local strategy annotation is satisfied. Note that this is possible since the conditions are local (to verify the automaton only needs the states at the global predecessor and class predecessor). The only complication here is the unary modalities $\mathcal{S}$ and $\mathcal{P}$. But from the Lemma 13 we know that there is a Data Automaton labelling every position with the correct modality. Since Büchi GDA subsume Data automata (Lemma 1) are closed under composition (Lemma 2) we conclude that this can be done. To verify that the local strategy annotation is accepting the automaton $\mathcal{A}'$ has to check that for all infinite paths in the annoation it is the case that the maximal infinitely occurring parity is even. Hence it remains to show that the automaton $\mathcal{A}'$ can verify this property. We define the acceptance language L of $\mathcal{A}'$ to be the set of all ω-words

$$\begin{matrix} f_1 & f_2 & f_3 & \\ \epsilon & s_1 & s_2 & \end{matrix} \cdots$$

over the alphabet $F \times \{g, c\}$ such that all sequence of states

$$p_0 = q_0, p_1 \ni f_1(p_0, s_1), p_2 \ni f_2(p_1, s_2), \ldots$$

sees a maximal infinitely occurring even priority. To see that L is ω-regular, notice that there is a nondeterministic Büchi automaton which will guess a path by guessing a state p_i in each f_i and verifies that the guessed path sees a maximal infinitely occurring odd priority. The complement automaton will then verify that all paths see a maximal even priority infinitely often. It is straightforward to see that all paths in the strategy annotation $(f_i)_{i\in\omega}$ on w has a maximal recurring even priority iff the strategy annotation is accepting iff for all paths π in the run of $\mathcal{A}'$ on w the marked path projection of π is in L.

In the special case when $\mathcal{A}$ is an alternating Büchi automaton, we show that L is in particular a deterministic Büchi language. We construct a deterministic Büchi automaton $\mathcal{B}$ recognizing L using the break-point construction. The states of $\mathcal{B}$ are of the form (E, F) where $E, F \subseteq Q$, and while reading an ω-word the set E stores the end states of all the paths leading upto the current position and $F \subseteq E$ stores the end states of paths which has not seen a Büchi state in Q since the last break-point. The Büchi states of $\mathcal{B}$ are all pairs where F is empty. On reaching a state (E, F) where F is empty (called a break-point) at the next position F is again set to be E. This way $\mathcal{B}$ makes sure that all paths see a Büchi state in Q infinitely often.

◀

I Proof of Theorem 15

Proof. Given a formula φ in the fragment we claim that there is a Büchi GDA $\mathcal{A}$ such that φ is satisfiable if and only if the language of $\mathcal{A}$ is nonempty. Since Büchi GDA are closed under composition (Lemma 2) it is enough to verify the base cases; which is when φ belongs to the forward fragment or the backward fragment. In both cases we know that there is an equivalent Büchi GDA by Lemma 10 and Proposition 14. Since non-emptiness of Büchi GDA is decidable by Theorem 8 the theorem follows. ◀