Weak MSO+U over infinite trees

Mikołaj Bojańczyk[1] and Szymon Toruńczyk[2]

1 **University of Warsaw**
2 **INRIA and ENS Cachan**

Abstract

We prove that, over infinite trees, satisfiability is decidable for Weak Monadic Second-Order Logic extended by the unbounding quantifier U. We develop an automaton model, prove that it is effectively equivalent to the logic, and that the automaton model has decidable emptiness.

1998 ACM Subject Classification F.1.1 Models of Computation, F.4.1 Mathematical Logic

Keywords and phrases Infinite trees, distance automata, MSO+U, profinite words

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

Introduction. The general topic of this paper is monadic second-order logic extended with the unbounding quantifier. The unbounding quantifier is a kind of set quantifier, which says that a formula $\varphi(X)$ holds for arbitrarily large finite sets X:

$$\mathsf{U}X\varphi(X) \qquad \stackrel{\text{def}}{=} \qquad \bigwedge_{n\in\mathbb{N}} \exists X\ n \le |X| < \infty \ \wedge\ \varphi(X).$$

The unbounding quantifier was introduced in [1], along with some rudimentary decidability results. The quantifier is part of a research program, which investigates the notion of "regular language" for infinite words and trees. The general theme of the research program is that some features, such as the unbounding quantifier, can be added to monadic second-order logic over infinite objects, while preserving properties one would expect from a regular language. For instance, consider a language L of infinite words. Define a Myhill-Nerode-like equivalence relation $\sim_L$ on finite words:

$$w \sim_L w' \qquad \text{if} \qquad \text{for every finite word } u \text{ and every infinite word } v,$$

$$uwv \in L \iff uw'v \in L.$$

One can show that if L is defined in monadic second-order logic with the unbounding quantifier (MSO+U), then $\sim_L$ has finitely many equivalence classes. Furthermore, each equivalence class is a regular language of finite words. The research program is discussed in [3].

The expressive power of the logic MSO+U is still not properly understood. It is an open problem whether satisfiability is decidable over infinite words.

So far, research has dealt with fragments of the logic. The paper [4] introduces two classes of automata on infinite words, called ωB- and ωS-automata, and proves that they correspond to fragments of MSO+U with restricted quantifier use. It is not clear if there can be an automaton model for the whole logic MSO+U, as opposed to an automaton model for fragments of the logic. These doubts are based on the paper [11], which proves that MSO+U can define non-Borel languages of infinite words. This implies that there can be no nondeterministic automaton model for MSO+U that has a Borel acceptance condition, which excludes all known nondeterministic automata models that use counters. One has to keep in mind that the non-Borel result still leaves room for automata; a distant analogy is that parity automata on infinite trees recognize non-Borel sets.

The topological problems described above disappear when one considers *weak* monadic second-order logic (WMSO), where set quantifiers are restricted to finite sets. In countable structures, such as infinite words or trees, formulas of WMSO, even extended with the unbounding quantifier, can only define Borel languages. Over infinite words, and without the unbounding quantifier, WMSO has the same expressive power as MSO, thanks to the McNaughton/Safra determinization theorem. This coincidence fails when the unbounding quantifier is introduced: WMSO+U is strictly less powerful than MSO+U. The crucial advantage of the weak logic is that it supports the classical automaton-logic connection: it admits an automaton model, the max-automaton from [2]. The automaton-logic connection also works for other extensions of WMSO on infinite words, see [5]. The topological complexity of WMSO+U has been studied in [6].

Content of this paper. The goal of this paper is the following theorem:

▶ **Theorem 1.** *Satisfiability is decidable for* WMSO+U *over infinite trees.*

We prove the theorem in three steps.

1. In Section 1, we define a new automaton model for infinite trees, called a nested limsup automaton, which has the same expressive power as WMSO+U, and show effective translations from the logic to the automaton and back again.
2. In Section 2, we define a second new automaton model for infinite trees, called a puzzle, which is more expressive than a nested limsup automaton, and show an effective translation from nested limsup automata to puzzles.
3. In Section 3, we provide a decision procedure for nonemptiness of puzzles.

The proof, especially step 3, is maybe more interesting than the result itself. The general theme is to extend concepts of automata theory from finite sets to infinite sets equipped with compact metric topologies.

Related work. The automata models studied in this paper work on infinite objects. Two of the models, namely the ωB- and ωS-automata from [4], have natural counterparts working on finite words, called B- and S-automata. These finite word counterparts have recently seen a lot of interest. Instead of defining boolean-valued functions, which accept or rejects words, B- and S-automata define number-valued functions, which map words to numbers. These number-valued functions on finite words have been studied in depth by Colcombet in [8], under the name of regular cost functions. The theory of regular cost functions looks very promising, see [10] and [9] for some developments.

On a technical level, this paper uses profinite words [12] to model the limit behavior of finite words. This approach has been successfully applied in [13], as an alternative for cost functions in the study of the limitedness problem – B- or S-automata do define boolean-valued functions, which accept or reject profinite words.

Acknowledgement. We are very grateful to the anonymous reviewer for his detailed remarks.

1 WMSO+U and Nested Limsup Automata

In Section 1 and 2, when talking about a tree over an alphabet A, we mean a full infinite binary tree with nodes labeled by A. (In Section 3, we switch to edge-labeled graphs.)

WMSO+U. A tree is interpreted as a logical structure, with unary predicates for labels, and two binary predicates for left and right successors. To express properties of this logical structure, we use weak monadic second-order logic, which means that formulas can quantify over nodes and *finite* sets of nodes. We use the convention where first-order variables are denoted x, y, z and set variables are denoted X, Y, Z. Also, we allow the unbounding quantifier U defined in the introduction.

▶ Running example. Consider an alphabet $A = \{a, b, c\}$. Define a *b-factor* in a tree t to be a connected set of nodes with label b. Being a b-factor is definable in WMSO:

$$\text{bfactor}(X) \overset{\text{def}}{=} \exists x\ x \in X \wedge \big(\forall z\ z \in X \implies \big(x \leq z\ \wedge\ \forall y\ x \leq y \leq z \implies (y \in X \wedge b(y))\big)\big).$$

Here, $\leq$ denotes the ancestor relation – the transitive reflexive closure of the parent relation – which is definable in WMSO. The running example in this paper is the tree language over A, call it L, which contains a tree if and only if the root has label a, and for every node x:

(a) If x has label a, then its subtree has b-factors of unbounded size.
(b) If x has label b or c, then in its subtree, the size of b-factors is bounded.

The language L is defined by the following formula of WMSO+U

$$(\exists x\ \forall y\ (x \leq y)\ \wedge\ a(x))\quad \wedge\quad \big(\forall x\ a(x) \iff \mathsf{U}X\big((\text{bfactor}(X) \wedge \forall y\ (y \in X \Rightarrow y \geq x)\big)\big).$$

Let L be the set of trees satisfying the property above. What does a tree $t \in L$ look like? Observe first that every b-factor has to be finite, since an infinite b-factor contains finite b-factors of unbounded size, violating condition (b). Also, a node with label b or c cannot have a descendant with label a. This is because a tree with b-factors of bounded size cannot have a subtree with b-factors of unbounded size. It follows that all the nodes with label a form a connected set, call it X, which contains the root. There must be b-factors of unbounded size below every node from X, however every such b-factor must be finite, and have bounded size b-factors in its subtree. It follows that every node from X has at least one child in X, and the size of b-factors with parents in X is unbounded. An example is depicted in Figure 1. In the figure, we distinguish the maximal b-factors and call them $F_1, F_2, \ldots$, because they will get a lot of attention in the later analysis. The language L contains no regular tree, because in a regular tree either b-factors have bounded size, or some b-factor is infinite. In particular, L is not a regular language of infinite trees. Observe that in Figure 1, the only part of the tree that behaves in a non-regular way is the b-factors $F_1, F_2, \ldots$. ◀

1.1 Nested Limsup Automata

In this section, we define an automaton model which, over infinite trees, has the same expressive power as WMSO+U. The automaton is obtained by nesting two types of automata: prefix automata and limsup automata. We begin by defining prefix automata and limsup automata, then we show how they are nested.

Prefix automata. A prefix automaton is used to test regular properties of a finite prefix of a tree. Typical languages recognized by this kind of automaton are reachability properties "some node has label a", or "there is an antichain with five labels a". A *prefix* of a tree is an ancestor-closed set of nodes in the tree. A *prefix automaton* is given by the following ingredients:

- An *input alphabet* A.

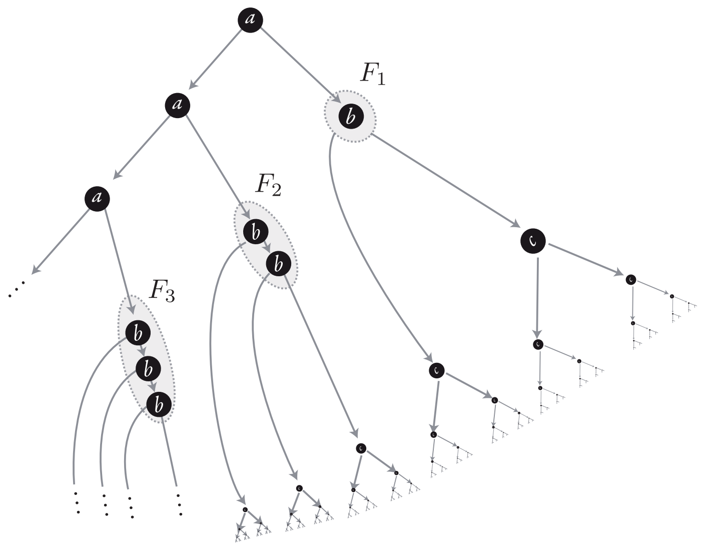


Figure 1 A tree $t \in L$. Every c node has only c descendants.

- A finite set of *states* Q, together with an initial state $q_I \in Q$.
- A *(nondeterministic) transition relation*

$$\delta \subseteq Q \times A \times Q \times Q.$$

- A set of *accepting states* $F \subseteq Q$

The automaton accepts an infinite tree if there is a finite prefix $X \subseteq \{0,1\}^*$ and a run $\rho : X \to Q$, such that ρ respects the transition relation, has the initial state in the root, and all maximal nodes of X have labels in the accepting set F.

A prefix automaton has an existential nature: it tests if there exists a finite prefix with a certain (regular) property. In particular, languages recognized by prefix automata are open sets, under the usual topology over infinite trees.

Atomic limsup automata. We now define a second kind of automaton, called an atomic limsup automaton. A typical language recognized by this kind of automaton is “for every $n \in \mathbb{N}$, there is some path in the tree with at least n labels a”. Observe that this typical language is not the same as “there is some path with infinitely many labels a”.

The general idea is that the automaton has a counter, which stores natural numbers. The transition function chooses states in a top-down deterministic fashion. The transition function also induces a labeling of edges in the tree by sequences of counter operations. There are two counter operations: increment (written *inc*) and reset (written *reset*). Unlike the model for WMSO+U on infinite words defined in [2], there is no max operation here. The automaton accepts an input tree if the counter has unbounded values, ranging over nodes in the tree. We give a formal definition below.

An *atomic limsup automaton* is given by the following ingredients:

- An *input alphabet* A.
- A finite set of *states* Q, together with an initial state $q_I \in Q$.
- A *(top-down deterministic) transition function*

$$\delta : Q \times A \to (\{inc, reset\}^* \times Q)^2.$$

Let t be a tree over the input alphabet A. Using the deterministic transition function δ and the initial state in the root, one labels in a unique way the nodes of t by states and the edges of t by sequences in $\{inc, reset\}^*$. Suppose that the counter has value 0 in the root. For any finite path π in t, by reading the operations along the path, we get a counter value. The automaton accepts the tree t if the counter value is unbounded, when ranging over all finite paths in the tree. In other words, the automaton accepts if there are arbitrarily long sequences of increments that are not interrupted by reset.

Nested limsup automata. We now combine the two automata above into a single model, by using nesting. We define nested limsup automata by induction on the *nesting depth*. A nested limsup automaton of nesting depth 1 is either a prefix automaton, or an atomic limsup automaton.

An automaton of nesting depth $k+1$ is defined as follows. Suppose that $\mathcal{A}_1, \ldots, \mathcal{A}_n$ are nested limsup automata of nesting depth k, over a common input alphabet A. Let $\mathcal{B}$ be either a prefix automaton, or an atomic limsup automaton, with input alphabet $\{0,1\}^n$. Then the expression $\mathcal{B}[\mathcal{A}_1, \ldots, \mathcal{A}_n]$ defines a nested limsup automaton. This new automaton has nesting depth $k+1$ and input alphabet A. When does it accept a tree t? Consider the tree $\hat{t}$ over alphabet $\{0,1\}^n$, where the label of a node x is a bit-vector, which has 1 on coordinate $i \in \{1, \ldots, n\}$ if and only if $\mathcal{A}_i$ accepts the subtree of t rooted in x. The automaton $\mathcal{B}[\mathcal{A}_1, \ldots, \mathcal{A}_n]$ accepts t if and only if the automaton $\mathcal{B}$ accepts the tree $\hat{t}$.

Observe that nested limsup automata are closed under complementation – the complement of $\mathcal{A}$ is recognized by an automaton $\mathcal{B}[\mathcal{A}]$, where $\mathcal{B}$ is a prefix automaton checking for 0 at the root.

Like all nested models of automata, nested limsup automata are something of a hybrid, sitting between logical formulas and automata.

▶ Running example. We now present a nested limsup automaton which recognizes the complement of the language L from the running example. Consider first an auxiliary automaton $\mathcal{B}$, a limsup automaton, which increments its counter whenever it sees a b, and resets it whenever it sees a or c. Since a large b-factor must contain a long path, the automaton $\mathcal{B}$ accepts a tree if and only if the tree has b-factors of unbounded size. A tree belongs to the complement of L if and only if the root is not labeled by an a, or if there is some node x, such that:

- The label of x is a, and $\mathcal{B}$ rejects the subree of x; or
- The label of x is either b or c, and $\mathcal{B}$ accepts the subtree of x.

Therefore, the complement of L is recognized by a limsup automaton nested inside a prefix automaton. ◀

1.2 Equivalence

The model of nested limsup automata is designed to be equivalent to WMSO+U, as stated in the following theorem.

▶ **Theorem 2.** *A language of infinite trees is definable in* WMSO+U *if and only if it is recognized by a nested limsup automaton. Translations both ways are effective.*

The proof of this theorem is in part I of the appendix. The proof ideas are based on [2].

Recall that our goal in this paper is to decide satisfiability of WMSO+U. The above theorem reduces the satisfiability problem of WMSO+U to the emptiness problem for nested limsup automata. However, due to the nesting operation, nested limsup automata are still too difficult to solve for emptiness. That is why, in the next section, we present a further reduction, which removes the nesting in a nested limsup automaton.

2 Puzzles

We now turn to the second automaton model in this paper, which is called a puzzle. The name is silly because we do not expect this model to be relevant outside this paper.

2.1 Puzzles, a denested version of nested limsup automata.

The ingredients of a *puzzle* are:

- a finite set Q of *states*
- a finite set C of *counters*
- an input alphabet A
- an *initial state* $q_I \in Q$
- a (nondeterministic) *transition relation*

$$\delta \subseteq Q \times A \times (\{\mathit{inc}, \mathit{reset}, \mathit{cut}\} \times C)^* \times Q)^2$$

- an *unbounding acceptance condition* $q \in Q \mapsto \mathsf{U}_q \subseteq C$, which maps each state q to the set of counters that are called *unbounded in* q.
- a *parity acceptance condition* $q \in Q \mapsto \Omega_q \in \mathbb{N}$, which maps each state to a natural number, called its *parity rank.*

Given an input tree t over the input alphabet, a *run* of the puzzle is an infinite binary tree where the nodes are labeled by states, the root has the initial state, and the edges are labeled by $(\{\mathit{inc}, \mathit{reset}, \mathit{cut}\} \times C)^*$, in a way consistent with the transition relation of the puzzle.

Observe that there is a new counter operation, called *cut*. The idea is that in the acceptance condition, the lim sup operation is only calculated along paths without cut. More formally, for a sequence of counter operations

$$\sigma \in (\{\mathit{inc}, \mathit{reset}, \mathit{cut}\} \times C)^*$$

we define the value of σ on counter c, denoted by $\mathrm{val}(\sigma, c)$, to be the maximal number n, such that some prefix of σ without a cut on counter c has n increments on counter c that are not interrupted by a reset on counter c. For example,

$$\mathrm{val}(\sigma, c) = 2 \qquad \text{for } \sigma = \mathit{inc}(c)\mathit{cut}(d)\mathit{inc}(c)\mathit{cut}(c)\mathit{inc}(d)\mathit{inc}(c)\mathit{inc}(c)\mathit{inc}(c)\mathit{reset}(c).$$

even though there are 3 consecutive increments on c after the cut on c. For a finite path π in a run ρ, we define

$$\mathrm{val}(\rho, \pi, c) \overset{\text{def}}{=} \mathrm{val}(\sigma, c) \qquad \text{where } \sigma \text{ is the sequence of edge labels on } \pi.$$

Finally, for a node x in a run ρ, we define

$$\mathrm{val}(\rho, x, c) \overset{\text{def}}{=} \sup\{\mathrm{val}(\rho, \pi, c) : \pi \text{ is a finite path originating in } x\ \} \in \mathbb{N} \cup \{\infty\}.$$

A run ρ is *accepting* if on every path, the parity acceptance condition is satisfied, and

$$\mathsf{U}_q = \{c \in C : \mathrm{val}(\rho, x, c) = \infty\} \qquad \text{for every node } x \text{ with state } q. \tag{1}$$

The key differences between a puzzle and a nested limsup automaton are:

- The set of bounded counters is tested in every subtree, as defined in (1);
- The model is not nested, but nondeterministic;
- There is the new *cut* counter operation.

▶ Running example. We define a puzzle which recognizes the language L from the running example (for simplicity, we ignore the condition on the root label). The states Q are q_a, q_b and q_c. There is one counter, call it d. State q_b increments the counter, which corresponds to counting the size of a path in a b-block, while the other states q_a and q_c reset the counter. This behavior is captured by the following set of transitions:

$$\{(q_\sigma, \sigma, (reset(d), q_0), (reset(d), q_1)) : \sigma \in \{a, c\}, q_0, q_1 \in Q\} \cup \\ \{(q_b, b, (inc(d), q_0), (inc(d), q_1)) : q_0, q_1 \in Q\}.$$

In this particular puzzle, the parity acceptance condition plays no role, and all states have accepting parity rank 0. Also, this puzzle does not use the *cut* operation. The key role is played by the unbounding acceptance condition, which is defined by

$$\mathsf{U}_{q_a} = \{d\} \qquad \mathsf{U}_{q_b} = \emptyset \qquad \mathsf{U}_{q_c} = \emptyset.$$

In other words, any node with state q_a in an accepting run must have unbounded values of the counter in its subtree, and every other node must have bounded values of the counter in its subtree. ◀

▶ **Theorem 3.** *For every nested limsup automaton one can compute a puzzle that recognizes the same language.*

The proof of this theorem is in part II of the appendix. The theorem can be interpreted as trading nesting for nondeterminism. From the point of view of deciding emptiness, this is a good trade: nesting is cumbersome for an emptiness algorithm, while nondeterminism is irrelevant.

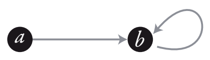
The converse of Theorem 3 fails: thanks to the parity condition, puzzles recognize non-Borel tree languages, while WMSO+U defines only Borel tree languages. Another reason is shown in the appendix: languages recognized by puzzles are not closed under complements.

3 Emptiness for puzzles

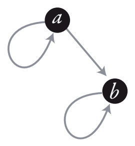
This section is about the emptiness procedure for puzzles.

▶ **Theorem 4.** *Emptiness is decidable for puzzles.*

The general idea is that even though an accepting run of a puzzle is an infinite object, there should be some way of drawing it in a finite way. This idea works for Büchi automata, because every nonempty Büchi automaton accepts the unfolding of a lasso graph such as:



This idea also works for parity tree automata, because every nonempty parity tree automaton accepts the unfolding of some finite graphs, such as:



Runs as graphs. In the proof of Theorem 4, we will treat a run ρ of a puzzle as an edge-labeled graph G_ρ. The graph G_ρ has the same nodes as ρ. It has no labels on the nodes. An edge in the graph is labeled by the word

$$\sigma \in Q \cdot (\{inc, reset, cut\} \times C)^* \cdot Q,$$

which begins with the state in the source node of the edge, followed by the sequence of counter operations on the edge, and ending with the state in target node of the edge. From now on, when writing ρ, we will refer to the graph G_ρ. The labels on the edges of G_ρ are words over the alphabet

$$\Lambda \overset{\text{def}}{=} Q \cup \{inc, reset, cut\} \times C.$$

We fix this alphabet for the rest of this section.

▶ Running example. Recall the tree t from Figure 1. The puzzle in the running example has only one run over any tree, and over t the run is accepting. The part of this run that concerns that b-factor F_n is illustrated in Figure 2. In the rest of this section, we will try to define a limit F_ω. ◀

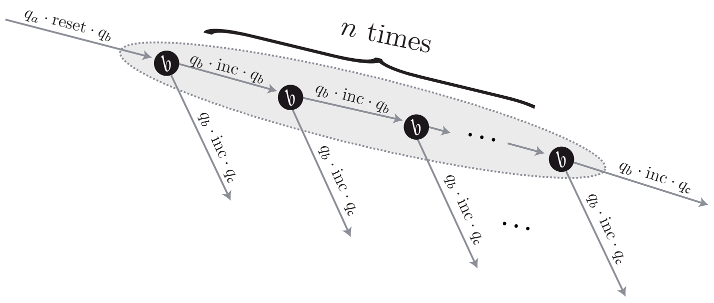

$q_a \cdot$ reset $\cdot q_b$
n times
$q_b \cdot$ inc $\cdot q_b$
$q_b \cdot$ inc $\cdot q_b$
$q_b \cdot$ inc $\cdot q_c$
$q_b \cdot$ inc $\cdot q_c$
$q_b \cdot$ inc $\cdot q_c$
$q_b \cdot$ inc $\cdot q_c$
$q_b \cdot$ inc $\cdot q_c$
b
...

Figure 2 The run of the puzzle inside the b-factor F_n of the tree t from Figure 1.

Factor. A *factor* in a tree is a connected set of nodes. Every factor has a root node, which is the least node in the factor. A *port* in a factor is a node outside the factor that has its parent in the factor. A *root-to-port path* in a factor is a path from the root to some port, seen as a sequence of edges.

Signature. The signature of a (finite or infinite) path in a run is the concatenation of all the labels on that path, which is a word over the alphabet Λ. We use the letters σ or τ to denote signatures of paths.

▶ Running example. In Figure 2, the signature of the rightmost root-to-port path in F_n is

$$\sigma_n \overset{\text{def}}{=} (q_b \cdot inc \cdot q_b)^{n-1} \cdot (q_b \cdot inc \cdot q_c). \quad ◀$$

For signatures of factors, we use multisets, which are sets where the number of occurrences an element can be in $\mathbb{N} \cup \{\infty\}$. Consider a *finite* factor F. The *signature* of the factor is the multiset of path signatures, ranging over root-to-port paths in the factor. All path signatures in this multiset have the same source state, namely the state in the root of the factor. This state is called the root state of a factor signature. It is important that factor signatures describe finite factors. In an infinite factor, it may be the case that a path is not included in root-to-port paths. We use the letter Σ to denote factor signatures.

▶ Running example. The signature of the factor in Figure 2 is the multiset, call it Σ_n, which contains all path signatures $\sigma_1, \ldots, \sigma_{n-1}$ once, and the path signature σ_n twice. ◀

3.1 Limits of signatures

The key technique in this paper is to use limits. We are mainly interested in the limits of signatures, both signatures of paths, and signatures of factors. In this section, we establish the notion of limit that we use. Our approach to limits of path signatures is to treat path signatures as a special case of profinite words. Our approach to limits of factor signatures is to use a variant of Hausdorff distance on multisets of profinite words. The definitions follow.

Profinite words. Consider the following distance on finite words over the alphabet Λ.

$$\text{distance}(\sigma, \tau) = \max\{\frac{1}{2^n} : \text{ some DFA of } n \text{ states accepts } \sigma \text{ but not } \tau\}.$$

It is not difficult to see that this is indeed a distance, even an ultrametric:

$$\text{distance}(\sigma_1, \sigma_2) \leq \max(\text{distance}(\sigma_1, \tau), \text{distance}(\tau, \sigma_2)) \qquad \text{for every } \sigma_1, \sigma_2, \tau \in \Lambda^*.$$

A sequence of words $(\tau_n)_n$ is called *Cauchy* if for every $\varepsilon > 0$ there is some n such that all the words $\tau_n, \tau_{n+1}, \ldots$ lie in a ball of diameter ε.

▶ Running example. Recall the sequence $(\sigma_n)_n$ of signatures of rightmost paths in the factors F_n. This sequence is not Cauchy, because even-numbered words have an even number of increments, and odd-numbered words do not, and evenness can be tested by a DFA of 2 states. However, the sequence $(\sigma_n)_n$ has several Cauchy subsequences, including the sequences $(\sigma_{n!})_n$ and $(\sigma_{n!+1})_n$. ◀

Consider two Cauchy sequences $(\sigma_n)_n$ and $(\tau_n)_n$ to be *equivalent* if

$$\sigma_1, \tau_1, \sigma_2, \tau_2, \ldots$$

is also a Cauchy sequence. This is an equivalence relation, call it $\sim$. An equivalence class of this relation is called a *profinite word* (see [12] for more on profinite words). The set of profinite words is denoted by $\widehat{\Lambda^*}$. We model signatures of paths and their limits by profinite words. Here are the key properties of profinite words that we use:

1. It makes sense to say that a profinite word belongs or does not belong to a regular language $L \subseteq \Lambda^*$. Indeed, if (σ_n) is a Cauchy sequence, then either all but finitely many elements belong to L, or all but finitely many elements do not belong to L. Therefore, it makes sense to say that a Cauchy sequence belongs or does not belong to a regular language. Also this property is preserved when going to an equivalent Cauchy sequence. In particular, it makes sense to say that a profinite word does at least one increment on some counter c, or does at least 4 increments, or begins with state q, because all of these are regular properties.
2. There is a distance on profinite words, namely:

$$\text{distance}((\sigma_n)_n, (\tau_n)_n) = \lim_{n\to\infty} \text{distance}(\sigma_n, \tau_n).$$

   By the triangle inequality, the above limit exists for Cauchy sequences and does not depend on the choice of a sequence in a class of $\sim$. Equipped with this distance, the set of profinite words is a compact metric space. This means that every sequence has a converging subsequence. Also, for every regular language $L \subseteq \Lambda^*$, there is some distance ε such that any two words at distance at most ε either both belong to L, or both do not belong to L.

3. It makes sense to concatenate profinite words. This is because the relation $\sim$ is a congruence with respect to concatenation of sequences:

$$(\sigma_n)_n \sim (\sigma'_n)_n \quad \text{and} \quad (\tau_n)_n \sim (\tau'_n)_n, \qquad \text{implies} \qquad (\sigma_n \cdot \tau_n)_n \sim (\sigma'_n \cdot \tau'_n)_n.$$

▶ Running example. Recall the Cauchy sequence $(\sigma_{n!})_n$. We write σ_ω for the profinite word represented by this sequence. This profinite word begins with letter q_b and ends with letter q_c. Also, for every n, there are more than n increments in σ_ω, which is something that can only happen in a profinite word. ◀

Hausdorff distance on sets. So far, we have defined a compact metric space to model path signatures, namely the set of profinite words over Λ. We now want to do the same thing for multisets of path signatures. Our approach is to use a multiset variant of the Hausdorff distance on sets. We begin by recalling the distance on sets (not multisets), because this definition is easier to digest. A metric on a set A (we are interested in the case when A is the set of profinite words over Λ) can be lifted to a metric on closed subsets of A, using the Hausdorff distance. For two closed subsets $X, Y \subseteq A$, their Hausdorff distance is defined by

$$\text{distance}(X,Y) \stackrel{\text{def}}{=} \quad \max\big(\sup_{x\in X}\inf_{y\in Y}\text{distance}(x,y), \sup_{y\in Y}\inf_{x\in X}\text{distance}(x,y)\big).$$

This is a metric on closed subsets. This definition can be extended to so-called *closed multisets*. The precise definition of closed multisets and their metric is given in the appendix. As an example, consider multisets of real numbers. Like any finite multiset, the multiset

$$X_n = \{\frac{1}{n}, \frac{1}{n^2}, \ldots, \frac{1}{n^n}\}$$

is closed. The sequence $(X_n)_n$ tends to the (closed) multiset where 0 appears infinitely often.

▶ Running example. Consider the signature Σ_n of the factor F_n. One can prove that the sequence $(\Sigma_{n!})_n$ is Cauchy. Its limit is the multiset, call it Σ_ω, where every σ_n appears once, and every limit of a subsequence of (σ_n) appears infinitely often. Among others, for every $k \in \mathbb{N}$, $\sigma_{\omega+k}$ appears infinitely often. ◀

3.2 Signature graphs

We are now ready to define the key concept of this paper, which is a signature graph. A signature graph is used to represent limits of accepting runs. A signature graph is going to have labeled parallel edges, so it is really a multigraph. When talking about an *edge labeled multigraph*, we mean a directed graph with edges labeled by some alphabet A. The edges form a multiset, so for any label σ and pair of vertices x, y, the number of edges from x to y with label σ may potentially be $0, 1, \ldots$, or countably infinite.

Definition of a signature graph. A *path signature* is a profinite word over Λ that begins with a state (called the *source* state) and ends with a state (called the *target* state). A *factor signature* is a closed multiset of path signatures, which agree on the source state. A *signature graph* is a multigraph with edges labeled by path signatures, subject to the following consistency condition. For every node x, there is some state $q \in Q$, such that all edges entering x have target state q, and all edges leaving x have source state q. This state q is called the *node label* of x, although technically speaking a signature graph supplies only edge labels, and the node labels are derived information.

In a signature graph, the labeling assigns signature paths to individual edges. However, using the monoid structure of path signatures, we can assign a path signature to every finite edge path, by concatenating the labels of the edges in the path.

Fans. Suppose that x is a node in an edge labeled graph. We define the *fan* of x to be the multiset of labels of edges leaving x. If $\mathscr{P}$ is a family of multisets over A, then we say that a graph has *fans in* $\mathscr{P}$ if the fan of each node is in $\mathscr{P}$. In the proof, when dealing with signature graphs, we are interested in two sets $\mathscr{P}$ of factor signatures. The first set, call it

$$\mathscr{P}_{\text{fin}}$$

is the set of factor signatures of finite factors that appear in some run of the puzzle, not necessarily accepting. This set depends on the transitions and counter in the puzzle. It does not depend, however, on the acceptance conditions (boundedness and parity) in the puzzle, because these are only used to distinguish accepting runs. The second set is the closure of the first set, under the Hausdorff distance, in the space of closed multisets of profinite words:

$$\overline{\mathscr{P}_{\text{fin}}}.$$

Limit accepting signature graph. Recall that thanks to the properties of the profinite monoid, it makes sense to say that a path signature does an increment/cut/reset on some counter. We say that a path signature has *value* ω *on counter* c if for every $n \in \mathbb{N}$, the path signature has value at least n on counter c (recall that the value refers to the maximal number of increments, not interrupted by a reset, before the cut). Also, one can ask about the maximum rank, in the parity acceptance condition, of states visited by the limit path signature. For a node x in a signature graph G, define $\mathsf{U}(G, x)$ to be the set of counters U_q, where q is the state in the label of x.

We now present the key definition in the emptiness procedure for puzzles, the definition of a limit accepting signature graph. The idea is that a limit accepting signature graph is the limit of a converging sequence of accepting runs.

A signature graph G with a distinguished *root node* is called *limit accepting* if

1. The root node is labeled by the initial state of the automaton,
2. Every node in G is reachable from the root node,
3. The parity condition is satisfied on every infinite path,
4. For every node x and counter c, counter c belongs to $\mathsf{U}(G, x)$ if and only if
   a. There is an infinite path from x, such that every prefix of the path can be extended to a finite path that does not cut c, and reaches a node whose fan contains an edge with ω on counter c; or
   b. There is an infinite path from x, which does not cut c, resets it finitely often, and increments it infinitely often.

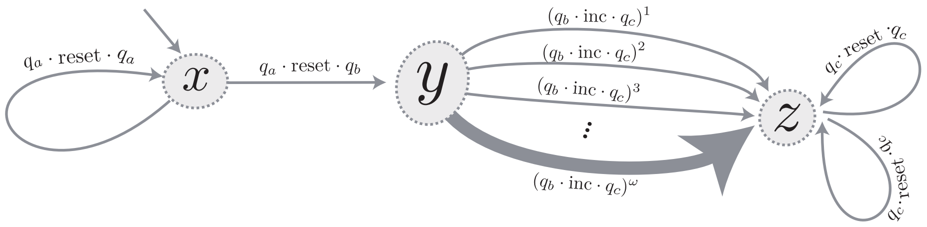

$q_a \cdot \text{reset} \cdot q_a$
x
$q_a \cdot \text{reset} \cdot q_b$
y
$(q_b \cdot \text{inc} \cdot q_c)^1$
$(q_b \cdot \text{inc} \cdot q_c)^2$
$(q_b \cdot \text{inc} \cdot q_c)^3$
$\vdots$
$(q_b \cdot \text{inc} \cdot q_c)^\omega$
z
$q_c \cdot \text{reset} \cdot q_c$
$q_c \cdot \text{reset} \cdot q_c$

Figure 3 This signature graph represents the accepting run in Figure 2, in the following sense. Node x represents all nodes with label a. The self-loop in x stands for the lefmost path in the graph from Figure 2. Node y represents a limit of the factors F_n. The fan of y is Σ_ω – the limit of the fans $(\Sigma_{n!})_n$. The thick edge stands for infinitely many edges, including infinitely may with label $(q_b \cdot inc \cdot q_c)^\omega$. Node z together with its two self-loops stands for a subtree with infinitely many c's.

Main technical theorem. To prove Theorem 4, we present a stronger result, Theorem 5, which is the main technical contribution of this paper.

▶ **Theorem 5.** *The following conditions are equivalent.*

1. *There is a limit accepting signature graph with fans in $\overline{\mathscr{P}_{\mathrm{fin}}}$,*
2. *There is a limit accepting signature graph with fans in $\overline{\mathscr{P}_{\mathrm{fin}}}$ with finitely many nodes,*
3. *The puzzle has an accepting run.*

Furthermore, given a puzzle one can decide if the conditions hold.

▶ Running example. By Theorem 5, the puzzle from the running example should have a limit accepting signature graph with fans in $\overline{\mathscr{P}_{\mathrm{fin}}}$, with finitely many nodes. Such a graph is illustrated in Figure 3, and has 3 nodes, but infinitely many edges. ◀

The proof of Theorem 5 is in part III of the appendix. A rough sketch is as follows.

- **Implication from 1 to 2**. The key point is that we can design an automaton model, closely resembling alternating automata on graphs, which recognizes limit accepting signature graphs. This automaton model shares the following property with alternating automata on graphs: a nonempty automaton accepts a graph with finitely many nodes.
- **Implication from 2 to 3.** The key point is to get rid of the limits, and replace them by actual finite pieces of runs. The idea is of course to use finite pieces of runs from a sequence approximating the limit, but the implementation of this idea requires some technical effort. We use a notion of bisimulation that is adapted to converging sequences.
- **Implication from 3 to 1.** The key point is to extract limits from an arbitrary accepting run of a puzzle. For this, we use a version of Ramsey's theorem adapted to metric spaces.
- **Decidability.** The key point is to compute a finite representation of the set $\overline{\mathscr{P}_{\mathrm{fin}}}$. In the end, we reduce this to the domination problem for B-automata over finite trees [10].

We believe that the technique of limits of graphs is quite general, and can be applied to other automaton models for trees.

References

1 M. Bojańczyk. A bounding quantifier. In *CSL*, pages 41–55, 2004.

2 M. Bojańczyk. Weak MSO with the unbounding quantifier. In *STACS*, pages 159–170, 2009.

3 M. Bojańczyk. Beyond ω-regular languages. In *STACS*, pages 11–16, 2010.

4 M. Bojańczyk and T. Colcombet. Bounds in ω-regularity. In *LICS*, pages 285–296, 2006.

5 M. Bojańczyk and S. Toruńczyk. Deterministic automata and extensions of weak mso. In *FSTTCS*, pages 73–84, 2009.

6 J. Cabessa, J. Duparc, A. Facchini, and F. Murlak. The wadge hierarchy of max-regular languages. In *FSTTCS*, pages 121–132, 2009.

7 T. Colcombet. Factorisation forests for infinite words. In *FCT'07*, 2007.

8 T. Colcombet. The theory of stabilisation monoids and regular cost functions. In *ICALP (2)*, pages 139–150, 2009.

9 T. Colcombet, D. Kuperberg, and S. Lombardy. Regular temporal cost functions. In *ICALP (2)*, pages 563–574, 2010.

10 T. Colcombet and C. Löding. Regular cost functions over finite trees. In *LICS*, pages 70–79, 2010.

11 S. Hummel, M. Skrzypczak, and S. Toruńczyk. On the topological complexity of MSO+U and related automata models. In *MFCS*, pages 429–440, 2010.

12 J-E. Pin. Profinite methods in automata theory. In *STACS*, pages 31–50, 2009.

13 S. Toruńczyk. *Languages of profinite words and the limitedness problem.* PhD thesis, Warsaw University, 2011.