Punctual Presentability in Certain Classes of Algebraic Structures

Dariusz Kaloci\'nski
Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland

Luca San Mauro
Department of Philosophy, University of Bari, Italy

Michał Wrocławski
Faculty of Philosophy, University of Warsaw, Poland

Abstract

Punctual structure theory is a rapidly emerging subfield of computable structure theory which aims at understanding the primitive recursive content of algebraic structures. A structure with domain $\mathbb{N}$ is punctual if its relations and functions are (uniformly) primitive recursive. One of the fundamental problems of this area is to understand which computable members of a given class of structures admit a punctual presentation. We investigate such a problem for a number of familiar classes of algebraic structures, paying special attention to the case of trees, presented both in a relational and functional signature.

2012 ACM Subject Classification Theory of computation → Computability

Keywords and phrases fully primitive recursive structures, punctual presentability, trees, injection structures

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.65

Funding *Dariusz Kaloci\'nski*: supported by the National Science Centre Poland grant under the agreement no. 2023/49/B/HS1/03930.
Luca San Mauro: is a member of INDAM-GNSAGA.
Michał Wrocławski: supported by the National Science Centre Poland grant under the agreement no. 2023/49/B/HS1/03930.

Acknowledgements We would like to thank the reviewers as well as Nikolay Bazhenov, Ivan Georgiev and Stefan Vatev for helpful discussions.

1 Introduction

Computable structure theory is a vast research program which aims at analyzing the algorithmic content of algebraic structures through the tools of computability theory (for an excellent introduction to this field, see [15]). The fundamental concept of when a structure is computably presented dates back to the seminal work of Mal'cev [14] and Rabin [16] in the 1960s: A structure with domain the set $\mathbb{N}$ of the natural numbers is computable if its relations and functions are uniformly Turing computable. Then, a countably infinite structure is computably presentable (or, it has computable copy) if it is isomorphic to some computable structure.

A classic problem in computable structure theory is to understand, for familiar classes of structures $\mathfrak{K}$, which members of $\mathfrak{K}$ are computably presentable. Sometimes the answer is that *every* member of $\mathfrak{K}$ has a computable copy: this is the case for, e.g., vector spaces over $\mathbb{Q}$ and algebraically closed fields of a given characteristic. On the other hand, it is an immediate consequence of Tennebaum's theorem that, among models of Peano Arithmetic, only one is computably presentable: the standard model. Yet, in most cases, characterizing the computable members of $\mathfrak{K}$ is a delicate task, which often requires to individuate nice invariants for $\mathfrak{K}$ (if any) and to determine how hard is to compute such invariants.

A parallel endeavour is to explore when a computable algebraic structure has a feasible presentation. The problem can be formalized in a number of different ways leading, e.g., to the study of algebraic structures presented by finite state automata [13], or to polynomial-time algebra [7, 1]. Kalimullin, Melnikov, and Ng [12] initiated the systematic study of punctual presentations, that lies somewhere in the between of computability and complexity theory:

▶ **Definition 1.** *A structure with domain* $\mathbb{N}$ *is* punctual *(or, fully primitive recursive) if its relations and functions are uniformly primitive recursive.*

Then, clearly, a structure is punctually presentable, if it is isomorphic to some punctual structure. Intuitively – and by relying on a sort of restricted Church-Turing thesis – a structure is punctual, if there is algorithmic with no unbounded search which is able to decide any quantifier-free question about the structure (that is, for any such question, one knows in advance how much time is needed to compute an answer).

Punctual structure theory rapidly emerged as an intriguing subfield of computable structure theory [10, 11, 4, 3] and it also serves as a theoretical underpinning for the study of online algorithms [2] (i.e., algorithms in which the input is received and processed piece by piece without having access from the start to the complete problem data).

One of notable results is that the index set of computable structures that are punctually presentable is Σ^1_1-complete [5]. However, if we restrict the class of structures is some natural way, it may be the case that every computable member from that class is punctually presentable (and thus the corresponding index set is trivial).

▶ **Definition 2.** *We say that a class of structures* $\mathfrak{K}$ *is* punctually robust*, if every computable member of* $\mathfrak{K}$ *admits a punctual presentation.*

In [12], it is proven that the following classes of structures are punctually robust: equivalence structures, linear orders, torsion-free abelian groups, Boolean algebras, and abelian p-groups; on the other hand, there are computable undirected graphs, computable torsion abelian groups, and computable Archimedean ordered abelian groups with no punctual copy. In this work, we contribute to this line of research, exhibiting new examples of both classes that are punctually robust and classes that are not.

Kalimullin, Melnikov, and Ng [12] showed that there exists a computable undirected graph with no punctual presentation. However, many natural classes of graphs admit such presentations. In Section 3, using a technique introduced in [12] for showing that equivalence structures are punctually presentable, we isolate one such class.

▶ **Proposition 3.** *Every computable digraph with an infinite semi-transversal is punctually presentable.*

For structures with a functional signature, the situation may change drastically. One of the simplest types of purely functional structures are the so-called injection structures. They are of the form $\mathcal{A} = (\mathbb{N}, f)$, where f is 1-1. Injection structures have already been considered in the computable setting (see, e.g., [6]) and in punctual structure theory with an emphasis on punctual categoricity in [10]. In Section 4 we prove the following:

▶ **Theorem 4.** *The class of injection structures is not punctually robust.*

The structure witnessing that injections structures are not punctually robust will consist only of cycles. One can observe that the relational counterpart of this structure is punctually presentable (this follows from Proposition 3). We supplement Theorem 4 by a list of sufficient

conditions for an injection structure to be punctually presentable, as well as by a list of conditions equivalent to their punctual presentability. These supplementary results are omitted due to space constraints and will appear in the extended version.[1]

A similar contrast emerges for trees. We show that the punctual robustness (or, lack thereof) of the class of trees depends on how they are presented. More precisely, in Section 5 we consider the notion of a tree represented as a function mapping each child to its parent and looping back at the root. We then prove the main result of the paper:

▶ **Theorem 5.** *The class of functional trees is not punctually robust.*

2 Preliminaries

We denote the set of natural numbers by $\mathbb{N}$. We let $[a, b) = \{x \in \mathbb{N} : a \leq x < b\}$. We assume a fixed computable enumeration $p_0, p_1, \ldots$ of all primitive recursive unary functions. We abbreviate *primitive recursive* as *p.r.* By f^n be denote the n-fold composition of f with itself. If F is a partial function or a relation, χ_F denotes a graph of F. A structure is any tuple $\mathcal{A} = (A, (R_i)_{i\in I}, (f_j)_{j\in J}, (c_k)_{k\in K})$, where $A \neq \emptyset$ is the domain of $\mathcal{A}$, in symbols $dom(\mathcal{A})$, while R_i, f_j, c_k are relations, functions, constants on A. By $\mathcal{A} \cong \mathcal{B}$ we mean that $\mathcal{A}$ and $\mathcal{B}$ are isomorphic, and by $\mathcal{A} \hookrightarrow \mathcal{B}$ that $\mathcal{A}$ embeds into $\mathcal{B}$. All structures in this article are countably infinite, except finite structures that appear in constructions.

▶ **Definition 6** (computable structure). *A structure is a computable if its domain, as well as all relations, functions, and constants from its signature are uniformly computable.*

Punctual structures were defined in the introduction (Definition 1).

Notice that $(\mathbb{N}, p_0), (\mathbb{N}, p_1), \ldots$ is a computable enumeration of all punctual structures with just one unary functional symbol.

▶ **Definition 7** (punctual presentability). *A structure $\mathcal{A}$ is a copy (presentation) of $\mathcal{B}$ if $\mathcal{A}$ and $\mathcal{B}$ are isomorphic. We say that $\mathcal{A}$ is punctually presentable if $\mathcal{A}$ has a punctual presentation.*

3 Directed graphs

In this section, we individuate a class of computable directed graphs (from now on, digraphs) which are punctually presentable. For a digraph $G = (\mathbb{N}, E_G)$, we denote by L_G the collection of G-nodes with loops, i.e., $L_G := \{x : (x, x) \in E_G\}$. We denote by $G \restriction_c$ the restriction of G to the initial segment $\{x : 0 \leq x \leq c\}$.

▶ **Definition 8** (semi-transversal). *Let $G = (\mathbb{N}, E_G)$ be a computable digraph. The* semi-transversal *of G, denoted as τ_G, is the collection of numbers that are not adjacent to any smaller number, i.e.,*

$$\tau_G := \{x : (\forall y < x)(\{(y, x), (x, y)\} \cap E_G = \emptyset)\}.$$

It is immediate to observe that τ_G is computable, whenever G is computable. As aforementioned, the following result elaborates on ideas presented in [12] when dealing with equivalence structures. The proof may serve as a gentle introduction to the way of thinking about punctual copies and as a warm-up to the more complex constructions of this paper.

[1] Following a suggestion from one of the reviewers, the authors have realized that some results on injection structures were obtained earlier [8]. Essentially, Theorem 4 should be attributed to Cenzer and Remmel (see, Lemma 3.12 and Theorem 3.13 in [8]). For completeness of the presentation, we include our proof of this theorem in the punctual setting.

▶ **Proposition 3.** *Every computable digraph with an infinite semi-transversal is punctually presentable.*

Proof. Let $G = (\mathbb{N}, E_G)$ be a computable digraph so that τ_G is an infinite set. For the sake of exposition, we assume that both $L_G \cap \tau_G$ and $(\mathbb{N} \setminus L_G) \cap \tau_G$ are infinite (the other cases are treated similarly and are somehow simpler). We will build by stages a punctual copy P_G of G and a bijection $f : \mathbb{N} \to \mathbb{N}$ witnessing that $G \cong P_G$.

The underlying idea of the proof is rather straightforward: At any given stage of the construction, we monitor a finite fragment of G. In particular, we aim at determining if a given number c belongs to τ_G. This information is computable but, in general, not primitive recursive. So to ensure that P_G will be punctual, while waiting to know if $c \in \tau_G$, we let several numbers x to be inactive (meaning that such numbers will belong to the semi-transversal of P_G). If we see that $c \notin \tau_G$, we extend the desired isomorphism by letting $f(c)$ be a fresh number and ensuring that $G \restriction_c$ embeds into the active part of P_G. On the other hand, if $c \in \tau_G$, we will let $f(c)$ be a suitable inactive number z (which returns active right after this action). Let us now be formal.

Construction

To record which fragment of G is currently monitored, we use a parameter c that will be updated as the construction proceeds. Moreover, during the construction some numbers x will be marked as *inactive*: specifically, by declaring x to be *no loop-inactive* (*NL*-inactive), we let x be non-adjacent (in P_G) to any number $\leq x$; by declaring x as *loop-inactive* (*L*-inactive), we let x non-adjacent (in P_G) to any number $< x$ but we also let $(x, x) \in P_G$. *Active* numbers are those that are neither *NL*-inactive nor *L*-inactive.

At stage 0, all numbers in $\mathbb{N}$ are active; we let c be 0 and we immediately move to the next stage. At all stages $s > 0$, we see if the following condition holds:

$$(\forall x \leq c)(\chi_{E_G}(x, c) \downarrow \text{ and } \chi_{E_G}(c, x) \downarrow \text{ in less than } s \text{ stages}). \tag{$\star$}$$

If $(\star)$ is not met, we declare $2s$ to be *NL*-inactive and $2s + 1$ to be *L*-inactive. Then, we move to the next stage. Otherwise, if $(\star)$ is met, we can determine whether c belongs to τ_G. Then, we distinguish two cases:

1. If $c \notin \tau_G$, we define $f(c) = 2s$ and we mirror an initial segment of the G-neighborhood of c by forming the P_G-edges that are needed to ensure that, for all $x \leq c$, the following equation holds:

$$(x, c) \in E_G \Leftrightarrow (f(x), f(c)) \in E_{P_G} \ \& \ (c, x) \in E_G \Leftrightarrow (f(c), f(x)) \in E_{P_G}. \tag{$\dagger$}$$

   Finally, we declare $2s + 1$ *NL*-inactive.
2. We distinguish two subcases:
   a. If $c \in L_G \cap \tau_G$, we define $f(c) = z$ for the least number z which is *L*-inactive, and we mark z as active (if there is no such number, we define $f(c) = 2s + 1$). Then, we declare $2s$ to be NL-inactive.
   b. If $c \in (\mathbb{N} \setminus L_G) \cap \tau_G$, the situation is symmetric: We define $f(c) = z$ for the least number z which is NL-inactive, and we mark z as active (if there is no such number, we define $f(c) = 2s$), and we declare $2s + 1$ to be *L*-inactive.

   In both cases 1. and 2., we increase c by one and we move to the next stage.

Verification

From the construction, it follows immediately that P_G is punctual: indeed, to decide if $(u, v) \in E_{P_G}$, for $u < v$, it suffices to run the construction until stage $\lfloor v/2 \rfloor$ and each stage s requires at most s steps to be completed. To see that P_G is isomorphic to G, we begin by arguing that f is bijection.

▶ **Lemma 9.** *The function $f : \mathbb{N} \to \mathbb{N}$ is a bijection.*

Proof. Say that a stage s is *expansionary*, if at stage s the condition $(\star)$ holds. It is not hard to see that there are infinitely many expansionary stages: indeed, since G is computable, for any choice of the parameter c, there will a stage at which we are able to entirely compute $G \upharpoonright_c$. Now, observe that the parameter c starts as 0 and is increased by one whenever $f(c)$ is defined. Thus, f is total.

Next, let $u < v$ be numbers on which f is defined at stages $s_u < s_v$. By construction, $f(u) \in \{2s_u, 2s_{u+1}, z\}$, for some z which is NL-inactive or L-inactive at the beginning of stage s_u; on the other hand, $f(v) \in \{2s_v, 2s_{v+1}, z'\}$, for some z' which is NL-inactive or L-inactive at stage s_v. Observe that $z \neq z'$, since z returns active as soon as it enters the range of f. Thus, f is injective.

To deduce that f is surjective, it suffices to prove that all numbers are eventually active: indeed, if a number $z \in \{2s, 2s + 1\}$ is active at all stages, then s must an expansionary stage at which z enters the range of f; similarly, if at stage s some number z returns active, after being NL-inactive or L-inactive, then z simultaneously enters the range of f. Towards a contradiction, suppose there is a least number u which is declared, say, L-inactive and eventually remains so. Since $L_G \cap \tau_G$ is infinite, then there must be a stage s at which the condition $(\star)$ is met and we perform action $2.a$, by which z (being the least L-inactive number) enters the range of f, contradicting our hypothesis. Hence, f is surjective. ◀

Finally, we shall prove that the bijection f yields an isomorphism from G to P_G. Suppose that, for a pair of numbers $u < v$,

$$(u, v) \in E_G \not\Leftrightarrow (f(u), f(v)) \in E_{P_G}.$$

Let s_u and s_v be the stages at which $f(u)$ and $f(v)$ are, respectively, defined. By construction, $s_u < s_v$, so that $f(u)$ is already defined at stage s_v. Moreover, at stage s_v, we decide whether $(f(u), f(v)) \in E_{P_G}$: if $v \notin \tau_G$, then condition $(\dagger)$ ensures that $(f(u), f(v)) \in E_{P_G}$ if and only if $(u, v) \in E_G$; on the other hand, if $v \in \tau_G$, we have that $(u, v) \notin E_G$, but $f(v)$ is chosen from the inactive numbers so that $f(u)$ is non-adjacent to $f(v)$. Similarly, one proves that, for all u, $(u, u) \in E_G$ if and only $(f(u), f(u)) \in E_{P_G}$. So f is an isomorphism from G to E_{P_G}. ◀

As an immediate corollary of Proposition 3, one obtains that the following classes of algebraic structures $\mathfrak{K}$ are punctually robust: equivalence structures, strongly locally finite graphs, and – more generally – graphs with infinitely many components. In fact, Proposition 3 may serve to obtain the punctual robustness of the classes of locally finite graphs and of graph-theoretic trees (the analog result holds in the setting of feasible presentations, see Cenzer and Remmel [9]). Let us give a brief informal discussion of these cases.

Recall that a *dominating* set for a graph $G = (V_G, E_G)$ is a set $D \subseteq V_G$ so that all vertices of $V_G \setminus D$ has a neighbor in D. The *domination number* $\gamma(G)$ is the minimum size of a dominating set for G. It is simple to show that every computably presentable graph G with $\gamma(G) = \infty$ has a computable copy H with an infinite semi-transversal. Hence, by

Proposition 3, all computable graphs with infinite domination numbers admit a punctual copy. This comprises the class of infinite but locally finite graphs, as is shown by a simple application of the pigeonhole principle.

We conclude the section by observing that the class of graph-theoretic trees, (i.e., acyclic graphs) is also punctually robust. Let T be a such tree. If T has infinite domination number, then T must be punctually presentable. On the other hand, if $\gamma(T) < \infty$, then it must contain a vertex with infinite neighborhood; a standard construction proves that such T is punctually presentable (we omit the details for space reasons). In contrast, in Section 5 we will show that there is a computable functional tree which admits no punctual copy.

4 Injection structures

▶ **Definition 10** (injection structure). *(A, f) is an* injection structure*, if $f : A \to A$ is injective.*

Let $f : A \to A$. A finite sequence of pairwise distinct elements $a_1, ..., a_n \in A$ is an f-cycle of length n if, for each $i = 1, ..., n-1$, $f(a_i) = a_{i+1}$, and also $f(a_n) = a_1$. If f is clear from the context, we may refer to a cycle without specifying f.

▶ **Definition 11** (cyclic injection structure). *$f : A \to A$ is* cyclic*, if f is injective and every element of A belongs to some f-cycle. $\mathcal{A} = (\mathbb{N}, f)$ is a* cyclic injection structure*, if f is cyclic. Such $\mathcal{A}$ is called* simple*, if $\mathcal{A}$ contains at most one cycle of length l, for all $l \in \mathbb{N}$.*

▶ **Definition 12** ($\mathbb{N}$-chain, $\mathbb{Z}$-chain). *Let $f : A \to A$. If $(a_n)_{n \in I}$ is an indexed family of elements of A such that $a_i \neq a_j$ for all $i \neq j \in I$, and $f(a_n) = a_{n+1}$ for all $n \in I$, then we call (a_n) an* $\mathbb{N}$-chain *if $I = \mathbb{N}$ and there is no such x that $f(x) = a_0$, and a* $\mathbb{Z}$-chain *if $I = \mathbb{Z}$.*

If $f : \mathbb{N} \to \mathbb{N}$ is an injection, then every $n \in \mathbb{N}$ belongs to a cycle, an $\mathbb{N}$-chain, or a $\mathbb{Z}$-chain. In the remaining part of this section we prove the following:[2]

▶ **Theorem 4.** *The class of injection structures is not punctually robust.*

We construct by stages a computable simple cyclic injection structure $\mathcal{A} = (\mathbb{N}, f)$ not isomorphic to any $(\mathbb{N}, p_i)$. For each $i \in \mathbb{N}$, we have the following requirement:

R_i : if $(\mathbb{N}, p_i)$ is a simple cyclic injection structure, then there is $n \in \mathbb{N}$ so that p_i has a cycle of length n but f does not.

Initially, for $\mathcal{A} = \emptyset$, and all requirement are inactive. At a given stage we have finitely many active requirements. Active requirements may later be deactivated: deactivating a requirement R_i will guarantee that R_i is, and will remain, satisfied. Some requirements R_i may stay eventually active, but we will argue that in that case they are satisfied vacuously (i.e., the antecedent of R_i is false).

[2] As a side remark, Theorem 3.16 from [8] says that *any* computable injective structure has a polynomial-time computable copy with the domain equal to $Bin(\omega)$, the set of all binary representations of natural numbers, or to $Tal(\omega)$, the set of all unary (tally) representations of natural numbers; call such a copy *fully polynomial-time*. The statement of Theorem 3.16 [8] turns out to be too general because the structures constructed in the proofs of our Theorem 3, or Theorem 3.13 [8] for that matter, are computable injection structures with no punctual presentation, and hence they cannot have fully polynomial-time copies either (as any fully polynomial-time structure is punctual). However, a slight weakening of the statement of Theorem 3.16 [8] can be achieved, namely: every computable injection structure which is not cyclic is punctually presentable. We omit the proof due to space constraints.

Stage s of the construction. Let $R_{i_1}, \ldots, R_{i_k}$ be the list of active requirements, and let $c_{i_1}, \ldots, c_{i_k}$ be the corresponding witnesses. If there is a fresh active requirement in the list, execute the strategy for the fresh requirement (see below). Then, for each active non-fresh requirement, execute the corresponding strategy (also see below). If the list of active requirements is empty or, during stage s, we have not discovered any cycle of length s while executing strategies for fresh or non-fresh R_{i_j}, then we add a fresh (i.e., composed from distinct least numbers $a_1, a_2, \ldots, a_s \notin \mathcal{A}$) cycle of length s to $\mathcal{A}$ and add the least inactive requirement to the list of active requirements and call it fresh.

Strategy for non-fresh $R_{i,j}$. Compute $c_{i_j}, p_{i_j}(c_{i_j}), p^2_{i_j}(c_{i_j}), \ldots, p^{s-1}_{i_j}(c_{i_j})$. The following outcomes are possible:

1. elements of the sequence are pairwise different from each other and then we shall call such a sequence a straight line of length s,
2. there is some l such that $1 \leq l \leq s-1$ and $p^l_{i_j}(c_{i_j}) = c_{i_j}$ and then for the least such l, we shall call $c_{i_j}, p_{i_j}(c_{i_j}), p^2_{i_j}(c_{i_j}), \ldots, p^l_{i_j}(c_{i_j})$ a cycle of length l,
3. there are some l and l' such that $1 \leq l < l' \leq s-1$ and $p^l_{i_j}(c_{i_j}) = p^{l'}_{i_j}(c_{i_j})$ and then for the least such l' we shall call such sequence a cycle with a tail of length $l'+1$.

If precisely after $s-1$-th iteration of function p_{i_j} on its corresponding witness we close a cycle or a cycle with a tail, then we deactivate R_{i_j}.

Strategy for fresh $R_{i,j}$. We define α_p to be the sequence $a_p, p_{i_j}(a_p), p^2_{i_j}(a_p), \ldots, p^{s-1}_{i_j}(a_p)$, for $p = 1, \ldots s$, where a_p is the least number which is currently not in the domain of $\mathcal{A}$ and has not appeared anywhere in calculations of any of the sequences α_p.

Now we check if any of the following outcomes occurs and act accordingly:

- if some α_p is a line of length s, set $c_{i_j} := a_p$ as the witness for R_{i_j} and call R_{i_j} non-fresh,
- if some α_p is a cycle of length s or a cycle with a tail of any length, deactivate R_{i_j},
- if for every α_p we obtained a cycle of length smaller than s, deactivate R_{i_j}.

Verification

▶ **Lemma 13.** *$\mathcal{A}$ is a computable simple cyclic injection structure.*

Proof. Clearly, $\mathcal{A}$ is computable. It is a cyclic injection structure since whenever we add new elements to the structure, they are in a cycle. A cycle of length s can only be added at stage s and only once. Hence, the structure is simple. ◀

▶ **Lemma 14.** *If a requirement R_i is deactivated during some stage s, then from that moment it always remains satisfied.*

Proof. There are several possible cases of how R_i was deactivated at stage s. Below we consider all of them:

- We discovered a cycle with a tail generated by the function p_i starting from its corresponding witness. In this case R_i is satisfied because p_i is not a cyclic function,
- We discovered a cycle of length s generated by p_i during stage s. In this case R_i is satisfied because we do not have a cycle of such length in $\mathcal{A}$ and hence $\mathcal{A}$ is not isomorphic to $(\mathbb{N}, p_i)$.
- We tried s different witnesses $a_1, \ldots, a_s$ for R_i and for each potential witness a we generated a cycle of length shorter than s starting with a. As a consequence of the pigeon-hole principle it is necessary that there are some witnesses a and b generating cycles of the same length. Furthermore, whenever we choose a new witness, we take a number which has not appeared anywhere in earlier calculations. Hence, the cycle containing a and the cycle containing b are not the same and they are both in $(\mathbb{N}, p_i)$. We conclude that this is not a simple cyclic injection structure and hence R_i is satisfied. ◀

▶ **Lemma 15.** *If a requirement R_i is not deactivated at any stage, then it is satisfied.*

Proof. We assume that R_i was activated at some stage s. Then at stage $s+1$ it acted as a fresh requirement and it was deactivated unless we discovered a witness c_i such that the sequence $c_i, p_i(c_i), p_i^2(c_i), \ldots, p_i^s(c_i)$ is a straight line of length $s+1$. In that case, at every stage $t \geq s+1$, requirement R_i is deactivated if and only if we discover a cycle or a cycle with a tail of length t starting with c_i. If this does not happen at any stage, then the line starting with c_i is becoming longer at every stage and never closes. Hence, $(\mathbb{N}, p_i)$ contains an $\mathbb{N}$-chain with an element c_i. So this structure is not cyclic and R_i is satisfied. ◀

▶ **Lemma 16.** *The domain of $\mathcal{A}$ is $\mathbb{N}$.*

Proof. Whenever we add a new element to $\mathcal{A}$, it is the least natural number which is currently not there. It follows that the domain of $\mathcal{A}$ is either all of $\mathbb{N}$ or some initial segment of it.

It remains to show that the $dom(\mathcal{A})$ is infinite. Suppose not. Let a be the largest (as a number) element in $\mathcal{A}$. Suppose that a was added at some stage s and that at the end of stage s the only active requirements were $R_{i_1}, \ldots, R_{i_k}$. We consider stages $s+1, \ldots, s+i_k+1$. We observe that there are i_k+1 of them and this is more than the number of active requirements. Since each requirement gets deactivated at most once, it follows that there is some t such that $s+1 \leq t \leq s+i_k+1$ such that none of the requirements is deactivated at stage t. This implies that at stage t we have not discovered any new cycle of length t. Hence at stage t we add a cycle of length t to $\mathcal{A}$ and this cycle consists of new elements larger than a. Hence a is not the largest natural number in $\mathcal{A}$ contrary to our assumption. ◀

▶ **Lemma 17.** *$\mathcal{A}$ is not punctually presentable.*

Proof. Suppose not. Then $\mathcal{A} \cong (\mathbb{N}, p_i)$, for some $i \in \mathbb{N}$. By Lemma 13, $\mathcal{A}$ is a simple cyclic injection structure. By Lemmas 14 and 15, R_i is satisfied. Since $(\mathbb{N}, p_i)$ is also a simple cyclic injection structure, there is some cycle length occurring in p_i but not in f. This contradicts the assumption that $(\mathbb{N}, f) \cong (\mathbb{N}, p_i)$. Hence, $\mathcal{A}$ is not punctually presentable. ◀

5 Functional trees

In this section we prove the main result of the paper:

▶ **Theorem 5.** *The class of functional trees is not punctually robust.*

▶ **Definition 18** (functional tree)**.** *Let $A \neq \emptyset$ be a set and let $T : A \to A$. (A, T) is a* functional tree*, if there is a unique r such that $T(r) = r$, and for every $x \in A$ there exists $i \in \mathbb{N}$ such that $T^i(x) = r$. The unique r is called the root, and is denoted by $r(A, T)$.*

Before we start, we need several technical notions.

First, observe that a functional tree (P, T) may be viewed as a partial order $(P, \leq)$ defined as follows: $x \leq y \Leftrightarrow \exists i \in \mathbb{N}\, T^i(x) = y$. This allows us to use a convenient order-theoretic notation when speaking about trees. However, we should keep in mind that this is just a manner of speaking and that in this section we deal with functional trees. Given a partial order $(P, \leq)$, we define $P_{\leq x} = \{y \in P : y \leq x\}$ and $P_{\geq x} = \{y \in P : y \geq x\}$. The corresponding strict partial order is denoted by $<$. Elements $x, y \in P$ are adjacent, $Adj(x, y)$, if and only if $x < y \wedge \neg\exists z \in P\, x < z < y$. To avoid confusion, we sometimes write $\leq_T$ or Adj_T to indicate that the ordering or adjacency relation is induced by T.

▶ **Definition 19** (branching node, branching, binary branching). *$x \in T$ is a* branching node *of T (or x branches in T) if it has at least two children in T. Such an x induces a unique subtree of T, called a branching and defined as $br(x,T) = \{y \in T : Adj(y,x)\} \cup T_{\geq x}$. If x is a branching node, we define $|br(x,t)|$, the length of $br(x,t)$, as the length of $T_{\geq x}$. By a* binary branching *we mean a tree with exacly two leaves sharing a parent.*

▶ **Definition 20** (uniquely branching tree). *We say that a tree T is uniquely branching if for every $n \in \mathbb{N}$, there exists at most one branching node $x \in T$ such that $|T_{\geq x}| = n$.*

▶ **Definition 21** (level). *We say that a* branching *node $x \in T$ belongs to the level n of T if the set $T_{>x}$ contains precisely n branching nodes. We denote the level n of T by $T[n]$. We sometimes refer to the members of $T[n]$ as n-level nodes.*

Keep in mind that we number levels $0, 1, \ldots$. Therefore, the first level is level 0. Level $T[n]$ may be empty but if $T[n] \neq \emptyset$ then $T[k] \neq \emptyset$, for $k < n$.

▶ **Definition 22** (i-level subtree). *Let T be a tree such that $T[i] \neq \emptyset$. We define $T[\leq i]$ as the least subtree of T containing all nodes at levels $\leq i$ together with their children.*

Notice that $T[\leq i]$ is the sum of the branchings $br(x,T)$ for all $x \in T[j]$ such that $j \leq i$.

Recall that a binary tree is a tree in which every internal node has at most two children. In a proper binary tree, every internal node has exactly two children.

▶ **Lemma 23.** *Let $(T, \leq)$ be a finite proper binary tree and let $F \subseteq T$ be such that, at every level of T except the first, at most one node is in F. Then there exists a leaf $x \in T$ such that $T_{\geq x} \cap F = \emptyset$.*

Proof. By assumption, $r(T) \notin F$. Suppose we have a path $x_n, x_{n-1}, \ldots, x_0 = r(T)$ such that $x_i \notin F$, for $i = 0, 1, \ldots, n$. If x_n is a leaf, we are done. If not, x_n has two children. These two children are at the same level and one of them is outside F. Let x_{n+1} to be a child outside F. ◁

▶ **Definition 24** (attaching a tree to a leaf). *Let $T, \hat{T}$ be disjoint finite trees and let $z \in T$ be a leaf. T' is obtained from T by attaching $\hat{T}$ to z in T if $dom(T') = dom(T) \cup (dom(\hat{T}) \setminus \{r(\hat{T})\})$ and $x, y \in dom(T')$ satisfy $Adj_{T'}(x,y)$ iff $Adj_T(x,y) \vee Adj_{\hat{T}}(x,y) \vee Adj_{\hat{T}}(x, r(\hat{T})) \wedge y = z$.*

▶ **Definition 25** (functional semitree). *Let A be a finite set and let $T : A \to A$ be a partial function. We say that (A,T) is a* functional semitree *if there is a unique r such that $T(r) = r$, and $(A, \chi_T \setminus \{(r,r)\})$ is an acyclic directed graph. The unique r is called the root, and is denoted by $r(A,T)$.*

The difference between a semitree and a tree is that the former may contain nodes that are not connected to the root via any path. We will use the letters t and q, possibly with decorations, to refer to *finite* trees and to semitrees, respectively.

▶ **Definition 26** (rooted part of T). *$R(T)$ denotes the largest subtree of a semitree T.*

At every stage of the construction, the main tree T that we build is approximated by a finite tree. As we will see, the growth of T is modulated by a set of nodes F.

▶ **Definition 27** (closed node, open node). *A node $y \in T$ is closed at a given stage if $T_{\geq y} \cap F \neq \emptyset$ at that stage. A node y is open if y is not closed, that is $T_{\geq y} \cap F = \emptyset$.*

▶ **Definition 28** (leaf extension). *Let t be a tree with a distinguished set of nodes F. We say that t' is a* leaf extension *of a tree t (in symbols: $t' \sqsupseteq t$) if, for some $k > 0$, there exist disjoint (from t and from each other) finite trees $\tilde{t}_1, \ldots, \tilde{t}_k$, and pairwise distinct open leaves $x_1, \ldots, x_k \in t$, such that t' is obtained from t by simultaneously attaching each $\tilde{t}_i$ to x_i in t. We say that t' is a* proper leaf extension *of t (in symbols: $t' \sqsupset t$) if $t' \sqsupseteq$ and $t' \neq t$.*

▶ **Definition 29** (matching). *(h, t') is a* matching of q into t *(in symbols: $(h, t') : p \hookrightarrow t$), if $t' \sqsupseteq t$ and $h : R(q) \hookrightarrow t'$ is an embedding such that $img(h) \supseteq dom(t') \setminus dom(t)$. A matching (h, t') of q into t is proper, if $t' \sqsupset t$. We say that q is* (properly) matchable with t *if there exists a (proper) matching (h, t') of q into t.*

▶ **Lemma 30.** *Let P be an infinite tree and let t be a finite tree whose level n is the maximal one (that is, $t[n] \neq \emptyset$ and $t[n+1] = \emptyset$) and $t[\leq n] = t$. Let q be a finite subtree of P. If q is not matchable with t, then $P \ncong T$, for every infinite tree $T \supset t$ such that $T[\leq n] = t$.*

Proof. Suppose there is an infinite tree $T \supset t$ such that $T[\leq n] = t$ and $P \cong T$ via $h : P \to T$. Let $h(q)$ be the isomorphic image of q (hence, a subtree of T). Let $t' = T \restriction dom(t) \cup dom(h(q))$. t' is a leaf extension of t. Since q is a tree, $R(q) = q$. We observe that $(h \restriction q, t')$ is a matching of q into t. ◀

▶ **Definition 31** (outside branching). *Suppose q is matchable with t. We say that q* branches outside *t at x if x branches in q and $|q_{\geq x}|$ is greater than the length of every branching in t. We say that q branches outside t if it branches outside t for some x.*

▶ **Definition 32** (inside branching). *Suppose q is matchable with t. We say that q* branches inside *t at x if x branches in $R(q)$, $br(x, q)$ embeds in t and for every matching $(h, t') : q \hookrightarrow t$, $h(q_{\leq x}) \cap t' \setminus t \neq \emptyset$ (i.e., h maps some descendants of x to $t' \setminus t$).*

We are ready to start the proof of Theorem 5.

We build an infinite computable functional tree T with domain $\mathbb{N}$ such that, for all $i \in \mathbb{N}$, the following requirements R_i are satisfied: $\mathcal{T} \ncong \mathcal{P}_i$, where $\mathcal{P}_i = (\mathbb{N}, p_i)$. T will be binary, uniquely branching and it will grow only through its leaves. T will be approximated by a sequence of finite trees $T_0 \subseteq T_1 \subseteq \ldots$, with $T = \bigcup_{s \in \mathbb{N}} T_s$. Occasionally, when it does not lead to confusion, we take the liberty to use the symbol T to refer to T at a given stage s, i.e. to T_s. During the construction, we approximate $\mathcal{P}_i$ by looking at certain specific finite approximations of $\mathcal{P}_i$, which we call anticipations. We make this notion precise in Definition 34. For now, it sufficient to know that an anticipation of $\mathcal{P}_i$ simply looks at portions of $\mathcal{P}_i$ that are large compared with the current approximation of $\mathcal{T}$. We simultaneously build a dynamic set $F \subset T$. We will put elements into F but will never withdraw them.

▶ Remark 33. An approximation $\mathcal{P}_{i,s} = (A, p_i \restriction A)$ of a functional tree $\mathcal{P}_i$ may actually be a semi-tree. This is because A may contain nodes which are connected to the root via nodes from $\mathbb{N} \setminus A$.

Satisfaction of R_i will be based on two strategies that we describe below. In the description of these strategies, we mention a tree t. The role of this tree will become clear when we describe the *StrategySelection* procedure. When a strategy is called, t is given as one of the inputs and used t to compute anticipations of $\mathcal{P}_i$. These anticipations depend on the number of stage s and t, hence their name: (s, t)-anticipations.

Outside-branch Strategy

We say that the outside-branch strategy for p_i is ready at stage s for a tree t if the (s, t)-anticipation q of p_i is a functional semitree and q branches outside T_{s-1}. The strategy is called only when it is ready in the specified sense. The outside-branch strategy is called with arguments q, T_{s-1}, where q is some (s, t)-anticipation of p_i that branches outside T_{s-1}.

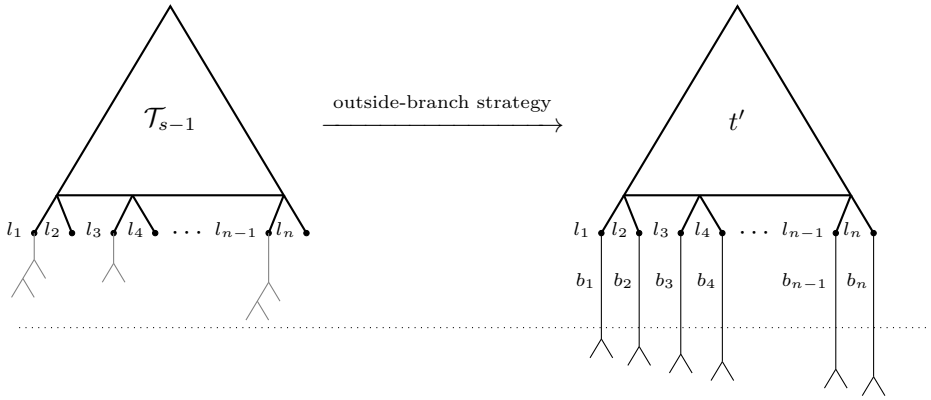

Figure 1 Outside-branch strategy.

The idea of the strategy is as follows (see, also, Figure 1). By the definition of the outside-branching, q contains a branching b that is longer than every branching in T_{s-1} (hence the word "outside-branching"). If we wanted to embed b into T, we would have to extend T_{s-1} to include the branching length of b in the extension of T_{s-1}. We purposely extend T_{s-1} to a leaf extension $t' \sqsupset T_{s-1}$ that omits the branching length of b. Recall that the construction is arranged in such a way that T grows only through its leaves and whenever we add new branchings into T, their length is greater than any branching that was in T so far. Hence, once we omit a given branching length in T, T will not have any branching of this length at all. Therefore, if we set $T_s = t'$, we would kill the potential isomorphism $\mathcal{T} \cong \mathcal{P}_i$ permanently.

More precisely, we produce t' in the following way. Let $d = max\{|q_{\geq x}| : x \text{ branches in } q\}$. Let $\{l_1, \ldots, l_n\}$ be the set of all open leaves of T_{s-1}. We make binary branchings $b_1, \ldots, b_n$ such that for all i, j, $|(T_{s-1})_{\geq l_i}| + |b_i| > d$ and $i \neq j \Rightarrow |(T_{s-1})_{\geq l_i}| + |b_i| \neq |(T_{s-1})_{\geq l_j}| + |b_j|$. The strategy outputs t' obtained from T_{s-1} by simultaneous attachment of each b_i to l_i. We make sure t' is extended by the least fresh numbers not occurring in T_{s-1}.

The outside-branch strategy is illustrated in Figure 1. T_{s-1} is shown on the left in black. The tree $R(q)$ (or rather a leaf extension of T_{s-1} into which $R(q)$ embeds) is shown explicitly only in part using gray subtrees on the left. The output of the outside-branch strategy is t' shown on the right which has very long branchings attached to $l_1, \ldots, l_n$, the open leaves of T_{s-1}, so that t' omits some branching lengths of $R(q)$. The horizontal dotted line indicates that the lengths of the branchings below it are greater than the lengths of all the branchings in the tree on the left.

Inside-branch Strategy

We say that the inside-branch strategy for p_i is ready at stage s for a tree t if the (s,t)-anticipation q of p_i and T_{s-1} satisfy the following conditions:

$$q \text{ is a functional semitree,} \tag{1}$$

$$T_{s-1} \text{ and } R(q) \text{ have level } i+1, \tag{2}$$

$$R(q)[\leq i+1] \cong T[\leq i+1], \tag{3}$$

$$q \text{ branches inside } T_{s-1}. \tag{4}$$

The strategy is called only when it is ready in the specified sense. We call it with arguments q, T_{s-1}, where q is some (s,t)-anticipation of p_i that branches inside T_{s-1}.

The idea of the inside-branch strategy is as follows. By the definition of the inside-branching, q contains a branching node x such that $br(x, q)$ embeds into T_{s-1} (hence the word "inside-branching"). Since T_{s-1} is uniquely branching, there is only one way of embedding this branching to T_{s-1}. Suppose this unique embedding maps x to $y \in T_{s-1}$. By the definition of the inside-branching, there is no way of embedding the subtree $q_{\leq x}$ into the subtree $(T_{s-1})_{\leq y}$ without prolonging some paths in the subtree $(T_{s-1})_{\leq y}$. Hence, if we decide to put y (or some of its ancestors) into F, we would stop growing $(T_{s-1})_{\leq y}$, that is, we would have (in the limit) $T_{\leq y} = (T_{s-1})_{\leq y}$, thus killing the isomorphism $\mathcal{T} \cong \mathcal{P}_i$ permanently.

The details of the strategy follow. Suppose that the inside-branch strategy for p_i is ready and thus (1)-(4) hold. The strategy will stop the growth of T from exactly one node at level $i+1$ of T. Since q branches inside T_{s-1}, let x be a node of T_{s-1} at which q branches inside T_{s-1}. If x belongs to level $j > i+1$, then the path connecting x to the root must intersect level $i+1$ and the point of intersection is the branching node that we put into F; note that this point is unique because T is uniquely branching and (2) and (3) hold. If x belongs to level $i+1$, we simply put x to F; again, this x is unique by the same reasoning. If x belongs to level $\leq i$ but not to level $i+1$, it means that T_{s-1} did not have chance to grow up to level $i+1$ from x (and will not have such a chance anymore, because we never withdraw elements from F) and therefore we do not have to take any action.

Construction

The construction is arranged in stages $s = 0, 1, \ldots$. For $s = 0$, we set $T_0(0) = 0$ with domain $\{0\}$ and $F = \emptyset$. No R_i is satisfied at stage 0. At each subsequent stage $s > 0$, we start with a finite tree T_{s-1}. We make a call to $StrategySelection(s, T_{s-1})$ and after it finishes working, we go to the next stage. This ends the construction.

StrategySelection is a recursive procedure that selects which strategy to perform and with which arguments. It uses a specific way of growing our approximations.

▶ **Definition 34** (anticipation). *Let $p : \mathbb{N} \to \mathbb{N}$, $s \in \mathbb{N}$ and let T be a finite tree. Let $C = [0, s) \cup [0, |T| + 1)$ and $D = \bigcup_{l=0}^{H(T)+s} p^l(C)$. We say that $p \restriction D$ is the (s, T)-anticipation of p.*

We say that p_i is ready at stage s for a tree t if either the outside-branch or the inside-branch strategy for p_i is ready at stage s for t.

***StrategySelection*(n, t).** Check whether there exists an unsatisfied R_i with $i < n$ such that p_i is ready at stage s for t. If there is no such i, return from the current call to *StrategySelection* with permission. Otherwise, let $k < n$ be the least such i.

If the (s, t)-anticipation of p_k branches inside T_{s-1}, perform the inside-branch strategy with the (s, t)-anticipation of p_k, T_{s-1} as arguments (we say that p_k triggers the inside-branch strategy). Declare R_i as satisfied and return from the current call to *StrategySelection* without permission.

If it is not the case that the (s, t)-anticipation of p_k branches inside T_{s-1}, then the (s, t)-anticipation of p_k branches outside T_{s-1}. In that case we let t' be the output of the outside-branch strategy with the (s, t)-anticipation of p_k and T_{s-1} as arguments, and we call $StrategySelection(k, t')$ (we say that p_k asks for permission). If the call returns with permission, we set $T_s = t'$ (we say that p_k triggers the outside-branch strategy), declare that R_k is satisfied, and we return without permission.

Verification

It is clear that T is computable and that if it is infinite then its domain is equal to $\mathbb{N}$.

▶ **Lemma 35.** *T is infinite and has infinitely many branchings of distinct length.*

Proof. We show by induction on the number of levels that T has infinitely many levels.

Suppose the current T has no levels at all, i.e. the domain of T is $\{0\}$. 0 is not a branching node of T and therefore it cannot trigger the inside-branch strategy. So $0 \notin F$ forever. However, at some stage $u \geq s$, we will see p_i, with $i < u$, such that the anticipation of p_i will branch outside T and since 0 will be an open leaf of T at that stage, it will trigger the outside-branch strategy and add level 0 to T. If T has only one level, i.e., level 0, the unique branching node in T will never be put into F (because the inside-branch strategy may affect only levels ≥ 1). As above, we can show that T will grow up to level 1.

Now, let us suppose that T has exactly n levels, where $n > 1$ (that is, levels $0, 1, \ldots, n-1$). The construction may put into F at most one node from each of the levels $1, 2, \ldots, n-1$. Once it does, it never withdraws them from F. Suppose we are at a stage s_0 after which the construction do not add any new nodes from levels $1, 2, \ldots, n-1$ into F. This means that it will never be the case that, for some $0 \leq i \leq n-2$, R_i is unsatisfied and the inside-branch strategy for p_i is ready (if we later encountered such an i, the inside-branch strategy would put some element from levels $1, 2, \ldots, n-1$ into F, contradicting our choice of s_0). If T has grown by the time we reached stage s_0, we are done. So suppose otherwise. Clearly, at some later stage s, some p_j, $j < s$, with unsatisfied R_j will show us an anticipation q such that q is a functional semitree and q branches outside T. Choose the least such j at stage s. Clearly, the outside-branch strategy for p_j is ready at stage s (and R_j is unsatisfied). Moreover, no outside-branch strategy for p_k with $k < j$ is ready because of the choice of j. As for the inside-branch strategy for p_k with $k < j$, either R_k is satisfied or the inside-branch strategy for p_k not ready – this follows from the choice of s_0. Therefore, p_k will be granted permission. By the arrangement of the inside-branch strategy and by Lemma 23, T has open leaves. Therefore, the outside-branch strategy for p_k will extend T by another level.

We have already shown that the outside-branch strategy acts infinitely often. Since it always adds new branchings, there must be infinitely many branchings in T. Clearly, the strategy also guarantees that any two different branchings do not have the same length. ◀

▶ **Lemma 36.** *For every $i \in \mathbb{N}$, if $\mathcal{P}_i$ is a functional tree, then R_i is eventually satisfied.*

Proof. Fix a punctual structure $\mathcal{P}_i$ and assume that there exists a stage s_0 after which for every $j < i$, either R_i is satisfied or p_j is not ready (this holds vacuously for $i = 0$). Suppose that $\mathcal{P}_i$ is a uniquely branching functional tree (if not, $\mathcal{T} \not\cong \mathcal{P}_i$). We will show that R_i will be eventually satisfied.

Choose a stage s_0 as above and assume that $i < s_0$. Suppose that at the beginning of s_0 the requirement R_i has not yet been declared as satisfied (that is, neither of the two strategies has been triggered before s_0).

If at some stage $u \geq s_0$, we see that an anticipation q of p_i is not matchable with T_{u-1}, then R_i is satisfied by Lemma 30. In the lemma, we take $P = p_i$, $t = T_{u-1}$ and q. Let n be the maximal level of t. By the construction $t[\leq n] = t$. The infinite tree T that we are building will be such that $T \supset T_{u-1} = t$ and $T[\leq n] = t$. Hence, the premises of the lemma are satisfied. It means that $\mathcal{T} \not\cong \mathcal{P}_i$ and thus R_i is satisfied. We can therefore assume that, for $u \geq s_0$, any considered anticipation of p_i is matchable with T_{u-1}.

Now, if at some stage $u \geq s_0$ the requirement R_i is still not satisfied but p_i is ready at stage u for T_{u-1}, i will be the least j with that property. So, according to *StrategySelection*, either it will trigger an inside-branch strategy, or it will ask for permission and receive it, thus

triggering an outside-branch strategy. Now, it may happen that p_i becomes ready because the outside-branch strategy for p_i is ready but the inside-branch strategy is not. In that case, according to the *StrategySelection*, p_i will receive permission and thus R_i will become satisfied.

So assume p_i never becomes ready for this reason after stage s_0. Observe that if anticipations of p_i and T_s always fail to satisfy (1), (2) or (3), then $\mathcal{T} \not\cong \mathcal{P}_i$. So assume that after stage s_0 these conditions are always satisfied. Therefore, it remains to prove that at some stage $u \geq s_0$, the inside-branch strategy is triggered.

It is important to note that once the level $i+1$ of T is done, T does not change up to level $i+1$ at any later stage (it may only add some nodes from levels $\leq i+1$ to F but this will not change the structure of T up to level $i+1$).

Consider stage $u \geq s_0$. The construction calls *StrategySelection*(u, T_{u-1}). Let q be the initial anticipation of p_i at stage u, namely the (u, T_{u-1})-anticipation of p_i. By the definition of anticipation, q contains more nodes than T_{u-1}. Therefore, the situation in which $R(q) = q$ and $R(q) \hookrightarrow T_{u-1}$ is not possible. So we have two remaining cases:

(a) $R(q) \hookrightarrow T_{u-1}$ and $R(q) \neq q$. In this case, there are nodes $z \in q \setminus R(q)$ that are disconnected from the root. Moreover, by the definition of anticipation, if $z \in q \setminus R(q)$ and z does not have children in q, then z gives rise to a chain of length $H(T_{u-1}) + u$ in q.

(b) $R(q) \not\hookrightarrow T_{u-1}$. In this case, q must be properly matchable with T_{u-1}. To see why, recall that q is matchable with T_{u-1}. So $R(q)$ embeds in a leaf extension T' of T_{u-1}. But any such leaf extension must be proper – otherwise $R(q) \hookrightarrow T_{u-1}$. From this it follows that q branches inside T_{u-1}. For take a proper matching of q into T_{u-1} that embeds q into $T' \sqsupset T_{u-1}$. Take the least $z \in T' \setminus T_{u-1}$. This z is connected to the root of T_{u-1} via a path that overlaps with initial $i+1$ levels of T_{u-1}. But every such path (the one in T_{u-1} or its preimage in q) contains branching nodes because q contains a copy of initial $i+1$ levels of T_{u-1}. Therefore, q branches inside T_{u-1}, and thus q is ready at stage u for T_{u-1}. Hence the inside branch strategy will handle R_i.

Assume that (b) holds. Therefore, the inside-branch strategy for p_i is ready at stage u for T_{u-1}. Hence, R_i becomes satisfied immediately after *StrategySelection* is called.

Assume that we are in case (a). Then p_i is not ready at stage u for T_{u-1}. But other p_j, with $j > i$ will become ready later on, say at stage v. If such p_j triggers an inside-branch strategy to satisfy R_j, our tree does not grow and we therefore do not miss an opportunity to satisfy R_i. However, if p_j, with $j > i$ wants to trigger an outside-branch strategy (which must happen eventually), it will produce $T' \supset T_{u-1}$ and ask for permission. So before p_j is allowed to act by extending the current tree from T_{u-1} to $T' \supset T_{u-1}$, we first check whether p_i is ready at stage v for the (bigger) tree T' that we would grow if we allowed p_j to act first. If it is not ready, it means that we can permit p_j to trigger an outside-branch strategy and set $T_v = T'$. For it means that all the long disconnected chains that we had in q remain disconnected, even after prolonging them up to length $H(T') + v$ in the (v, T')-anticipation of p_i. Notice that if we did not insist on asking for permission, we would set $T_v := T' \supset T_{u-1}$ and it could happen that prolonging a bit the disconnected chains in q would change q into a tree that embeds into T_v and we would miss the opportunity to satisfy R_i. Asking for permission prevents this.

Finally, it suffices to observe that, since p_i is a tree, at some point these disconnected chains will connect to the root, either at the beginning of *StrategySelection* or after being asked for permission by some other p_j, with $j > i$. Since these chains are kept very long (longer than any path in the current T), once they become connected, they necessarily stick out of T and thus branch inside it. At this point the inside-branch strategy is triggered and R_i is satisfied. ◀

References

1 Pavel E. Alaev and Viktor L. Selivanov. Polynomial computability of fields of algebraic numbers. *Doklady Mathematics*, 98(1):341–343, 2018. doi:10.1134/S1064562418050137.

2 Nikolay Bazhenov, Rod Downey, Iskander Kalimullin, and Alexander Melnikov. Foundations of online structure theory. *Bulletin of Symbolic Logic*, 25(2):141–181, 2019. Publisher: Cambridge University Press. doi:10.1017/bsl.2019.20.

3 Nikolay Bazhenov, Iskander Kalimullin, Alexander Melnikov, and Keng Meng Ng. Online presentations of finitely generated structures. *Theoretical Computer Science*, 844:195–216, 2020. doi:10.1016/j.tcs.2020.08.021.

4 Nikolay Bazhenov, Keng Meng Ng, Luca San Mauro, and Andrea Sorbi. Primitive recursive equivalence relations and their primitive recursive complexity. *Computability*, 11(3-4):187–221, 2022. doi:10.3233/COM-210375.

5 Nilkolay Bazhenov, Matthew Harrison-Trainor, Iskander Kalimullin, Alexander Melnikov, and Keng Meng Ng. Automatic and polynomial-time algebraic structures. *The Journal of Symbolic Logic*, 84(4):1630–1669, 2019. doi:10.1017/jsl.2019.26.

6 Douglas Cenzer, Valentina Harizanov, and Jeffrey B. Remmel. Computability-theoretic properties of injection structures. *Algebra and Logic*, 53(1):39–69, 2014. doi:10.1007/s10469-014-9270-0.

7 Douglas Cenzer and Jeffrey B. Remmel. Polynomial-time versus recursive models. *Annals of Pure and Applied Logic*, 54(1):17–58, 1991. doi:10.1016/0168-0072(91)90008-A.

8 Douglas Cenzer and Jeffrey B. Remmel. Polynomial-time abelian groups. *Annals of Pure and Applied Logic*, 56(1):313–363, 1992. doi:10.1016/0168-0072(92)90076-C.

9 Douglas Cenzer and Jeffrey B. Remmel. Feasible graphs with standard universe. *Annals of Pure and Applied Logic*, 94(1):21–35, 1998. doi:10.1016/S0168-0072(97)00064-X.

10 Rod Downey, Noam Greenberg, Alexander Melnikov, Keng Meng Ng, and Daniel Turetsky. Punctual categoricity and universality. *The Journal of Symbolic Logic*, 85(4):1427–1466, 2020. doi:10.1017/jsl.2020.51.

11 Noam Greenberg, Matthew Harrison-Trainor, Alexander Melnikov, and Dan Turetsky. Non-density in punctual computability. *Annals of Pure and Applied Logic*, 172(9):102985, 2021. doi:10.1016/j.apal.2021.102985.

12 Iskander Kalimullin, Alexander Melnikov, and Keng Meng Ng. Algebraic structures computable without delay. *Theoretical Computer Science*, 674:73–98, 2017. doi:10.1016/j.tcs.2017.01.029.

13 Bakhadyr Khoussainov and Anil Nerode. Automatic presentations of structures. In Daniel Leivant, editor, *Logic and Computational Complexity*, volume 960 of *Lecture Notes in Computer Science*, pages 367–392. Springer Berlin Heidelberg, 1995. doi:10.1007/3-540-60178-3_93.

14 Anatolii I. Mal'tsev. Constructive algebras i. *Russian Mathematical Surveys*, 16(3):77, 1961. doi:10.1070/RM1961v016n03ABEH001120.

15 Antonio Montalbán. *Computable structure theory: Within the arithmetic*. Cambridge University Press, 2021.

16 Michael O. Rabin. Computable algebra, general theory and theory of computable fields. *Transactions of the American Mathematical Society*, 95(2):341–360, 1960. doi:10.2307/1993295.